Sage Reference Manual: Schemes

Release 6.8

The Sage Development Team

July 29, 2015

1 Scheme implementation overview **1**
1.1 TODO List . . . 2

2 Schemes **3**

3 The Spec functor **13**

4 Scheme obtained by gluing two other schemes **15**

5 Points on schemes **17**

6 Ambient Spaces **19**

7 Algebraic schemes **23**

8 Hypersurfaces in affine and projective space **51**

9 Set of homomorphisms between two schemes **53**

10 Scheme morphism **59**

11 Divisors on schemes **69**

12 Divisor groups **73**

13 Affine Schemes **75**
13.1 Affine n space over a ring . . . 75
13.2 Points on affine varieties . . . 80
13.3 Morphisms on affine varieties . . . 84
13.4 Enumeration of rational points on affine schemes . . . 95
13.5 Set of homomorphisms between two affine schemes . . . 97

14 Projective Schemes **101**
14.1 Projective n space over a ring . . . 101
14.2 Points on projective varieties . . . 108
14.3 Morphisms on projective varieties . . . 120
14.4 Morphisms on projective varieties (Cython helper) . . . 159
14.5 Enumeration of rational points on projective schemes . . . 159
14.6 Set of homomorphisms between two projective schemes . . . 162
14.7 Automorphism groups of endomorphisms of the projective line . . . 165
14.8 Sage functions to compute minimal models of rational functions . . . 175

15 Products of Projective Spaces **179**
15.1 Products of projective spaces 179
15.2 Set of homomorphisms 184
15.3 Polynomial morphisms for products of projective spaces 185
15.4 Points for products of projective spaces 185
15.5 Wehler K3 Surfaces 187

16 Toric Varieties **207**
16.1 Toric varieties 207
16.2 Fano toric varieties 241
16.3 Library of toric varieties 258
16.4 Toric divisors and divisor classes 273
16.5 Toric rational divisor classes 292
16.6 The Chow group of a toric variety 294
16.7 Toric ideals 305
16.8 Morphisms of Toric Varieties 308
16.9 Weierstrass form of a toric elliptic curve. 328
16.10 Map to the Weierstrass form of a toric elliptic curve. 337
16.11 Weierstrass for Elliptic Curves in Higher Codimension 341
16.12 Set of homomorphisms between two toric varieties. 343
16.13 Enumerate Points of a Toric Variety 348

17 Indices and Tables **359**

Bibliography **361**

CHAPTER

ONE

SCHEME IMPLEMENTATION OVERVIEW

Various parts of schemes were implemented by Volker Braun, David Joyner, David Kohel, Andrey Novoseltsev, and William Stein.

AUTHORS:

- David Kohel (2006-01-03): initial version
- William Stein (2006-01-05)
- William Stein (2006-01-20)
- Andrey Novoseltsev (2010-09-24): update due to addition of toric varieties.
- **Scheme:** A scheme whose datatype might not be defined in terms of algebraic equations: e.g. the Jacobian of a curve may be represented by means of a Scheme.
- **AlgebraicScheme:** A scheme defined by means of polynomial equations, which may be reducible or defined over a ring other than a field. In particular, the defining ideal need not be a radical ideal, and an algebraic scheme may be defined over Spec(R).
- **AmbientSpaces:** Most effective models of algebraic scheme will be defined not by generic gluings, but by embeddings in some fixed ambient space.
- **AffineSpace:** Affine spaces and their affine subschemes form the most important universal objects from which algebraic schemes are built. The affine spaces form universal objects in the sense that a morphism is uniquely determined by the images of its coordinate functions and any such images determine a well-defined morphism.

  By default affine spaces will embed in some ordinary projective space, unless it is created as an affine patch of another object.
- **ProjectiveSpace:** Projective spaces are the most natural ambient spaces for most projective objects. They are locally universal objects.
- **ProjectiveSpace_ordinary (not implemented)** The ordinary projective spaces have the standard weights $[1, .., 1]$ on their coefficients.
- **ProjectiveSpace_weighted (not implemented):** A special subtype for non-standard weights.
- **ToricVariety:** Toric varieties are (partial) compactifications of algebraic tori $(\mathbf{C}^*)^n$ compatible with torus action. Affine and projective spaces are examples of toric varieties, but it is not envisioned that these special cases should inherit from `ToricVariety`.
- **AlgebraicScheme_subscheme_affine:** An algebraic scheme defined by means of an embedding in a fixed ambient affine space.
- **AlgebraicScheme_subscheme_projective:** An algebraic scheme defined by means of an embedding in a fixed ambient projective space.

- **QuasiAffineScheme (not yet implemented):** An open subset $U = X \setminus Z$ of a closed subset X of affine space; note that this is mathematically a quasi-projective scheme, but its ambient space is an affine space and its points are represented by affine rather than projective points.

Note: AlgebraicScheme_quasi is implemented, as a base class for this.

- **QuasiProjectiveScheme (not yet implemented):** An open subset of a closed subset of projective space; this datatype stores the defining polynomial, polynomials, or ideal defining the projective closure X plus the closed subscheme Z of X whose complement $U = X \setminus Z$ is the quasi-projective scheme.

Note: The quasi-affine and quasi-projective datatype lets one create schemes like the multiplicative group scheme $\mathbb{G}_m = \mathbb{A}^1 \setminus \{(0)\}$ and the non-affine scheme $\mathbb{A}^2 \setminus \{(0,0)\}$. The latter is not affine and is not of the form $\mathrm{Spec}(R)$.

1.1 TODO List

- **PointSets and points over a ring:** For algebraic schemes X/S and T/S over S, one can form the point set $X(T)$ of morphisms from $T \to X$ over S.

```
sage: PP.<X,Y,Z> = ProjectiveSpace(2, QQ)
sage: PP
Projective Space of dimension 2 over Rational Field
```

The first line is an abuse of language – returning the generators of the coordinate ring by `gens()`.

A projective space object in the category of schemes is a locally free object – the images of the generator functions *locally* determine a point. Over a field, one can choose one of the standard affine patches by the condition that a coordinate function $X_i \neq 0$

```
sage: PP(QQ)
Set of rational points of Projective Space
of dimension 2 over Rational Field
sage: PP(QQ)([-2,3,5])
(-2/5 : 3/5 : 1)
```

Over a ring, this is not true, e.g. even over an integral domain which is not a PID, there may be no *single* affine patch which covers a point.

```
sage: R.<x> = ZZ[]
sage: S.<t> = R.quo(x^2+5)
sage: P.<X,Y,Z> = ProjectiveSpace(2, S)
sage: P(S)
Set of rational points of Projective Space of dimension 2 over
Univariate Quotient Polynomial Ring in t over Integer Ring with
modulus x^2 + 5
```

In order to represent the projective point $(2 : 1 + t) = (1 - t : 3)$ we note that the first representative is not well-defined at the prime $pp = (2, 1+t)$ and the second element is not well-defined at the prime $qq = (1-t, 3)$, but that $pp + qq = (1)$, so globally the pair of coordinate representatives is well-defined.

```
sage: P( [2, 1+t] )
(2 : t + 1 : 1)
```

In fact, we need a test `R.ideal([2,1+t]) == R.ideal([1])` in order to make this meaningful.

CHAPTER

TWO

SCHEMES

AUTHORS:

- William Stein, David Kohel, Kiran Kedlaya (2008): added zeta_series
- Volker Braun (2011-08-11): documenting, improving, refactoring.

class sage.schemes.generic.scheme.**AffineScheme**(*R*, *S=None*, *category=None*)

Bases: sage.structure.unique_representation.UniqueRepresentation, sage.schemes.generic.scheme.Scheme

Class for general affine schemes.

TESTS:

```
sage: from sage.schemes.generic.scheme import AffineScheme
sage: A = QQ['t']
sage: X_abs = AffineScheme(A); X_abs
Spectrum of Univariate Polynomial Ring in t over Rational Field
sage: X_rel = AffineScheme(A, QQ); X_rel
Spectrum of Univariate Polynomial Ring in t over Rational Field

sage: X_abs == X_rel
False
sage: X_abs.base_ring()
Integer Ring
sage: X_rel.base_ring()
Rational Field
```

See also:

For affine spaces over a base ring and subschemes thereof, see sage.schemes.generic.algebraic_scheme.AffineSpace.

Element

alias of SchemeTopologicalPoint_prime_ideal

base_extend(*R*)

Extend the base ring/scheme.

INPUT:

- R – an affine scheme or a commutative ring

EXAMPLES:

```
sage: Spec_ZZ = Spec(ZZ);  Spec_ZZ
Spectrum of Integer Ring
sage: Spec_ZZ.base_extend(QQ)
Spectrum of Rational Field
```

coordinate_ring()

Return the underlying ring of this scheme.

OUTPUT:

A commutative ring.

EXAMPLES:

```
sage: Spec(QQ).coordinate_ring()
Rational Field
sage: Spec(PolynomialRing(QQ, 3, 'x')).coordinate_ring()
Multivariate Polynomial Ring in x0, x1, x2 over Rational Field
```

dimension()

Return the absolute dimension of this scheme.

OUTPUT:

Integer.

EXAMPLES:

```
sage: S = Spec(ZZ)
sage: S.dimension_absolute()
1
sage: S.dimension()
1
```

dimension_absolute()

Return the absolute dimension of this scheme.

OUTPUT:

Integer.

EXAMPLES:

```
sage: S = Spec(ZZ)
sage: S.dimension_absolute()
1
sage: S.dimension()
1
```

dimension_relative()

Return the relative dimension of this scheme over its base.

OUTPUT:

Integer.

EXAMPLES:

```
sage: S = Spec(ZZ)
sage: S.dimension_relative()
0
```

hom(*x*, *Y=None*)

Return the scheme morphism from `self` to `Y` defined by `x`.

INPUT:

- `x` – anything that determines a scheme morphism; if `x` is a scheme, try to determine a natural map to `x`

- Y – the codomain scheme (optional); if Y is not given, try to determine Y from context
- check – boolean (optional, default: True); whether to check the defining data for consistency

OUTPUT:

The scheme morphism from self to Y defined by x.

EXAMPLES:

We construct the inclusion from $\mathrm{Spec}(\mathbf{Q})$ into $\mathrm{Spec}(\mathbf{Z})$ induced by the inclusion from $\mathbf{Z}$ into $\mathbf{Q}$:

```
sage: X = Spec(QQ)
sage: X.hom(ZZ.hom(QQ))
Affine Scheme morphism:
  From: Spectrum of Rational Field
  To:   Spectrum of Integer Ring
  Defn: Ring Coercion morphism:
          From: Integer Ring
          To:   Rational Field
```

TESTS:

We can construct a morphism to an affine curve (trac ticket #7956):

```
sage: S.<p,q> = QQ[]
sage: A1.<r> = AffineSpace(QQ,1)
sage: A1_emb = Curve(p-2)
sage: A1.hom([2,r],A1_emb)
Scheme morphism:
  From: Affine Space of dimension 1 over Rational Field
  To:   Affine Curve over Rational Field defined by p - 2
  Defn: Defined on coordinates by sending (r) to
        (2, r)
```

is_noetherian()

Return True if self is Noetherian, False otherwise.

EXAMPLES:

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

class sage.schemes.generic.scheme.**Scheme**(*X=None*, *category=None*)

Bases: sage.structure.parent.Parent

The base class for all schemes.

INPUT:

- X – a scheme, scheme morphism, commutative ring, commutative ring morphism, or None (optional). Determines the base scheme. If a commutative ring is passed, the spectrum of the ring will be used as base.
- category – the category (optional). Will be automatically construted by default.

EXAMPLES:

```
sage: from sage.schemes.generic.scheme import Scheme
sage: Scheme(ZZ)
<class 'sage.schemes.generic.scheme.Scheme_with_category'>
```

A scheme is in the category of all schemes over its base:

```
sage: ProjectiveSpace(4, QQ).category()
Category of schemes over Rational Field
```

There is a special and unique $Spec(\mathbf{Z})$ that is the default base scheme:

```
sage: Spec(ZZ).base_scheme() is Spec(QQ).base_scheme()
True
```

base_extend(*Y*)

Extend the base of the scheme.

Derived clases must override this method.

EXAMPLES:

```
sage: from sage.schemes.generic.scheme import Scheme
sage: X = Scheme(ZZ)
sage: X.base_scheme()
Spectrum of Integer Ring
sage: X.base_extend(QQ)
Traceback (most recent call last):
...
NotImplementedError
```

base_morphism()

Return the structure morphism from `self` to its base scheme.

OUTPUT:

A scheme morphism.

EXAMPLES:

```
sage: A = AffineSpace(4, QQ)
sage: A.base_morphism()
Scheme morphism:
  From: Affine Space of dimension 4 over Rational Field
  To:   Spectrum of Rational Field
  Defn: Structure map

sage: X = Spec(QQ)
sage: X.base_morphism()
Scheme morphism:
  From: Spectrum of Rational Field
  To:   Spectrum of Integer Ring
  Defn: Structure map
```

base_ring()

Return the base ring of the scheme self.

OUTPUT:

A commutative ring.

EXAMPLES:

```
sage: A = AffineSpace(4, QQ)
sage: A.base_ring()
Rational Field

sage: X = Spec(QQ)
```

```
sage: X.base_ring()
Integer Ring
```

base_scheme()

Return the base scheme.

OUTPUT:

A scheme.

EXAMPLES:

```
sage: A = AffineSpace(4, QQ)
sage: A.base_scheme()
Spectrum of Rational Field

sage: X = Spec(QQ)
sage: X.base_scheme()
Spectrum of Integer Ring
```

coordinate_ring()

Return the coordinate ring.

OUTPUT:

The global coordinate ring of this scheme, if defined. Otherwise raise a ValueError.

EXAMPLES:

```
sage: R.<x, y> = QQ[]
sage: I = (x^2 - y^2)*R
sage: X = Spec(R.quotient(I))
sage: X.coordinate_ring()
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x^2 - y^2
```

count_points(n)

Count points over finite fields.

INPUT:

- n – integer.

OUTPUT:

An integer. The number of points over $\mathbf{F}_q, \ldots, \mathbf{F}_{q^n}$ on a scheme over a finite field $\mathbf{F}_q$.

Note: This is currently only implemented for schemes over prime order finite fields.

EXAMPLES:

```
sage: P.<x> = PolynomialRing(GF(3))
sage: C = HyperellipticCurve(x^3+x^2+1)
sage: C.count_points(4)
[6, 12, 18, 96]
sage: C.base_extend(GF(9,'a')).count_points(2)
[12, 96]
```

dimension()

Return the absolute dimension of this scheme.

OUTPUT:

Integer.

EXAMPLES:

```
sage: R.<x, y> = QQ[]
sage: I = (x^2 - y^2)*R
sage: X = Spec(R.quotient(I))
sage: X.dimension_absolute()
Traceback (most recent call last):
...
NotImplementedError
sage: X.dimension()
Traceback (most recent call last):
...
NotImplementedError
```

dimension_absolute()

Return the absolute dimension of this scheme.

OUTPUT:

Integer.

EXAMPLES:

```
sage: R.<x, y> = QQ[]
sage: I = (x^2 - y^2)*R
sage: X = Spec(R.quotient(I))
sage: X.dimension_absolute()
Traceback (most recent call last):
...
NotImplementedError
sage: X.dimension()
Traceback (most recent call last):
...
NotImplementedError
```

dimension_relative()

Return the relative dimension of this scheme over its base.

OUTPUT:

Integer.

EXAMPLES:

```
sage: R.<x, y> = QQ[]
sage: I = (x^2 - y^2)*R
sage: X = Spec(R.quotient(I))
sage: X.dimension_relative()
Traceback (most recent call last):
...
NotImplementedError
```

hom(*x*, *Y=None*, *check=True*)

Return the scheme morphism from `self` to `Y` defined by `x`.

INPUT:

- `x` – anything that determines a scheme morphism; if `x` is a scheme, try to determine a natural map to `x`
- `Y` – the codomain scheme (optional); if `Y` is not given, try to determine `Y` from context

- check – boolean (optional, default: True); whether to check the defining data for consistency

OUTPUT:

The scheme morphism from self to Y defined by x.

EXAMPLES:

```
sage: P = ProjectiveSpace(ZZ, 3)
sage: P.hom(Spec(ZZ))
Scheme morphism:
  From: Projective Space of dimension 3 over Integer Ring
  To:   Spectrum of Integer Ring
  Defn: Structure map
```

identity_morphism()

Return the identity morphism.

OUTPUT:

The identity morphism of the scheme self.

EXAMPLES:

```
sage: X = Spec(QQ)
sage: X.identity_morphism()
Scheme endomorphism of Spectrum of Rational Field
  Defn: Identity map
```

point(*v*, *check=True*)

Create a point.

INPUT:

- v – anything that defines a point
- check – boolean (optional, default: True); whether to check the defining data for consistency

OUTPUT:

A point of the scheme.

EXAMPLES:

```
sage: A2 = AffineSpace(QQ,2)
sage: A2.point([4,5])
(4, 5)

sage: R.<t> = PolynomialRing(QQ)
sage: E = EllipticCurve([t + 1, t, t, 0, 0])
sage: E.point([0, 0])
(0 : 0 : 1)
```

point_homset(*S=None*)

Return the set of S-valued points of this scheme.

INPUT:

- S – a commutative ring.

OUTPUT:

The set of morphisms $Spec(S)oX$.

EXAMPLES:

```
sage: P = ProjectiveSpace(ZZ, 3)
sage: P.point_homset(ZZ)
Set of rational points of Projective Space of dimension 3 over Integer Ring
sage: P.point_homset(QQ)
Set of rational points of Projective Space of dimension 3 over Rational Field
sage: P.point_homset(GF(11))
Set of rational points of Projective Space of dimension 3 over
Finite Field of size 11
```

TESTS:

```
sage: P = ProjectiveSpace(QQ,3)
sage: P.point_homset(GF(11))
Traceback (most recent call last):
...
ValueError: There must be a natural map S --> R, but
S = Rational Field and R = Finite Field of size 11
```

point_set(*S=None*)

Return the set of S-valued points of this scheme.

INPUT:

- S – a commutative ring.

OUTPUT:

The set of morphisms $Spec(S)oX$.

EXAMPLES:

```
sage: P = ProjectiveSpace(ZZ, 3)
sage: P.point_homset(ZZ)
Set of rational points of Projective Space of dimension 3 over Integer Ring
sage: P.point_homset(QQ)
Set of rational points of Projective Space of dimension 3 over Rational Field
sage: P.point_homset(GF(11))
Set of rational points of Projective Space of dimension 3 over
Finite Field of size 11
```

TESTS:

```
sage: P = ProjectiveSpace(QQ,3)
sage: P.point_homset(GF(11))
Traceback (most recent call last):
...
ValueError: There must be a natural map S --> R, but
S = Rational Field and R = Finite Field of size 11
```

structure_morphism()

Return the structure morphism from `self` to its base scheme.

OUTPUT:

A scheme morphism.

EXAMPLES:

```
sage: A = AffineSpace(4, QQ)
sage: A.base_morphism()
Scheme morphism:
  From: Affine Space of dimension 4 over Rational Field
  To:   Spectrum of Rational Field
```

```
  Defn: Structure map

sage: X = Spec(QQ)
sage: X.base_morphism()
Scheme morphism:
  From: Spectrum of Rational Field
  To:   Spectrum of Integer Ring
  Defn: Structure map
```

union (X)

Return the disjoint union of the schemes `self` and `X`.

EXAMPLES:

```
sage: S = Spec(QQ)
sage: X = AffineSpace(1, QQ)
sage: S.union(X)
Traceback (most recent call last):
...
NotImplementedError
```

zeta_series (n, t)

Return the zeta series.

Compute a power series approximation to the zeta function of a scheme over a finite field.

INPUT:

- `n` – the number of terms of the power series to compute
- `t` – the variable which the series should be returned

OUTPUT:

A power series approximating the zeta function of self

EXAMPLES:

```
sage: P.<x> = PolynomialRing(GF(3))
sage: C = HyperellipticCurve(x^3+x^2+1)
sage: R.<t> = PowerSeriesRing(Integers())
sage: C.zeta_series(4,t)
1 + 6*t + 24*t^2 + 78*t^3 + 240*t^4 + O(t^5)
sage: (1+2*t+3*t^2)/(1-t)/(1-3*t) + O(t^5)
1 + 6*t + 24*t^2 + 78*t^3 + 240*t^4 + O(t^5)
```

Note that this function depends on count_points, which is only defined for prime order fields for general schemes. Nonetheless, since trac ticket #15108 and trac ticket #15148, it supports hyperelliptic curves over non-prime fields:

```
sage: C.base_extend(GF(9,'a')).zeta_series(4,t)
1 + 12*t + 120*t^2 + 1092*t^3 + 9840*t^4 + O(t^5)
```

`sage.schemes.generic.scheme.`**is_AffineScheme** (x)

Return True if x is an affine scheme.

EXAMPLES:

```
sage: from sage.schemes.generic.scheme import is_AffineScheme
sage: is_AffineScheme(5)
False
sage: E = Spec(QQ)
```

```
sage: is_AffineScheme(E)
True
```

sage.schemes.generic.scheme.**is_Scheme**(*x*)

Test whether x is a scheme.

INPUT:

- x – anything.

OUTPUT:

Boolean. Whether x derives from Scheme.

EXAMPLES:

```
sage: from sage.schemes.generic.scheme import is_Scheme
sage: is_Scheme(5)
False
sage: X = Spec(QQ)
sage: is_Scheme(X)
True
```

CHAPTER

THREE

THE SPEC FUNCTOR

AUTHORS:

- William Stein (2006): initial implementation
- Peter Bruin (2014): rewrite Spec as a functor

`sage.schemes.generic.spec.`**`Spec`**`(`*R*, *S=None*`)`

Apply the Spec functor to R.

INPUT:

- `R` – either a commutative ring or a ring homomorphism
- `S` – a commutative ring (optional), the base ring

OUTPUT:

- `AffineScheme` – the affine scheme $\mathrm{Spec}(R)$

EXAMPLES:

```
sage: Spec(QQ)
Spectrum of Rational Field
sage: Spec(PolynomialRing(QQ, 'x'))
Spectrum of Univariate Polynomial Ring in x over Rational Field
sage: Spec(PolynomialRing(QQ, 'x', 3))
Spectrum of Multivariate Polynomial Ring in x0, x1, x2 over Rational Field
sage: X = Spec(PolynomialRing(GF(49,'a'), 3, 'x')); X
Spectrum of Multivariate Polynomial Ring in x0, x1, x2 over Finite Field in a of size 7^2
sage: TestSuite(X).run()
```

Applying `Spec` twice to the same ring gives identical output (see trac ticket #17008):

```
sage: A = Spec(ZZ); B = Spec(ZZ)
sage: A is B
True
```

A `TypeError` is raised if the input is not a commutative ring:

```
sage: Spec(5)
Traceback (most recent call last):
...
TypeError: x (=5) is not in Category of commutative rings
sage: Spec(FreeAlgebra(QQ,2, 'x'))
Traceback (most recent call last):
...
TypeError: x (=Free Algebra on 2 generators (x0, x1) over Rational Field) is not in Category of
```

TESTS:

```
sage: X = Spec(ZZ)
sage: X
Spectrum of Integer Ring
sage: X.base_scheme()
Spectrum of Integer Ring
sage: X.base_ring()
Integer Ring
sage: X.dimension()
1
sage: Spec(QQ,QQ).base_scheme()
Spectrum of Rational Field
sage: Spec(RDF,QQ).base_scheme()
Spectrum of Rational Field
```

class `sage.schemes.generic.spec.`**`SpecFunctor`**(*base_ring=None*)

Bases: `sage.categories.functor.Functor`, `sage.structure.unique_representation.UniqueReprese`

The Spec functor.

CHAPTER

FOUR

SCHEME OBTAINED BY GLUING TWO OTHER SCHEMES

class `sage.schemes.generic.glue.`**`GluedScheme`**(*f*, *g*, *check=True*)

Bases: `sage.schemes.generic.scheme.Scheme`

INPUT:

- `f` - open immersion from a scheme U to a scheme X
- `g` - open immersion from U to a scheme Y

OUTPUT: The scheme obtained by gluing X and Y along the open set U.

Note: Checking that f and g are open immersions is not implemented.

`gluing_maps`()

CHAPTER

FIVE

POINTS ON SCHEMES

class sage.schemes.generic.point.**SchemePoint**(*S*, *parent=None*)

Bases: sage.structure.element.Element

Base class for points on a scheme, either topological or defined by a morphism.

scheme()

Return the scheme on which self is a point.

EXAMPLES:

```
sage: from sage.schemes.generic.point import SchemePoint
sage: S = Spec(ZZ)
sage: P = SchemePoint(S)
sage: P.scheme()
Spectrum of Integer Ring
```

class sage.schemes.generic.point.**SchemeRationalPoint**(*f*)

Bases: sage.schemes.generic.point.SchemePoint

INPUT:

- f - a morphism of schemes

morphism()

class sage.schemes.generic.point.**SchemeTopologicalPoint**(*S*)

Bases: sage.schemes.generic.point.SchemePoint

Base class for topological points on schemes.

class sage.schemes.generic.point.**SchemeTopologicalPoint_affine_open**(*u*, *x*)

Bases: sage.schemes.generic.point.SchemeTopologicalPoint

INPUT:

- u – morphism with domain an affine scheme U
- x – topological point on U

affine_open()

Return the affine open subset U.

embedding_of_affine_open()

Return the embedding from the affine open subset U into this scheme.

point_on_affine()

Return the scheme point on the affine open U.

class sage.schemes.generic.point.**SchemeTopologicalPoint_prime_ideal**(*S*, *P*, *check=False*)

Bases: sage.schemes.generic.point.SchemeTopologicalPoint

INPUT:

- S – an affine scheme
- P – a prime ideal of the coordinate ring of S, or anything that can be converted into such an ideal

TESTS:

```
sage: from sage.schemes.generic.point import SchemeTopologicalPoint_prime_ideal
sage: S = Spec(ZZ)
sage: P = SchemeTopologicalPoint_prime_ideal(S, 3); P
Point on Spectrum of Integer Ring defined by the Principal ideal (3) of Integer Ring
sage: SchemeTopologicalPoint_prime_ideal(S, 6, check=True)
Traceback (most recent call last):
...
ValueError: The argument Principal ideal (6) of Integer Ring must be a prime ideal of Integer Ri
sage: SchemeTopologicalPoint_prime_ideal(S, ZZ.ideal(7))
Point on Spectrum of Integer Ring defined by the Principal ideal (7) of Integer Ring
```

We define a parabola in the projective plane as a point corresponding to a prime ideal:

```
sage: P2.<x, y, z> = ProjectiveSpace(2, QQ)
sage: SchemeTopologicalPoint_prime_ideal(P2, y*z-x^2)
Point on Projective Space of dimension 2 over Rational Field defined by the Ideal (-x^2 + y*z) c
```

prime_ideal()

Return the prime ideal that defines this scheme point.

EXAMPLES:

```
sage: from sage.schemes.generic.point import SchemeTopologicalPoint_prime_ideal
sage: P2.<x, y, z> = ProjectiveSpace(2, QQ)
sage: pt = SchemeTopologicalPoint_prime_ideal(P2, y*z-x^2)
sage: pt.prime_ideal()
Ideal (-x^2 + y*z) of Multivariate Polynomial Ring in x, y, z over Rational Field
```

sage.schemes.generic.point.**is_SchemeRationalPoint**(*x*)

sage.schemes.generic.point.**is_SchemeTopologicalPoint**(*x*)

CHAPTER

SIX

AMBIENT SPACES

class `sage.schemes.generic.ambient_space.AmbientSpace(n, R=Integer Ring)`
Bases: `sage.schemes.generic.scheme.Scheme`

Base class for ambient spaces over a ring.

INPUT:

- `n` - dimension
- `R` - ring

ambient_space()
Return the ambient space of the scheme self, in this case self itself.

EXAMPLES:

```
sage: P = ProjectiveSpace(4, ZZ)
sage: P.ambient_space() is P
True

sage: A = AffineSpace(2, GF(3))
sage: A.ambient_space()
Affine Space of dimension 2 over Finite Field of size 3
```

base_extend(R)
Return the natural extension of `self` over `R`.

INPUT:

- `R` – a commutative ring, such that there is a natural map from the base ring of self to `R`.

OUTPUT:

- an ambient space over `R` of the same structure as `self`.

Note: A `ValueError` is raised if there is no such natural map. If you need to drop this condition, use `self.change_ring(R)`.

EXAMPLES:

```
sage: P.<x, y, z> = ProjectiveSpace(2, ZZ)
sage: PQ = P.base_extend(QQ); PQ
Projective Space of dimension 2 over Rational Field
sage: PQ.base_extend(GF(5))
Traceback (most recent call last):
...
ValueError: no natural map from the base ring (=Rational Field)
to R (=Finite Field of size 5)!
```

change_ring(R)

Return an ambient space over ring R and otherwise the same as self.

INPUT:

- `R` – commutative ring

OUTPUT:

- ambient space over `R`

Note: There is no need to have any relation between R and the base ring of self, if you want to have such a relation, use `self.base_extend(R)` instead.

TESTS:

```
sage: from sage.schemes.generic.ambient_space import AmbientSpace
sage: A = AmbientSpace(5)
sage: A.change_ring(QQ)
Traceback (most recent call last):
...
NotImplementedError: ambient spaces must override "change_ring" method!
```

defining_polynomials()

Return the defining polynomials of the scheme self. Since self is an ambient space, this is an empty list.

EXAMPLES:

```
sage: ProjectiveSpace(2, QQ).defining_polynomials()
()
sage: AffineSpace(0, ZZ).defining_polynomials()
()
```

dimension()

Return the absolute dimension of this scheme.

EXAMPLES:

```
sage: A2Q = AffineSpace(2, QQ)
sage: A2Q.dimension_absolute()
2
sage: A2Q.dimension()
2
sage: A2Z = AffineSpace(2, ZZ)
sage: A2Z.dimension_absolute()
3
sage: A2Z.dimension()
3
```

dimension_absolute()

Return the absolute dimension of this scheme.

EXAMPLES:

```
sage: A2Q = AffineSpace(2, QQ)
sage: A2Q.dimension_absolute()
2
sage: A2Q.dimension()
2
sage: A2Z = AffineSpace(2, ZZ)
sage: A2Z.dimension_absolute()
3
```

```
sage: A2Z.dimension()
3
```

dimension_relative()

Return the relative dimension of this scheme over its base.

EXAMPLES:

```
sage: A2Q = AffineSpace(2, QQ)
sage: A2Q.dimension_relative()
2
sage: A2Z = AffineSpace(2, ZZ)
sage: A2Z.dimension_relative()
2
```

gen(*n=0*)

Return the n-th generator of the coordinate ring of the scheme self.

EXAMPLES:

```
sage: P.<x, y, z> = ProjectiveSpace(2, ZZ)
sage: P.gen(1)
y
```

gens()

Return the generators of the coordinate ring of the scheme self.

EXAMPLES:

```
sage: AffineSpace(0, QQ).gens()
()

sage: P.<x, y, z> = ProjectiveSpace(2, GF(5))
sage: P.gens()
(x, y, z)
```

is_projective()

Return whether this ambient space is projective n-space.

EXAMPLES:

```
sage: AffineSpace(3,QQ).is_projective()
False
sage: ProjectiveSpace(3,QQ).is_projective()
True
```

ngens()

Return the number of generators of the coordinate ring of the scheme self.

EXAMPLES:

```
sage: AffineSpace(0, QQ).ngens()
0

sage: ProjectiveSpace(50, ZZ).ngens()
51
```

sage.schemes.generic.ambient_space.**is_AmbientSpace**(*x*)

Return True if x is an ambient space.

EXAMPLES:

```
sage: from sage.schemes.generic.ambient_space import is_AmbientSpace
sage: is_AmbientSpace(ProjectiveSpace(3, ZZ))
True
sage: is_AmbientSpace(AffineSpace(2, QQ))
True
sage: P.<x, y, z> = ProjectiveSpace(2, ZZ)
sage: is_AmbientSpace(P.subscheme([x+y+z]))
False
```

CHAPTER

SEVEN

ALGEBRAIC SCHEMES

An algebraic scheme is defined by a set of polynomials in some suitable affine or projective coordinates. Possible ambient spaces are

- Affine spaces (`AffineSpace`),
- Projective spaces (`ProjectiveSpace`), or
- Toric varieties (`ToricVariety`).

Note that while projective spaces are of course toric varieties themselves, they are implemented differently in Sage due to efficiency considerations. You still can create a projective space as a toric variety if you wish.

In the following, we call the corresponding subschemes affine algebraic schemes, projective algebraic schemes, or toric algebraic schemes. In the future other ambient spaces, perhaps by means of gluing relations, may be intoduced.

Generally, polynomials $p_0, p_1, \dots, p_n$ define an ideal $I = \langle p_0, p_1, \dots, p_n \rangle$. In the projective and toric case, the polynomials (and, therefore, the ideal) must be homogeneous. The associated subscheme $V(I)$ of the ambient space is, roughly speaking, the subset of the ambient space on which all polynomials vanish simultaneously.

Warning: You should not construct algebraic scheme objects directly. Instead, use `.subscheme()` methods of ambient spaces. See below for examples.

EXAMPLES:

We first construct the ambient space, here the affine space $\mathbf{Q}^2$:

```
sage: A2 = AffineSpace(2, QQ, 'x, y')
sage: A2.coordinate_ring().inject_variables()
Defining x, y
```

Now we can write polynomial equations in the variables x and y. For example, one equation cuts out a curve (a one-dimensional subscheme):

```
sage: V = A2.subscheme([x^2+y^2-1]); V
Closed subscheme of Affine Space of dimension 2
over Rational Field defined by:
  x^2 + y^2 - 1
sage: V.dimension()
1
```

Here is a more complicated example in a projective space:

```
sage: P3 = ProjectiveSpace(3, QQ, 'x')
sage: P3.inject_variables()
Defining x0, x1, x2, x3
sage: Q = matrix([[x0, x1, x2], [x1, x2, x3]]).minors(2); Q
```

```
[-x1^2 + x0*x2, -x1*x2 + x0*x3, -x2^2 + x1*x3]
sage: twisted_cubic = P3.subscheme(Q)
sage: twisted_cubic
Closed subscheme of Projective Space of dimension 3
over Rational Field defined by:
  -x1^2 + x0*x2,
  -x1*x2 + x0*x3,
  -x2^2 + x1*x3
sage: twisted_cubic.dimension()
1
```

Note that there are 3 equations in the 3-dimensional ambient space, yet the subscheme is 1-dimensional. One can show that it is not possible to eliminate any of the equations, that is, the twisted cubic is **not** a complete intersection of two polynomial equations.

Let us look at one affine patch, for example the one where $x_0 = 1$

```
sage: patch = twisted_cubic.affine_patch(0)
sage: patch
Closed subscheme of Affine Space of dimension 3
over Rational Field defined by:
  -x0^2 + x1,
  -x0*x1 + x2,
  -x1^2 + x0*x2
sage: patch.embedding_morphism()
Scheme morphism:
  From: Closed subscheme of Affine Space of dimension 3
  over Rational Field defined by:
  -x0^2 + x1,
  -x0*x1 + x2,
  -x1^2 + x0*x2
  To:   Closed subscheme of Projective Space of dimension 3
  over Rational Field defined by:
  -x1^2 + x0*x2,
  -x1*x2 + x0*x3,
  -x2^2 + x1*x3
  Defn: Defined on coordinates by sending (x0, x1, x2) to
        (1 : x0 : x1 : x2)
```

AUTHORS:

- David Kohel (2005): initial version.
- William Stein (2005): initial version.
- Andrey Novoseltsev (2010-05-17): subschemes of toric varieties.
- Volker Braun (2010-12-24): documentation of schemes and refactoring. Added coordinate neighborhoods and is_smooth()
- Ben Hutz (2014): subschemes of cartesian products of projective space

class `sage.schemes.generic.algebraic_scheme.`**`AlgebraicScheme`**`(A)`

Bases: `sage.schemes.generic.scheme.Scheme`

An algebraic scheme presented as a subscheme in an ambient space.

This is the base class for all algebraic schemes, that is, schemes defined by equations in affine, projective, or toric ambient spaces.

ambient_space()

Return the ambient space of this algebraic scheme.

EXAMPLES:

```
sage: A.<x, y> = AffineSpace(2, GF(5))
sage: S = A.subscheme([])
sage: S.ambient_space()
Affine Space of dimension 2 over Finite Field of size 5

sage: P.<x, y, z> = ProjectiveSpace(2, ZZ)
sage: S = P.subscheme([x-y, x-z])
sage: S.ambient_space() is P
True
```

coordinate_ring()

Return the coordinate ring of this algebraic scheme. The result is cached.

OUTPUT:

The coordinate ring. Usually a polynomial ring, or a quotient thereof.

EXAMPLES:

```
sage: P.<x, y, z> = ProjectiveSpace(2, ZZ)
sage: S = P.subscheme([x-y, x-z])
sage: S.coordinate_ring()
Quotient of Multivariate Polynomial Ring in x, y, z over Integer Ring by the ideal (x - y, x
```

embedding_center()

Return the distinguished point, if there is any.

If the scheme Y was constructed as a neighbourhood of a point $p \in X$, then `embedding_morphism()` returns a local isomorphism $f : Y \to X$ around the preimage point $f^{-1}(p)$. The latter is returned by `embedding_center()`.

OUTPUT:

A point of `self`. Raises `AttributeError` if there is no distinguished point, depending on how `self` was constructed.

EXAMPLES:

```
sage: P3.<w,x,y,z> = ProjectiveSpace(QQ,3)
sage: X = P3.subscheme( (w^2-x^2)*(y^2-z^2) )
sage: p = [1,-1,3,4]
sage: nbhd = X.neighborhood(p); nbhd
Closed subscheme of Affine Space of dimension 3 over Rational Field defined by:
  x0^2*x2^2 - x1^2*x2^2 + 6*x0^2*x2 - 6*x1^2*x2 + 2*x0*x2^2 +
  2*x1*x2^2 - 7*x0^2 + 7*x1^2 + 12*x0*x2 + 12*x1*x2 - 14*x0 - 14*x1
sage: nbhd.embedding_center()
(0, 0, 0)
sage: nbhd.embedding_morphism()(nbhd.embedding_center())
(1/4 : -1/4 : 3/4 : 1)
sage: nbhd.embedding_morphism()
Scheme morphism:
  From: Closed subscheme of Affine Space of dimension 3 over Rational Field defined by:
  x0^2*x2^2 - x1^2*x2^2 + 6*x0^2*x2 - 6*x1^2*x2 + 2*x0*x2^2 +
  2*x1*x2^2 - 7*x0^2 + 7*x1^2 + 12*x0*x2 + 12*x1*x2 - 14*x0 - 14*x1
  To:   Closed subscheme of Projective Space of dimension 3 over Rational Field defined by:
  w^2*y^2 - x^2*y^2 - w^2*z^2 + x^2*z^2
  Defn: Defined on coordinates by sending (x0, x1, x2) to
        (x0 + 1 : x1 - 1 : x2 + 3 : 4)
```

embedding_morphism()

Return the default embedding morphism of `self`.

If the scheme Y was constructed as a neighbourhood of a point $p \in X$, then `embedding_morphism()` returns a local isomorphism $f : Y \to X$ around the preimage point $f^{-1}(p)$. The latter is returned by `embedding_center()`.

If the algebraic scheme Y was not constructed as a neighbourhood of a point, then the embedding in its `ambient_space()` is returned.

OUTPUT:

A scheme morphism whose `domain()` is `self`.

- By default, it is the tautological embedding into its own ambient space `ambient_space()`.
- If the algebraic scheme (which itself is a subscheme of an auxiliary `ambient_space()`) was constructed as a patch or neighborhood of a point then the embedding is the embedding into the original scheme.
- A `NotImplementedError` is raised if the construction of the embedding morphism is not implemented yet.

EXAMPLES:

```
sage: A2.<x,y> = AffineSpace(QQ,2)
sage: C = A2.subscheme(x^2+y^2-1)
sage: C.embedding_morphism()
  Scheme morphism:
  From: Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x^2 + y^2 - 1
  To:   Affine Space of dimension 2 over Rational Field
  Defn: Defined on coordinates by sending (x, y) to
        (x, y)
sage: P1xP1.<x,y,u,v> = toric_varieties.P1xP1()
sage: P1 = P1xP1.subscheme(x-y)
sage: P1.embedding_morphism()
Scheme morphism:
From: Closed subscheme of 2-d CPR-Fano toric variety covered
      by 4 affine patches defined by:
x - y
To:   2-d CPR-Fano toric variety covered by 4 affine patches
Defn: Defined on coordinates by sending [x : y : u : v] to
      [y : y : u : v]
```

So far, the embedding was just in the own ambient space. Now a bit more interesting examples:

```
sage: P2.<x,y,z> = ProjectiveSpace(QQ,2)
sage: X = P2.subscheme((x^2-y^2)*z)
sage: p = (1,1,0)
sage: nbhd = X.neighborhood(p)
sage: nbhd
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  -x0^2*x1 - 2*x0*x1
```

Note that $p = (1, 1, 0)$ is a singular point of X. So the neighborhood of p is not just affine space. The :meth:neighborhood‘ method returns a presentation of the neighborhood as a subscheme of an auxiliary 2-dimensional affine space:

```
sage: nbhd.ambient_space()
Affine Space of dimension 2 over Rational Field
```

But its `embedding_morphism()` is not into this auxiliary affine space, but the original subscheme X:

```
sage: nbhd.embedding_morphism()
Scheme morphism:
  From: Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  -x0^2*x1 - 2*x0*x1
  To:   Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
  x^2*z - y^2*z
  Defn: Defined on coordinates by sending (x0, x1) to
        (1 : x0 + 1 : x1)
```

A couple more examples:

```
sage: patch1 = P1xP1.affine_patch(1)
sage: patch1
2-d affine toric variety
sage: patch1.embedding_morphism()
  Scheme morphism:
  From: 2-d affine toric variety
  To:   2-d CPR-Fano toric variety covered by 4 affine patches
  Defn: Defined on coordinates by sending [y : u] to
        [1 : y : u : 1]
sage: subpatch = P1.affine_patch(1)
sage: subpatch
Closed subscheme of 2-d affine toric variety defined by:
  -y + 1
sage: subpatch.embedding_morphism()
Scheme morphism:
  From: Closed subscheme of 2-d affine toric variety defined by:
  -y + 1
  To:   Closed subscheme of 2-d CPR-Fano toric variety covered
        by 4 affine patches defined by:
  x - y
  Defn: Defined on coordinates by sending [y : u] to
        [1 : y : u : 1]
```

is_projective()

Return True if self is presented as a subscheme of an ambient projective space.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: PP.<x,y,z,w> = ProjectiveSpace(3,QQ)
sage: f = x^3 + y^3 + z^3 + w^3
sage: R = f.parent()
sage: I = [f] + [f.derivative(zz) for zz in PP.gens()]
sage: V = PP.subscheme(I)
sage: V.is_projective()
True
sage: AA.<x,y,z,w> = AffineSpace(4,QQ)
sage: V = AA.subscheme(I)
sage: V.is_projective()
False
```

Note that toric varieties are implemented differently than projective spaces. This is why this method returns `False` for toric varieties:

```
sage: PP.<x,y,z,w> = toric_varieties.P(3)
sage: V = PP.subscheme(x^3 + y^3 + z^3 + w^3)
sage: V.is_projective()
False
```

ngens()

Return the number of generators of the ambient space of this algebraic scheme.

EXAMPLES:

```
sage: A.<x, y> = AffineSpace(2, GF(5))
sage: S = A.subscheme([])
sage: S.ngens()
2
sage: P.<x, y, z> = ProjectiveSpace(2, ZZ)
sage: S = P.subscheme([x-y, x-z])
sage: P.ngens()
3
```

class sage.schemes.generic.algebraic_scheme.**AlgebraicScheme_quasi**(X, Y)

Bases: sage.schemes.generic.algebraic_scheme.AlgebraicScheme

The quasi-affine or quasi-projective scheme $X - Y$, where X and Y are both closed subschemes of a common ambient affine or projective space.

> **Warning:** You should not create objects of this class directly. The preferred method to construct such subschemes is to use `complement()` method of algebraic schemes.

OUTPUT:

An instance of AlgebraicScheme_quasi.

EXAMPLES:

```
sage: P.<x, y, z> = ProjectiveSpace(2, ZZ)
sage: S = P.subscheme([])
sage: T = P.subscheme([x-y])
sage: T.complement(S)
Quasi-projective subscheme X - Y of Projective Space of dimension 2 over
Integer Ring, where X is defined by:
  (no polynomials)
and Y is defined by:
  x - y
```

X()

Return the scheme X such that self is represented as $X - Y$.

EXAMPLES:

```
sage: P.<x, y, z> = ProjectiveSpace(2, ZZ)
sage: S = P.subscheme([])
sage: T = P.subscheme([x-y])
sage: U = T.complement(S)
sage: U.X() is S
True
```

Y()

Return the scheme Y such that self is represented as $X - Y$.

EXAMPLES:

```
sage: P.<x, y, z> = ProjectiveSpace(2, ZZ)
sage: S = P.subscheme([])
sage: T = P.subscheme([x-y])
sage: U = T.complement(S)
sage: U.Y() is T
True
```

rational_points(*F=None*, *bound=0*)

Return the set of rational points on this algebraic scheme over the field F.

EXAMPLES:

```
sage: A.<x, y> = AffineSpace(2, GF(7))
sage: S = A.subscheme([x^2-y])
sage: T = A.subscheme([x-y])
sage: U = T.complement(S)
sage: U.rational_points()
[(2, 4), (3, 2), (4, 2), (5, 4), (6, 1)]
sage: U.rational_points(GF(7^2, 'b'))
[(2, 4), (3, 2), (4, 2), (5, 4), (6, 1), (b, b + 4), (b + 1, 3*b + 5), (b + 2, 5*b + 1),
(b + 3, 6), (b + 4, 2*b + 6), (b + 5, 4*b + 1), (b + 6, 6*b + 5), (2*b, 4*b + 2),
(2*b + 1, b + 3), (2*b + 2, 5*b + 6), (2*b + 3, 2*b + 4), (2*b + 4, 6*b + 4),
(2*b + 5, 3*b + 6), (2*b + 6, 3), (3*b, 2*b + 1), (3*b + 1, b + 2), (3*b + 2, 5),
(3*b + 3, 6*b + 3), (3*b + 4, 5*b + 3), (3*b + 5, 4*b + 5), (3*b + 6, 3*b + 2),
(4*b, 2*b + 1), (4*b + 1, 3*b + 2), (4*b + 2, 4*b + 5), (4*b + 3, 5*b + 3),
(4*b + 4, 6*b + 3), (4*b + 5, 5), (4*b + 6, b + 2), (5*b, 4*b + 2), (5*b + 1, 3),
(5*b + 2, 3*b + 6), (5*b + 3, 6*b + 4), (5*b + 4, 2*b + 4), (5*b + 5, 5*b + 6),
(5*b + 6, b + 3), (6*b, b + 4), (6*b + 1, 6*b + 5), (6*b + 2, 4*b + 1), (6*b + 3, 2*b + 6),
(6*b + 4, 6), (6*b + 5, 5*b + 1), (6*b + 6, 3*b + 5)]
```

class sage.schemes.generic.algebraic_scheme.**AlgebraicScheme_subscheme**(*A*, *polynomials*)

Bases: sage.schemes.generic.algebraic_scheme.AlgebraicScheme

An algebraic scheme presented as a closed subscheme is defined by explicit polynomial equations. This is as opposed to a general scheme, which could, e.g., be the Neron model of some object, and for which we do not want to give explicit equations.

INPUT:

- A - ambient space (e.g. affine or projective n-space)
- **polynomials - single polynomial, ideal or iterable of defining** polynomials; in any case polynomials must belong to the coordinate ring of the ambient space and define valid polynomial functions (e.g. they should be homogeneous in the case of a projective space)

OUTPUT:

- algebraic scheme

EXAMPLES:

```
sage: from sage.schemes.generic.algebraic_scheme import AlgebraicScheme_subscheme
sage: P.<x, y, z> = ProjectiveSpace(2, QQ)
sage: P.subscheme([x^2-y*z])
Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
  x^2 - y*z
sage: AlgebraicScheme_subscheme(P, [x^2-y*z])
Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
  x^2 - y*z
```

Jacobian()

Return the Jacobian ideal.

This is the ideal generated by

- the $d \times d$ minors of the Jacobian matrix, where d is the `codimension()` of the algebraic scheme, and
- the defining polynomials of the algebraic scheme. Note that some authors do not include these in the definition of the Jacobian ideal. An example of a reference that does include the defining equations is [LazarsfeldJacobian].

OUTPUT:

An ideal in the coordinate ring of the ambient space.

REFERENCES:

EXAMPLES:

```
sage: P3.<w,x,y,z> = ProjectiveSpace(3, QQ)
sage: twisted_cubic = P3.subscheme(matrix([[w, x, y],[x, y, z]]).minors(2))
sage: twisted_cubic.Jacobian()
Ideal (-x^2 + w*y, -x*y + w*z, -y^2 + x*z, x*z, -2*w*z, w*y, 3*w*y, -2*w*x,
w^2, y*z, -2*x*z, w*z, 3*w*z, -2*w*y, w*x, z^2, -2*y*z, x*z, 3*x*z, -2*w*z,
w*y) of Multivariate Polynomial Ring in w, x, y, z over Rational Field
sage: twisted_cubic.defining_ideal()
Ideal (-x^2 + w*y, -x*y + w*z, -y^2 + x*z) of Multivariate Polynomial Ring
in w, x, y, z over Rational Field
```

Jacobian_matrix()

Return the matrix $\frac{\partial f_i}{\partial x_j}$ of (formal) partial derivatives.

OUTPUT:

A matrix of polynomials.

EXAMPLES:

```
sage: P3.<w,x,y,z> = ProjectiveSpace(3, QQ)
sage: twisted_cubic = P3.subscheme(matrix([[w, x, y],[x, y, z]]).minors(2))
sage: twisted_cubic.Jacobian_matrix()
[   y -2*x    w    0]
[   z   -y   -x    w]
[   0    z -2*y    x]
```

base_extend(R)

Return the base change to the ring R of this scheme.

EXAMPLES:

```
sage: P.<x, y, z> = ProjectiveSpace(2, GF(11))
sage: S = P.subscheme([x^2-y*z])
sage: S.base_extend(GF(11^2, 'b'))
Closed subscheme of Projective Space of dimension 2 over Finite Field in b of size 11^2 defi
  x^2 - y*z
sage: S.base_extend(ZZ)
Traceback (most recent call last):
...
ValueError: no natural map from the base ring (=Finite Field of size 11) to R (=Integer Ring
```

change_ring(R)

Returns a new projective subscheme whose base ring is self coerced to R.

EXAMPLES:

```
sage: P.<x,y>=ProjectiveSpace(QQ,1)
sage: X=P.subscheme([3*x^2-y^2])
sage: H=Hom(X,X)
sage: X.change_ring(GF(3))
Closed subscheme of Projective Space of dimension 1 over Finite Field of size 3 defined by:
-y^2
```

codimension()

Return the codimension of the algebraic subscheme.

OUTPUT:

Integer.

EXAMPLES:

```
sage: PP.<x,y,z,w,v> = ProjectiveSpace(4,QQ)
sage: V = PP.subscheme(x*y)
sage: V.codimension()
1
sage: V.dimension()
3
```

complement(*other=None*)

Return the scheme-theoretic complement other - self, where self and other are both closed algebraic subschemes of the same ambient space.

If other is unspecified, it is taken to be the ambient space of self.

EXAMPLES:

```
sage: A.<x, y, z> = AffineSpace(3, ZZ)
sage: X = A.subscheme([x+y-z])
sage: Y = A.subscheme([x-y+z])
sage: Y.complement(X)
Quasi-affine subscheme X - Y of Affine Space of
dimension 3 over Integer Ring, where X is defined by:
  x + y - z
and Y is defined by:
  x - y + z
sage: Y.complement()
Quasi-affine subscheme X - Y of Affine Space of
dimension 3 over Integer Ring, where X is defined by:
  (no polynomials)
and Y is defined by:
  x - y + z
sage: P.<x, y, z> = ProjectiveSpace(2, QQ)
sage: X = P.subscheme([x^2+y^2+z^2])
sage: Y = P.subscheme([x*y+y*z+z*x])
sage: Y.complement(X)
Quasi-projective subscheme X - Y of Projective Space of
dimension 2 over Rational Field, where X is defined by:
  x^2 + y^2 + z^2
and Y is defined by:
  x*y + x*z + y*z
sage: Y.complement(P)
Quasi-projective subscheme X - Y of Projective Space of
dimension 2 over Rational Field, where X is defined by:
  (no polynomials)
```

```
and Y is defined by:
  x*y + x*z + y*z
```

defining_ideal()

Return the ideal that defines this scheme as a subscheme of its ambient space.

OUTPUT:

An ideal in the coordinate ring of the ambient space.

EXAMPLES:

```
sage: P.<x, y, z> = ProjectiveSpace(2, ZZ)
sage: S = P.subscheme([x^2-y*z, x^3+z^3])
sage: S.defining_ideal()
Ideal (x^2 - y*z, x^3 + z^3) of Multivariate Polynomial Ring in x, y, z over Integer Ring
```

defining_polynomials()

Return the polynomials that define this scheme as a subscheme of its ambient space.

OUTPUT:

A tuple of polynomials in the coordinate ring of the ambient space.

EXAMPLES:

```
sage: P.<x, y, z> = ProjectiveSpace(2, ZZ)
sage: S = P.subscheme([x^2-y*z, x^3+z^3])
sage: S.defining_polynomials()
(x^2 - y*z, x^3 + z^3)
```

intersection(*other*)

Return the scheme-theoretic intersection of self and other in their common ambient space.

EXAMPLES:

```
sage: A.<x, y> = AffineSpace(2, ZZ)
sage: X = A.subscheme([x^2-y])
sage: Y = A.subscheme([y])
sage: X.intersection(Y)
Closed subscheme of Affine Space of dimension 2 over Integer Ring defined by:
  x^2 - y,
  y
```

irreducible_components()

Return the irreducible components of this algebraic scheme, as subschemes of the same ambient space.

OUTPUT:

an immutable sequence of irreducible subschemes of the ambient space of this scheme

The components are cached.

EXAMPLES:

We define what is clearly a union of four hypersurfaces in ¶$^4_{\mathbf{Q}}$ then find the irreducible components:

```
sage: PP.<x,y,z,w,v> = ProjectiveSpace(4,QQ)
sage: V = PP.subscheme( (x^2 - y^2 - z^2)*(w^5 -  2*v^2*z^3)* w * (v^3 - x^2*z) )
sage: V.irreducible_components()
[
Closed subscheme of Projective Space of dimension 4 over Rational Field defined by:
w,
```

```
Closed subscheme of Projective Space of dimension 4 over Rational Field defined by:
x^2 - y^2 - z^2,
Closed subscheme of Projective Space of dimension 4 over Rational Field defined by:
x^2*z - v^3,
Closed subscheme of Projective Space of dimension 4 over Rational Field defined by:
w^5 - 2*z^3*v^2
]
```

We verify that the irrelevant ideal isn't accidently returned (see trac 6920):

```
sage: PP.<x,y,z,w> = ProjectiveSpace(3,QQ)
sage: f = x^3 + y^3 + z^3 + w^3
sage: R = f.parent()
sage: I = [f] + [f.derivative(zz) for zz in PP.gens()]
sage: V = PP.subscheme(I)
sage: V.irreducible_components()
[

]
```

The same polynomial as above defines a scheme with a nontrivial irreducible component in affine space (instead of the empty scheme as above):

```
sage: AA.<x,y,z,w> = AffineSpace(4,QQ)
sage: V = AA.subscheme(I)
sage: V.irreducible_components()
[
Closed subscheme of Affine Space of dimension 4 over Rational Field defined by:
  w,
  z,
  y,
  x
]
```

rational_points(*bound=0, F=None*)

Return the rational points on the algebraic subscheme.

EXAMPLES:

Enumerate over a projective scheme over a number field:

```
sage: u = QQ['u'].0
sage: K.<v> = NumberField(u^2 + 3)
sage: A.<x,y> = ProjectiveSpace(K,1)
sage: X=A.subscheme(x^2 - y^2)
sage: X.rational_points(3)
[(-1 : 1), (1 : 1)]
```

One can enumerate points up to a given bound on a projective scheme over the rationals:

```
sage: E = EllipticCurve('37a')
sage: E.rational_points(bound=8)
[(-1 : -1 : 1), (-1 : 0 : 1), (0 : -1 : 1), (0 : 0 : 1), (0 : 1 : 0), (1/4 : -5/8 : 1),
(1/4 : -3/8 : 1), (1 : -1 : 1), (1 : 0 : 1), (2 : -3 : 1), (2 : 2 : 1)]
```

For a small finite field, the complete set of points can be enumerated.

```
sage: Etilde = E.base_extend(GF(3))
sage: Etilde.rational_points()
[(0 : 0 : 1), (0 : 1 : 0), (0 : 2 : 1), (1 : 0 : 1),
 (1 : 2 : 1), (2 : 0 : 1), (2 : 2 : 1)]
```

The class of hyperelliptic curves does not (yet) support desingularization of the places at infinity into two points:

```
sage: FF = FiniteField(7)
sage: P.<x> = PolynomialRing(FiniteField(7))
sage: C = HyperellipticCurve(x^8+x+1)
sage: C.rational_points()
[(0 : 1 : 0), (0 : 1 : 1), (0 : 6 : 1), (2 : 0 : 1),
 (4 : 0 : 1), (6 : 1 : 1), (6 : 6 : 1)]
```

TODO:

1. The above algorithms enumerate all projective points and test whether they lie on the scheme; Implement a more naive sieve at least for covers of the projective line.

2. Implement Stoll's model in weighted projective space to resolve singularities and find two points (1 : 1 : 0) and (-1 : 1 : 0) at infinity.

reduce()

Return the corresponding reduced algebraic space associated to this scheme.

EXAMPLES: First we construct the union of a doubled and tripled line in the affine plane over $\mathbf{Q}$

```
sage: A.<x,y> = AffineSpace(2, QQ)
sage: X = A.subscheme([(x-1)^2*(x-y)^3]); X
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x^5 - 3*x^4*y + 3*x^3*y^2 - x^2*y^3 - 2*x^4 + 6*x^3*y
  - 6*x^2*y^2 + 2*x*y^3 + x^3 - 3*x^2*y + 3*x*y^2 - y^3
sage: X.dimension()
1
```

Then we compute the corresponding reduced scheme:

```
sage: Y = X.reduce(); Y
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x^2 - x*y - x + y
```

Finally, we verify that the reduced scheme Y is the union of those two lines:

```
sage: L1 = A.subscheme([x-1]); L1
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x - 1
sage: L2 = A.subscheme([x-y]); L2
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x - y
sage: W = L1.union(L2); W             # taken in ambient space
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x^2 - x*y - x + y
sage: Y == W
True
```

union(*other*)

Return the scheme-theoretic union of self and other in their common ambient space.

EXAMPLES: We construct the union of a line and a tripled-point on the line.

```
sage: A.<x,y> = AffineSpace(2, QQ)
sage: I = ideal([x,y])^3
sage: P = A.subscheme(I)
sage: L = A.subscheme([y-1])
```

```
sage: S = L.union(P); S
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
y^4 - y^3,
x*y^3 - x*y^2,
x^2*y^2 - x^2*y,
x^3*y - x^3
sage: S.dimension()
1
sage: S.reduce()
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
y^2 - y,
x*y - x
```

We can also use the notation "+" for the union:

```
sage: A.subscheme([x]) + A.subscheme([y^2 - (x^3+1)])
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
x^4 - x*y^2 + x
```

Saving and loading:

```
sage: loads(S.dumps()) == S
True
```

weil_restriction()

Compute the Weil restriction of this variety over some extension field. If the field is a finite field, then this computes the Weil restriction to the prime subfield.

A Weil restriction of scalars - denoted $Res_{L/k}$ - is a functor which, for any finite extension of fields L/k and any algebraic variety X over L, produces another corresponding variety $Res_{L/k}(X)$, defined over k. It is useful for reducing questions about varieties over large fields to questions about more complicated varieties over smaller fields.

This function does not compute this Weil restriction directly but computes on generating sets of polynomial ideals:

Let d be the degree of the field extension L/k, let a a generator of L/k and p the minimal polynomial of L/k. Denote this ideal by I.

Specifically, this function first maps each variable x to its representation over k: $\sum_{i=0}^{d-1} a^i x_i$. Then each generator of I is evaluated over these representations and reduced modulo the minimal polynomial p. The result is interpreted as a univariate polynomial in a and its coefficients are the new generators of the returned ideal.

If the input and the output ideals are radical, this is equivalent to the statement about algebraic varieties above.

OUTPUT: Affine subscheme - the Weil restriction of `self`.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<w> = NumberField(x^5-2)
sage: R.<x> = K[]
sage: L.<v> = K.extension(x^2+1)
sage: A.<x,y> = AffineSpace(L,2)
sage: X = A.subscheme([y^2-L(w)*x^3-v])
sage: X.weil_restriction()
Closed subscheme of Affine Space of dimension 4 over Number Field in w
with defining polynomial x^5 - 2 defined by:
  (-w)*z0^3 + (3*w)*z0*z1^2 + z2^2 - z3^2,
```

```
  (-3*w)*z0^2*z1 + (w)*z1^3 + 2*z2*z3 - 1
sage: X.weil_restriction().ambient_space() is A.weil_restriction()
True

sage: A.<x,y,z> = AffineSpace(GF(5^2,'t'),3)
sage: X = A.subscheme([y^2-x*z, z^2+2*y])
sage: X.weil_restriction()
Closed subscheme of Affine Space of dimension 6 over Finite Field of
size 5 defined by:
  z2^2 - 2*z3^2 - z0*z4 + 2*z1*z5,
  2*z2*z3 + z3^2 - z1*z4 - z0*z5 - z1*z5,
  z4^2 - 2*z5^2 + 2*z2,
  2*z4*z5 + z5^2 + 2*z3
```

class `sage.schemes.generic.algebraic_scheme.`**`AlgebraicScheme_subscheme_affine`**(*A*, *polynomials*)

Bases: `sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme`

Construct an algebraic subscheme of affine space.

> **Warning:** You should not create objects of this class directly. The preferred method to construct such subschemes is to use `subscheme()` method of `affine space`.

INPUT:

- `A` – ambient `affine space`
- `polynomials` – single polynomial, ideal or iterable of defining polynomials.

EXAMPLES:

```
sage: A3.<x, y, z> = AffineSpace(3, QQ)
sage: A3.subscheme([x^2-y*z])
Closed subscheme of Affine Space of dimension 3 over Rational Field defined by:
  x^2 - y*z
```

TESTS:

```
sage: from sage.schemes.generic.algebraic_scheme import AlgebraicScheme_subscheme_affine
sage: AlgebraicScheme_subscheme_affine(A3, [x^2-y*z])
Closed subscheme of Affine Space of dimension 3 over Rational Field defined by:
  x^2 - y*z
```

`dimension`()

Return the dimension of the affine algebraic subscheme.

OUTPUT:

Integer.

EXAMPLES:

```
sage: A.<x,y> = AffineSpace(2, QQ)
sage: A.subscheme([]).dimension()
2
sage: A.subscheme([x]).dimension()
1
sage: A.subscheme([x^5]).dimension()
```

```
1
sage: A.subscheme([x^2 + y^2 - 1]).dimension()
1
sage: A.subscheme([x*(x-1), y*(y-1)]).dimension()
0
```

Something less obvious:

```
sage: A.<x,y,z,w> = AffineSpace(4, QQ)
sage: X = A.subscheme([x^2, x^2*y^2 + z^2, z^2 - w^2, 10*x^2 + w^2 - z^2])
sage: X
Closed subscheme of Affine Space of dimension 4 over Rational Field defined by:
  x^2,
  x^2*y^2 + z^2,
  z^2 - w^2,
  10*x^2 - z^2 + w^2
sage: X.dimension()
1
```

is_smooth(*point=None*)

Test whether the algebraic subscheme is smooth.

INPUT:

- `point` – A point or `None` (default). The point to test smoothness at.

OUTPUT:

Boolean. If no point was specified, returns whether the algebraic subscheme is smooth everywhere. Otherwise, smoothness at the specified point is tested.

EXAMPLES:

```
sage: A2.<x,y> = AffineSpace(2,QQ)
sage: cuspidal_curve = A2.subscheme([y^2-x^3])
sage: cuspidal_curve
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  -x^3 + y^2
sage: smooth_point = cuspidal_curve.point([1,1])
sage: smooth_point in cuspidal_curve
True
sage: singular_point = cuspidal_curve.point([0,0])
sage: singular_point in cuspidal_curve
True
sage: cuspidal_curve.is_smooth(smooth_point)
True
sage: cuspidal_curve.is_smooth(singular_point)
False
sage: cuspidal_curve.is_smooth()
False
```

projective_embedding(*i=None, PP=None*)

Returns a morphism from this affine scheme into an ambient projective space of the same dimension.

INPUT:

- `i` – integer (default: dimension of self = last coordinate) determines which projective embedding to compute. The embedding is that which has a 1 in the i-th coordinate, numbered from 0.
- **`PP` – (default: None) ambient projective space, i.e., ambient space** of codomain of morphism; this is constructed if it is not given.

EXAMPLES:

```
sage: A.<x, y, z> = AffineSpace(3, ZZ)
sage: S = A.subscheme([x*y-z])
sage: S.projective_embedding()
Scheme morphism:
  From: Closed subscheme of Affine Space of dimension 3 over Integer Ring defined by:
  x*y - z
  To:   Closed subscheme of Projective Space of dimension 3 over Integer Ring defined by:
  x0*x1 - x2*x3
  Defn: Defined on coordinates by sending (x, y, z) to
        (x : y : z : 1)

sage: A.<x, y, z> = AffineSpace(3, ZZ)
sage: P = ProjectiveSpace(3,ZZ,'u')
sage: S = A.subscheme([x^2-y*z])
sage: S.projective_embedding(1,P)
Scheme morphism:
  From: Closed subscheme of Affine Space of dimension 3 over Integer
Ring defined by:
  x^2 - y*z
  To:   Closed subscheme of Projective Space of dimension 3 over Integer
Ring defined by:
  u0^2 - u2*u3
  Defn: Defined on coordinates by sending (x, y, z) to
        (x : 1 : y : z)
```

class sage.schemes.generic.algebraic_scheme.**AlgebraicScheme_subscheme_affine_toric**(*toric_variety*, *polynomials*)

Bases: sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_toric

Construct an algebraic subscheme of an affine toric variety.

> **Warning:** You should not create objects of this class directly. The preferred method to construct such subschemes is to use subscheme() method of toric varieties.

INPUT:

- toric_variety – ambient affine toric variety;
- polynomials – single polynomial, list, or ideal of defining polynomials in the coordinate ring of toric_variety.

OUTPUT:

A algebraic subscheme of an affine toric variety.

TESTS:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1.inject_variables()
Defining s, t, x, y
sage: import sage.schemes.generic.algebraic_scheme as SCM
sage: X = SCM.AlgebraicScheme_subscheme_toric(
...         P1xP1, [x*s + y*t, x^3+y^3])
sage: X
Closed subscheme of 2-d CPR-Fano toric variety
covered by 4 affine patches defined by:
```

```
  s*x + t*y,
  x^3 + y^3
```

A better way to construct the same scheme as above:

```
sage: P1xP1.subscheme([x*s + y*t, x^3+y^3])
Closed subscheme of 2-d CPR-Fano toric variety
covered by 4 affine patches defined by:
  s*x + t*y,
  x^3 + y^3
```

dimension()

Return the dimension of `self`.

OUTPUT:

- integer.

EXAMPLES:

```
sage: P1xP1.<s0,s1,t0,t1> = toric_varieties.P1xP1()
sage: P1 = P1xP1.subscheme(s0-s1)
sage: P1.dimension()
1
```

A more complicated example where the ambient toric variety is not smooth:

```
sage: X.<x,y> = toric_varieties.A2_Z2()
sage: X.is_smooth()
False
sage: Y = X.subscheme([x*y, x^2])
sage: Y
Closed subscheme of 2-d affine toric variety defined by:
  x*y,
  x^2
sage: Y.dimension()
1
```

is_smooth(*point=None*)

Test whether the algebraic subscheme is smooth.

INPUT:

- `point` – A point or `None` (default). The point to test smoothness at.

OUTPUT:

Boolean. If no point was specified, returns whether the algebraic subscheme is smooth everywhere. Otherwise, smoothness at the specified point is tested.

EXAMPLES:

```
sage: A2.<x,y> = toric_varieties.A2()
sage: cuspidal_curve = A2.subscheme([y^2-x^3])
sage: cuspidal_curve
Closed subscheme of 2-d affine toric variety defined by:
  -x^3 + y^2
sage: cuspidal_curve.is_smooth([1,1])
True
sage: cuspidal_curve.is_smooth([0,0])
False
sage: cuspidal_curve.is_smooth()
```

```
False
sage: circle = A2.subscheme(x^2+y^2-1)
sage: circle.is_smooth([1,0])
True
sage: circle.is_smooth()
True
```

A more complicated example where the ambient toric variety is not smooth:

```
sage: X.<x,y> = toric_varieties.A2_Z2()     # 2-d affine space mod Z/2
sage: X.is_smooth()
False
sage: Y = X.subscheme([x*y, x^2])    # (twice the x=0 curve) mod Z/2
sage: Y
Closed subscheme of 2-d affine toric variety defined by:
  x*y,
  x^2
sage: Y.dimension()    # Y is a Weil divisor but not Cartier
1
sage: Y.is_smooth()
True
sage: Y.is_smooth([0,0])
True
```

class `sage.schemes.generic.algebraic_scheme.`**`AlgebraicScheme_subscheme_product_projective`**(*A, polynomials*)

Bases: `sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_projective`

See `AlgebraicScheme_subscheme` for documentation.

TESTS:

```
sage: from sage.schemes.generic.algebraic_scheme import AlgebraicScheme_subscheme
sage: P.<x, y, z> = ProjectiveSpace(2, QQ)
sage: P.subscheme([x^2-y*z])
Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
  x^2 - y*z
sage: AlgebraicScheme_subscheme(P, [x^2-y*z])
Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
  x^2 - y*z
```

`affine_patch`(*I, return_embedding=False*)

Return the I^{th} affine patch of this projective scheme where 'I' is a multi-index.

INPUT:

- `I` – a list or tuple of positive integers
- `return_embedding` – Boolean, if true the projective embedding is also returned

OUTPUT:

- An affine algebraic scheme
- An embedding into a product of projective space (optional)

EXAMPLES:

```
sage: PP.<x,y,z,w,u,v> = ProductProjectiveSpaces([3,1],QQ)
sage: W = PP.subscheme([y^2*z-x^3,z^2-w^2,u^3-v^3])
sage: W.affine_patch([0,1],True)
(Closed subscheme of Affine Space of dimension 4 over Rational Field defined by:
  x0^2*x1 - 1,
  x1^2 - x2^2,
  x3^3 - 1, Scheme morphism:
  From: Closed subscheme of Affine Space of dimension 4 over Rational Field defined by:
  x0^2*x1 - 1,
  x1^2 - x2^2,
  x3^3 - 1
  To:   Closed subscheme of Product of projective spaces P^3 x P^1 over Rational Field defin
  -x^3 + y^2*z,
  z^2 - w^2,
  u^3 - v^3
  Defn: Defined on coordinates by sending (x0, x1, x2, x3) to
        (1 : x0 : x1 : x2 , x3 : 1))
```

dimension()

Return the dimension of the algebraic subscheme.

OUTPUT:

Integer.

EXAMPLES:

```
sage: X.<x,y,z,w,u,v> = ProductProjectiveSpaces([2,2],QQ)
sage: L = (-w - v)*x + (-w*y - u*z)
sage: Q = (-u*w - v^2)*x^2 + ((-w^2 - u*w + (-u*v - u^2))*y + (-w^2 - u*v)*z)*x +
sage: W = X.subscheme([L,Q])
sage: W.dimension()
2
```

is_smooth(*point=None*)

Test whether the algebraic subscheme is smooth.

EXAMPLES:

```
sage: X.<x,y,z,w,u,v> = ProductProjectiveSpaces([2,2],QQ)
sage: L = (-w - v)*x + (-w*y - u*z)
sage: Q = (-u*w - v^2)*x^2 + ((-w^2 - u*w + (-u*v - u^2))*y + (-w^2 - u*v)*z)*x + \
((-w^2 - u*w - u^2)*y^2 + (-u*w - v^2)*z*y + (-w^2 + (-v - u)*w)*z^2)
sage: W = X.subscheme([L,Q])
sage: W.is_smooth()
Traceback (most recent call last):
...
NotImplementedError: Not Implemented
```

segre_embedding(*PP=None*)

Return the Segre embedding of `self` into the appropriate projective space.

INPUT:

- `PP` – (default: `None`) ambient image projective space; this is constructed if it is not given.

OUTPUT:

Hom from `self` to the appropriate subscheme of projective space

Todo

products with more than two components

EXAMPLES:

```
sage: X.<x,y,z,w,u,v> = ProductProjectiveSpaces([2,2],QQ)
sage: P = ProjectiveSpace(QQ,8,'t')
sage: L = (-w - v)*x + (-w*y - u*z)
sage: Q = (-u*w - v^2)*x^2 + ((-w^2 - u*w + (-u*v - u^2))*y + (-w^2 - u*v)*z)*x + \
((-w^2 - u*w - u^2)*y^2 + (-u*w - v^2)*z*y + (-w^2 + (-v - u)*w)*z^2)
sage: W = X.subscheme([L,Q])
sage: phi = W.segre_embedding(P)
sage: phi.codomain().ambient_space() == P
True
```

class `sage.schemes.generic.algebraic_scheme.`**`AlgebraicScheme_subscheme_projective`**(*A*, *polynomials*)

Bases: `sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme`

Construct an algebraic subscheme of projective space.

> **Warning:** You should not create objects of this class directly. The preferred method to construct such subschemes is to use `subscheme()` method of `projective space`.

INPUT:

- `A` – ambient `projective space`.
- `polynomials` – single polynomial, ideal or iterable of defining homogeneous polynomials.

EXAMPLES:

```
sage: P.<x, y, z> = ProjectiveSpace(2, QQ)
sage: P.subscheme([x^2-y*z])
Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
  x^2 - y*z
```

TESTS:

```
sage: from sage.schemes.generic.algebraic_scheme import AlgebraicScheme_subscheme_projective
sage: AlgebraicScheme_subscheme_projective(P, [x^2-y*z])
Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
  x^2 - y*z
```

`affine_patch`(*i*, *AA=None*)

Return the i^{th} affine patch of this projective scheme. This is the intersection with this i^{th} affine patch of its ambient space.

INPUT:

- `i` – integer between 0 and dimension of self, inclusive.
- **`AA` – (default: None) ambient affine space, this is constructed** if it is not given.

OUTPUT:

An affine algebraic scheme with fixed `embedding_morphism()` equal to the default `projective_embedding()` map‘.

EXAMPLES:

```
sage: PP = ProjectiveSpace(2, QQ, names='X,Y,Z')
sage: X,Y,Z = PP.gens()
sage: C = PP.subscheme(X^3*Y + Y^3*Z + Z^3*X)
sage: U = C.affine_patch(0)
sage: U
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x0^3*x1 + x1^3 + x0
sage: U.embedding_morphism()
Scheme morphism:
  From: Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x0^3*x1 + x1^3 + x0
  To:   Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
  X^3*Y + Y^3*Z + X*Z^3
  Defn: Defined on coordinates by sending (x0, x1) to
        (1 : x0 : x1)
sage: U.projective_embedding() is U.embedding_morphism()
True

sage: A.<x,y,z> = AffineSpace(QQ,3)
sage: X = A.subscheme([x-y*z])
sage: Y = X.projective_embedding(1).codomain()
sage: Y.affine_patch(1,A).ambient_space() == A
True

sage: P.<u,v,w> = ProjectiveSpace(2,ZZ)
sage: S = P.subscheme([u^2-v*w])
sage: A.<x, y> = AffineSpace(2, ZZ)
sage: S.affine_patch(1, A)
Closed subscheme of Affine Space of dimension 2 over Integer Ring
defined by:
  x^2 - y
```

dimension()

Return the dimension of the projective algebraic subscheme.

OUTPUT:

Integer.

EXAMPLES:

```
sage: P2.<x,y,z> = ProjectiveSpace(2, QQ)
sage: P2.subscheme([]).dimension()
2
sage: P2.subscheme([x]).dimension()
1
sage: P2.subscheme([x^5]).dimension()
1
sage: P2.subscheme([x^2 + y^2 - z^2]).dimension()
1
sage: P2.subscheme([x*(x-z), y*(y-z)]).dimension()
0
```

Something less obvious:

```
sage: P3.<x,y,z,w,t> = ProjectiveSpace(4, QQ)
sage: X = P3.subscheme([x^2, x^2*y^2 + z^2*t^2, z^2 - w^2, 10*x^2 + w^2 - z^2])
sage: X
Closed subscheme of Projective Space of dimension 4 over Rational Field defined by:
  x^2,
```

```
  x^2*y^2 + z^2*t^2,
  z^2 - w^2,
  10*x^2 - z^2 + w^2
sage: X.dimension()
1
```

is_smooth(*point=None*)

Test whether the algebraic subscheme is smooth.

INPUT:

- `point` – A point or `None` (default). The point to test smoothness at.

OUTPUT:

Boolean. If no point was specified, returns whether the algebraic subscheme is smooth everywhere. Otherwise, smoothness at the specified point is tested.

EXAMPLES:

```
sage: P2.<x,y,z> = ProjectiveSpace(2,QQ)
sage: cuspidal_curve = P2.subscheme([y^2*z-x^3])
sage: cuspidal_curve
Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
  -x^3 + y^2*z
sage: cuspidal_curve.is_smooth([1,1,1])
True
sage: cuspidal_curve.is_smooth([0,0,1])
False
sage: cuspidal_curve.is_smooth()
False
sage: P2.subscheme([y^2*z-x^3+z^3+1/10*x*y*z]).is_smooth()
True
```

TESTS:

```
sage: H = P2.subscheme(x)
sage: H.is_smooth()   # one of the few cases where the cone over the subvariety is smooth
True
```

neighborhood(*point*)

Return an affine algebraic subscheme isomorphic to a neighborhood of the `point`.

INPUT:

- `point` – a point of the projective subscheme.

OUTPUT:

An affine algebraic scheme (polynomial equations in affine space) `result` such that

- `embedding_morphism` is an isomorphism to a neighborhood of `point`
- `embedding_center` is mapped to `point`.

EXAMPLES:

```
sage: P.<x,y,z>= ProjectiveSpace(QQ,2)
sage: S = P.subscheme(x+2*y+3*z)
sage: s = S.point([0,-3,2]); s
(0 : -3/2 : 1)
sage: patch = S.neighborhood(s); patch
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
```

```
  x0 + 3*x1
sage: patch.embedding_morphism()
Scheme morphism:
  From: Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x0 + 3*x1
  To:   Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
  x + 2*y + 3*z
  Defn: Defined on coordinates by sending (x0, x1) to
        (x0 : -3/2 : x1 + 1)
sage: patch.embedding_center()
(0, 0)
sage: patch.embedding_morphism()([0,0])
(0 : -3/2 : 1)
sage: patch.embedding_morphism()(patch.embedding_center())
(0 : -3/2 : 1)
```

class `sage.schemes.generic.algebraic_scheme.`**`AlgebraicScheme_subscheme_toric`**(*toric_variety*, *polynomials*)

Bases: `sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme`

Construct an algebraic subscheme of a toric variety.

> **Warning:** You should not create objects of this class directly. The preferred method to construct such subschemes is to use `subscheme()` method of `toric varieties`.

INPUT:

- `toric_variety` – ambient `toric variety`;
- `polynomials` – single polynomial, list, or ideal of defining polynomials in the coordinate ring of `toric_variety`.

OUTPUT:

- `algebraic subscheme of a toric variety`.

TESTS:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1.inject_variables()
Defining s, t, x, y
sage: import sage.schemes.generic.algebraic_scheme as SCM
sage: X = SCM.AlgebraicScheme_subscheme_toric(
...         P1xP1, [x*s + y*t, x^3+y^3])
sage: X
Closed subscheme of 2-d CPR-Fano toric variety
covered by 4 affine patches defined by:
  s*x + t*y,
  x^3 + y^3
```

A better way to construct the same scheme as above:

```
sage: P1xP1.subscheme([x*s + y*t, x^3+y^3])
Closed subscheme of 2-d CPR-Fano toric variety
covered by 4 affine patches defined by:
  s*x + t*y,
  x^3 + y^3
```

affine_algebraic_patch(*cone=None*, *names=None*)

Return the affine patch corresponding to `cone` as an affine algebraic scheme.

INPUT:

- `cone` – a `Cone` σ of the fan. It can be omitted for an affine toric variety, in which case the single generating cone is used.

OUTPUT:

An `affine algebraic subscheme` corresponding to the patch $Spec(\sigma^\vee \cap M)$ associated to the cone σ.

See also `affine_patch()`, which expresses the patches as subvarieties of affine toric varieties instead.

REFERENCES:

David A. Cox, "The Homogeneous Coordinate Ring of a Toric Variety", Lemma 2.2. http://www.arxiv.org/abs/alg-geom/9210008v2

EXAMPLES:

```
sage: P2.<x,y,z> = toric_varieties.P2()
sage: cone = P2.fan().generating_cone(0)
sage: V = P2.subscheme(x^3+y^3+z^3)
sage: V.affine_algebraic_patch(cone)
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  z0^3 + z1^3 + 1

sage: cone = Cone([(0,1),(2,1)])
sage: A2Z2.<x,y> = AffineToricVariety(cone)
sage: A2Z2.affine_algebraic_patch()
Closed subscheme of Affine Space of dimension 3 over Rational Field defined by:
  -z0*z1 + z2^2
sage: V = A2Z2.subscheme(x^2+y^2-1)
sage: patch = V.affine_algebraic_patch();  patch
Closed subscheme of Affine Space of dimension 3 over Rational Field defined by:
  -z0*z1 + z2^2,
  z0 + z1 - 1
sage: nbhd_patch = V.neighborhood([1,0]).affine_algebraic_patch();  nbhd_patch
Closed subscheme of Affine Space of dimension 3 over Rational Field defined by:
  -z0*z1 + z2^2,
  z0 + z1 - 1
sage: nbhd_patch.embedding_center()
(0, 1, 0)
```

Here we got two defining equations. The first one describes the singularity of the ambient space and the second is the pull-back of $x^2 + y^2 - 1$

```
sage: lp = LatticePolytope([(1,0,0),(1,1,0),(1,1,1),(1,0,1),(-2,-1,-1)],
...                        lattice=ToricLattice(3))
sage: X.<x,y,u,v,t> = CPRFanoToricVariety(Delta_polar=lp)
sage: Y = X.subscheme(x*v+y*u+t)
sage: cone = Cone([(1,0,0),(1,1,0),(1,1,1),(1,0,1)])
sage: Y.affine_algebraic_patch(cone)
Closed subscheme of Affine Space of dimension 4 over Rational Field defined by:
  z0*z2 - z1*z3,
  z1 + z3 + 1
```

affine_patch (*i*)

Return the i-th affine patch of self as an affine toric algebraic scheme.

INPUT:

- i – integer, index of a generating cone of the fan of the ambient space of self.

OUTPUT:

- subscheme of an affine toric variety corresponding to the pull-back of self by the embedding morphism of the i-th affine patch of the ambient space of self.

The result is cached, so the i-th patch is always the same object in memory.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: patch1 = P1xP1.affine_patch(1)
sage: patch1.embedding_morphism()
Scheme morphism:
  From: 2-d affine toric variety
  To:   2-d CPR-Fano toric variety covered by 4 affine patches
  Defn: Defined on coordinates by sending [t : x] to
        [1 : t : x : 1]
sage: P1xP1.inject_variables()
Defining s, t, x, y
sage: P1 = P1xP1.subscheme(x-y)
sage: subpatch = P1.affine_patch(1)
sage: subpatch
Closed subscheme of 2-d affine toric variety defined by:
  x - 1
```

dimension ()

Return the dimension of self.

OUTPUT:

Integer. If self is empty, -1 is returned.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1.inject_variables()
Defining s, t, x, y
sage: P1 = P1xP1.subscheme(s-t)
sage: P1.dimension()
1
sage: P1xP1.subscheme([s-t, (s-t)^2]).dimension()
1
sage: P1xP1.subscheme([s, t]).dimension()
-1
```

fan ()

Return the fan of the ambient space.

OUTPUT:

A fan.

EXAMPLES:

```
sage: P2.<x,y,z> = toric_varieties.P(2)
sage: E = P2.subscheme([x^2+y^2+z^2])
```

```
sage: E.fan()
Rational polyhedral fan in 2-d lattice N
```

is_smooth(*point=None*)

Test whether the algebraic subscheme is smooth.

INPUT:

- `point` – A point or `None` (default). The point to test smoothness at.

OUTPUT:

Boolean. If no point was specified, returns whether the algebraic subscheme is smooth everywhere. Otherwise, smoothness at the specified point is tested.

EXAMPLES:

```
sage: P2.<x,y,z> = toric_varieties.P2()
sage: cuspidal_curve = P2.subscheme([y^2*z-x^3])
sage: cuspidal_curve
Closed subscheme of 2-d CPR-Fano toric variety covered by 3 affine patches defined by:
  -x^3 + y^2*z
sage: cuspidal_curve.is_smooth([1,1,1])
True
sage: cuspidal_curve.is_smooth([0,0,1])
False
sage: cuspidal_curve.is_smooth()
False
```

Any sufficiently generic cubic hypersurface is smooth:

```
sage: P2.subscheme([y^2*z-x^3+z^3+1/10*x*y*z]).is_smooth()
True
```

A more complicated example:

```
sage: dP6.<x0,x1,x2,x3,x4,x5> = toric_varieties.dP6()
sage: disjointP1s = dP6.subscheme(x0*x3)
sage: disjointP1s.is_smooth()
True
sage: intersectingP1s = dP6.subscheme(x0*x1)
sage: intersectingP1s.is_smooth()
False
```

A smooth hypersurface in a compact singular toric variety:

```
sage: lp = LatticePolytope([(1,0,0),(1,1,0),(1,1,1),(1,0,1),(-2,-1,-1)],
...                        lattice=ToricLattice(3))
sage: X.<x,y,u,v,t> = CPRFanoToricVariety(Delta_polar=lp)
sage: Y = X.subscheme(x*v+y*u+t)
sage: cone = Cone([(1,0,0),(1,1,0),(1,1,1),(1,0,1)])
sage: Y.is_smooth()
True
```

neighborhood(*point*)

Return an toric algebraic scheme isomorphic to neighborhood of the `point`.

INPUT:

- `point` – a point of the toric algebraic scheme.

OUTPUT

An affine toric algebraic scheme (polynomial equations in an affine toric variety) with fixed embedding_morphism() and embedding_center().

EXAMPLES:

```
sage: P.<x,y,z>= toric_varieties.P2()
sage: S = P.subscheme(x+2*y+3*z)
sage: s = S.point([0,-3,2]); s
[0 : -3 : 2]
sage: patch = S.neighborhood(s); patch
Closed subscheme of 2-d affine toric variety defined by:
  x + 2*y + 6
sage: patch.embedding_morphism()
Scheme morphism:
  From: Closed subscheme of 2-d affine toric variety defined by:
  x + 2*y + 6
  To:   Closed subscheme of 2-d CPR-Fano toric variety covered by 3 affine patches defined b
  x + 2*y + 3*z
  Defn: Defined on coordinates by sending [x : y] to
        [-2*y - 6 : y : 2]
sage: patch.embedding_center()
[0 : -3]
sage: patch.embedding_morphism()(patch.embedding_center())
[0 : -3 : 2]
```

A more complicated example:

```
sage: dP6.<x0,x1,x2,x3,x4,x5> = toric_varieties.dP6()
sage: twoP1 = dP6.subscheme(x0*x3)
sage: patch = twoP1.neighborhood([0,1,2, 3,4,5]); patch
Closed subscheme of 2-d affine toric variety defined by:
  3*x0
sage: patch.embedding_morphism()
Scheme morphism:
  From: Closed subscheme of 2-d affine toric variety defined by:
  3*x0
  To:   Closed subscheme of 2-d CPR-Fano toric variety covered by 6 affine patches defined b
  x0*x3
  Defn: Defined on coordinates by sending [x0 : x1] to
        [0 : x1 : 2 : 3 : 4 : 5]
sage: patch.embedding_center()
[0 : 1]
sage: patch.embedding_morphism()(patch.embedding_center())
[0 : 1 : 2 : 3 : 4 : 5]
```

sage.schemes.generic.algebraic_scheme.**is_AlgebraicScheme**(*x*)

Test whether x is an algebraic scheme.

INPUT:

- x – anything.

OUTPUT:

Boolean. Whether x is an an algebraic scheme, that is, a subscheme of an ambient space over a ring defined by polynomial equations.

EXAMPLES:

```
sage: A2 = AffineSpace(2, QQ, 'x, y')
sage: A2.coordinate_ring().inject_variables()
Defining x, y
```

```
sage: V = A2.subscheme([x^2+y^2]); V
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x^2 + y^2
sage: from sage.schemes.generic.algebraic_scheme import is_AlgebraicScheme
sage: is_AlgebraicScheme(V)
True
```

Affine space is itself not an algebraic scheme, though the closed subscheme defined by no equations is:

```
sage: from sage.schemes.generic.algebraic_scheme import is_AlgebraicScheme
sage: is_AlgebraicScheme(AffineSpace(10, QQ))
False
sage: V = AffineSpace(10, QQ).subscheme([]); V
Closed subscheme of Affine Space of dimension 10 over Rational Field defined by:
  (no polynomials)
sage: is_AlgebraicScheme(V)
True
```

We create a more complicated closed subscheme:

```
sage: A,x = AffineSpace(10, QQ).objgens()
sage: X = A.subscheme([sum(x)]); X
Closed subscheme of Affine Space of dimension 10 over Rational Field defined by:
x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
sage: is_AlgebraicScheme(X)
True
```

```
sage: is_AlgebraicScheme(QQ)
False
sage: S = Spec(QQ)
sage: is_AlgebraicScheme(S)
False
```

CHAPTER

EIGHT

HYPERSURFACES IN AFFINE AND PROJECTIVE SPACE

AUTHORS:

- William Stein <wstein@gmail.com> (2005-12-08)
- David Kohel <kohel@maths.usyd.edu.au> (2005-12-08)
- Alex Ghitza <aghitza@alum.mit.edu> (2009-04-17)

class `sage.schemes.generic.hypersurface.`**`AffineHypersurface`**(*poly*, *ambient=None*)

Bases: `sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_affine`

The affine hypersurface defined by the given polynomial.

EXAMPLES:

```
sage: A.<x, y, z> = AffineSpace(ZZ, 3)
sage: AffineHypersurface(x*y-z^3, A)
Affine hypersurface defined by -z^3 + x*y in Affine Space of dimension 3 over Integer Ring

sage: A.<x, y, z> = QQ[]
sage: AffineHypersurface(x*y-z^3)
Affine hypersurface defined by -z^3 + x*y in Affine Space of dimension 3 over Rational Field
```

`defining_polynomial`()

Return the polynomial equation that cuts out this affine hypersurface.

EXAMPLES:

```
sage: R.<x, y, z> = ZZ[]
sage: H = AffineHypersurface(x*z+y^2)
sage: H.defining_polynomial()
y^2 + x*z
```

class `sage.schemes.generic.hypersurface.`**`ProjectiveHypersurface`**(*poly*, *ambient=None*)

Bases: `sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_projective`

The projective hypersurface defined by the given polynomial.

EXAMPLES:

```
sage: P.<x, y, z> = ProjectiveSpace(ZZ, 2)
sage: ProjectiveHypersurface(x-y, P)
Projective hypersurface defined by x - y in Projective Space of dimension 2 over Integer Ring

sage: R.<x, y, z> = QQ[]
sage: ProjectiveHypersurface(x-y)
Projective hypersurface defined by x - y in Projective Space of dimension 2 over Rational Field
```

defining_polynomial()

Return the polynomial equation that cuts out this projective hypersurface.

EXAMPLES:

```
sage: R.<x, y, z> = ZZ[]
sage: H = ProjectiveHypersurface(x*z+y^2)
sage: H.defining_polynomial()
y^2 + x*z
```

sage.schemes.generic.hypersurface.**is_Hypersurface**(*self*)

Return True if self is a hypersurface, i.e. an object of the type ProjectiveHypersurface or AffineHypersurface.

EXAMPLES:

```
sage: from sage.schemes.generic.hypersurface import is_Hypersurface
sage: R.<x, y, z> = ZZ[]
sage: H = ProjectiveHypersurface(x*z+y^2)
sage: is_Hypersurface(H)
True

sage: H = AffineHypersurface(x*z+y^2)
sage: is_Hypersurface(H)
True

sage: H = ProjectiveSpace(QQ, 5)
sage: is_Hypersurface(H)
False
```

CHAPTER

NINE

SET OF HOMOMORPHISMS BETWEEN TWO SCHEMES

For schemes X and Y, this module implements the set of morphisms $Hom(X, Y)$. This is done by SchemeHomset_generic.

As a special case, the Hom-sets can also represent the points of a scheme. Recall that the K-rational points of a scheme X over k can be identified with the set of morphisms $Spec(K) \to X$. In Sage the rational points are implemented by such scheme morphisms. This is done by SchemeHomset_points and its subclasses.

Note: You should not create the Hom-sets manually. Instead, use the Hom() method that is inherited by all schemes.

AUTHORS:

- William Stein (2006): initial version.
- Volker Braun (2011-08-11): significant improvement and refactoring.
- Ben Hutz (June 2012): added support for projective ring

class sage.schemes.generic.homset.**SchemeHomsetFactory**

Bases: sage.structure.factory.UniqueFactory

Factory for Hom-sets of schemes.

EXAMPLES:

```
sage: A2 = AffineSpace(QQ,2)
sage: A3 = AffineSpace(QQ,3)
sage: Hom = A3.Hom(A2)
```

The Hom-sets are uniquely determined by domain and codomain:

```
sage: Hom is copy(Hom)
True
sage: Hom is A3.Hom(A2)
True
```

The Hom-sets are identical if the domains and codomains are identical:

```
sage: loads(Hom.dumps()) is Hom
True
sage: A3_iso = AffineSpace(QQ,3)
sage: A3_iso is A3
True
sage: Hom_iso = A3_iso.Hom(A2)
sage: Hom_iso is Hom
True
```

TESTS:

```
sage: Hom.base()
Integer Ring
sage: Hom.base_ring()
Integer Ring
```

create_key_and_extra_args (*X*, *Y*, *category=None*, *base=Integer Ring*, *check=True*, *as_point_homset=False*)

Create a key that uniquely determines the Hom-set.

INPUT:

- X – a scheme. The domain of the morphisms.
- Y – a scheme. The codomain of the morphisms.
- category – a category for the Hom-sets (default: schemes over given base).
- base – a scheme or a ring. The base scheme of domain and codomain schemes. If a ring is specified, the spectrum of that ring will be used as base scheme.
- check – boolean (default: True).

EXAMPLES:

```
sage: A2 = AffineSpace(QQ,2)
sage: A3 = AffineSpace(QQ,3)
sage: A3.Hom(A2)    # indirect doctest
Set of morphisms
  From: Affine Space of dimension 3 over Rational Field
  To:   Affine Space of dimension 2 over Rational Field
sage: from sage.schemes.generic.homset import SchemeHomsetFactory
sage: SHOMfactory = SchemeHomsetFactory('test')
sage: key, extra = SHOMfactory.create_key_and_extra_args(A3,A2,check=False)
sage: key
(..., ..., Category of schemes over Integer Ring, False)
sage: extra
{'X': Affine Space of dimension 3 over Rational Field,
 'Y': Affine Space of dimension 2 over Rational Field,
 'base_ring': Integer Ring,
 'check': False}
```

create_object (*version*, *key*, ***extra_args*)

Create a SchemeHomset_generic.

INPUT:

- version – object version. Currently not used.
- key – a key created by create_key_and_extra_args().
- extra_args – a dictionary of extra keyword arguments.

EXAMPLES:

```
sage: A2 = AffineSpace(QQ,2)
sage: A3 = AffineSpace(QQ,3)
sage: A3.Hom(A2) is A3.Hom(A2)    # indirect doctest
True
sage: from sage.schemes.generic.homset import SchemeHomsetFactory
sage: SHOMfactory = SchemeHomsetFactory('test')
sage: SHOMfactory.create_object(0, [id(A3), id(A2), A3.category(), False],
....:                           check=True, X=A3, Y=A2, base_ring=QQ)
Set of morphisms
```

```
  From: Affine Space of dimension 3 over Rational Field
  To:   Affine Space of dimension 2 over Rational Field
```

class sage.schemes.generic.homset.**SchemeHomset_generic**(*X*, *Y*, *category=None*, *check=True*, *base=None*)

Bases: sage.categories.homset.HomsetWithBase

The base class for Hom-sets of schemes.

INPUT:

- X – a scheme. The domain of the Hom-set.
- Y – a scheme. The codomain of the Hom-set.
- category – a category (optional). The category of the Hom-set.
- check – boolean (optional, default=``True``). Whether to check the defining data for consistency.

EXAMPLES:

```
sage: from sage.schemes.generic.homset import SchemeHomset_generic
sage: A2 = AffineSpace(QQ,2)
sage: Hom = SchemeHomset_generic(A2, A2); Hom
Set of morphisms
  From: Affine Space of dimension 2 over Rational Field
  To:   Affine Space of dimension 2 over Rational Field
sage: Hom.category()
Category of endsets of schemes over Rational Field
```

Element

alias of SchemeMorphism

natural_map()

Return a natural map in the Hom space.

OUTPUT:

A SchemeMorphism if there is a natural map from domain to codomain. Otherwise, a NotImplementedError is raised.

EXAMPLES:

```
sage: A = AffineSpace(4, QQ)
sage: A.structure_morphism()    # indirect doctest
Scheme morphism:
  From: Affine Space of dimension 4 over Rational Field
  To:   Spectrum of Rational Field
  Defn: Structure map
```

class sage.schemes.generic.homset.**SchemeHomset_points**(*X*, *Y*, *category=None*, *check=True*, *base=Integer Ring*)

Bases: sage.schemes.generic.homset.SchemeHomset_generic

Set of rational points of the scheme.

Recall that the K-rational points of a scheme X over k can be identified with the set of morphisms $Spec(K)oX$. In Sage, the rational points are implemented by such scheme morphisms.

If a scheme has a finite number of points, then the homset is supposed to implement the Python iterator interface. See SchemeHomset_points_toric_field for example.

INPUT:

See SchemeHomset_generic.

EXAMPLES:

```
sage: from sage.schemes.generic.homset import SchemeHomset_points
sage: SchemeHomset_points(Spec(QQ), AffineSpace(ZZ,2))
Set of rational points of Affine Space of dimension 2 over Rational Field
```

cardinality()

Return the number of points.

OUTPUT:

An integer or infinity.

EXAMPLES:

```
sage: toric_varieties.P2().point_set().cardinality()
+Infinity

sage: P2 = toric_varieties.P2(base_ring=GF(3))
sage: P2.point_set().cardinality()
13
```

extended_codomain()

Return the codomain with extended base, if necessary.

OUTPUT:

The codomain scheme, with its base ring extended to the codomain. That is, the codomain is of the form $Spec(R)$ and the base ring of the domain is extended to R.

EXAMPLES:

```
sage: P2 = ProjectiveSpace(QQ,2)
sage: K.<a> = NumberField(x^2 + x - (3^3-3))
sage: K_points = P2(K);  K_points
Set of rational points of Projective Space of dimension 2
over Number Field in a with defining polynomial x^2 + x - 24

sage: K_points.codomain()
Projective Space of dimension 2 over Rational Field

sage: K_points.extended_codomain()
Projective Space of dimension 2 over Number Field in a with
defining polynomial x^2 + x - 24
```

list()

Return a tuple containing all points.

OUTPUT:

A tuple containing all points of the toric variety.

EXAMPLE:

```
sage: P1 = toric_varieties.P1(base_ring=GF(3))
sage: P1.point_set().list()
([0 : 1], [1 : 0], [1 : 1], [1 : 2])
```

value_ring()

Return R for a point Hom-set $X(Spec(R))$.

OUTPUT:

A commutative ring.

EXAMPLES:

```
sage: P2 = ProjectiveSpace(ZZ,2)
sage: P2(QQ).value_ring()
Rational Field
```

`sage.schemes.generic.homset.is_SchemeHomset(H)`

Test whether `H` is a scheme Hom-set.

EXAMPLES:

```
sage: f = Spec(QQ).identity_morphism();  f
Scheme endomorphism of Spectrum of Rational Field
  Defn: Identity map
sage: from sage.schemes.generic.homset import is_SchemeHomset
sage: is_SchemeHomset(f)
False
sage: is_SchemeHomset(f.parent())
True
sage: is_SchemeHomset('a string')
False
```

CHAPTER

TEN

SCHEME MORPHISM

Note: You should never create the morphisms directy. Instead, use the `hom()` and `Hom()` methods that are inherited by all schemes.

If you want to extend the Sage library with some new kind of scheme, your new class (say, `myscheme`) should provide a method

- `myscheme._morphism(*args, **kwds)` returning a morphism between two schemes in your category, usually defined via polynomials. Your morphism class should derive from `SchemeMorphism_polynomial`. These morphisms will usually be elements of the Hom-set `SchemeHomset_generic`.

Optionally, you can also provide a special Hom-set class for your subcategory of schemes. If you want to do this, you should also provide a method

- `myscheme._homset(*args, **kwds)` returning a Hom-set, which must be an element of a derived class of $class$: ' $sage.schemes.generic.homset.SchemeHomset_generic$. If your new Hom-set class does not use `myscheme._morphism` then you do not have to provide it.

Note that points on schemes are morphisms $Spec(K) \to X$, too. But we typically use a different notation, so they are implemented in a different derived class. For this, you should implement a method

- `myscheme._point(*args, **kwds)` returning a point, that is, a morphism $Spec(K) \to X$. Your point class should derive from `SchemeMorphism_point`.

Optionally, you can also provide a special Hom-set for the points, for example the point Hom-set can provide a method to enumerate all points. If you want to do this, you should also provide a method

- `myscheme._point_homset(*args, **kwds)` returning the `homset` of points. The Hom-sets of points are implemented in classes named `SchemeHomset_points_...`. If your new Hom-set class does not use `myscheme._point` then you do not have to provide it.

AUTHORS:

- David Kohel, William Stein
- William Stein (2006-02-11): fixed bug where P(0,0,0) was allowed as a projective point.
- Volker Braun (2011-08-08): Renamed classes, more documentation, misc cleanups.
- Ben Hutz (June 2012): added support for projective ring
- Simon King (2013-10): copy the changes of `Morphism` that have been introduced in trac ticket #14711.

class `sage.schemes.generic.morphism.`**`SchemeMorphism`**(*parent*, *codomain=None*)

Bases: `sage.structure.element.Element`

Base class for scheme morphisms

INPUT:

•parent – the parent of the morphism.

Todo

Currently, SchemeMorphism copies code from Map rather than inheriting from it. This is to work around a bug in Cython: We want to create a common sub-class of ModuleElement and SchemeMorphism, but Cython would currently confuse cpdef attributes of the two base classes. Proper inheritance should be used as soon as this bug is fixed.

EXAMPLES:

```
sage: X = Spec(ZZ)
sage: Hom = X.Hom(X)
sage: from sage.schemes.generic.morphism import SchemeMorphism
sage: f = SchemeMorphism(Hom)
sage: type(f)
<class 'sage.schemes.generic.morphism.SchemeMorphism'>
```

TESTS:

```
sage: A2 = AffineSpace(QQ,2)
sage: A2.structure_morphism().domain()
Affine Space of dimension 2 over Rational Field
sage: A2.structure_morphism().category()
Category of homsets of schemes
```

category()

Return the category of the Hom-set.

OUTPUT:

A category.

EXAMPLES:

```
sage: A2 = AffineSpace(QQ,2)
sage: A2.structure_morphism().category()
Category of homsets of schemes
```

category_for()

Return the category which this morphism belongs to.

EXAMPLES:

```
sage: A2 = AffineSpace(QQ,2)
sage: A2.structure_morphism().category_for()
Category of schemes
```

glue_along_domains(*other*)

Glue two morphism

INPUT:

•other – a scheme morphism with the same domain.

OUTPUT:

Assuming that self and other are open immersions with the same domain, return scheme obtained by gluing along the images.

EXAMPLES:

We construct a scheme isomorphic to the projective line over $\mathrm{Spec}(\mathbf{Q})$ by gluing two copies of $\mathbb{A}^1$ minus a point:

```
sage: R.<x,y> = PolynomialRing(QQ, 2)
sage: S.<xbar, ybar> = R.quotient(x*y - 1)
sage: Rx = PolynomialRing(QQ, 'x')
sage: i1 = Rx.hom([xbar])
sage: Ry = PolynomialRing(QQ, 'y')
sage: i2 = Ry.hom([ybar])
sage: Sch = Schemes()
sage: f1 = Sch(i1)
sage: f2 = Sch(i2)
```

Now f1 and f2 have the same domain, which is a $\mathbb{A}^1$ minus a point. We glue along the domain:

```
sage: P1 = f1.glue_along_domains(f2)
sage: P1
Scheme obtained by gluing X and Y along U, where
  X: Spectrum of Univariate Polynomial Ring in x over Rational Field
  Y: Spectrum of Univariate Polynomial Ring in y over Rational Field
  U: Spectrum of Quotient of Multivariate Polynomial Ring in x, y
  over Rational Field by the ideal (x*y - 1)

sage: a, b = P1.gluing_maps()
sage: a
Affine Scheme morphism:
 From: Spectrum of Quotient of Multivariate Polynomial Ring in x, y
       over Rational Field by the ideal (x*y - 1)
  To:   Spectrum of Univariate Polynomial Ring in x over Rational Field
  Defn: Ring morphism:
          From: Univariate Polynomial Ring in x over Rational Field
          To:   Quotient of Multivariate Polynomial Ring in x, y over
                Rational Field by the ideal (x*y - 1)
          Defn: x |--> xbar
sage: b
Affine Scheme morphism:
  From: Spectrum of Quotient of Multivariate Polynomial Ring in x, y
        over Rational Field by the ideal (x*y - 1)
  To:   Spectrum of Univariate Polynomial Ring in y over Rational Field
  Defn: Ring morphism:
          From: Univariate Polynomial Ring in y over Rational Field
          To:   Quotient of Multivariate Polynomial Ring in x, y over
                Rational Field by the ideal (x*y - 1)
          Defn: y |--> ybar
```

is_endomorphism()

Return wether the morphism is an endomorphism.

OUTPUT:

Boolean. Whether the domain and codomain are identical.

EXAMPLES:

```
sage: X = AffineSpace(QQ,2)
sage: X.structure_morphism().is_endomorphism()
False
sage: X.identity_morphism().is_endomorphism()
True
```

class sage.schemes.generic.morphism.**SchemeMorphism_id**(*X*)

Bases: sage.schemes.generic.morphism.SchemeMorphism

Return the identity morphism from X to itself.

INPUT:

- X – the scheme.

EXAMPLES:

```
sage: X = Spec(ZZ)
sage: X.identity_morphism()  # indirect doctest
Scheme endomorphism of Spectrum of Integer Ring
  Defn: Identity map
```

class sage.schemes.generic.morphism.**SchemeMorphism_point** (*parent*, *codomain=None*)

Bases: sage.schemes.generic.morphism.SchemeMorphism

Base class for rational points on schemes.

Recall that the K-rational points of a scheme X over k can be identified with the set of morphisms $Spec(K)oX$. In Sage, the rational points are implemented by such scheme morphisms.

EXAMPLES:

```
sage: from sage.schemes.generic.morphism import SchemeMorphism
sage: f = SchemeMorphism(Spec(ZZ).Hom(Spec(ZZ)))
sage: type(f)
<class 'sage.schemes.generic.morphism.SchemeMorphism'>
```

change_ring (*R*, ***kwds*)

Returns a new SchemeMorphism_point which is self coerced to R. If *check* is true, then the initialization checks are performed. The user may specify the embedding into R with a keyword.

INPUT:

- R – ring

kwds:

- check – Boolean
- embedding – field embedding from the base ring of self to R

OUTPUT: SchemeMorphism_point

EXAMPLES:

```
sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: X = P.subscheme(x^2-y^2)
sage: X(23,23,1).change_ring(GF(13))
(10 : 10 : 1)

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: P(-2/3,1).change_ring(CC)
(-0.666666666666667 : 1.00000000000000)

sage: P.<x,y> = ProjectiveSpace(ZZ,1)
sage: P(152,113).change_ring(Zp(5))
(2 + 5^2 + 5^3 + O(5^20) : 3 + 2*5 + 4*5^2 + O(5^20))

sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2-x+1)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: Q = P([a+1,1])
```

```
sage: emb = K.embeddings(QQbar)
sage: Q.change_ring(QQbar, embedding = emb[0])
(1.5000000000000000? - 0.866025403784439?*I : 1)
sage: Q.change_ring(QQbar, embedding = emb[1])
(1.5000000000000000? + 0.866025403784439?*I : 1)
```

```
sage: K.<v> = QuadraticField(2)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: Q = P([v,1])
sage: Q.change_ring(QQbar)
(-1.414213562373095? : 1)
```

scheme()

Return the scheme whose point is represented.

OUTPUT:

A scheme.

EXAMPLES:

```
sage: A = AffineSpace(2, QQ)
sage: a = A(1,2)
sage: a.scheme()
Affine Space of dimension 2 over Rational Field
```

class sage.schemes.generic.morphism.**SchemeMorphism_polynomial**(*parent*, *polys*, *check=True*)

Bases: sage.schemes.generic.morphism.SchemeMorphism

A morphism of schemes determined by polynomials that define what the morphism does on points in the ambient space.

INPUT:

- parent – Hom-set whose domain and codomain are affine schemes.
- polys – a list/tuple/iterable of polynomials defining the scheme morphism.
- check – boolean (optional, default:True). Whether to check the input for consistency.

EXAMPLES:

An example involving the affine plane:

```
sage: R.<x,y> = QQ[]
sage: A2 = AffineSpace(R)
sage: H = A2.Hom(A2)
sage: f = H([x-y, x*y])
sage: f([0,1])
(-1, 0)
```

An example involving the projective line:

```
sage: R.<x,y> = QQ[]
sage: P1 = ProjectiveSpace(R)
sage: H = P1.Hom(P1)
sage: f = H([x^2+y^2,x*y])
sage: f([0,1])
(1 : 0)
```

Some checks are performed to make sure the given polynomials define a morphism:

```
sage: f = H([exp(x),exp(y)])
Traceback (most recent call last):
...
TypeError: polys (=[e^x, e^y]) must be elements of
Multivariate Polynomial Ring in x, y over Rational Field
```

base_ring()

Return the base ring of `self`, that is, the ring over which the coefficients of `self` is given as polynomials.

OUTPUT:

- ring

EXAMPLES:

```
sage: P.<x,y>=ProjectiveSpace(QQ,1)
sage: H=Hom(P,P)
sage: f=H([3/5*x^2,6*y^2])
sage: f.base_ring()
Rational Field

sage: R.<t>=PolynomialRing(ZZ,1)
sage: P.<x,y>=ProjectiveSpace(R,1)
sage: H=Hom(P,P)
sage: f=H([3*x^2,y^2])
sage: f.base_ring()
Multivariate Polynomial Ring in t over Integer Ring
```

change_ring(*R*, ***kwds*)

Returns a new `SchemeMorphism_polynomial` which is `self` coerced to `R`. If `check` is `True`, then the initialization checks are performed. The user may specify the embedding into `R` with a keyword.

INPUT:

- `R` – ring

kwds:

- `check` – Boolean
- `embedding` – field embedding from the base ring of `self` to `R`

OUTPUT:

- A new :class: $SchemeMorphism_polynomial$ which is `self` coerced to `R`.

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(ZZ,1)
sage: H = Hom(P,P)
sage: f = H([3*x^2,y^2])
sage: f.change_ring(GF(3))
Traceback (most recent call last):
...
ValueError: polys (=[0, y^2]) must be of the same degree

sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = Hom(P,P)
sage: f = H([5/2*x^3 + 3*x*y^2-y^3,3*z^3 + y*x^2, x^3-z^3])
sage: f.change_ring(GF(3))
Scheme endomorphism of Projective Space of dimension 2 over Finite Field of size 3
    Defn: Defined on coordinates by sending (x : y : z) to
        (x^3 - y^3 : x^2*y : x^3 - z^3)
```

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: X = P.subscheme([5*x^2-y^2])
sage: H = Hom(X,X)
sage: f = H([x,y])
sage: f.change_ring(GF(3))
Scheme endomorphism of Closed subscheme of Projective Space of dimension
1 over Finite Field of size 3 defined by:
    -x^2 - y^2
    Defn: Defined on coordinates by sending (x : y) to
        (x : y)

Check that :trac:'16834' is fixed::

sage: A.<x,y,z> = AffineSpace(RR,3)
sage: h = Hom(A,A)
sage: f = h([x^2+1.5,y^3,z^5-2.0])
sage: f.change_ring(CC)
Scheme endomorphism of Affine Space of dimension 3 over Complex Field with 53 bits of precis
Defn: Defined on coordinates by sending (x, y, z) to
    (x^2 + 1.50000000000000, y^3, z^5 - 2.00000000000000)

sage: A.<x,y> = ProjectiveSpace(ZZ,1)
sage: B.<u,v> = AffineSpace(QQ,2)
sage: h = Hom(A,B)
sage: f = h([x^2, y^2])
sage: f.change_ring(QQ)
Scheme morphism:
    From: Projective Space of dimension 1 over Rational Field
    To:   Affine Space of dimension 2 over Rational Field
    Defn: Defined on coordinates by sending (x : y) to
    (x^2, y^2)

sage: A.<x,y> = AffineSpace(QQ,2)
sage: H = Hom(A,A)
sage: f = H([3*x^2/y,y^2/x])
sage: f.change_ring(RR)
Scheme endomorphism of Affine Space of dimension 2 over Real Field with
53 bits of precision
Defn: Defined on coordinates by sending (x, y) to
        (3.00000000000000*x^2/y, y^2/x)

sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^3-x+1)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = End(P)
sage: f = H([x^2 + a*x*y + a^2*y^2,y^2])
sage: emb = K.embeddings(QQbar)
sage: f.change_ring(QQbar, embedding=emb[0])
Scheme endomorphism of Projective Space of dimension 1 over Algebraic
Field
   Defn: Defined on coordinates by sending (x : y) to
         (x^2 + (-1.324717957244746?)*x*y + 1.754877666246693?*y^2 : y^2)
sage: f.change_ring(QQbar, embedding=emb[1])
Scheme endomorphism of Projective Space of dimension 1 over Algebraic
Field
   Defn: Defined on coordinates by sending (x : y) to
         (x^2 + (0.6623589786223730? - 0.5622795120623013?*I)*x*y +
```

```
(0.1225611668766537? - 0.744861766619745?*I)*y^2 : y^2)

sage: K.<v> = QuadraticField(2)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = End(P)
sage: f = H([x^2+v*y^2,y^2])
sage: f.change_ring(QQbar)
Scheme endomorphism of Projective Space of dimension 1 over Algebraic
Field
  Defn: Defined on coordinates by sending (x : y) to
        (x^2 + (-1.414213562373095?)*y^2 : y^2)
```

coordinate_ring()

Returns the coordinate ring of the ambient projective space the multivariable polynomial ring over the base ring

OUTPUT:

- ring

EXAMPLES:

```
sage: P.<x,y>=ProjectiveSpace(QQ,1)
sage: H=Hom(P,P)
sage: f=H([3/5*x^2,6*y^2])
sage: f.coordinate_ring()
Multivariate Polynomial Ring in x, y over Rational Field

sage: R.<t>=PolynomialRing(ZZ,1)
sage: P.<x,y>=ProjectiveSpace(R,1)
sage: H=Hom(P,P)
sage: f=H([3*x^2,y^2])
sage: f.coordinate_ring()
Multivariate Polynomial Ring in x, y over Multivariate Polynomial Ring
in t over Integer Ring
```

defining_polynomials()

Return the defining polynomials.

OUTPUT:

An immutable sequence of polynomials that defines this scheme morphism.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: A.<x,y> = AffineSpace(R)
sage: H = A.Hom(A)
sage: H([x^3+y, 1-x-y]).defining_polynomials()
[x^3 + y, -x - y + 1]
```

class `sage.schemes.generic.morphism.`**`SchemeMorphism_spec`**(*parent*, *phi*, *check=True*)

Bases: `sage.schemes.generic.morphism.SchemeMorphism`

Morphism of spectra of rings

INPUT:

- `parent` – Hom-set whose domain and codomain are affine schemes.
- `phi` – a ring morphism with matching domain and codomain.

•check – boolean (optional, default:True). Whether to check the input for consistency.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(QQ)
sage: phi = R.hom([QQ(7)]); phi
Ring morphism:
  From: Univariate Polynomial Ring in x over Rational Field
  To:   Rational Field
  Defn: x |--> 7

sage: X = Spec(QQ); Y = Spec(R)
sage: f = X.hom(phi); f
Affine Scheme morphism:
  From: Spectrum of Rational Field
  To:   Spectrum of Univariate Polynomial Ring in x over Rational Field
  Defn: Ring morphism:
          From: Univariate Polynomial Ring in x over Rational Field
          To:   Rational Field
          Defn: x |--> 7

sage: f.ring_homomorphism()
Ring morphism:
  From: Univariate Polynomial Ring in x over Rational Field
  To:   Rational Field
  Defn: x |--> 7
```

ring_homomorphism()

Return the underlying ring homomorphism.

OUTPUT:

A ring homomorphism.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(QQ)
sage: phi = R.hom([QQ(7)])
sage: X = Spec(QQ); Y = Spec(R)
sage: f = X.hom(phi)
sage: f.ring_homomorphism()
Ring morphism:
  From: Univariate Polynomial Ring in x over Rational Field
  To:   Rational Field
  Defn: x |--> 7
```

class sage.schemes.generic.morphism.**SchemeMorphism_structure_map**(*parent*, *codomain=None*)

Bases: sage.schemes.generic.morphism.SchemeMorphism

The structure morphism

INPUT:

•parent – Hom-set with codomain equal to the base scheme of the domain.

EXAMPLES:

```
sage: Spec(ZZ).structure_morphism()    # indirect doctest
Scheme endomorphism of Spectrum of Integer Ring
  Defn: Structure map
```

sage.schemes.generic.morphism.**is_SchemeMorphism**(*f*)

Test whether f is a scheme morphism.

INPUT:

- f – anything.

OUTPUT:

Boolean. Return True if f is a scheme morphism or a point on an elliptic curve.

EXAMPLES:

```
sage: A.<x,y> = AffineSpace(QQ,2); H = A.Hom(A)
sage: f = H([y,x^2+y]); f
Scheme endomorphism of Affine Space of dimension 2 over Rational Field
  Defn: Defined on coordinates by sending (x, y) to
        (y, x^2 + y)
sage: from sage.schemes.generic.morphism import is_SchemeMorphism
sage: is_SchemeMorphism(f)
True
```

CHAPTER

ELEVEN

DIVISORS ON SCHEMES

AUTHORS:

- William Stein
- David Kohel
- David Joyner
- Volker Braun (2010-07-16): Documentation, doctests, coercion fixes, bugfixes.

EXAMPLES:

```
sage: x,y,z = ProjectiveSpace(2, GF(5), names='x,y,z').gens()
sage: C = Curve(y^2*z^7 - x^9 - x*z^8)
sage: pts = C.rational_points(); pts
[(0 : 0 : 1), (0 : 1 : 0), (2 : 2 : 1), (2 : 3 : 1), (3 : 1 : 1), (3 : 4 : 1)]
sage: D1 = C.divisor(pts[0])*3
sage: D2 = C.divisor(pts[1])
sage: D3 = 10*C.divisor(pts[5])
sage: D1.parent() is D2.parent()
True
sage: D = D1 - D2 + D3; D
3*(x, y) - (x, z) + 10*(x + 2*z, y + z)
sage: D[1][0]
-1
sage: D[1][1]
Ideal (x, z) of Multivariate Polynomial Ring in x, y, z over Finite Field of size 5
sage: C.divisor([(3, pts[0]), (-1, pts[1]), (10,pts[5])])
3*(x, y) - (x, z) + 10*(x + 2*z, y + z)
```

sage.schemes.generic.divisor.**CurvePointToIdeal**(*C*, *P*)

Return the vanishing ideal of a point on a curve.

EXAMPLES:

```
sage: x,y = AffineSpace(2, QQ, names='xy').gens()
sage: C = Curve(y^2 - x^9 - x)
sage: from sage.schemes.generic.divisor import CurvePointToIdeal
sage: CurvePointToIdeal(C, (0,0))
Ideal (x, y) of Multivariate Polynomial Ring in x, y over Rational Field
```

class sage.schemes.generic.divisor.**Divisor_curve**(*v*, *parent=None*, *check=True*, *reduce=True*)

Bases: sage.schemes.generic.divisor.Divisor_generic

For any curve C, use `C.divisor(v)` to construct a divisor on C. Here v can be either

- a rational point on C

- a list of rational points
- a list of 2-tuples (c, P), where c is an integer and P is a rational point.

TODO: Divisors shouldn't be restricted to rational points. The problem is that the divisor group is the formal sum of the group of points on the curve, and there's no implemented notion of point on E/K that has coordinates in L. This is what should be implemented, by adding an appropriate class to `schemes/generic/morphism.py`.

EXAMPLES:

```
sage: E = EllipticCurve([0, 0, 1, -1, 0])
sage: P = E(0,0)
sage: 10*P
(161/16 : -2065/64 : 1)
sage: D = E.divisor(P)
sage: D
(x, y)
sage: 10*D
10*(x, y)
sage: E.divisor([P, P])
2*(x, y)
sage: E.divisor([(3,P), (-4,5*P)])
3*(x, y) - 4*(x - 1/4*z, y + 5/8*z)
```

coefficient(*P*)

Return the coefficient of a given point P in this divisor.

EXAMPLES:

```
sage: x,y = AffineSpace(2, GF(5), names='xy').gens()
sage: C = Curve(y^2 - x^9 - x)
sage: pts = C.rational_points(); pts
[(0, 0), (2, 2), (2, 3), (3, 1), (3, 4)]
sage: D = C.divisor(pts[0])
sage: D.coefficient(pts[0])
1
sage: D = C.divisor([(3,pts[0]), (-1,pts[1])]); D
3*(x, y) - (x - 2, y - 2)
sage: D.coefficient(pts[0])
3
sage: D.coefficient(pts[1])
-1
```

support()

Return the support of this divisor, which is the set of points that occur in this divisor with nonzero coefficients.

EXAMPLES:

```
sage: x,y = AffineSpace(2, GF(5), names='xy').gens()
sage: C = Curve(y^2 - x^9 - x)
sage: pts = C.rational_points(); pts
[(0, 0), (2, 2), (2, 3), (3, 1), (3, 4)]
sage: D = C.divisor_group()([(3,pts[0]), (-1, pts[1])]); D
3*(x, y) - (x - 2, y - 2)
sage: D.support()
[(0, 0), (2, 2)]
```

TESTS:

This checks that trac ticket #10732 is fixed:

```
sage: R.<x, y, z> = GF(5)[]
sage: C = Curve(x^7 + y^7 + z^7)
sage: pts = C.rational_points()
sage: D = C.divisor([(2, pts[0])])
sage: D.support()
[(0 : 4 : 1)]
sage: (D + D).support()
[(0 : 4 : 1)]
sage: E = C.divisor([(-3, pts[1]), (1, pts[2])])
sage: (D - 2*E).support()
[(0 : 4 : 1), (1 : 2 : 1), (2 : 1 : 1)]
sage: (D - D).support()
[]
```

class `sage.schemes.generic.divisor.`**`Divisor_generic`**(*v*, *parent*, *check=True*, *reduce=True*)

Bases: `sage.structure.formal_sum.FormalSum`

A Divisor.

`scheme`()

Return the scheme that this divisor is on.

EXAMPLES:

```
sage: A.<x, y> = AffineSpace(2, GF(5))
sage: C = Curve(y^2 - x^9 - x)
sage: pts = C.rational_points(); pts
[(0, 0), (2, 2), (2, 3), (3, 1), (3, 4)]
sage: D = C.divisor(pts[0])*3 - C.divisor(pts[1]); D
3*(x, y) - (x - 2, y - 2)
sage: D.scheme()
Affine Curve over Finite Field of size 5 defined by -x^9 + y^2 - x
```

`sage.schemes.generic.divisor.`**`is_Divisor`**(*x*)

Test whether `x` is an instance of `Divisor_generic`

INPUT:

- `x` – anything.

OUTPUT:

`True` or `False`.

EXAMPLES:

```
sage: from sage.schemes.generic.divisor import is_Divisor
sage: x,y = AffineSpace(2, GF(5), names='xy').gens()
sage: C = Curve(y^2 - x^9 - x)
sage: is_Divisor( C.divisor([]) )
True
sage: is_Divisor("Ceci n'est pas un diviseur")
False
```

CHAPTER

TWELVE

DIVISOR GROUPS

AUTHORS:

- David Kohel (2006): Initial version
- Volker Braun (2010-07-16): Documentation, doctests, coercion fixes, bugfixes.

`sage.schemes.generic.divisor_group.`**`DivisorGroup`**`(`*scheme*, *base_ring=None*`)`

Return the group of divisors on the scheme.

INPUT:

- `scheme` – a scheme.
- `base_ring` – usually either $\mathbf{Z}$ (default) or $\mathbf{Q}$. The coefficient ring of the divisors. Not to be confused with the base ring of the scheme!

OUTPUT:

An instance of `DivisorGroup_generic`.

EXAMPLES:

```
sage: from sage.schemes.generic.divisor_group import DivisorGroup
sage: DivisorGroup(Spec(ZZ))
Group of ZZ-Divisors on Spectrum of Integer Ring
sage: DivisorGroup(Spec(ZZ), base_ring=QQ)
Group of QQ-Divisors on Spectrum of Integer Ring
```

class `sage.schemes.generic.divisor_group.`**`DivisorGroup_curve`**`(`*scheme*, *base_ring*`)`

Bases: `sage.schemes.generic.divisor_group.DivisorGroup_generic`

Special case of the group of divisors on a curve.

class `sage.schemes.generic.divisor_group.`**`DivisorGroup_generic`**`(`*scheme*, *base_ring*`)`

Bases: `sage.structure.formal_sum.FormalSums`

The divisor group on a variety.

`base_extend``(`*R*`)`

EXAMPLES:

```
sage: from sage.schemes.generic.divisor_group import DivisorGroup
sage: DivisorGroup(Spec(ZZ),ZZ).base_extend(QQ)
Group of QQ-Divisors on Spectrum of Integer Ring
sage: DivisorGroup(Spec(ZZ),ZZ).base_extend(GF(7))
Group of (Finite Field of size 7)-Divisors on Spectrum of Integer Ring
```

Divisor groups are unique:

```
sage: A.<x, y> = AffineSpace(2, CC)
sage: C = Curve(y^2 - x^9 - x)
sage: DivisorGroup(C,ZZ).base_extend(QQ) is DivisorGroup(C,QQ)
True
```

scheme()

Return the scheme supporting the divisors.

EXAMPLES:

```
sage: from sage.schemes.generic.divisor_group import DivisorGroup
sage: Div = DivisorGroup(Spec(ZZ))    # indirect test
sage: Div.scheme()
Spectrum of Integer Ring
```

sage.schemes.generic.divisor_group.**is_DivisorGroup**(*x*)

Return whether x is a DivisorGroup_generic.

INPUT:

- x – anything.

OUTPUT:

True or False.

EXAMPLES:

```
sage: from sage.schemes.generic.divisor_group import is_DivisorGroup, DivisorGroup
sage: Div = DivisorGroup(Spec(ZZ), base_ring=QQ)
sage: is_DivisorGroup(Div)
True
sage: is_DivisorGroup('not a divisor')
False
```

CHAPTER THIRTEEN

AFFINE SCHEMES

13.1 Affine n space over a ring

sage.schemes.affine.affine_space.**AffineSpace**(*n*, *R=None*, *names='x'*)

Return affine space of dimension n over the ring R.

EXAMPLES:

The dimension and ring can be given in either order:

```
sage: AffineSpace(3, QQ, 'x')
Affine Space of dimension 3 over Rational Field
sage: AffineSpace(5, QQ, 'x')
Affine Space of dimension 5 over Rational Field
sage: A = AffineSpace(2, QQ, names='XY'); A
Affine Space of dimension 2 over Rational Field
sage: A.coordinate_ring()
Multivariate Polynomial Ring in X, Y over Rational Field
```

Use the divide operator for base extension:

```
sage: AffineSpace(5, names='x')/GF(17)
Affine Space of dimension 5 over Finite Field of size 17
```

The default base ring is **Z**:

```
sage: AffineSpace(5, names='x')
Affine Space of dimension 5 over Integer Ring
```

There is also an affine space associated to each polynomial ring:

```
sage: R = GF(7)['x,y,z']
sage: A = AffineSpace(R); A
Affine Space of dimension 3 over Finite Field of size 7
sage: A.coordinate_ring() is R
True
```

class sage.schemes.affine.affine_space.**AffineSpace_field**(*n*, *R*, *names*)

Bases: sage.schemes.affine.affine_space.AffineSpace_generic

EXAMPLES:

```
sage: AffineSpace(3, Zp(5), 'y')
Affine Space of dimension 3 over 5-adic Ring with capped relative precision 20
```

points_of_bounded_height(*bound*)

Returns an iterator of the points in self of absolute height of at most the given bound. Bound check is strict for the rational field. Requires self to be affine space over a number field. Uses the Doyle-Krumm algorithm for computing algebraic numbers up to a given height [Doyle-Krumm].

INPUT:

•`bound` - a real number

OUTPUT:

•an iterator of points in self

EXAMPLES:: sage: A.<x,y> = AffineSpace(QQ,2) sage: list(A.points_of_bounded_height(3)) [(0, 0), (1, 0), (-1, 0), (1/2, 0), (-1/2, 0), (2, 0), (-2, 0), (0, 1), (1, 1), (-1, 1), (1/2, 1), (-1/2, 1), (2, 1), (-2, 1), (0, -1), (1, -1), (-1, -1), (1/2, -1), (-1/2, -1), (2, -1), (-2, -1), (0, 1/2), (1, 1/2), (-1, 1/2), (1/2, 1/2), (-1/2, 1/2), (2, 1/2), (-2, 1/2), (0, -1/2), (1, -1/2), (-1, -1/2), (1/2, -1/2), (-1/2, -1/2), (2, -1/2), (-2, -1/2), (0, 2), (1, 2), (-1, 2), (1/2, 2), (-1/2, 2), (2, 2), (-2, 2), (0, -2), (1, -2), (-1, -2), (1/2, -2), (-1/2, -2), (2, -2), (-2, -2)]

```
sage: u = QQ['u'].0
sage: A.<x,y> = AffineSpace(NumberField(u^2 - 2,'v'), 2)
sage: len(list(A.points_of_bounded_height(6)))
121
```

weil_restriction()

Compute the Weil restriction of this affine space over some extension field. If the field is a finite field, then this computes the Weil restriction to the prime subfield.

OUTPUT: Affine space of dimension `d * self.dimension_relative()` over the base field of `self.base_ring()`.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<w> = NumberField(x^5-2)
sage: AK.<x,y> = AffineSpace(K,2)
sage: AK.weil_restriction()
Affine Space of dimension 10 over Rational Field
sage: R.<x> = K[]
sage: L.<v> = K.extension(x^2+1)
sage: AL.<x,y> = AffineSpace(L,2)
sage: AL.weil_restriction()
Affine Space of dimension 4 over Number Field in w with defining
polynomial x^5 - 2
```

class sage.schemes.affine.affine_space.**AffineSpace_finite_field**(*n*, *R*, *names*)

Bases: `sage.schemes.affine.affine_space.AffineSpace_field`

EXAMPLES:

```
sage: AffineSpace(3, Zp(5), 'y')
Affine Space of dimension 3 over 5-adic Ring with capped relative precision 20
```

class sage.schemes.affine.affine_space.**AffineSpace_generic**(*n*, *R*, *names*)

Bases: `sage.schemes.generic.ambient_space.AmbientSpace`, `sage.schemes.generic.scheme.AffineScheme`

Affine space of dimension n over the ring R.

EXAMPLES:

```
sage: X.<x,y,z> = AffineSpace(3, QQ)
sage: X.base_scheme()
```

```
Spectrum of Rational Field
sage: X.base_ring()
Rational Field
sage: X.category()
Category of schemes over Rational Field
sage: X.structure_morphism()
Scheme morphism:
  From: Affine Space of dimension 3 over Rational Field
  To:   Spectrum of Rational Field
  Defn: Structure map
```

Loading and saving:

```
sage: loads(X.dumps()) == X
True
```

We create several other examples of affine spaces:

```
sage: AffineSpace(5, PolynomialRing(QQ, 'z'), 'Z')
Affine Space of dimension 5 over Univariate Polynomial Ring in z over Rational Field

sage: AffineSpace(RealField(), 3, 'Z')
Affine Space of dimension 3 over Real Field with 53 bits of precision

sage: AffineSpace(Qp(7), 2, 'x')
Affine Space of dimension 2 over 7-adic Field with capped relative precision 20
```

Even 0-dimensional affine spaces are supported:

```
sage: AffineSpace(0)
Affine Space of dimension 0 over Integer Ring
```

change_ring(R)

Return an affine space over ring R and otherwise the same as self.

INPUT:

- R – commutative ring

OUTPUT:

- affine space over R

Note: There is no need to have any relation between R and the base ring of self, if you want to have such a relation, use `self.base_extend(R)` instead.

EXAMPLES:

```
sage: A.<x, y, z> = AffineSpace(3, ZZ)
sage: AQ = A.change_ring(QQ); AQ
Affine Space of dimension 3 over Rational Field
sage: AQ.change_ring(GF(5))
Affine Space of dimension 3 over Finite Field of size 5
```

coordinate_ring()

Return the coordinate ring of this scheme, if defined.

EXAMPLES:

```
sage: R = AffineSpace(2, GF(9,'alpha'), 'z').coordinate_ring(); R
Multivariate Polynomial Ring in z0, z1 over Finite Field in alpha of size 3^2
```

```
sage: AffineSpace(3, R, 'x').coordinate_ring()
Multivariate Polynomial Ring in x0, x1, x2 over Multivariate Polynomial Ring in z0, z1 over
```

ngens()

Return the number of generators of self, i.e. the number of variables in the coordinate ring of self.

EXAMPLES:

```
sage: AffineSpace(3, QQ).ngens()
3
sage: AffineSpace(7, ZZ).ngens()
7
```

projective_embedding(*i=None*, *PP=None*)

Returns a morphism from this space into an ambient projective space of the same dimension.

INPUT:

- `i` – integer (default: dimension of self = last coordinate) determines which projective embedding to compute. The embedding is that which has a 1 in the i-th coordinate, numbered from 0.
- `PP` – (default: None) ambient projective space, i.e., codomain of morphism; this is constructed if it is not given.

EXAMPLES:

```
sage: AA = AffineSpace(2, QQ, 'x')
sage: pi = AA.projective_embedding(0); pi
Scheme morphism:
  From: Affine Space of dimension 2 over Rational Field
  To:   Projective Space of dimension 2 over Rational Field
  Defn: Defined on coordinates by sending (x0, x1) to
        (1 : x0 : x1)
sage: z = AA(3,4)
sage: pi(z)
(1/4 : 3/4 : 1)
sage: pi(AA(0,2))
(1/2 : 0 : 1)
sage: pi = AA.projective_embedding(1); pi
Scheme morphism:
  From: Affine Space of dimension 2 over Rational Field
  To:   Projective Space of dimension 2 over Rational Field
  Defn: Defined on coordinates by sending (x0, x1) to
        (x0 : 1 : x1)
sage: pi(z)
(3/4 : 1/4 : 1)
sage: pi = AA.projective_embedding(2)
sage: pi(z)
(3 : 4 : 1)

sage: A.<x,y> = AffineSpace(ZZ,2)
sage: A.projective_embedding(2).codomain().affine_patch(2) == A
True
```

rational_points(*F=None*)

Return the list of F-rational points on the affine space self, where F is a given finite field, or the base ring of self.

EXAMPLES:

```
sage: A = AffineSpace(1, GF(3))
sage: A.rational_points()
[(0), (1), (2)]
sage: A.rational_points(GF(3^2, 'b'))
[(0), (b), (b + 1), (2*b + 1), (2), (2*b), (2*b + 2), (b + 2), (1)]

sage: AffineSpace(2, ZZ).rational_points(GF(2))
[(0, 0), (1, 0), (0, 1), (1, 1)]
```

TESTS:

```
sage: AffineSpace(2, QQ).rational_points()
Traceback (most recent call last):
...
TypeError: Base ring (= Rational Field) must be a finite field.
sage: AffineSpace(1, GF(3)).rational_points(ZZ)
Traceback (most recent call last):
...
TypeError: Second argument (= Integer Ring) must be a finite field.
```

subscheme(*X*)

Return the closed subscheme defined by X.

INPUT:

- `X` - a list or tuple of equations

EXAMPLES:

```
sage: A.<x,y> = AffineSpace(QQ, 2)
sage: X = A.subscheme([x, y^2, x*y^2]); X
Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x,
  y^2,
  x*y^2

sage: X.defining_polynomials ()
(x, y^2, x*y^2)
sage: I = X.defining_ideal(); I
Ideal (x, y^2, x*y^2) of Multivariate Polynomial Ring in x, y over Rational Field
sage: I.groebner_basis()
[y^2, x]
sage: X.dimension()
0
sage: X.base_ring()
Rational Field
sage: X.base_scheme()
Spectrum of Rational Field
sage: X.structure_morphism()
Scheme morphism:
  From: Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x,
  y^2,
  x*y^2
  To:   Spectrum of Rational Field
  Defn: Structure map
sage: X.dimension()
0
```

sage.schemes.affine.affine_space.**is_AffineSpace**(*x*)

Returns True if x is an affine space, i.e., an ambient space $\mathbb{A}^n_R$, where R is a ring and $n \geq 0$ is an integer.

EXAMPLES:

```
sage: from sage.schemes.affine.affine_space import is_AffineSpace
sage: is_AffineSpace(AffineSpace(5, names='x'))
True
sage: is_AffineSpace(AffineSpace(5, GF(9,'alpha'), names='x'))
True
sage: is_AffineSpace(Spec(ZZ))
False
```

13.2 Points on affine varieties

Scheme morphism for points on affine varieties

AUTHORS:

- David Kohel, William Stein
- Volker Braun (2011-08-08): Renamed classes, more documentation, misc cleanups.
- Ben Hutz (2013)

class `sage.schemes.affine.affine_point.`**`SchemeMorphism_point_affine`**(*X*, *v*, *check=True*)

Bases: `sage.schemes.generic.morphism.SchemeMorphism_point`

A rational point on an affine scheme.

INPUT:

- `X` – a subscheme of an ambient affine space over a ring R.
- `v` – a list/tuple/iterable of coordinates in R.
- `check` – boolean (optional, default:`True`). Whether to check the input for consistency.

EXAMPLES:

```
sage: A = AffineSpace(2, QQ)
sage: A(1,2)
(1, 2)
```

`global_height`(*prec=None*)

Returns the logarithmic height of the point.

INPUT:

- `prec` – desired floating point precision (default: default RealField precision).

OUTPUT:

- a real number

EXAMPLES:

```
sage: P.<x,y>=AffineSpace(QQ,2)
sage: Q=P(41,1/12)
sage: Q.global_height()
3.71357206670431
```

```
sage: P=AffineSpace(ZZ,4,'x')
sage: Q=P(3,17,-51,5)
sage: Q.global_height()
3.93182563272433

sage: R.<x>=PolynomialRing(QQ)
sage: k.<w>=NumberField(x^2+5)
sage: A=AffineSpace(k,2,'z')
sage: A([3,5*w+1]).global_height(prec=100)
2.4181409534757389986565376694
```

Todo

p-adic heights

add heights to integer.pyx and remove special case

homogenize(*n*)

Return the homogenization of `self` at the `nth` coordinate.

INPUT:

- `n` – integer between 0 and dimension of self, inclusive.

OUTPUT:

- A point in the projectivization of the codomain of `self`

EXAMPLES:

```
sage: A.<x,y> = AffineSpace(ZZ,2)
sage: Q = A(2,3)
sage: Q.homogenize(2).dehomogenize(2) == Q
True

::

sage: A.<x,y> = AffineSpace(QQ,2)
sage: Q = A(2,3)
sage: P = A(0,1)
sage: Q.homogenize(2).codomain() == P.homogenize(2).codomain()
True
```

nth_iterate(*f, n*)

Returns the point $f^n(self)$

INPUT:

- `f` – a `SchemeMorphism_polynomial` with `self` if `f.domain()`
- `n` – a positive integer.

OUTPUT:

- a point in `f.codomain()`

EXAMPLES:

```
sage: A.<x,y>=AffineSpace(QQ,2)
sage: H=Hom(A,A)
sage: f=H([(x-2*y^2)/x,3*x*y])
sage: A(9,3).nth_iterate(f,3)
(-104975/13123, -9566667)
```

```
sage: A.<x,y>=AffineSpace(ZZ,2)
sage: X=A.subscheme([x-y^2])
sage: H=Hom(X,X)
sage: f=H([9*y^2,3*y])
sage: X(9,3).nth_iterate(f,4)
(59049, 243)
```

orbit (*f*, *N*)

Returns the orbit of self by f. If n is an integer it returns $[self, f(self), \ldots, f^n(self)]$.

If n is a list or tuple $n = [m, k]$ it returns $[f^m(self), \ldots, f^k(self)]$.

INPUT:

- `f` – a `SchemeMorphism_polynomial` with `self` in `f.domain()`
- `n` – a non-negative integer or list or tuple of two non-negative integers

OUTPUT:

- a list of points in `f.codomain()`

EXAMPLES:

```
sage: A.<x,y>=AffineSpace(QQ,2)
sage: H=Hom(A,A)
sage: f=H([(x-2*y^2)/x,3*x*y])
sage: A(9,3).orbit(f,3)
[(9, 3), (-1, 81), (13123, -243), (-104975/13123, -9566667)]

sage: A.<x>=AffineSpace(QQ,1)
sage: H=Hom(A,A)
sage: f=H([(x-2)/x])
sage: A(1/2).orbit(f,[1,3])
[(-3), (5/3), (-1/5)]

sage: A.<x,y>=AffineSpace(ZZ,2)
sage: X=A.subscheme([x-y^2])
sage: H=Hom(X,X)
sage: f=H([9*y^2,3*y])
sage: X(9,3).orbit(f,(0,4))
[(9, 3), (81, 9), (729, 27), (6561, 81), (59049, 243)]
```

class `sage.schemes.affine.affine_point.`**`SchemeMorphism_point_affine_field`**(*X*, *v*, *check=True*)

Bases: `sage.schemes.affine.affine_point.SchemeMorphism_point_affine`

The Python constructor.

See `SchemeMorphism_point_affine` for details.

TESTS:

```
sage: from sage.schemes.affine.affine_point import SchemeMorphism_point_affine
sage: A3.<x,y,z> = AffineSpace(QQ, 3)
sage: SchemeMorphism_point_affine(A3(QQ), [1,2,3])
(1, 2, 3)
```

weil_restriction()

Compute the Weil restriction of this point over some extension field. If the field is a finite field, then this computes the Weil restriction to the prime subfield.

A Weil restriction of scalars - denoted $Res_{L/k}$ - is a functor which, for any finite extension of fields L/k and any algebraic variety X over L, produces another corresponding variety $Res_{L/k}(X)$, defined over k. It is useful for reducing questions about varieties over large fields to questions about more complicated varieties over smaller fields. This functor applied to a point gives the equivalent point on the Weil restriction of its codomain.

OUTPUT: Scheme point on the Weil restriction of the codomain of `self`.

EXAMPLES:

```
sage: A.<x,y,z> = AffineSpace(GF(5^3,'t'),3)
sage: X = A.subscheme([y^2-x*z, z^2+y])
sage: Y = X.weil_restriction()
sage: P = X([1,-1,1])
sage: Q = P.weil_restriction();Q
(1, 0, 0, 4, 0, 0, 1, 0, 0)
sage: Q.codomain() == Y
True

sage: R.<x> = QQ[]
sage: K.<w> = NumberField(x^5-2)
sage: R.<x> = K[]
sage: L.<v> = K.extension(x^2+w)
sage: A.<x,y> = AffineSpace(L,2)
sage: P = A([w^3-v,1+w+w*v])
sage: P.weil_restriction()
(w^3, -1, w + 1, w)
```

class `sage.schemes.affine.affine_point.`**`SchemeMorphism_point_affine_finite_field`**(*X*, *v*, *check=True*)

Bases: `sage.schemes.affine.affine_point.SchemeMorphism_point_affine_field`

The Python constructor.

See `SchemeMorphism_point_affine` for details.

TESTS:

```
sage: from sage.schemes.affine.affine_point import SchemeMorphism_point_affine
sage: A3.<x,y,z> = AffineSpace(QQ, 3)
sage: SchemeMorphism_point_affine(A3(QQ), [1,2,3])
(1, 2, 3)
```

`orbit_structure`(*f*)

Every point is preperiodic over a finite field. This function returns the pair $[m, n]$ where m is the preperiod and n is the period of the point `self` by `f`.

INPUT:

- `f` – a `SceneMorphism_polynomial` with `self` in `f.domain()`

OUTPUT:

- a list $[m, n]$ of integers

EXAMPLES:

```
sage: P.<x,y,z> = AffineSpace(GF(5),3)
sage: H = Hom(P,P)
sage: f = H([x^2 + y^2,y^2,z^2 + y * z])
sage: P(1,1,1).orbit_structure(f)
[0, 6]
```

```
sage: P.<x,y,z> = AffineSpace(GF(7),3)
sage: X = P.subscheme(x^2 - y^2)
sage: H = Hom(X,X)
sage: f = H([x^2,y^2,z^2])
sage: X(1,1,2).orbit_structure(f)
[0, 2]
```

```
sage: P.<x,y> = AffineSpace(GF(13),2)
sage: H = Hom(P,P)
sage: f = H([x^2 - y^2,y^2])
sage: P(3,4).orbit_structure(f)
[2, 6]
```

13.3 Morphisms on affine varieties

A morphism of schemes determined by rational functions that define what the morphism does on points in the ambient affine space.

AUTHORS:

- David Kohel, William Stein
- Volker Braun (2011-08-08): Renamed classes, more documentation, misc cleanups.
- Ben Hutz (2013-03) iteration functionality and new directory structure for affine/projective

class sage.schemes.affine.affine_morphism.**SchemeMorphism_polynomial_affine_space**(*parent*, *polys*, *check=True*)

Bases: sage.schemes.generic.morphism.SchemeMorphism_polynomial

A morphism of schemes determined by rational functions that define what the morphism does on points in the ambient affine space.

EXAMPLES:

```
sage: RA.<x,y> = QQ[]
sage: A2 = AffineSpace(RA)
sage: RP.<u,v,w> = QQ[]
sage: P2 = ProjectiveSpace(RP)
sage: H = A2.Hom(P2)
sage: f = H([x, y, 1])
sage: f
Scheme morphism:
  From: Affine Space of dimension 2 over Rational Field
  To:   Projective Space of dimension 2 over Rational Field
  Defn: Defined on coordinates by sending (x, y) to
        (x : y : 1)
```

dynatomic_polynomial(*period*)

For a map $f : \mathbb{A}^1 \to \mathbb{A}^1$ this function computes the (affine) dynatomic polynomial. The dynatomic polynomial is the analog of the cyclotomic polynomial and its roots are the points of formal period n.

ALGORITHM:

Homogenize to a map $f : \mathbb{P}^1 \to \mathbb{P}^1$ and compute the dynatomic polynomial there. Then, dehomogenize.

INPUT:

- `period` – a positive integer or a list/tuple $[m, n]$ where m is the preperiod and n is the period

OUTPUT:

- If possible, a single variable polynomial in the coordinate ring of `self`. Otherwise a fraction field element of the coordinate ring of `self`

EXAMPLES:

```
sage: A.<x,y> = AffineSpace(QQ,2)
sage: H = Hom(A,A)
sage: f = H([x^2+y^2,y^2])
sage: f.dynatomic_polynomial(2)
Traceback (most recent call last):
...
TypeError: Does not make sense in dimension >1

sage: A.<x> = AffineSpace(ZZ,1)
sage: H = Hom(A,A)
sage: f = H([(x^2+1)/x])
sage: f.dynatomic_polynomial(4)
2*x^12 + 18*x^10 + 57*x^8 + 79*x^6 + 48*x^4 + 12*x^2 + 1

sage: A.<x> = AffineSpace(CC,1)
sage: H = Hom(A,A)
sage: f = H([(x^2+1)/(3*x)])
sage: f.dynatomic_polynomial(3)
13.0000000000000*x^6 + 117.000000000000*x^4 + 78.0000000000000*x^2 +
1.00000000000000

sage: A.<x> = AffineSpace(QQ,1)
sage: H = Hom(A,A)
sage: f = H([x^2-10/9])
sage: f.dynatomic_polynomial([2,1])
531441*x^4 - 649539*x^2 - 524880

sage: A.<x> = AffineSpace(CC,1)
sage: H = Hom(A,A)
sage: f = H([x^2+CC.0])
sage: f.dynatomic_polynomial(2)
x^2 + x + 1.00000000000000 + 1.00000000000000*I

sage: K.<c> = FunctionField(QQ)
sage: A.<x> = AffineSpace(K,1)
sage: f = Hom(A,A)([x^2 + c])
sage: f.dynatomic_polynomial(4)
x^12 + 6*c*x^10 + x^9 + (15*c^2 + 3*c)*x^8 + 4*c*x^7 + (20*c^3 + 12*c^2 + 1)*x^6
+ (6*c^2 + 2*c)*x^5 + (15*c^4 + 18*c^3 + 3*c^2 + 4*c)*x^4 + (4*c^3 + 4*c^2 + 1)*x^3
+ (6*c^5 + 12*c^4 + 6*c^3 + 5*c^2 + c)*x^2 + (c^4 + 2*c^3 + c^2 + 2*c)*x
+ c^6 + 3*c^5 + 3*c^4 + 3*c^3 + 2*c^2 + 1
```

global_height (*prec=None*)

Returns the maximum of the heights of the coefficients in any of the coordinate functions of `self`.

INPUT:

- `prec` – desired floating point precision (default: default RealField precision).

OUTPUT:

- a real number

EXAMPLES:

```
sage: A.<x>=AffineSpace(QQ,1)
sage: H=Hom(A,A)
sage: f=H([1/1331*x^2+4000]);
sage: f.global_height()
8.29404964010203

sage: R.<x>=PolynomialRing(QQ)
sage: k.<w>=NumberField(x^2+5)
sage: A.<x,y>=AffineSpace(k,2)
sage: H=Hom(A,A)
sage: f=H([13*w*x^2+4*y, 1/w*y^2]);
sage: f.global_height(prec=100)
3.3696683136785869233538671082
```

Todo

add heights to integer.pyx and remove special case

homogenize(*n*)

Return the homogenization of `self`. If `self.domain()` is a subscheme, the domain of the homogenized map is the projective embedding of `self.domain()`. The domain and codomain can be homogenized at different coordinates: `n[0]` for the domain and `n[1]` for the codomain.

INPUT:

- **n – a tuple of nonnegative integers. If n is an integer, then the two values of** the tuple are assumed to be the same.

OUTPUT:

- `SchemMorphism_polynomial_projective_space`

EXAMPLES:

```
sage: A.<x,y> = AffineSpace(ZZ,2)
sage: H = Hom(A,A)
sage: f = H([(x^2-2)/x^5,y^2])
sage: f.homogenize(2)
Scheme endomorphism of Projective Space of dimension 2 over Integer Ring
  Defn: Defined on coordinates by sending (x0 : x1 : x2) to
        (x0^2*x2^5 - 2*x2^7 : x0^5*x1^2 : x0^5*x2^2)

sage: A.<x,y> = AffineSpace(CC,2)
sage: H = Hom(A,A)
sage: f = H([(x^2-2)/(x*y),y^2-x])
sage: f.homogenize((2,0))
Scheme morphism:
  From: Projective Space of dimension 2 over Complex Field with 53 bits of precision
  To:   Projective Space of dimension 2 over Complex Field with 53 bits of precision
  Defn: Defined on coordinates by sending (x0 : x1 : x2) to
        (x0*x1*x2^2 : x0^2*x2^2 + (-2.00000000000000)*x2^4 : x0*x1^3 - x0^2*x1*x2)

sage: A.<x,y> = AffineSpace(ZZ,2)
sage: X = A.subscheme([x-y^2])
sage: H = Hom(X,X)
sage: f = H([9*y^2,3*y])
sage: f.homogenize(2)
Scheme endomorphism of Closed subscheme of Projective Space of dimension 2 over Integer Ring
  -x1^2 + x0*x2
```

```
  Defn: Defined on coordinates by sending (x0 : x1 : x2) to
        (9*x0*x2 : 3*x1*x2 : x2^2)

sage: R.<t> = PolynomialRing(ZZ)
sage: A.<x,y> = AffineSpace(R,2)
sage: H = Hom(A,A)
sage: f = H([(x^2-2)/y,y^2-x])
sage: f.homogenize((2,0))
Scheme morphism:
  From: Projective Space of dimension 2 over Univariate Polynomial Ring in t over Integer Ri
  To:   Projective Space of dimension 2 over Univariate Polynomial Ring in t over Integer Ri
  Defn: Defined on coordinates by sending (x0 : x1 : x2) to
        (x1*x2^2 : x0^2*x2 + (-2)*x2^3 : x1^3 - x0*x1*x2)

sage: A.<x> = AffineSpace(QQ,1)
sage: H = End(A)
sage: f = H([x^2-1])
sage: f.homogenize((1,0))
Scheme morphism:
  From: Projective Space of dimension 1 over Rational Field
  To:   Projective Space of dimension 1 over Rational Field
  Defn: Defined on coordinates by sending (x0 : x1) to
        (x1^2 : x0^2 - x1^2)

R.<a> = PolynomialRing(QQbar)
A.<x,y> = AffineSpace(R,2)
H = End(A)
f = H([QQbar(sqrt(2))*x*y,a*x^2])
f.homogenize(2)
Scheme endomorphism of Projective Space of dimension 2 over Univariate
Polynomial Ring in a over Algebraic Field
  Defn: Defined on coordinates by sending (x0 : x1 : x2) to
        (1.414213562373095?*x0*x1 : a*x0^2 : x2^2)

sage: P.<x,y,z> = AffineSpace(QQ,3)
sage: H = End(P)
sage: f = H([x^2 - 2*x*y + z*x, z^2 -y^2 , 5*z*y])
sage: f.homogenize(2).dehomogenize(2) == f
True

sage: K.<c> = FunctionField(QQ)
sage: A.<x> = AffineSpace(K,1)
sage: f = Hom(A,A)([x^2 + c])
sage: f.homogenize(1)
Scheme endomorphism of Projective Space of dimension 1 over Rational function field in c ove
  Defn: Defined on coordinates by sending (x0 : x1) to
        (x0^2 + c*x1^2 : x1^2)
```

jacobian()

Returns the Jacobian matrix of partial derivitive of `self` in which the `(i,j)` entry of the Jacobian matrix is the partial derivative `diff(functions[i], variables[j])`.

OUTPUT:

- matrix with coordinates in the coordinate ring of `self`

EXAMPLES:

```
sage: A.<z> = AffineSpace(QQ,1)
sage: H = End(A)
sage: f = H([z^2-3/4])
sage: f.jacobian()
[2*z]

sage: A.<x,y> = AffineSpace(QQ,2)
sage: H = End(A)
sage: f = H([x^3 - 25*x + 12*y,5*y^2*x - 53*y + 24])
sage: f.jacobian()
[ 3*x^2 - 25          12]
[      5*y^2 10*x*y - 53]

sage: A.<x,y> = AffineSpace(ZZ,2)
sage: H = End(A)
sage: f = H([(x^2 - x*y)/(1+y),(5+y)/(2+x)])
sage: f.jacobian()
[          (2*x - y)/(y + 1) (-x^2 - x)/(y^2 + 2*y + 1)]
[  (-y - 5)/(x^2 + 4*x + 4)                  1/(x + 2)]
```

multiplier(*P*, *n*, *check=True*)

Returns the multiplier of `self` at the point `P` of period `n`. `self` must be an endomorphism.

INPUT:

- `P` - a point on domain of `self`
- `n` - a positive integer, the period of `P`
- `check` – verify that `P` has period `n`, Default:True

OUTPUT:

- a square matrix of size `self.codomain().dimension_relative()` in the `base_ring` of `self`

EXAMPLES:

```
sage: P.<x,y> = AffineSpace(QQ,2)
sage: H = End(P)
sage: f = H([x^2,y^2])
sage: f.multiplier(P([1,1]),1)
[2 0]
[0 2]

sage: P.<x,y,z> = AffineSpace(QQ,3)
sage: H = End(P)
sage: f = H([x,y^2,z^2 - y])
sage: f.multiplier(P([1/2,1,0]),2)
[1 0 0]
[0 4 0]
[0 0 0]

sage: P.<x> = AffineSpace(CC,1)
sage: H = End(P)
sage: f = H([x^2 + 1/2])
sage: f.multiplier(P([0.5 + 0.5*I]),1)
[1.00000000000000 + 1.00000000000000*I]
```

```
sage: R.<t> = PolynomialRing(CC,1)
sage: P.<x> = AffineSpace(R,1)
sage: H = End(P)
sage: f = H([x^2 - t^2 + t])
sage: f.multiplier(P([-t + 1]),1)
[(-2.00000000000000)*t + 2.00000000000000]

sage: P.<x,y> = AffineSpace(QQ,2)
sage: X=P.subscheme([x^2-y^2])
sage: H = End(X)
sage: f = H([x^2,y^2])
sage: f.multiplier(X([1,1]),1)
[2 0]
[0 2]
```

nth_iterate(*P*, *n*)

Returns the point $self f^n(P)$

INPUT:

- P – a point in `self.domain()`
- n – a positive integer.

OUTPUT:

- a point in `self.codomain()`

EXAMPLES:

```
sage: A.<x,y>=AffineSpace(QQ,2)
sage: H=Hom(A,A)
sage: f=H([(x-2*y^2)/x,3*x*y])
sage: f.nth_iterate(A(9,3),3)
(-104975/13123, -9566667)

sage: A.<x,y>=AffineSpace(ZZ,2)
sage: X=A.subscheme([x-y^2])
sage: H=Hom(X,X)
sage: f=H([9*y^2,3*y])
sage: f.nth_iterate(X(9,3),4)
(59049, 243)

sage: R.<t>=PolynomialRing(QQ)
sage: A.<x,y>=AffineSpace(FractionField(R),2)
sage: H=Hom(A,A)
sage: f=H([(x-t*y^2)/x,t*x*y])
sage: f.nth_iterate(A(1,t),3)
((-t^16 + 3*t^13 - 3*t^10 + t^7 + t^5 + t^3 - 1)/(t^5 + t^3 - 1), -t^9 - t^7 + t^4)
```

nth_iterate_map(*n*)

This function returns the nth iterate of `self`

ALGORITHM:

Uses a form of successive squaring to reducing computations.

Todo

This could be improved.

INPUT:

- n - a positive integer.

OUTPUT:

- A map between Affine spaces

EXAMPLES:

```
sage: A.<x,y>=AffineSpace(ZZ,2)
sage: H=Hom(A,A)
sage: f=H([(x^2-2)/(2*y),y^2-3*x])
sage: f.nth_iterate_map(2)
Scheme endomorphism of Affine Space of dimension 2 over Integer Ring
  Defn: Defined on coordinates by sending (x, y) to
        ((x^4 - 4*x^2 - 8*y^2 + 4)/(8*y^4 - 24*x*y^2), (2*y^5 - 12*x*y^3
+ 18*x^2*y - 3*x^2 + 6)/(2*y))

sage: A.<x>=AffineSpace(QQ,1)
sage: H=Hom(A,A)
sage: f=H([(3*x^2-2)/(x)])
sage: f.nth_iterate_map(3)
Scheme endomorphism of Affine Space of dimension 1 over Rational Field
  Defn: Defined on coordinates by sending (x) to
        ((2187*x^8 - 6174*x^6 + 6300*x^4 - 2744*x^2 + 432)/(81*x^7 -
168*x^5 + 112*x^3 - 24*x))

sage: A.<x,y>=AffineSpace(ZZ,2)
sage: X=A.subscheme([x-y^2])
sage: H=Hom(X,X)
sage: f=H([9*x^2,3*y])
sage: f.nth_iterate_map(2)
Scheme endomorphism of Closed subscheme of Affine Space of dimension 2
over Integer Ring defined by:
  -y^2 + x
  Defn: Defined on coordinates by sending (x, y) to
        (729*x^4, 9*y)
```

orbit (P, n)

Returns the orbit of P by `self`. If n is an integer it returns $[P, self f(P), \ldots, self f^n(P)]$.

If n is a list or tuple $n = [m, k]$ it returns $[self f^m(P), \ldots, self f^k(P)]$

INPUT:

- P – a point in `self.domain()`
- n – a non-negative integer or list or tuple of two non-negative integers

OUTPUT:

- a list of points in `self.codomain()`

EXAMPLES:

```
sage: A.<x,y>=AffineSpace(QQ,2)
sage: H=Hom(A,A)
sage: f=H([(x-2*y^2)/x,3*x*y])
sage: f.orbit(A(9,3),3)
[(9, 3), (-1, 81), (13123, -243), (-104975/13123, -9566667)]
```

```
sage: A.<x>=AffineSpace(QQ,1)
sage: H=Hom(A,A)
sage: f=H([(x-2)/x])
sage: f.orbit(A(1/2),[1,3])
[(-3), (5/3), (-1/5)]
```

```
sage: A.<x,y>=AffineSpace(ZZ,2)
sage: X=A.subscheme([x-y^2])
sage: H=Hom(X,X)
sage: f=H([9*y^2,3*y])
sage: f.orbit(X(9,3),(0,4))
[(9, 3), (81, 9), (729, 27), (6561, 81), (59049, 243)]
```

```
sage: R.<t>=PolynomialRing(QQ)
sage: A.<x,y>=AffineSpace(FractionField(R),2)
sage: H=Hom(A,A)
sage: f=H([(x-t*y^2)/x,t*x*y])
sage: f.orbit(A(1,t),3)
[(1, t), (-t^3 + 1, t^2), ((-t^5 - t^3 + 1)/(-t^3 + 1), -t^6 + t^3),
((-t^16 + 3*t^13 - 3*t^10 + t^7 + t^5 + t^3 - 1)/(t^5 + t^3 - 1), -t^9 -
t^7 + t^4)]
```

class `sage.schemes.affine.affine_morphism.`**`SchemeMorphism_polynomial_affine_space_field`**(*parent*, *polys*, *check=T*

Bases: `sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space`

The Python constructor.

See `SchemeMorphism_polynomial` for details.

INPUT:

- `parent` – Hom
- `polys` – list or tuple of polynomial or rational functions
- `check` – Boolean

OUTPUT:

- `SchemeMorphism_polynomial_affine_space`

EXAMPLES:

```
sage: A.<x,y>=AffineSpace(ZZ,2)
sage: H=Hom(A,A)
sage: H([3/5*x^2,y^2/(2*x^2)])
Traceback (most recent call last):
...
TypeError: polys (=[3/5*x^2, y^2/(2*x^2)]) must be rational functions in
Multivariate Polynomial Ring in x, y over Integer Ring
```

```
sage: A.<x,y>=AffineSpace(ZZ,2)
sage: H=Hom(A,A)
sage: H([3*x^2/(5*y),y^2/(2*x^2)])
Scheme endomorphism of Affine Space of dimension 2 over Integer Ring
  Defn: Defined on coordinates by sending (x, y) to
        (3*x^2/(5*y), y^2/(2*x^2))
```

```
sage: A.<x,y>=AffineSpace(QQ,2)
```

```
sage: H=Hom(A,A)
sage: H([3/2*x^2,y^2])
Scheme endomorphism of Affine Space of dimension 2 over Rational Field
  Defn: Defined on coordinates by sending (x, y) to
        (3/2*x^2, y^2)
```

```
sage: A.<x,y>=AffineSpace(QQ,2)
sage: X=A.subscheme([x-y^2])
sage: H=Hom(X,X)
sage: H([9/4*x^2,3/2*y])
Scheme endomorphism of Closed subscheme of Affine Space of dimension 2
over Rational Field defined by:
  -y^2 + x
  Defn: Defined on coordinates by sending (x, y) to
        (9/4*x^2, 3/2*y)
```

```
sage: P.<x,y,z>=ProjectiveSpace(ZZ,2)
sage: H=Hom(P,P)
sage: f=H([5*x^3 + 3*x*y^2-y^3,3*z^3 + y*x^2, x^3-z^3])
sage: f.dehomogenize(2)
Scheme endomorphism of Affine Space of dimension 2 over Integer Ring
  Defn: Defined on coordinates by sending (x0, x1) to
        ((5*x0^3 + 3*x0*x1^2 - x1^3)/(x0^3 - 1), (x0^2*x1 + 3)/(x0^3 - 1))
```

weil_restriction()

Compute the Weil restriction of this morphism over some extension field. If the field is a finite field, then this computes the Weil restriction to the prime subfield.

A Weil restriction of scalars - denoted $Res_{L/k}$ - is a functor which, for any finite extension of fields L/k and any algebraic variety X over L, produces another corresponding variety $Res_{L/k}(X)$, defined over k. It is useful for reducing questions about varieties over large fields to questions about more complicated varieties over smaller fields. Since it is a functor it also applied to morphisms. In particular, the functor applied to a morphism gives the equivalent morphism from the Weil restriction of the domain to the Weil restriction of the codomain.

OUTPUT: Scheme morphism on the Weil restrictions of the domain and codomain of `self`.

EXAMPLES:

```
sage: K.<v> = QuadraticField(5)
sage: A.<x,y> = AffineSpace(K,2)
sage: H = End(A)
sage: f = H([x^2-y^2,y^2])
sage: f.weil_restriction()
Scheme endomorphism of Affine Space of dimension 4 over Rational Field
  Defn: Defined on coordinates by sending (z0, z1, z2, z3) to
        (z0^2 + 5*z1^2 - z2^2 - 5*z3^2, 2*z0*z1 - 2*z2*z3, z2^2 + 5*z3^2, 2*z2*z3)
```

```
sage: K.<v> = QuadraticField(5)
sage: PS.<x,y> = AffineSpace(K,2)
sage: H = Hom(PS,PS)
sage: f = H([x,y])
sage: F = f.weil_restriction()
sage: P = PS(2,1)
sage: Q = P.weil_restriction()
sage: f(P).weil_restriction() == F(Q)
True
```

class sage.schemes.affine.affine_morphism.**SchemeMorphism_polynomial_affine_space_finite_field**

Bases: sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space_field

The Python constructor.

See SchemeMorphism_polynomial for details.

INPUT:

- parent – Hom
- polys – list or tuple of polynomial or rational functions
- check – Boolean

OUTPUT:

- SchemeMorphism_polynomial_affine_space

EXAMPLES:

```
sage: A.<x,y>=AffineSpace(ZZ,2)
sage: H=Hom(A,A)
sage: H([3/5*x^2,y^2/(2*x^2)])
Traceback (most recent call last):
...
TypeError: polys (=[3/5*x^2, y^2/(2*x^2)]) must be rational functions in
Multivariate Polynomial Ring in x, y over Integer Ring

sage: A.<x,y>=AffineSpace(ZZ,2)
sage: H=Hom(A,A)
sage: H([3*x^2/(5*y),y^2/(2*x^2)])
Scheme endomorphism of Affine Space of dimension 2 over Integer Ring
  Defn: Defined on coordinates by sending (x, y) to
        (3*x^2/(5*y), y^2/(2*x^2))


sage: A.<x,y>=AffineSpace(QQ,2)
sage: H=Hom(A,A)
sage: H([3/2*x^2,y^2])
Scheme endomorphism of Affine Space of dimension 2 over Rational Field
  Defn: Defined on coordinates by sending (x, y) to
        (3/2*x^2, y^2)


sage: A.<x,y>=AffineSpace(QQ,2)
sage: X=A.subscheme([x-y^2])
sage: H=Hom(X,X)
sage: H([9/4*x^2,3/2*y])
Scheme endomorphism of Closed subscheme of Affine Space of dimension 2
over Rational Field defined by:
  -y^2 + x
  Defn: Defined on coordinates by sending (x, y) to
        (9/4*x^2, 3/2*y)

sage: P.<x,y,z>=ProjectiveSpace(ZZ,2)
sage: H=Hom(P,P)
sage: f=H([5*x^3 + 3*x*y^2-y^3,3*z^3 + y*x^2, x^3-z^3])
sage: f.dehomogenize(2)
Scheme endomorphism of Affine Space of dimension 2 over Integer Ring
```

```
Defn: Defined on coordinates by sending (x0, x1) to
      ((5*x0^3 + 3*x0*x1^2 - x1^3)/(x0^3 - 1), (x0^2*x1 + 3)/(x0^3 - 1))
```

cyclegraph()

returns Digraph of all orbits of self mod p. For subschemes, only points on the subscheme whose image are also on the subscheme are in the digraph.

OUTPUT:

- a digraph

EXAMPLES:

```
sage: P.<x,y>=AffineSpace(GF(5),2)
sage: H=Hom(P,P)
sage: f=H([x^2-y,x*y+1])
sage: f.cyclegraph()
Looped digraph on 25 vertices
```

```
sage: P.<x>=AffineSpace(GF(3^3,'t'),1)
sage: H=Hom(P,P)
sage: f=H([x^2-1])
sage: f.cyclegraph()
Looped digraph on 27 vertices
```

```
sage: P.<x,y>=AffineSpace(GF(7),2)
sage: X=P.subscheme(x-y)
sage: H=Hom(X,X)
sage: f=H([x^2,y^2])
sage: f.cyclegraph()
Looped digraph on 7 vertices
```

orbit_structure(*P*)

Every point is preperiodic over a finite field. This function returns the pair $[m, n]$ where m is the preperiod and n is the period of the point `P` by `self`.

INPUT:

- `P` – a point in `self.domain()`

OUTPUT:

- a list $[m, n]$ of integers

EXAMPLES:

```
sage: A.<x,y> = AffineSpace(GF(13),2)
sage: H = Hom(A,A)
sage: f = H([x^2 - 1, y^2])
sage: f.orbit_structure(A(2,3))
[1, 6]
```

```
sage: A.<x,y,z> = AffineSpace(GF(49, 't'),3)
sage: H = Hom(A,A)
sage: f = H([x^2 - z, x - y + z, y^2 - x^2])
sage: f.orbit_structure(A(1,1,2))
[7, 6]
```

13.4 Enumeration of rational points on affine schemes

Naive algorithms for enumerating rational points over **Q** or finite fields over for general schemes.

> **Warning:** Incorrect results and infinite loops may occur if using a wrong function. (For instance using an affine function for a projective scheme or a finite field function for a scheme defined over an infinite field.)

EXAMPLES:

Affine, over **Q**:

```
sage: from sage.schemes.affine.affine_rational_point import enum_affine_rational_field
sage: A.<x,y,z> = AffineSpace(3,QQ)
sage: S = A.subscheme([2*x-3*y])
sage: enum_affine_rational_field(S,2)
[(0, 0, -2), (0, 0, -1), (0, 0, -1/2), (0, 0, 0),
 (0, 0, 1/2), (0, 0, 1), (0, 0, 2)]
```

Affine over a finite field:

```
sage: from sage.schemes.affine.affine_rational_point import enum_affine_finite_field
sage: A.<w,x,y,z> = AffineSpace(4,GF(2))
sage: enum_affine_finite_field(A(GF(2)))
[(0, 0, 0, 0), (0, 0, 0, 1), (0, 0, 1, 0), (0, 0, 1, 1), (0, 1, 0, 0),
 (0, 1, 0, 1), (0, 1, 1, 0), (0, 1, 1, 1), (1, 0, 0, 0), (1, 0, 0, 1),
 (1, 0, 1, 0), (1, 0, 1, 1), (1, 1, 0, 0), (1, 1, 0, 1), (1, 1, 1, 0),
 (1, 1, 1, 1)]
```

AUTHORS:

- David R. Kohel <kohel@maths.usyd.edu.au>: original version.
- John Cremona and Charlie Turner <charlotteturner@gmail.com> (06-2010): improvements to clarity and documentation.

sage.schemes.affine.affine_rational_point.**enum_affine_finite_field**(*X*)

Enumerates affine points on scheme `X` defined over a finite field.

INPUT:

- `X` - a scheme defined over a finite field or a set of abstract rational points of such a scheme.

OUTPUT:

- a list containing the affine points of `X` over the finite field, sorted.

EXAMPLES:

```
sage: F = GF(7)
sage: A.<w,x,y,z> = AffineSpace(4,F)
sage: C = A.subscheme([w^2+x+4,y*z*x-6,z*y+w*x])
sage: from sage.schemes.affine.affine_rational_point import enum_affine_finite_field
sage: enum_affine_finite_field(C(F))
[]
sage: C = A.subscheme([w^2+x+4,y*z*x-6])
sage: enum_affine_finite_field(C(F))
[(0, 3, 1, 2), (0, 3, 2, 1), (0, 3, 3, 3), (0, 3, 4, 4), (0, 3, 5, 6),
(0, 3, 6, 5), (1, 2, 1, 3), (1, 2, 2, 5), (1, 2, 3, 1), (1, 2, 4, 6),
(1, 2, 5, 2), (1, 2, 6, 4), (2, 6, 1, 1), (2, 6, 2, 4), (2, 6, 3, 5),
(2, 6, 4, 2), (2, 6, 5, 3), (2, 6, 6, 6), (3, 1, 1, 6), (3, 1, 2, 3),
(3, 1, 3, 2), (3, 1, 4, 5), (3, 1, 5, 4), (3, 1, 6, 1), (4, 1, 1, 6),
```

```
(4, 1, 2, 3), (4, 1, 3, 2), (4, 1, 4, 5), (4, 1, 5, 4), (4, 1, 6, 1),
(5, 6, 1, 1), (5, 6, 2, 4), (5, 6, 3, 5), (5, 6, 4, 2), (5, 6, 5, 3),
(5, 6, 6, 6), (6, 2, 1, 3), (6, 2, 2, 5), (6, 2, 3, 1), (6, 2, 4, 6),
(6, 2, 5, 2), (6, 2, 6, 4)]

sage: A.<x,y,z> = AffineSpace(3,GF(3))
sage: S = A.subscheme(x+y)
sage: enum_affine_finite_field(S)
[(0, 0, 0), (0, 0, 1), (0, 0, 2), (1, 2, 0), (1, 2, 1), (1, 2, 2),
(2, 1, 0), (2, 1, 1), (2, 1, 2)]
```

ALGORITHM:

Checks all points in affine space to see if they lie on X.

> **Warning:** If `X` is defined over an infinite field, this code will not finish!

AUTHORS:

- John Cremona and Charlie Turner (06-2010)

`sage.schemes.affine.affine_rational_point.`**`enum_affine_number_field`**(*X*, *B*)

Enumerates affine points on scheme `X` defined over a number field. Simply checks all of the points of absolute height up to `B` and adds those that are on the scheme to the list.

INPUT:

- `X` - a scheme defined over a number field
- `B` - a real number

OUTPUT:

- a list containing the affine points of `X` of absolute height up to `B`, sorted.

EXAMPLES:

```
sage: from sage.schemes.affine.affine_rational_point import enum_affine_number_field
sage: u = QQ['u'].0
sage: K = NumberField(u^2 + 2,'v')
sage: A.<x,y,z> = AffineSpace(K, 3)
sage: X = A.subscheme([y^2 - x])
sage: enum_affine_number_field(X(K),4)
[(0, 0, -1), (0, 0, -v), (0, 0, -1/2*v), (0, 0, 0), (0, 0, 1/2*v), (0, 0, v), (0, 0, 1),
(1, -1, -1), (1, -1, -v), (1, -1, -1/2*v), (1, -1, 0), (1, -1, 1/2*v), (1, -1, v), (1, -1, 1),
(1, 1, -1), (1, 1, -v), (1, 1, -1/2*v), (1, 1, 0), (1, 1, 1/2*v), (1, 1, v), (1, 1, 1)]

sage: u = QQ['u'].0
sage: K = NumberField(u^2 + 3,'v')
sage: A.<x,y> = AffineSpace(K,2)
sage: X=A.subscheme(x-y)
sage: from sage.schemes.affine.affine_rational_point import enum_affine_number_field
sage: enum_affine_number_field(X,3)
[(-1, -1), (-1/2*v - 1/2, -1/2*v - 1/2), (1/2*v - 1/2, 1/2*v - 1/2), (0, 0), (-1/2*v + 1/2, -1/2
(1/2*v + 1/2, 1/2*v + 1/2), (1, 1)]
```

`sage.schemes.affine.affine_rational_point.`**`enum_affine_rational_field`**(*X*, *B*)

Enumerates affine rational points on scheme `X` (defined over $\mathbf{Q}$) up to bound `B`.

INPUT:

- X - a scheme or set of abstract rational points of a scheme;
- B - a positive integer bound.

OUTPUT:

- a list containing the affine points of X of height up to B, sorted.

EXAMPLES:

```
sage: A.<x,y,z> = AffineSpace(3,QQ)
sage: from sage.schemes.affine.affine_rational_point import enum_affine_rational_field
sage: enum_affine_rational_field(A(QQ),1)
[(-1, -1, -1), (-1, -1, 0), (-1, -1, 1), (-1, 0, -1), (-1, 0, 0), (-1, 0, 1),
(-1, 1, -1), (-1, 1, 0), (-1, 1, 1), (0, -1, -1), (0, -1, 0), (0, -1, 1),
(0, 0, -1), (0, 0, 0), (0, 0, 1), (0, 1, -1), (0, 1, 0), (0, 1, 1), (1, -1, -1),
(1, -1, 0), (1, -1, 1), (1, 0, -1), (1, 0, 0), (1, 0, 1), (1, 1, -1), (1, 1, 0),
(1, 1, 1)]
```

```
sage: A.<w,x,y,z> = AffineSpace(4,QQ)
sage: S = A.subscheme([x^2-y*z+3,w^3+z+y^2])
sage: enum_affine_rational_field(S(QQ),2)
[]
sage: enum_affine_rational_field(S(QQ),3)
[(-2, 0, -3, -1)]
```

```
sage: A.<x,y> = AffineSpace(2,QQ)
sage: C = Curve(x^2+y-x)
sage: enum_affine_rational_field(C,10)
[(-2, -6), (-1, -2), (0, 0), (1, 0), (2, -2), (3, -6)]
```

AUTHORS:

- David R. Kohel <kohel@maths.usyd.edu.au>: original version.
- Charlie Turner (06-2010): small adjustments.

13.5 Set of homomorphisms between two affine schemes

For schemes X and Y, this module implements the set of morphisms $Hom(X, Y)$. This is done by SchemeHomset_generic.

As a special case, the Hom-sets can also represent the points of a scheme. Recall that the K-rational points of a scheme X over k can be identified with the set of morphisms $Spec(K) \to X$. In Sage the rational points are implemented by such scheme morphisms. This is done by SchemeHomset_points and its subclasses.

Note: You should not create the Hom-sets manually. Instead, use the Hom() method that is inherited by all schemes.

AUTHORS:

- William Stein (2006): initial version.

class sage.schemes.affine.affine_homset.**SchemeHomset_points_affine**(*X*, *Y*, *category=None*, *check=True*, *base=Integer Ring*)

Bases: sage.schemes.generic.homset.SchemeHomset_points

Set of rational points of an affine variety.

INPUT:

See SchemeHomset_generic.

EXAMPLES:

```
sage: from sage.schemes.affine.affine_homset import SchemeHomset_points_affine
sage: SchemeHomset_points_affine(Spec(QQ), AffineSpace(ZZ,2))
Set of rational points of Affine Space of dimension 2 over Rational Field
```

points (*B=0*)

Return some or all rational points of an affine scheme.

INPUT:

- B – integer (optional, default: 0). The bound for the height of the coordinates.

OUTPUT:

- If the base ring is a finite field: all points of the scheme, given by coordinate tuples.
- If the base ring is $\mathbf{Q}$ or $\mathbf{Z}$: the subset of points whose coordinates have height B or less.

EXAMPLES: The bug reported at #11526 is fixed:

```
sage: A2 = AffineSpace(ZZ,2)
sage: F = GF(3)
sage: A2(F).points()
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]
```

```
sage: R = ZZ
sage: A.<x,y> = R[]
sage: I = A.ideal(x^2-y^2-1)
sage: V = AffineSpace(R,2)
sage: X = V.subscheme(I)
sage: M = X(R)
sage: M.points(1)
[(-1, 0), (1, 0)]
```

```
sage: u = QQ['u'].0
sage: K.<v> = NumberField(u^2 + 3)
sage: A.<x,y> = AffineSpace(K,2)
sage: len(A(K).points(9))
361
```

```
sage: A.<x,y> = AffineSpace(QQ,2)
sage: E = A.subscheme([x^2 + y^2 - 1, y^2 - x^3 + x^2 + x - 1])
sage: E(A.base_ring()).points()
[(-1, 0), (0, -1), (0, 1), (1, 0)]
```

class sage.schemes.affine.affine_homset.**SchemeHomset_points_spec** (*X*, *Y*, *category=None*, *check=True*, *base=None*)

Bases: sage.schemes.generic.homset.SchemeHomset_generic

Set of rational points of an affine variety.

INPUT:

See SchemeHomset_generic.

EXAMPLES:

```
sage: from sage.schemes.affine.affine_homset import SchemeHomset_points_spec
sage: SchemeHomset_points_spec(Spec(QQ), Spec(QQ))
Set of rational points of Spectrum of Rational Field
```

CHAPTER

FOURTEEN

PROJECTIVE SCHEMES

14.1 Projective n space over a ring

EXAMPLES:

We construct projective space over various rings of various dimensions.

The simplest projective space:

```
sage: ProjectiveSpace(0)
Projective Space of dimension 0 over Integer Ring
```

A slightly bigger projective space over $\mathbf{Q}$:

```
sage: X = ProjectiveSpace(1000, QQ); X
Projective Space of dimension 1000 over Rational Field
sage: X.dimension()
1000
```

We can use "over" notation to create projective spaces over various base rings.

```
sage: X = ProjectiveSpace(5)/QQ; X
Projective Space of dimension 5 over Rational Field
sage: X/CC
Projective Space of dimension 5 over Complex Field with 53 bits of precision
```

The third argument specifies the printing names of the generators of the homogenous coordinate ring. Using the method $.objgens()$ you can obtain both the space and the generators as ready to use variables.

```
sage: P2, vars = ProjectiveSpace(10, QQ, 't').objgens()
sage: vars
(t0, t1, t2, t3, t4, t5, t6, t7, t8, t9, t10)
```

You can alternatively use the special syntax with < and >.

```
sage: P2.<x,y,z> = ProjectiveSpace(2, QQ)
sage: P2
Projective Space of dimension 2 over Rational Field
sage: P2.coordinate_ring()
Multivariate Polynomial Ring in x, y, z over Rational Field
```

The first of the three lines above is just equivalent to the two lines:

```
sage: P2 = ProjectiveSpace(2, QQ, 'xyz')
sage: x,y,z = P2.gens()
```

For example, we use x, y, z to define the intersection of two lines.

```
sage: V = P2.subscheme([x+y+z, x+y-z]); V
Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
 x + y + z,
 x + y - z
sage: V.dimension()
0
```

AUTHORS:

- Ben Hutz: (June 2012): support for rings
- Ben Hutz (9/2014): added support for cartesian products

sage.schemes.projective.projective_space.**ProjectiveSpace**(*n*, *R=None*, *names='x'*)

Return projective space of dimension n over the ring R.

EXAMPLES: The dimension and ring can be given in either order.

```
sage: ProjectiveSpace(3, QQ)
Projective Space of dimension 3 over Rational Field
sage: ProjectiveSpace(5, QQ)
Projective Space of dimension 5 over Rational Field
sage: P = ProjectiveSpace(2, QQ, names='XYZ'); P
Projective Space of dimension 2 over Rational Field
sage: P.coordinate_ring()
Multivariate Polynomial Ring in X, Y, Z over Rational Field
```

The divide operator does base extension.

```
sage: ProjectiveSpace(5)/GF(17)
Projective Space of dimension 5 over Finite Field of size 17
```

The default base ring is $\mathbf{Z}$.

```
sage: ProjectiveSpace(5)
Projective Space of dimension 5 over Integer Ring
```

There is also an projective space associated each polynomial ring.

```
sage: R = GF(7)['x,y,z']
sage: P = ProjectiveSpace(R); P
Projective Space of dimension 2 over Finite Field of size 7
sage: P.coordinate_ring()
Multivariate Polynomial Ring in x, y, z over Finite Field of size 7
sage: P.coordinate_ring() is R
True

sage: ProjectiveSpace(3, Zp(5), 'y')
Projective Space of dimension 3 over 5-adic Ring with capped relative precision 20

sage: ProjectiveSpace(2,QQ,'x,y,z')
Projective Space of dimension 2 over Rational Field

sage: PS.<x,y>=ProjectiveSpace(1,CC)
sage: PS
Projective Space of dimension 1 over Complex Field with 53 bits of precision
```

Projective spaces are not cached, i.e., there can be several with the same base ring and dimension (to facilitate gluing constructions).

class sage.schemes.projective.projective_space.**ProjectiveSpace_field**(*n*, *R=Integer Ring*, *names=None*)

Bases: sage.schemes.projective.projective_space.ProjectiveSpace_ring

EXAMPLES:

```
sage: ProjectiveSpace(3, Zp(5), 'y')
Projective Space of dimension 3 over 5-adic Ring with capped relative precision 20
```

points_of_bounded_height(*bound*, *prec=53*)

Returns an iterator of the points in self of absolute height of at most the given bound. Bound check is strict for the rational field. Requires self to be projective space over a number field. Uses the Doyle-Krumm algorithm for computing algebraic numbers up to a given height [Doyle-Krumm].

INPUT:

- `bound` - a real number
- `prec` - the precision to use to compute the elements of bounded height for number fields

OUTPUT:

- an iterator of points in self

> **Warning:** In the current implementation, the output of the [Doyle-Krumm] algorithm cannot be guaranteed to be correct due to the necessity of floating point computations. In some cases, the default 53-bit precision is considerably lower than would be required for the algorithm to generate correct output.

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: list(P.points_of_bounded_height(5))
[(0 : 1), (1 : 1), (-1 : 1), (1/2 : 1), (-1/2 : 1), (2 : 1), (-2 : 1), (1/3 : 1),
(-1/3 : 1), (3 : 1), (-3 : 1), (2/3 : 1), (-2/3 : 1), (3/2 : 1), (-3/2 : 1), (1/4 : 1),
(-1/4 : 1), (4 : 1), (-4 : 1), (3/4 : 1), (-3/4 : 1), (4/3 : 1), (-4/3 : 1), (1 : 0)]

sage: u = QQ['u'].0
sage: P.<x,y,z> = ProjectiveSpace(NumberField(u^2 - 2,'v'), 2)
sage: len(list(P.points_of_bounded_height(1.5)))
57
```

class sage.schemes.projective.projective_space.**ProjectiveSpace_finite_field**(*n*, *R=Integer Ring*, *names=None*)

Bases: sage.schemes.projective.projective_space.ProjectiveSpace_field

EXAMPLES:

```
sage: ProjectiveSpace(3, Zp(5), 'y')
Projective Space of dimension 3 over 5-adic Ring with capped relative precision 20
```

rational_points(*F=None*)

Return the list of F-rational points on the affine space self, where F is a given finite field, or the base ring of self.

EXAMPLES:

```
sage: P = ProjectiveSpace(1, GF(3))
sage: P.rational_points()
[(0 : 1), (1 : 1), (2 : 1), (1 : 0)]
sage: P.rational_points(GF(3^2, 'b'))
[(0 : 1), (b : 1), (b + 1 : 1), (2*b + 1 : 1), (2 : 1), (2*b : 1), (2*b + 2 : 1), (b + 2 : 1
```

rational_points_dictionary()

Return dictionary of points.

OUTPUT:

- dictionary

EXAMPLES:

```
sage: P1=ProjectiveSpace(GF(7),1,'x')
sage: P1.rational_points_dictionary()
{(0 : 1): 0,
 (1 : 0): 7,
 (1 : 1): 1,
 (2 : 1): 2,
 (3 : 1): 3,
 (4 : 1): 4,
 (5 : 1): 5,
 (6 : 1): 6}
```

class sage.schemes.projective.projective_space.**ProjectiveSpace_rational_field**(*n*, *R=Integer Ring*, *names=None*)

Bases: sage.schemes.projective.projective_space.ProjectiveSpace_field

EXAMPLES:

```
sage: ProjectiveSpace(3, Zp(5), 'y')
Projective Space of dimension 3 over 5-adic Ring with capped relative precision 20
```

rational_points(*bound=0*)

Returns the projective points $(x_0 : \cdots : x_n)$ over $\mathbf{Q}$ with $|x_i| \leq$ bound.

INPUT:

- bound - integer

EXAMPLES:

```
sage: PP = ProjectiveSpace(0,QQ)
sage: PP.rational_points(1)
[(1)]
sage: PP = ProjectiveSpace(1,QQ)
sage: PP.rational_points(2)
[(-2 : 1), (-1 : 1), (0 : 1), (1 : 1), (2 : 1), (-1/2 : 1), (1/2 : 1), (1 : 0)]
sage: PP = ProjectiveSpace(2,QQ)
sage: PP.rational_points(2)
[(-2 : -2 : 1), (-1 : -2 : 1), (0 : -2 : 1), (1 : -2 : 1), (2 : -2 : 1),
(-2 : -1 : 1), (-1 : -1 : 1), (0 : -1 : 1), (1 : -1 : 1), (2 : -1 : 1),
(-2 : 0 : 1), (-1 : 0 : 1), (0 : 0 : 1), (1 : 0 : 1), (2 : 0 : 1), (-2 :
1 : 1), (-1 : 1 : 1), (0 : 1 : 1), (1 : 1 : 1), (2 : 1 : 1), (-2 : 2 :
1), (-1 : 2 : 1), (0 : 2 : 1), (1 : 2 : 1), (2 : 2 : 1), (-1/2 : -1 :
1), (1/2 : -1 : 1), (-1 : -1/2 : 1), (-1/2 : -1/2 : 1), (0 : -1/2 : 1),
(1/2 : -1/2 : 1), (1 : -1/2 : 1), (-1/2 : 0 : 1), (1/2 : 0 : 1), (-1 :
```

```
1/2 : 1), (-1/2 : 1/2 : 1), (0 : 1/2 : 1), (1/2 : 1/2 : 1), (1 : 1/2 :
1), (-1/2 : 1 : 1), (1/2 : 1 : 1), (-2 : 1 : 0), (-1 : 1 : 0), (0 : 1 :
0), (1 : 1 : 0), (2 : 1 : 0), (-1/2 : 1 : 0), (1/2 : 1 : 0), (1 : 0 :
0)]
```

Note: The very simple algorithm works as follows: every point $(x_0 : \cdots : x_n)$ in projective space has a unique largest index i for which x_i is not zero. The algorithm then iterates downward on this index. We normalize by choosing x_i positive. Then, the points $x_0, \ldots, x_{i-1}$ are the points of affine i-space that are relatively prime to x_i. We access these by using the Tuples method.

AUTHORS:

- Benjamin Antieau (2008-01-12)

class `sage.schemes.projective.projective_space.`**`ProjectiveSpace_ring`**(*n*, *R=Integer Ring*, *names=None*)

Bases: `sage.schemes.generic.ambient_space.AmbientSpace`

Projective space of dimension n over the ring R.

EXAMPLES:

```
sage: X.<x,y,z,w> = ProjectiveSpace(3, QQ)
sage: X.base_scheme()
Spectrum of Rational Field
sage: X.base_ring()
Rational Field
sage: X.structure_morphism()
Scheme morphism:
  From: Projective Space of dimension 3 over Rational Field
  To:   Spectrum of Rational Field
  Defn: Structure map
sage: X.coordinate_ring()
Multivariate Polynomial Ring in x, y, z, w over Rational Field
```

Loading and saving:

```
sage: loads(X.dumps()) == X
True
```

`Lattes_map`(*E*, *m*)

Given an elliptic curve E and an integer m return the Lattes map associated to multiplication by m. In other words, the rational map on the quotient $E/\{\pm 1\} \cong \mathbb{P}^1$ associated to $[m] : E \to E$.

INPUT:

- `E` – an elliptic curve
- `m` – an integer

OUTPUT: an endomorphism of `self`.

Examples:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: E = EllipticCurve(QQ,[-1, 0])
sage: P.Lattes_map(E,2)
Scheme endomorphism of Projective Space of dimension 1 over Rational Field
```

```
  Defn: Defined on coordinates by sending (x : y) to
        (x^4 + 2*x^2*y^2 + y^4 : 4*x^3*y - 4*x*y^3)
```

affine_patch (*i*, *AA=None*)

Return the i^{th} affine patch of this projective space. This is an ambient affine space $\mathbb{A}^n_R$, where R is the base ring of self, whose "projective embedding" map is 1 in the i^{th} factor.

INPUT:

- `i` – integer between 0 and dimension of self, inclusive.
- **`AA` – (default: None) ambient affine space, this is constructed** if it is not given.

OUTPUT:

- An ambient affine space with fixed projective_embedding map.

EXAMPLES:

```
sage: PP = ProjectiveSpace(5) / QQ
sage: AA = PP.affine_patch(2)
sage: AA
Affine Space of dimension 5 over Rational Field
sage: AA.projective_embedding()
Scheme morphism:
  From: Affine Space of dimension 5 over Rational Field
  To:   Projective Space of dimension 5 over Rational Field
  Defn: Defined on coordinates by sending (x0, x1, x2, x3, x4) to
        (x0 : x1 : 1 : x2 : x3 : x4)
sage: AA.projective_embedding(0)
Scheme morphism:
  From: Affine Space of dimension 5 over Rational Field
  To:   Projective Space of dimension 5 over Rational Field
  Defn: Defined on coordinates by sending (x0, x1, x2, x3, x4) to
        (1 : x0 : x1 : x2 : x3 : x4)

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: P.affine_patch(0).projective_embedding(0).codomain() == P
True
```

cartesian_product (*other*)

Return the cartesian product of the projective spaces `self` and `other`.

INPUT:

- `other` - A projective space with the same base ring as `self`

OUTPUT:

- A cartesian product of projective spaces

EXAMPLES:

```
sage: P1 = ProjectiveSpace(QQ,1,'x')
sage: P2 = ProjectiveSpace(QQ,2,'y')
sage: PP = P1.cartesian_product(P2); PP
Product of projective spaces P^1 x P^2 over Rational Field
sage: PP.gens()
(x0, x1, y0, y1, y2)
```

change_ring (*R*)

Return a projective space over ring R and otherwise the same as self.

INPUT:

•R – commutative ring

OUTPUT:

•projective space over R

Note: There is no need to have any relation between R and the base ring of self, if you want to have such a relation, use `self.base_extend(R)` instead.

EXAMPLES:

```
sage: P.<x, y, z> = ProjectiveSpace(2, ZZ)
sage: PQ = P.change_ring(QQ); PQ
Projective Space of dimension 2 over Rational Field
sage: PQ.change_ring(GF(5))
Projective Space of dimension 2 over Finite Field of size 5
```

coordinate_ring()

Return the coordinate ring of this scheme.

EXAMPLES:

```
sage: ProjectiveSpace(3, GF(19^2,'alpha'), 'abcd').coordinate_ring()
Multivariate Polynomial Ring in a, b, c, d over Finite Field in alpha of size 19^2

sage: ProjectiveSpace(3).coordinate_ring()
Multivariate Polynomial Ring in x0, x1, x2, x3 over Integer Ring

sage: ProjectiveSpace(2, QQ, ['alpha', 'beta', 'gamma']).coordinate_ring()
Multivariate Polynomial Ring in alpha, beta, gamma over Rational Field
```

is_projective()

Return that this ambient space is projective n-space.

EXAMPLES:

```
sage: ProjectiveSpace(3,QQ).is_projective()
True
```

ngens()

Return the number of generators of self, i.e. the number of variables in the coordinate ring of self.

EXAMPLES:

```
sage: ProjectiveSpace(3, QQ).ngens()
4
sage: ProjectiveSpace(7, ZZ).ngens()
8
```

subscheme(*X*)

Return the closed subscheme defined by X.

INPUT:

•X - a list or tuple of equations

EXAMPLES:

```
sage: A.<x,y,z> = ProjectiveSpace(2, QQ)
sage: X = A.subscheme([x*z^2, y^2*z, x*y^2]); X
Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
```

```
  x*z^2,
  y^2*z,
  x*y^2
sage: X.defining_polynomials ()
(x*z^2, y^2*z, x*y^2)
sage: I = X.defining_ideal(); I
Ideal (x*z^2, y^2*z, x*y^2) of Multivariate Polynomial Ring in x, y, z over Rational Field
sage: I.groebner_basis()
[x*y^2, y^2*z,  x*z^2]
sage: X.dimension()
0
sage: X.base_ring()
Rational Field
sage: X.base_scheme()
Spectrum of Rational Field
sage: X.structure_morphism()
Scheme morphism:
  From: Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
  x*z^2,
  y^2*z,
  x*y^2
  To:   Spectrum of Rational Field
  Defn: Structure map

sage: TestSuite(X).run(skip=["_test_an_element", "_test_elements", "_test_elements_eq", "_te
```

`sage.schemes.projective.projective_space.`**`is_ProjectiveSpace`**(*x*)

Return True if x is a projective space, i.e., an ambient space $\mathbb{P}^n_R$, where R is a ring and $n \geq 0$ is an integer.

EXAMPLES:

```
sage: from sage.schemes.projective.projective_space import is_ProjectiveSpace
sage: is_ProjectiveSpace(ProjectiveSpace(5, names='x'))
True
sage: is_ProjectiveSpace(ProjectiveSpace(5, GF(9,'alpha'), names='x'))
True
sage: is_ProjectiveSpace(Spec(ZZ))
False
```

14.2 Points on projective varieties

Scheme morphism for points on projective varieties

AUTHORS:

- David Kohel, William Stein
- William Stein (2006-02-11): fixed bug where P(0,0,0) was allowed as a projective point.
- Volker Braun (2011-08-08): Renamed classes, more documentation, misc cleanups.
- Ben Hutz (June 2012) added support for projective ring; (March 2013) iteration functionality and new directory structure for affine/projective, height functionality

class `sage.schemes.projective.projective_point.`**`SchemeMorphism_point_abelian_variety_field`**(*X, v, chec*

Bases: `sage.structure.element.AdditiveGroupElement`, `sage.schemes.projective.projective_poi`

A rational point of an abelian variety over a field.

EXAMPLES:

```
sage: E = EllipticCurve([0,0,1,-1,0])
sage: origin = E(0)
sage: origin.domain()
Spectrum of Rational Field
sage: origin.codomain()
Elliptic Curve defined by y^2 + y = x^3 - x over Rational Field
```

class sage.schemes.projective.projective_point.**SchemeMorphism_point_projective_field**(*X*, *v*, *check=True*)

Bases: sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring

A rational point of projective space over a field.

INPUT:

- X – a homset of a subscheme of an ambient projective space over a field K
- v – a list or tuple of coordinates in K
- check – boolean (optional, default:True). Whether to check the input for consistency.

EXAMPLES:

```
sage: P = ProjectiveSpace(3, RR)
sage: P(2,3,4,5)
(0.400000000000000 : 0.600000000000000 : 0.800000000000000 : 1.00000000000000)
```

clear_denominators()

scales by the least common multiple of the denominators.

OUTPUT: None.

EXAMPLES:

```
sage: R.<t> = PolynomialRing(QQ)
sage: P.<x,y,z> = ProjectiveSpace(FractionField(R), 2)
sage: Q = P([t, 3/t^2, 1])
sage: Q.clear_denominators(); Q
(t^3 : 3 : t^2)

sage: R.<x> = PolynomialRing(QQ)
sage: K.<w> = NumberField(x^2 - 3)
sage: P.<x,y,z> = ProjectiveSpace(K, 2)
sage: Q = P([1/w, 3, 0])
sage: Q.clear_denominators(); Q
(w : 9 : 0)

sage: P.<x,y,z> = ProjectiveSpace(QQ, 2)
sage: X = P.subscheme(x^2 - y^2);
sage: Q = X([1/2, 1/2, 1]);
sage: Q.clear_denominators(); Q
(1 : 1 : 2)

sage: PS.<x,y> = ProjectiveSpace(QQ, 1)
sage: Q = PS.point([1, 2/3], False); Q
(1 : 2/3)
sage: Q.clear_denominators(); Q
(3 : 2)
```

normalize_coordinates()

Normalizes `self` so that the last non-zero coordinate is 1.

OUTPUT: None.

EXAMPLES:

```
sage: P.<x,y,z> = ProjectiveSpace(GF(5),2)
sage: Q = P.point([GF(5)(1), GF(5)(3), GF(5)(0)], False); Q
(1 : 3 : 0)
sage: Q.normalize_coordinates(); Q
(2 : 1 : 0)

sage: P.<x,y,z> = ProjectiveSpace(QQ, 2)
sage: X  =P.subscheme(x^2-y^2);
sage: Q = X.point([23, 23, 46], False); Q
(23 : 23 : 46)
sage: Q.normalize_coordinates(); Q
(1/2 : 1/2 : 1)
```

class `sage.schemes.projective.projective_point.`**SchemeMorphism_point_projective_finite_field**(*X*
v
c

Bases: `sage.schemes.projective.projective_point.SchemeMorphism_point_projective_field`

The Python constructor.

See `SchemeMorphism_point_projective_ring` for details.

This function still normalized points so that the rightmost non-zero coordinate is 1. The is to maintain current functionality with current implementations of curves in projectives space (plane, connic, elliptic, etc). The class:$SchemeMorphism_point_projective_ring$ is for general use.

EXAMPLES:

```
sage: P = ProjectiveSpace(2, QQ)
sage: P(2, 3/5, 4)
(1/2 : 3/20 : 1)

sage: P = ProjectiveSpace(3, QQ)
sage: P(0,0,0,0)
Traceback (most recent call last):
...
ValueError: [0, 0, 0, 0] does not define a valid point since all entries are 0

sage: P.<x, y, z> = ProjectiveSpace(2, QQ)
sage: X = P.subscheme([x^2-y*z])
sage: X([2,2,2])
(1 : 1 : 1)

sage: P = ProjectiveSpace(1, GF(7))
sage: Q=P([2, 1])
sage: Q[0].parent()
Finite Field of size 7
```

orbit_structure(f)

Every point is preperiodic over a finite field. This function returns the pair $[m, n]$ where m is the preperiod and n is the period of the point `self` by `f`.

INPUT:

- `f` – a `SceneMorphism_polynomial` with `self` in `f.domain()`

OUTPUT:

- a list $[m, n]$ of integers

EXAMPLES:

```
sage: P.<x,y,z> = ProjectiveSpace(GF(5),2)
sage: H = Hom(P,P)
sage: f = H([x^2 + y^2,y^2,z^2 + y*z])
sage: P(1,0,1).orbit_structure(f)
[0, 1]

sage: P.<x,y,z> = ProjectiveSpace(GF(17),2)
sage: X = P.subscheme(x^2-y^2)
sage: H = Hom(X,X)
sage: f = H([x^2,y^2,z^2])
sage: X(1,1,2).orbit_structure(f)
[3, 1]

sage: R.<t> = GF(13^3)
sage: P.<x,y> = ProjectiveSpace(R,1)
sage: H = Hom(P,P)
sage: f = H([x^2 - y^2,y^2])
sage: P(t,4).orbit_structure(f)
[11, 6]
```

class `sage.schemes.projective.projective_point.`**`SchemeMorphism_point_projective_ring`**(*X*, *v*, *check=True*)

Bases: `sage.schemes.generic.morphism.SchemeMorphism_point`

A rational point of projective space over a ring.

INPUT:

- `X` – a homset of a subscheme of an ambient projective space over a field K
- `v` – a list or tuple of coordinates in K
- `check` – boolean (optional, default:`True`). Whether to check the input for consistency.

EXAMPLES:

```
sage: P = ProjectiveSpace(2, ZZ)
sage: P(2,3,4)
(2 : 3 : 4)
```

`canonical_height`(*F*, ***kwds*)

Evaluates the (absolute) canonical height of `self` with respect to `F`. Must be over number field or order of a number field or `QQbar`. Specify either the number of terms of the series to evaluate or the error bound required.

ALGORITHM:

The sum of the Green's function at the archimedean places and the places of bad reduction.

INPUT:

- `F` - a projective morphism

kwds:

- `badprimes` - a list of primes of bad reduction (optional)

- N - positive integer. number of terms of the series to use in the local green functions (optional - default:10)
- prec - positive integer, float point or p-adic precision, default:100
- error_bound - a positive real number (optional)

OUTPUT: a real number

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(ZZ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,2*x*y]);
sage: Q = P(2,1)
sage: f.canonical_height(f(Q))
2.1965476757927038111992627081
sage: f.canonical_height(Q)
1.0979353871245941198040174712
```

Notice that preperiodic points may not be exactly 0.

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2-29/16*y^2,y^2]);
sage: Q = P(5,4)
sage: f.canonical_height(Q, N=30)
1.4989058602918874235833076226e-9
```

```
sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: X = P.subscheme(x^2-y^2);
sage: H = Hom(X,X)
sage: f = H([x^2,y^2,30*z^2]);
sage: Q = X([4,4,1])
sage: f.canonical_height(Q, badprimes=[2,3,5], prec=200)
2.7054056208276961889784303469356774912979228770208655455481
```

dehomogenize(*n*)

Dehomogenizes at the nth coordinate

INPUT:

- n – non-negative integer

OUTPUT:

- SchemeMorphism_point_affine

EXAMPLES:

```
sage: P.<x,y,z>=ProjectiveSpace(QQ,2)
sage: X=P.subscheme(x^2-y^2);
sage: Q=X(23,23,46)
sage: Q.dehomogenize(2)
(1/2, 1/2)
```

```
sage: R.<t>=PolynomialRing(QQ)
sage: S=R.quo(R.ideal(t^3))
sage: P.<x,y,z>=ProjectiveSpace(S,2)
sage: Q=P(t,1,1)
sage: Q.dehomogenize(1)
(tbar, 1)
```

```
sage: P.<x,y,z>=ProjectiveSpace(GF(5),2)
sage: Q=P(1,3,1)
sage: Q.dehomogenize(0)
(3, 1)
```

```
sage: P.<x,y,z>=ProjectiveSpace(GF(5),2)
sage: Q=P(1,3,0)
sage: Q.dehomogenize(2)
Traceback (most recent call last):
...
ValueError: Can't dehomogenize at 0 coordinate.
```

global_height (*prec=None*)

Returns the absolute logarithmic height of the point `self`.

INPUT:

- `prec` – desired floating point precision (default: default RealField precision).

OUTPUT:

- a real number

EXAMPLES:

```
sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: Q = P.point([4,4,1/30])
sage: Q.global_height()
4.78749174278205
```

```
sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: Q = P([4,1,30])
sage: Q.global_height()
3.40119738166216
```

```
sage: R.<x> = PolynomialRing(QQ)
sage: k.<w> = NumberField(x^2+5)
sage: A = ProjectiveSpace(k,2,'z')
sage: A([3,5*w+1,1]).global_height(prec=100)
2.4181409534757389986565376694
```

```
sage: P.<x,y,z> = ProjectiveSpace(QQbar,2)
sage: Q = P([QQbar(sqrt(3)),QQbar(sqrt(-2)),1])
sage: Q.global_height()
0.549306144334055
```

green_function (*G, v, **kwds*)

Evaluates the local Green's function with respect to the morphism `G` at the place `v` for `self` with `N` terms of the series or to within a given error bound. Must be over a number field or order of a number field. Note that this is the absolute local Green's function so is scaled by the degree of the base field.

Use `v=0` for the archimedean place over $\mathbf{Q}$ or field embedding. Non-archimedean places are prime ideals for number fields or primes over $\mathbf{Q}$.

ALGORITHM:

See Exercise 5.29 and Figure 5.6 of `The Arithmetic of Dynamics Systems`, Joseph H. Silverman, Springer, GTM 241, 2007.

INPUT:

- `G` - a projective morphism whose local Green's function we are computing

- `v` - non-negative integer. a place, use v=0 for the archimedean place

kwds:

- `N` - positive integer. number of terms of the series to use, default: 10
- `prec` - positive integer, float point or p-adic precision, default: 100
- `error_bound` - a positive real number

OUTPUT:

- a real number

EXAMPLES:

```
sage: P.<x,y>=ProjectiveSpace(QQ,1)
sage: H=Hom(P,P)
sage: f=H([x^2+y^2,x*y]);
sage: Q=P(5,1)
sage: f.green_function(Q,0,N=30)
1.6460930159932946233759277576

sage: P.<x,y>=ProjectiveSpace(QQ,1)
sage: H=Hom(P,P)
sage: f=H([x^2+y^2,x*y]);
sage: Q=P(5,1)
sage: Q.green_function(f,0,N=200,prec=200)
1.6460930160038721802875250367738355497198064992657997569827

sage: K.<w> = QuadraticField(3)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = Hom(P,P)
sage: f = H([17*x^2+1/7*y^2,17*w*x*y])
sage: f.green_function(P.point([w,2],False), K.places()[1])
1.7236334013785676107373093775
sage: print f.green_function(P([2,1]), K.ideal(7), N=7)
0.48647753726382832627633818586
sage: print f.green_function(P([w,1]), K.ideal(17), error_bound=0.001)
-0.70691993106090157426711999977
```

Todo

Implement general p-adic extensions so that the flip trick can be used for number fields.

is_preperiodic (*f*, *err=0.1*, *return_period=False*)

Determine if the point `self` is preperiodic with respect to the map `f`, i.e., if `self` has a finite forward orbit by `f`. This is only implemented for projective space (not subschemes). There are two optional keyword arguments: `error_bound` sets the error_bound used in the canonical height computation and `return_period` a boolean which controls if the period is returned if the point is preperiodic. If `return_period` is `True` and the `self` is not preperiodic, then $(0, 0)$ is returned for the period.

ALGORITHM:

We know that a point is preperiodic if and only if it has canonical height zero. However, we can only compute the canonical height up to numerical precision. This function first computes the canonical height of the point to the given error bound. If it is larger than that error bound, then it must not be preperiodic. If it is less than the error bound, then we expect preperiodic. In this case we begin computing the orbit stopping if either we determine the orbit is finite, or the height of the point is large enough that it must be wandering. We can determine the height cutoff by computing the height difference constant, i.e., the bound between the height and the canonical height of a point (which depends only on the map and not the point itself). If the height of the point is larger than the difference bound, then the canonical height cannot be zero so the point cannot be preperiodic.

INPUT:

•`f` – an endomorphism of `self.codomain()`

kwds:

•`error_bound` – a positive real number (optional - default: 0.1)

•`return_period` – boolean (optional - default: `False`)

OUTPUT:

•boolean - `True` if preperiodic

•**if return_period is `True`, then `(0,0)` if wandering, and `(m,n)`** if preperiod `m` and period `n`

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^3-3*x*y^2, y^3])
sage: Q = P(-1,1)
sage: Q.is_preperiodic(f)
True
```

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2-29/16*y^2, y^2])
sage: Q = P(1,4)
sage: Q.is_preperiodic(f, return_period=True)
(1, 3)
sage: Q = P(1,1)
sage: Q.is_preperiodic(f, return_period=True)
(0, 0)
```

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2+1)
sage: P.<x,y> = ProjectiveSpace(K, 1)
sage: H = End(P)
sage: f = H([x^5 + 5/4*x*y^4, y^5])
sage: Q = P([-1/2*a+1/2, 1])
sage: Q.is_preperiodic(f)
True
sage: Q = P([a, 1])
sage: Q.is_preperiodic(f)
False
```

```
sage: P.<x,y,z> = ProjectiveSpace(QQ, 2)
sage: H = Hom(P,P)
sage: f = H([-38/45*x^2 + (2*y - 7/45*z)*x + (-1/2*y^2 - 1/2*y*z + z^2),\
    -67/90*x^2 + (2*y + z*157/90)*x - y*z, z^2])
sage: Q = P([1,3,1])
sage: Q.is_preperiodic(f, return_period = True)
(0, 9)
```

```
sage: P.<x,y,z,w> = ProjectiveSpace(QQ,3)
sage: H = Hom(P,P)
sage: f = H([(-y - w)*x + (-13/30*y^2 + 13/30*w*y + w^2),-1/2*x^2 + (-y + 3/2*w)*x\
    + (-1/3*y^2 + 4/3*w*y),-3/2*z^2 + 5/2*z*w + w^2,w^2])
sage: Q = P([3,0,4/3,1])
```

```
sage: Q.is_preperiodic(f, return_period = True)
(2, 24)

sage: set_verbose(-1)
sage: P.<x,y,z> = ProjectiveSpace(QQbar,2)
sage: H = End(P)
sage: f = H([x^2,QQbar(sqrt(-1))*y^2,z^2])
sage: Q = P([1,1,1])
sage: Q.is_preperiodic(f)
True

sage: set_verbose(-1)
sage: P.<x,y,z> = ProjectiveSpace(QQbar,2)
sage: H = End(P)
sage: f = H([x^2,y^2,z^2])
sage: Q = P([QQbar(sqrt(-1)),1,1])
sage: Q.is_preperiodic(f)
True
```

local_height (*v*, *prec=None*)

Returns the maximum of the local height of the coordinates of `self`.

INPUT:

- `v` – a prime or prime ideal of the base ring
- `prec` – desired floating point precision (default: default RealField precision).

OUTPUT:

- a real number

EXAMPLES:

```
sage: P.<x,y,z>=ProjectiveSpace(QQ,2)
sage: Q=P.point([4,4,1/150],False)
sage: Q.local_height(5)
3.21887582486820

sage: P.<x,y,z>=ProjectiveSpace(QQ,2)
sage: Q=P([4,1,30])
sage: Q.local_height(2)
0.693147180559945
```

local_height_arch (*i*, *prec=None*)

Returns the maximum of the local heights at the `i`-th infinite place of `self`.

INPUT:

- `i` – an integer
- `prec` – desired floating point precision (default: default RealField precision).

OUTPUT:

- a real number

EXAMPLES:

```
sage: P.<x,y,z>=ProjectiveSpace(QQ,2)
sage: Q = P.point([4,4,1/150], False)
sage: Q.local_height_arch(0)
1.38629436111989
```

```
sage: P.<x,y,z>=ProjectiveSpace(QuadraticField(5, 'w'),2)
sage: Q = P.point([4,1,30], False)
sage: Q.local_height_arch(1)
3.4011973816621553754132366916075
```

multiplier(*f*, *n*, *check=True*)

Returns the multiplier of the projective point `self` of period n by the function f. f must be an endomorphism of projective space

INPUT:

- `f` - a endomorphism of `self.codomain()`
- `n` - a positive integer, the period of `self`
- `check` – check if `P` is periodic of period `n`, Default:True

OUTPUT:

- a square matrix of size `self.codomain().dimension_relative()` in the `base_ring` of `self`

EXAMPLES:

```
sage: P.<x,y,z,w>=ProjectiveSpace(QQ,3)
sage: H=Hom(P,P)
sage: f=H([x^2,y^2,4*w^2,4*z^2]);
sage: Q=P.point([4,4,1,1],False);
sage: Q.multiplier(f,1)
[ 2  0 -8]
[ 0  2 -8]
[ 0  0 -2]
```

normalize_coordinates()

Removes the gcd from the coordinates of `self` (including -1).

Warning: The gcd will depend on the base ring.

OUTPUT: None.

EXAMPLES:

```
sage: P = ProjectiveSpace(ZZ,2,'x')
sage: p = P([-5,-15,-20])
sage: p.normalize_coordinates(); p
(1 : 3 : 4)

sage: P = ProjectiveSpace(Zp(7),2,'x')
sage: p = P([-5,-15,-2])
sage: p.normalize_coordinates(); p
(5 + O(7^20) : 1 + 2*7 + O(7^20) : 2 + O(7^20))

sage: R.<t> = PolynomialRing(QQ)
sage: P = ProjectiveSpace(R,2,'x')
sage: p = P([3/5*t^3,6*t, t])
sage: p.normalize_coordinates(); p
(3/5*t^2 : 6 : 1)

sage: P.<x,y> = ProjectiveSpace(Zmod(20),1)
sage: Q = P(3,6)
sage: Q.normalize_coordinates()
```

```
sage: Q
(1 : 2)
```

Since the base ring is a polynomial ring over a field, only the gcd c is removed.

```
sage: R.<c> = PolynomialRing(QQ)
sage: P = ProjectiveSpace(R,1)
sage: Q = P(2*c,4*c)
sage: Q.normalize_coordinates();Q
(2 : 4)
```

A polynomial ring over a ring gives the more intuitive result.

```
sage: R.<c> = PolynomialRing(ZZ)
sage: P = ProjectiveSpace(R,1)
sage: Q = P(2*c,4*c)
sage: Q.normalize_coordinates();Q
(1 : 2)
```

```
sage: R.<t> = PolynomialRing(QQ,1)
sage: S = R.quotient_ring(R.ideal(t^3))
sage: P.<x,y> = ProjectiveSpace(S,1)
sage: Q = P(t,t^2)
sage: Q.normalize_coordinates()
sage: Q
(1 : tbar)
```

nth_iterate (*f*, *n*, *normalize=False*)

For a map `self` and a point P in `self.domain()` this function returns the nth iterate of P by `self`. If `normalize==True`, then the coordinates are automatically normalized.

INPUT:

- `f` – a SchmemMorphism_polynomial with `self` in `f.domain()`
- `n` – a positive integer.
- `normalize` – Boolean (optional Default: `False`)

OUTPUT:

- A point in `self.codomain()`

EXAMPLES:

```
sage: P.<x,y>=ProjectiveSpace(ZZ,1)
sage: H=Hom(P,P)
sage: f=H([x^2+y^2,2*y^2])
sage: P(1,1).nth_iterate(f,4)
(32768 : 32768)
```

```
sage: P.<x,y>=ProjectiveSpace(ZZ,1)
sage: H=Hom(P,P)
sage: f=H([x^2+y^2,2*y^2])
sage: P(1,1).nth_iterate(f,4,1)
(1 : 1)
```

```
sage: R.<t>=PolynomialRing(QQ)
sage: P.<x,y,z>=ProjectiveSpace(R,2)
sage: H=Hom(P,P)
sage: f=H([x^2+t*y^2,(2-t)*y^2,z^2])
sage: P(2+t,7,t).nth_iterate(f,2)
```

```
(t^4 + 2507*t^3 - 6787*t^2 + 10028*t + 16 : -2401*t^3 + 14406*t^2 -
28812*t + 19208 : t^4)
```

```
sage: P.<x,y,z>=ProjectiveSpace(ZZ,2)
sage: X=P.subscheme(x^2-y^2)
sage: H=Hom(X,X)
sage: f=H([x^2,y^2,z^2])
sage: X(2,2,3).nth_iterate(f,3)
(256 : 256 : 6561)
```

Todo

Is there a more efficient way to do this?

orbit (*f*, *N*, ***kwds*)

Returns the orbit of P by `self`. If n is an integer it returns $[P, self(P), \ldots, self^n(P)]$. If n is a list or tuple $n = [m,k]$ it returns $[self^m(P), \ldots, self^k(P)]$. Automatically normalize the points if `normalize=True`. Perform the checks on point initialization if `check=True`

INPUT:

- `f` – a `SchemeMorphism_polynomial` with `self` in `f.domain()`
- `N` – a non-negative integer or list or tuple of two non-negative integers

kwds:

- `check` – boolean (optional - default: `True`)
- `normalize` – boolean (optional - default: `False`)

OUTPUT:

- a list of points in `self.codomain()`

EXAMPLES:

```
sage: P.<x,y,z>=ProjectiveSpace(ZZ,2)
sage: H=Hom(P,P)
sage: f=H([x^2+y^2,y^2-z^2,2*z^2])
sage: P(1,2,1).orbit(f,3)
[(1 : 2 : 1), (5 : 3 : 2), (34 : 5 : 8), (1181 : -39 : 128)]
```

```
sage: P.<x,y,z>=ProjectiveSpace(ZZ,2)
sage: H=Hom(P,P)
sage: f=H([x^2+y^2,y^2-z^2,2*z^2])
sage: P(1,2,1).orbit(f,[2,4])
[(34 : 5 : 8), (1181 : -39 : 128), (1396282 : -14863 : 32768)]
```

```
sage: P.<x,y,z>=ProjectiveSpace(ZZ,2)
sage: X=P.subscheme(x^2-y^2)
sage: H=Hom(X,X)
sage: f=H([x^2,y^2,x*z])
sage: X(2,2,3).orbit(f,3,normalize=True)
[(2 : 2 : 3), (2 : 2 : 3), (2 : 2 : 3), (2 : 2 : 3)]
```

```
sage: P.<x,y>=ProjectiveSpace(QQ,1)
sage: H=Hom(P,P)
sage: f=H([x^2+y^2,y^2])
sage: P.point([1,2],False).orbit(f,4,check = False)
[(1 : 2), (5 : 4), (41 : 16), (1937 : 256), (3817505 : 65536)]
```

scale_by(*t*)

Scale the coordinates of the point `self` by t. A `TypeError` occurs if the point is not in the base_ring of the codomain after scaling.

INPUT:

- `t` – a ring element

OUTPUT: None.

EXAMPLES:

```
sage: R.<t> = PolynomialRing(QQ)
sage: P = ProjectiveSpace(R, 2, 'x')
sage: p = P([3/5*t^3, 6*t, t])
sage: p.scale_by(1/t); p
(3/5*t^2 : 6 : 1)

sage: R.<t> = PolynomialRing(QQ)
sage: S = R.quo(R.ideal(t^3))
sage: P.<x,y,z> = ProjectiveSpace(S, 2)
sage: Q = P(t,1,1)
sage: Q.scale_by(t);Q
(tbar^2 : tbar : tbar)

sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: Q = P(2,2,2)
sage: Q.scale_by(1/2);Q
(1 : 1 : 1)
```

14.3 Morphisms on projective varieties

A morphism of schemes determined by rational functions that define what the morphism does on points in the ambient projective space.

AUTHORS:

- David Kohel, William Stein
- William Stein (2006-02-11): fixed bug where P(0,0,0) was allowed as a projective point.
- Volker Braun (2011-08-08): Renamed classes, more documentation, misc cleanups.
- Ben Hutz (2013-03) iteration functionality and new directory structure for affine/projective, height functionality
- Brian Stout, Ben Hutz (Nov 2013) - added minimal model functionality
- Dillon Rose (2014-01): Speed enhancements

class `sage.schemes.projective.projective_morphism.`**SchemeMorphism_polynomial_projective_space**(

Bases: `sage.schemes.generic.morphism.SchemeMorphism_polynomial`

A morphism of schemes determined by rational functions that define what the morphism does on points in the ambient projective space.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: P1 = ProjectiveSpace(R)
sage: H = P1.Hom(P1)
sage: H([y,2*x])
Scheme endomorphism of Projective Space of dimension 1 over Rational Field
  Defn: Defined on coordinates by sending (x : y) to
        (y : 2*x)
```

An example of a morphism between projective plane curves (see trac ticket #10297):

```
sage: P2.<x,y,z> = ProjectiveSpace(QQ,2)
sage: f = x^3+y^3+60*z^3
sage: g = y^2*z-( x^3 - 6400*z^3/3)
sage: C = Curve(f)
sage: E = Curve(g)
sage: xbar,ybar,zbar = C.coordinate_ring().gens()
sage: H = C.Hom(E)
sage: H([zbar,xbar-ybar,-(xbar+ybar)/80])
Scheme morphism:
  From: Projective Curve over Rational Field defined by x^3 + y^3 + 60*z^3
  To:   Projective Curve over Rational Field defined by -x^3 + y^2*z + 6400/3*z^3
  Defn: Defined on coordinates by sending (x : y : z) to
        (z : x - y : -1/80*x - 1/80*y)
```

A more complicated example:

```
sage: P2.<x,y,z> = ProjectiveSpace(2,QQ)
sage: P1 = P2.subscheme(x-y)
sage: H12 = P1.Hom(P2)
sage: H12([x^2,x*z, z^2])
Scheme morphism:
  From: Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
  x - y
  To:   Projective Space of dimension 2 over Rational Field
  Defn: Defined on coordinates by sending (x : y : z) to
      (y^2 : y*z : z^2)
```

We illustrate some error checking:

```
sage: R.<x,y> = QQ[]
sage: P1 = ProjectiveSpace(R)
sage: H = P1.Hom(P1)
sage: f = H([x-y, x*y])
Traceback (most recent call last):
...
ValueError: polys (=[x - y, x*y]) must be of the same degree

sage: H([x-1, x*y+x])
Traceback (most recent call last):
...
ValueError: polys (=[x - 1, x*y + x]) must be homogeneous

sage: H([exp(x),exp(y)])
Traceback (most recent call last):
...
TypeError: polys (=[e^x, e^y]) must be elements of
Multivariate Polynomial Ring in x, y over Rational Field
```

automorphism_group(***kwds*)

Given a homogenous rational function, this calculates the subsgroup of $PGL2$ that is the automorphism group of `self`.

INPUT:

keywords:

- `starting_prime` – The first prime to use for CRT. default: 5.(optional)
- **`algorithm`– Choose `CRT`-Chinese Remainder Theorem- or `fixed_points` algorithm.** default: depends on `self`. (optional)
- **`return_functions`– Boolean - True returns elements as linear fractional transformations.** False returns elements as $PGL2$ matrices. default: False. (optional)
- **`iso_type` – Boolean - True returns the isomorphism type of the automorphism group.** default: False (optional)

OUTPUT:

- list - elements of automorphism group.

AUTHORS:

- Original algorithm written by Xander Faber, Michelle Manes, Bianca Viray
- Modified by Joao Alberto de Faria, Ben Hutz, Bianca Thompson

REFERENCES:

EXAMPLES:

```
sage: R.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(R)
sage: f = H([x^2-y^2,x*y])
sage: f.automorphism_group(return_functions=True)
[x, -x]

sage: R.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(R)
sage: f = H([x^2 + 5*x*y + 5*y^2,5*x^2 + 5*x*y + y^2])
sage: f.automorphism_group()
[
[1 0]  [0 2]
[0 1], [2 0]
]

sage: R.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(R)
sage: f=H([x^2-2*x*y-2*y^2,-2*x^2-2*x*y+y^2])
sage: f.automorphism_group(return_functions=True)
[x, 2/(2*x), -x - 1, -2*x/(2*x + 2), (-x - 1)/x, -1/(x + 1)]

sage: R.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(R)
sage: f = H([3*x^2*y - y^3,x^3 - 3*x*y^2])
sage: f.automorphism_group(algorithm='CRT',return_functions=True,iso_type=True)
([x, (x + 1)/(x - 1), (-x + 1)/(x + 1), -x, 1/x, -1/x, (x - 1)/(x + 1), (-x - 1)/(x - 1)], '

sage: A.<z> = AffineSpace(QQ,1)
sage: H = End(A)
sage: f = H([1/z^3])
sage: F = f.homogenize(1)
sage: F.automorphism_group()
```

```
[
[1 0]  [0 2]  [-1  0]  [ 0 -2]
[0 1], [2 0], [ 0  1], [ 2  0]
]
```

canonical_height (*P*, ***kwds*)

Evaluates the (absolute) canonical height of `P` with respect to `self`. Must be over number field or order of a number field. Specify either the number of terms of the series to evaluate or the error bound required.

ALGORITHM:

The sum of the Green's function at the archimedean places and the places of bad reduction.

INPUT:

- `P` – a projective point

kwds:

- `badprimes` - a list of primes of bad reduction (optional)
- `N` - positive integer. number of terms of the series to use in the local green functions (optional - default: 10)
- `prec` - positive integer, float point or p-adic precision, default: 100
- `error_bound` - a positive real number (optional)

OUTPUT:

- a real number

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(ZZ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,2*x*y]);
sage: f.canonical_height(P.point([5,4]), error_bound=0.001)
2.1970553519503404898926835324
sage: f.canonical_height(P.point([2,1]), error_bound=0.001)
1.0984430632822307984974382955
```

Notice that preperiodic points may not be exactly 0:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2-29/16*y^2,y^2]);
sage: f.canonical_height(P.point([1,4]), error_bound=0.000001)
1.9185995011736159021863458227e-7

sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: X = P.subscheme(x^2-y^2);
sage: H = Hom(X,X)
sage: f = H([x^2,y^2,4*z^2]);
sage: Q = X([4,4,1])
sage: f.canonical_height(Q, badprimes=[2])
0.0013538030870311431824555314882
```

conjugate (*M*)

Conjugates `self` by `M`, i.e. $M^{-1} \circ f \circ M$.

If possible the map will be defined over the same space as `self`. Otherwise, will try to coerce to the base_ring of `M`.

INPUT:

- M – a square invertible matrix

OUTPUT:

- a map from `self.domain()` to `self.codomain()`.

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(ZZ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2])
sage: f.conjugate(matrix([[1,2],[0,1]]))
Scheme endomorphism of Projective Space of dimension 1 over Integer Ring
  Defn: Defined on coordinates by sending (x : y) to
        (x^2 + 4*x*y + 3*y^2 : y^2)

sage: R.<x> = PolynomialRing(QQ)
sage: K.<i> = NumberField(x^2+1)
sage: P.<x,y> = ProjectiveSpace(ZZ,1)
sage: H = Hom(P,P)
sage: f = H([x^3+y^3,y^3])
sage: f.conjugate(matrix([[i,0],[0,-i]]))
Scheme endomorphism of Projective Space of dimension 1 over Integer Ring
  Defn: Defined on coordinates by sending (x : y) to
        (-x^3 + y^3 : -y^3)

sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2,y*z])
sage: f.conjugate(matrix([[1,2,3],[0,1,2],[0,0,1]]))
Scheme endomorphism of Projective Space of dimension 2 over Integer Ring
  Defn: Defined on coordinates by sending (x : y : z) to
        (x^2 + 4*x*y + 3*y^2 + 6*x*z + 9*y*z + 7*z^2 : y^2 + 2*y*z : y*z + 2*z^2)

sage: P.<x,y> = ProjectiveSpace(ZZ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2])
sage: f.conjugate(matrix([[2,0],[0,1/2]]))
Scheme endomorphism of Projective Space of dimension 1 over Multivariate
Polynomial Ring in x, y over Rational Field
  Defn: Defined on coordinates by sending (x : y) to
        (2*x^2 + 1/8*y^2 : 1/2*y^2)

sage: R.<x> = PolynomialRing(QQ)
sage: K.<i> = NumberField(x^2+1)
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([1/3*x^2+1/2*y^2,y^2])
sage: f.conjugate(matrix([[i,0],[0,-i]]))
Scheme endomorphism of Projective Space of dimension 1 over Multivariate
Polynomial Ring in x, y over Number Field in i with defining polynomial
x^2 + 1
  Defn: Defined on coordinates by sending (x : y) to
        ((1/3*i)*x^2 + (1/2*i)*y^2 : (-i)*y^2)
```

critical_height(***kwds*)

Compute the critical height of `self`. The critical height is defined by J. Silverman as the sum of the canonical heights of the critical points. This must be dimension 1 and defined over a number field or number field order.

INPUT:

kwds:

- `badprimes` - a list of primes of bad reduction (optional)
- `N` - positive integer. number of terms of the series to use in the local green functions (optional - Default: 10)
- `prec` - positive integer, float point or p-adic precision, Default: 100
- `error_bound` - a positive real number (optional)

OUTPUT: Real number

Examples:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^3+7*y^3, 11*y^3])
sage: f.critical_height()
1.1989273321156851418802151128

sage: K.<w> = QuadraticField(2)
sage: O = K.maximal_order()
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = Hom(P,P)
sage: f = H([x^2+w*y^2,y^2])
sage: f.critical_height()
0.16090842452312941163719755472
```

Postcritically finite maps have critical height 0:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^3-3/4*x*y^2 + 3/4*y^3, y^3])
sage: f.critical_height(error_bound=0.0001)
0.00000117385083669485564432459839996
```

critical_point_portrait(*check=True*)

If `self` is post-critically finite, return the critical point portrait of `self`. This is the directed graph of iterates starting with the critical points. Must be dimension 1. If `check` is True, then the map is first checked to see if it is postcrtically finite.

INPUT:

- check - Boolean

OUTPUT: a digraph

Examples:

```
sage: R.<z> = QQ[]
sage: K.<v> = NumberField(z^6 + 2*z^5 + 2*z^4 + 2*z^3 + z^2 + 1)
sage: PS.<x,y> = ProjectiveSpace(K,1)
sage: H = End(PS)
sage: f = H([x^2+v*y^2,y^2])
sage: f.critical_point_portrait(check = False) # long time
Looped digraph on 6 vertices

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
```

```
sage: f = H([x^5 + 5/4*x*y^4,y^5])
sage: f.critical_point_portrait(check = False)
Looped digraph on 5 vertices
```

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2 + 2*y^2,y^2])
sage: f.critical_point_portrait()
Traceback (most recent call last):
...
TypeError: Map be be post-critically finite
```

critical_points(*R=None*)

Returns the critical points of the endomorphism `self` defined over the ring `R` or the base ring of `self`. Must be dimension 1.

INPUT:

- `R` - a ring (optional)

OUTPUT: a list of projective space points defined over `R`

Examples:

```
sage: set_verbose(None)
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^3-2*x*y^2 + 2*y^3, y^3])
sage: f.critical_points()
[(1 : 0)]
sage: K.<w> = QuadraticField(6)
sage: f.critical_points(K)
[(-1/3*w : 1), (1/3*w : 1), (1 : 0)]
```

```
sage: set_verbose(None)
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([2*x^2-y^2, x*y])
sage: f.critical_points(QQbar)
[(-0.7071067811865475?*I : 1), (0.7071067811865475?*I : 1)]
```

degree()

This function returns the degree of `self`.

The degree is defined as the degree of the homogeneous polynomials that are the coordinates of `self`.

OUTPUT:

- A positive integer

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2])
sage: f.degree()
2
```

```
sage: P.<x,y,z> = ProjectiveSpace(CC,2)
sage: H = Hom(P,P)
sage: f = H([x^3+y^3,y^2*z,z*x*y])
```

```
sage: f.degree()
3

sage: R.<t> = PolynomialRing(QQ)
sage: P.<x,y,z> = ProjectiveSpace(R,2)
sage: H = Hom(P,P)
sage: f = H([x^2+t*y^2,(2-t)*y^2,z^2])
sage: f.degree()
2

sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: X = P.subscheme(x^2-y^2)
sage: H = Hom(X,X)
sage: f = H([x^2,y^2,z^2])
sage: f.degree()
2
```

dehomogenize(*n*)

Returns the standard dehomogenization at the `n[0]` coordinate for the domain and the `n[1]` coordinate for the codomain.

Note that the new function is defined over the fraction field of the base ring of `self`.

INPUT:

- **n – a tuple of nonnegative integers. If n is an integer, then the two values of** the tuple are assumed to be the same.

OUTPUT:

- `SchemeMorphism_polynomial_affine_space`

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(ZZ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2])
sage: f.dehomogenize(0)
Scheme endomorphism of Affine Space of dimension 1 over Integer Ring
  Defn: Defined on coordinates by sending (x) to
        (x^2/(x^2 + 1))

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2-y^2,y^2])
sage: f.dehomogenize((0,1))
Scheme endomorphism of Affine Space of dimension 1 over Rational Field
  Defn: Defined on coordinates by sending (x) to
        ((-x^2 + 1)/x^2)

sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2-z^2,2*z^2])
sage: f.dehomogenize(2)
Scheme endomorphism of Affine Space of dimension 2 over Rational Field
  Defn: Defined on coordinates by sending (x0, x1) to
        (1/2*x0^2 + 1/2*x1^2, 1/2*x1^2 - 1/2)

sage: R.<t> = PolynomialRing(QQ)
sage: P.<x,y,z> = ProjectiveSpace(FractionField(R),2)
sage: H = Hom(P,P)
```

```
sage: f = H([x^2+t*y^2,t*y^2-z^2,t*z^2])
sage: f.dehomogenize(2)
Scheme endomorphism of Affine Space of dimension 2 over Fraction Field
of Univariate Polynomial Ring in t over Rational Field
  Defn: Defined on coordinates by sending (x0, x1) to
        (1/t*x0^2 + x1^2, x1^2 - 1/t)
```

```
sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: X = P.subscheme(x^2-y^2)
sage: H = Hom(X,X)
sage: f = H([x^2,y^2,x*z])
sage: f.dehomogenize(2)
Scheme endomorphism of Closed subscheme of Affine Space of dimension 2
over Integer Ring defined by:
  x0^2 - x1^2
  Defn: Defined on coordinates by sending (x0, x1) to
        (x1^2/x0, x1^2/x0)
```

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2 - 2*x*y, y^2])
sage: f.dehomogenize(0).homogenize(0) == f
True
```

dynatomic_polynomial(*period*)

For a map $f : \mathbb{P}^1 \to \mathbb{P}^1$ this function computes the dynatomic polynomial.

The dynatomic polynomial is the analog of the cyclotomic polynomial and its roots are the points of formal period $period$. If possible the division is done in the coordinate ring of `self` and a polynomial is returned. In rings where that is not possible, a FractionField element will be returned. In certain cases, when the conversion back to a polynomial fails, a SymbolRing element will be returned.

ALGORITHM:

For a positive integer n, let $[F_n, G_n]$ be the coordinates of the nth iterate of f. Then construct

$$\Phi^*_n(f)(x,y) = \sum_{d|n} (yF_d(x,y) - xG_d(x,y))^{\mu(n/d)}$$

where μ is the Moebius function.

For a pair $[m, n]$, let $f^m = [F_m, G_m]$. Compute

$$\Phi^*_{m,n}(f)(x,y) = \Phi^*_n(f)(F_m, G_m)/\Phi^*_n(f)(F_{m-1}, G_{m-1})$$

REFERENCES:

INPUT:

- `period` – a positive integer or a list/tuple $[m, n]$ where m is the preperiod and n is the period

OUTPUT:

- If possible, a two variable polynomial in the coordinate ring of `self`. Otherwise a fraction field element of the coordinate ring of `self`. Or, a Symbolic Ring element.

Todo

- Do the division when the base ring is p-adic so that the output is a polynomial.
- Convert back to a polynomial when the base ring is a function field (not over $\mathbf{Q}$ or F_p)

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2])
sage: f.dynatomic_polynomial(2)
x^2 + x*y + 2*y^2

sage: P.<x,y> = ProjectiveSpace(ZZ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,x*y])
sage: f.dynatomic_polynomial(4)
2*x^12 + 18*x^10*y^2 + 57*x^8*y^4 + 79*x^6*y^6 + 48*x^4*y^8 + 12*x^2*y^10 + y^12

sage: P.<x,y> = ProjectiveSpace(CC,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,3*x*y])
sage: f.dynatomic_polynomial(3)
13.0000000000000*x^6 + 117.000000000000*x^4*y^2 +
78.0000000000000*x^2*y^4 + y^6

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2-10/9*y^2,y^2])
sage: f.dynatomic_polynomial([2,1])
x^4*y^2 - 11/9*x^2*y^4 - 80/81*y^6

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2-29/16*y^2,y^2])
sage: f.dynatomic_polynomial([2,3])
x^12 - 95/8*x^10*y^2 + 13799/256*x^8*y^4 - 119953/1024*x^6*y^6 +
8198847/65536*x^4*y^8 - 31492431/524288*x^2*y^10 +
172692729/16777216*y^12

sage: P.<x,y> = ProjectiveSpace(ZZ,1)
sage: H = Hom(P,P)
sage: f = H([x^2-y^2,y^2])
sage: f.dynatomic_polynomial([1,2])
x^2 - x*y

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^3-y^3,3*x*y^2])
sage: f.dynatomic_polynomial([0,4])==f.dynatomic_polynomial(4)
True

sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,x*y,z^2])
sage: f.dynatomic_polynomial(2)
Traceback (most recent call last):
...
TypeError: Does not make sense in dimension >1

sage: P.<x,y> = ProjectiveSpace(Qp(5),1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2])
```

```
sage: f.dynatomic_polynomial(2)
(x^4*y + (2 + O(5^20))*x^2*y^3 - x*y^4 + (2 + O(5^20))*y^5)/(x^2*y -
x*y^2 + y^3)

sage: L.<t> = PolynomialRing(QQ)
sage: P.<x,y> = ProjectiveSpace(L,1)
sage: H = Hom(P,P)
sage: f = H([x^2+t*y^2,y^2])
sage: f.dynatomic_polynomial(2)
x^2 + x*y + (t + 1)*y^2

sage: K.<c> = PolynomialRing(ZZ)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = Hom(P,P)
sage: f = H([x^2+ c*y^2,y^2])
sage: f.dynatomic_polynomial([1,2])
x^2 - x*y + (c + 1)*y^2

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2])
sage: f.dynatomic_polynomial(2)
x^2 + x*y + 2*y^2
sage: R.<X> = PolynomialRing(QQ)
sage: K.<c> = NumberField(X^2 + X + 2)
sage: PP = P.change_ring(K)
sage: ff = f.change_ring(K)
sage: p = PP((c,1))
sage: ff(ff(p)) == p
True

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,x*y])
sage: f.dynatomic_polynomial([2,2])
x^4 + 4*x^2*y^2 + y^4
sage: R.<X> = PolynomialRing(QQ)
sage: K.<c> = NumberField(X^4 + 4*X^2 + 1)
sage: PP = P.change_ring(K)
sage: ff = f.change_ring(K)
sage: p = PP((c,1))
sage: ff.nth_iterate(p,4) == ff.nth_iterate(p,2)
True

sage: P.<x,y> = ProjectiveSpace(CC, 1)
sage: H = Hom(P,P)
sage: f = H([x^2-CC.0/3*y^2,y^2])
sage: f.dynatomic_polynomial(2)
0.666666666666667*x^2 + 0.333333333333333*y^2

sage: L.<t> = PolynomialRing (QuadraticField(2).maximal_order())
sage: P.<x, y> = ProjectiveSpace (L.fraction_field() , 1 )
sage: H = Hom (P, P )
sage: f = H ([x^2 + (t ^ 2 + 1) * y^2 , y^2 ])
sage: f.dynatomic_polynomial(2)
x^2 + x*y + (t^2 + 2)*y^2
```

TESTS:

We check that the dynatomic polynomial has the right parent (see trac ticket #18409):

```
sage: P.<x,y> = ProjectiveSpace(QQbar,1)
sage: H = End(P)
sage: R = P.coordinate_ring()
sage: f = H([x^2-1/3*y^2,y^2])
sage: f.dynatomic_polynomial(2).parent()
Multivariate Polynomial Ring in x, y over Algebraic Field
```

```
sage: T.<v> = QuadraticField(33)
sage: S.<t> = PolynomialRing(T)
sage: P.<x,y> = ProjectiveSpace(FractionField(S),1)
sage: H = End(P)
sage: f = H([t*x^2-1/t*y^2,y^2])
sage: f.dynatomic_polynomial([1,2]).parent()
Multivariate Polynomial Ring in x, y over Fraction Field of Univariate Polynomial
Ring in t over Number Field in v with defining polynomial x^2 - 33
```

This one still does not work, some function fields still return Symoblic Ring elements:

```
sage: S.<t> = FunctionField(CC)
sage: P.<x,y> = ProjectiveSpace(S,1)
sage: H = End(P)
sage: R = P.coordinate_ring()
sage: f = H([t*x^2-1*y^2,t*y^2])
sage: f.dynatomic_polynomial([1,2]).parent()
Symbolic Ring
```

global_height (*prec=None*)

Returns the maximum of the absolute logarithmic heights of the coefficients in any of the coordinate functions of `self`.

INPUT:

- `prec` – desired floating point precision (default: default RealField precision).

OUTPUT:

- a real number

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([1/1331*x^2+1/4000*y^2,210*x*y]);
sage: f.global_height()
8.29404964010203
```

This function does not automatically normalize:

```
sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: H = Hom(P,P)
sage: f = H([4*x^2+100*y^2,210*x*y,10000*z^2]);
sage: f.global_height()
9.21034037197618
sage: f.normalize_coordinates()
sage: f.global_height()
8.51719319141624
```

```
sage: R.<z> = PolynomialRing(QQ)
sage: K.<w> = NumberField(z^2-2)
sage: O = K.maximal_order()
```

```
sage: P.<x,y> = ProjectiveSpace(O,1)
sage: H = Hom(P,P)
sage: f = H([2*x^2 + 3*O(w)*y^2,O(w)*y^2])
sage: f.global_height()
1.44518587894808
```

```
sage: P.<x,y> = ProjectiveSpace(QQbar,1)
sage: P2.<u,v,w> = ProjectiveSpace(QQbar,2)
sage: H = Hom(P,P2)
sage: f = H([x^2 + QQbar(I)*x*y + 3*y^2,y^2,QQbar(sqrt(5))*x*y])
sage: f.global_height()
1.09861228866811
```

green_function(*P*, *v*, ***kwds*)

Evaluates the local Green's function at the place `v` for `P` with `N` terms of the series or to within a given error bound. Must be over a number field or order of a number field. Note that this is absolute local greens function so is scaled by the degree of the base field.

Use `v=0` for the archimedean place over **Q** or field embedding. Non-archimedean places are prime ideals for number fields or primes over **Q**.

ALGORITHM:

See Exercise 5.29 and Figure 5.6 of `The Arithmetic of Dynamics Systems`, Joseph H. Silverman, Springer, GTM 241, 2007.

INPUT:

- `P` - a projective point
- `v` - non-negative integer. a place, use v=0 for the archimedean place

kwds:

- `N` - positive integer. number of terms of the series to use, (optional - default: 10)
- `prec` - positive integer, float point or p-adic precision, default: 100
- `error_bound` - a positive real number (optional)

OUTPUT:

- a real number

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,x*y])
sage: f.green_function(P.point([5,2],False),0,N=30)
1.7315451844777407992085512000
sage: f.green_function(P.point([2,1],False),0,N=30)
0.86577259223181088325226209926
sage: f.green_function(P.point([1,1],False),0,N=30)
0.43288629610862338612700146098
```

height_difference_bound(*prec=None*)

Returns an upper bound on the different bewtween the canonical height of a point with respect to `self` and the absolute height of the point. `self` must be a morphism.

ALGORITHM:

Uses a Nullstellensatz argument to compute the constant. For details: B. Hutz, Efficient determination of rational preperiodic points for endomorphisms of projective space, arxiv:1210.6246 (2012).

INPUT:

- `prec` - positive integer, float point, default: RealField default

OUTPUT:

- a real number

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2+y^2,x*y]);
sage: f.height_difference_bound()
1.38629436111989
```

This function does not automatically normalize.

```
sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: H = End(P)
sage: f = H([4*x^2+100*y^2,210*x*y,10000*z^2]);
sage: f.height_difference_bound()
11.0020998412042
sage: f.normalize_coordinates()
sage: f.height_difference_bound()
10.3089526606443
```

A number field example:

```
sage: R.<x> = QQ[]
sage: K.<c> = NumberField(x^3 - 2)
sage: P.<x,y,z> = ProjectiveSpace(K,2)
sage: H = End(P)
sage: f = H([1/(c+1)*x^2+c*y^2,210*x*y,10000*z^2])
sage: f.height_difference_bound()
11.0020998412042
```

::

```
sage: P.<x,y,z> = ProjectiveSpace(QQbar,2)
sage: H = End(P)
sage: f = H([x^2, QQbar(sqrt(-1))*y^2, QQbar(sqrt(3))*z^2])
sage: f.height_difference_bound()
3.43967790223022
```

`is_PGL_minimal`(*prime_list=None*)

Checks if `self` is a minimal model in its conjugacy class. See [Bruin-Molnar] and [Molnar] for a description of the algorithm.

INPUT:

- `prime_list` – list of primes to check minimality, if None, check all places

OUTPUT:

- Boolean - True if `self` is minimal, False otherwise.

EXAMPLES:

```
sage: PS.<X,Y> = ProjectiveSpace(QQ,1)
sage: H = End(PS)
sage: f = H([X^2+3*Y^2,X*Y])
sage: f.is_PGL_minimal()
True
```

```
sage: PS.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(PS)
sage: f = H([6*x^2+12*x*y+7*y^2,12*x*y])
sage: f.is_PGL_minimal()
False
```

```
sage: PS.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(PS)
sage: f = H([6*x^2+12*x*y+7*y^2,y^2])
sage: f.is_PGL_minimal()
Traceback (most recent call last):
...
TypeError: Affine minimality is only considered for maps not of the form
f or 1/f for a polynomial f.
```

is_morphism()

returns `True` if self is a morphism (no common zero of defining polynomials).

The map is a morphism if and only if the ideal generated by the defining polynomials is the unit ideal.

OUTPUT:

- Boolean

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2])
sage: f.is_morphism()
True
```

```
sage: P.<x,y,z> = ProjectiveSpace(RR,2)
sage: H = Hom(P,P)
sage: f = H([x*z-y*z,x^2-y^2,z^2])
sage: f.is_morphism()
False
```

```
sage: R.<t> = PolynomialRing(GF(5))
sage: P.<x,y,z> = ProjectiveSpace(R,2)
sage: H = Hom(P,P)
sage: f = H([x*z-t*y^2,x^2-y^2,t*z^2])
sage: f.is_morphism()
True
```

Map that is not morphism on projective space, but is over a subscheme:

```
sage: P.<x,y,z> = ProjectiveSpace(RR,2)
sage: X = P.subscheme([x*y + y*z])
sage: H = Hom(X,X)
sage: f = H([x*z-y*z,x^2-y^2,z^2])
sage: f.is_morphism()
True
```

is_postcritically_finite (*err=0.01*)

Determine if `self` is post-critially finite for `self` an endomorphism of $\mathbb{P}^1$, i.e., check if each critical point is preperiodic. The optional parameter `err` is passed into `is_preperiodic()` as part of the preperiodic check.

INPUT:

- `err` - positive real number (optional, Default: 0.01)

OUTPUT: Boolean

Examples:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2 - y^2, y^2])
sage: f.is_postcritically_finite()
True

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^3- y^3, y^3])
sage: f.is_postcritically_finite()
False

sage: R.<z> = QQ[]
sage: K.<v> = NumberField(z^8 + 3*z^6 + 3*z^4 + z^2 + 1)
sage: PS.<x,y> = ProjectiveSpace(K,1)
sage: H = End(PS)
sage: f = H([x^3+v*y^3,y^3])
sage: f.is_postcritically_finite() # long time
True

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([6*x^2+16*x*y+16*y^2,-3*x^2-4*x*y-4*y^2])
sage: f.is_postcritically_finite()
True
```

local_height (*v*, *prec=None*)

Returns the maximum of the local height of the coefficients in any of the coordinate functions of `self`.

INPUT:

- `v` – a prime or prime ideal of the base ring
- `prec` – desired floating point precision (default: default RealField precision).

OUTPUT:

- a real number

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([1/1331*x^2+1/4000*y^2,210*x*y]);
sage: f.local_height(1331)
7.19368581839511
```

This function does not automatically normalize:

```
sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = Hom(P,P)
sage: f = H([4*x^2+3/100*y^2,8/210*x*y,1/10000*z^2]);
sage: f.local_height(2)
2.77258872223978
sage: f.normalize_coordinates()
sage: f.local_height(2)
0.000000000000000
```

```
sage: R.<z> = PolynomialRing(QQ)
sage: K.<w> = NumberField(z^2-2)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = Hom(P,P)
sage: f = H([2*x^2 + w/3*y^2,1/w*y^2])
sage: f.local_height(K.ideal(3))
1.09861228866811
```

local_height_arch(*i*, *prec=None*)

Returns the maximum of the local height at the `i`-th infinite place of the coefficients in any of the coordinate functions of `self`.

INPUT:

- `i` – an integer
- `prec` – desired floating point precision (default: default RealField precision).

OUTPUT:

- a real number

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([1/1331*x^2+1/4000*y^2,210*x*y]);
sage: f.local_height_arch(0)
5.34710753071747
```

```
sage: R.<z> = PolynomialRing(QQ)
sage: K.<w> = NumberField(z^2-2)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = Hom(P,P)
sage: f = H([2*x^2 + w/3*y^2,1/w*y^2])
sage: f.local_height_arch(1)
0.6931471805599453094172321214582
```

minimal_model(*return_transformation=False*, *prime_list=None*)

Given `self` a scheme morphism on the projective line over the rationals, determine if `self` is minimal. In particular, determine if `self` is affine minimal, which is enough to decide if it is minimal or not. See Proposition 2.10 in [Bruin-Molnar].

REFERENCES:

[Bruin-Molnar], [Molnar]

INPUT:

- `self` – scheme morphism on the projective line defined over QQ.
- **`return_transformation` – a boolean value, default value True. This** signals a return of the `PGL_2` transformation to conjugate `self` to the calculated minimal model. default: False

- **prime_list – a list of primes, in case one only wants to determine minimality** at those specific primes.

OUTPUT:

- a scheme morphism on the projective line which is a minimal model of `self`.
- a $PGL(2, QQ)$ element which conjugates `self` to a minimal model

EXAMPLES:

```
sage: PS.<X,Y> = ProjectiveSpace(QQ,1)
sage: H = End(PS)
sage: f = H([X^2+3*Y^2,X*Y])
sage: f.minimal_model(return_transformation=True)
(
Scheme endomorphism of Projective Space of dimension 1 over Rational
Field
  Defn: Defined on coordinates by sending (X : Y) to
        (X^2 + 3*Y^2 : X*Y)
,
[1 0]
[0 1]
)
```

```
sage: PS.<X,Y> = ProjectiveSpace(QQ,1)
sage: H = End(PS)
sage: f = H([7365/2*X^4 + 6282*X^3*Y + 4023*X^2*Y^2 + 1146*X*Y^3 + 245/2*Y^4, -12329/2*X^4 -
sage: f.minimal_model(return_transformation=True)
(
Scheme endomorphism of Projective Space of dimension 1 over Rational
Field
  Defn: Defined on coordinates by sending (X : Y) to
        (22176*X^4 + 151956*X^3*Y + 390474*X^2*Y^2 + 445956*X*Y^3 +
190999*Y^4 : -12329*X^4 - 84480*X^3*Y - 217080*X^2*Y^2 - 247920*X*Y^3 -
106180*Y^4),
[2 3]
[0 1]
)
```

```
sage: PS.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(PS)
sage: f = H([6*x^2+12*x*y+7*y^2,12*x*y])
sage: f.minimal_model()
Scheme endomorphism of Projective Space of dimension 1 over Rational
Field
  Defn: Defined on coordinates by sending (x : y) to
        (x^2 + 12*x*y + 42*y^2 : 2*x*y)
```

```
sage: PS.<x,y> = ProjectiveSpace(ZZ,1)
sage: H = End(PS)
sage: f = H([6*x^2+12*x*y+7*y^2,12*x*y + 42*y^2])
sage: g,M=f.minimal_model(return_transformation=True)
sage: f.conjugate(M)==g
True
```

```
sage: PS.<X,Y> = ProjectiveSpace(QQ,1)
sage: H = End(PS)
sage: f = H([X+Y,X-3*Y])
sage: f.minimal_model()
Traceback (most recent call last):
```

```
...
NotImplementedError: Minimality is only for degree 2 or higher

sage: PS.<X,Y> = ProjectiveSpace(QQ,1)
sage: H = End(PS)
sage: f = H([X^2-Y^2,X^2+X*Y])
sage: f.minimal_model()
Traceback (most recent call last):
...
TypeError: The function is not a morphism
```

multiplier(*P*, *n*, *check=True*)

Returns the multiplier of `self` point `P` of period `n`. `self` must be an endomorphism.

INPUT:

- `P` - a point on domain of `self`
- `n` - a positive integer, the period of `P`
- `check` – verify that `P` has period `n`, Default:True

OUTPUT:

- a square matrix of size `self.codomain().dimension_relative()` in the `base_ring` of `self`

EXAMPLES:

```
sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = End(P)
sage: f = H([x^2,y^2,4*z^2]);
sage: Q = P.point([4,4,1],False);
sage: f.multiplier(Q,1)
[2 0]
[0 2]

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([7*x^2 - 28*y^2,24*x*y])
sage: f.multiplier(P(2,5),4)
[231361/20736]

sage: P.<x,y> = ProjectiveSpace(CC,1)
sage: H = End(P)
sage: f = H([x^3 - 25*x*y^2 + 12*y^3,12*y^3])
sage: f.multiplier(P(1,1),5)
[0.389017489711935]

sage: P.<x,y> = ProjectiveSpace(RR,1)
sage: H = End(P)
sage: f = H([x^2-2*y^2,y^2])
sage: f.multiplier(P(2,1),1)
[4.00000000000000]

sage: P.<x,y> = ProjectiveSpace(Qp(13),1)
sage: H = End(P)
sage: f = H([x^2-29/16*y^2,y^2])
sage: f.multiplier(P(5,4),3)
[6 + 8*13 + 13^2 + 8*13^3 + 13^4 + 8*13^5 + 13^6 + 8*13^7 + 13^8 +
8*13^9 + 13^10 + 8*13^11 + 13^12 + 8*13^13 + 13^14 + 8*13^15 + 13^16 +
```

```
8*13^17 + 13^18 + 8*13^19 + O(13^20)]

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2-y^2,y^2])
sage: f.multiplier(P(0,1),1)
Traceback (most recent call last):
...
ValueError: (0 : 1) is not periodic of period 1
```

multiplier_spectra (*n*, *formal=True*)

Computes the formal n multiplier spectra of self, which is the set of multipliers of the periodic points of formal period n of self included with the appropriate multiplicity. User can also specify to compute the n multiplier spectra instead which includes the multipliers of all periodic points of period n of self. self must be defined over projective space of dimension 1 over a number field.

INPUT:

- n - a positive integer, the period
- **formal - a Boolean. True specifies to find the formal n multiplier spectra of self. False** specifies to find the n multiplier spectra of self. Default: True

OUTPUT:

- a list of QQbar elements

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([4608*x^10 - 2910096*x^9*y + 325988068*x^8*y^2 + 31825198932*x^7*y^3 - 413980662
- 44439736715486*x^5*y^5 + 2317935971590902*x^4*y^6 - 15344764859590852*x^3*y^7 + 2561851642
+ 113578270285012470*x*y^9 - 150049940203963800*y^10, 4608*y^10])
sage: f.multiplier_spectra(1)
[0, -7198147681176255644585/256, 848446157556848459363/19683, -3323781962860268721722583135/
529278480109921/256, -4290991994944936653/2097152, 1061953534167447403/19683, -3086380435599
8291137267280816193O567/8192, -119820502365680843999, 3553497751559301575157261317/8192]

sage: set_verbose(None)
sage: z = QQ['z'].0
sage: K.<w> = NumberField(z^4 - 4*z^2 + 1,'z')
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = End(P)
sage: f = H([x^2 - w/4*y^2,y^2])
sage: f.multiplier_spectra(2,False)
[0, 5.931851652578137? + 0.?e-17*I, 0.0681483474218635? - 1.930649271699173?*I,
0.0681483474218635? + 1.930649271699173?*I]

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2 - 3/4*y^2,y^2])
sage: f.multiplier_spectra(2)
[1]

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2 - 7/4*y^2,y^2])
sage: f.multiplier_spectra(3)
[1, 1]
```

normalize_coordinates()

Scales by 1/gcd of the coordinate functions. Also, scales to clear any denominators from the coefficients. This is done in place.

OUTPUT:

- None.

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([5/4*x^3,5*x*y^2])
sage: f.normalize_coordinates(); f
Scheme endomorphism of Projective Space of dimension 1 over Rational
Field
  Defn: Defined on coordinates by sending (x : y) to
        (x^2 : 4*y^2)

sage: P.<x,y,z> = ProjectiveSpace(GF(7),2)
sage: X = P.subscheme(x^2-y^2)
sage: H = Hom(X,X)
sage: f = H([x^3+x*y^2,x*y^2,x*z^2])
sage: f.normalize_coordinates(); f
Scheme endomorphism of Closed subscheme of Projective Space of dimension
2 over Finite Field of size 7 defined by:
  x^2 - y^2
  Defn: Defined on coordinates by sending (x : y : z) to
        (2*y^2 : y^2 : z^2)
```

Note: gcd raises an error if the base_ring does not support gcds.

nth_iterate(*P*, *n*, *normalize=False*)

For a map `self` and a point P in `self.domain()` this function returns the nth iterate of P by `self`.

If `normalize` is `True`, then the coordinates are automatically normalized.

Todo

Is there a more efficient way to do this?

INPUT:

- `P` – a point in `self.domain()`
- `n` – a positive integer.
- `normalize` - Boolean (optional Default: `False`)

OUTPUT:

- A point in `self.codomain()`

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(ZZ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,2*y^2])
sage: Q = P(1,1)
sage: f.nth_iterate(Q,4)
(32768 : 32768)
```

```
sage: P.<x,y> = ProjectiveSpace(ZZ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,2*y^2])
sage: Q = P(1,1)
sage: f.nth_iterate(Q,4,1)
(1 : 1)
```

Is this the right behavior?

```
sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = Hom(P,P)
sage: f = H([x^2,2*y^2,z^2-x^2])
sage: Q = P(2,7,1)
sage: f.nth_iterate(Q,2)
(-16/7 : -2744 : 1)
```

```
sage: R.<t> = PolynomialRing(QQ)
sage: P.<x,y,z> = ProjectiveSpace(R,2)
sage: H = Hom(P,P)
sage: f = H([x^2+t*y^2,(2-t)*y^2,z^2])
sage: Q = P(2+t,7,t)
sage: f.nth_iterate(Q,2)
(t^4 + 2507*t^3 - 6787*t^2 + 10028*t + 16 : -2401*t^3 + 14406*t^2 -
28812*t + 19208 : t^4)
```

```
sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: X = P.subscheme(x^2-y^2)
sage: H = Hom(X,X)
sage: f = H([x^2,y^2,z^2])
sage: f.nth_iterate(X(2,2,3),3)
(256 : 256 : 6561)
```

```
sage: K.<c> = FunctionField(QQ)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = Hom(P,P)
sage: f = H([x^3-2*x*y^2 - c*y^3,x*y^2])
sage: f.nth_iterate(P(c,1),2)
((c^6 - 9*c^4 + 25*c^2 - c - 21)/(c^2 - 3) : 1)
```

nth_iterate_map(*n*)

For a map `self` this function returns the nth iterate of `self` as a function on `self.domain()`

ALGORITHM:

Uses a form of successive squaring to reducing computations.

Todo

This could be improved.

INPUT:

- `n` – a positive integer.

OUTPUT:

- A map between projective spaces

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2])
sage: f.nth_iterate_map(2)
Scheme endomorphism of Projective Space of dimension 1 over Rational
Field
  Defn: Defined on coordinates by sending (x : y) to
        (x^4 + 2*x^2*y^2 + 2*y^4 : y^4)
```

```
sage: P.<x,y> = ProjectiveSpace(CC,1)
sage: H = Hom(P,P)
sage: f = H([x^2-y^2,x*y])
sage: f.nth_iterate_map(3)
Scheme endomorphism of Projective Space of dimension 1 over Complex
Field with 53 bits of precision
  Defn: Defined on coordinates by sending (x : y) to
        (x^8 + (-7.00000000000000)*x^6*y^2 + 13.0000000000000*x^4*y^4 +
(-7.00000000000000)*x^2*y^6 + y^8 : x^7*y + (-4.00000000000000)*x^5*y^3
+ 4.00000000000000*x^3*y^5 - x*y^7)
```

```
sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: H = Hom(P,P)
sage: f = H([x^2-y^2,x*y,z^2+x^2])
sage: f.nth_iterate_map(2)
Scheme endomorphism of Projective Space of dimension 2 over Integer Ring
  Defn: Defined on coordinates by sending (x : y : z) to
        (x^4 - 3*x^2*y^2 + y^4 : x^3*y - x*y^3 : 2*x^4 - 2*x^2*y^2 + y^4
+ 2*x^2*z^2 + z^4)
```

```
sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: X = P.subscheme(x*z-y^2)
sage: H = Hom(X,X)
sage: f = H([x^2,x*z,z^2])
sage: f.nth_iterate_map(2)
Scheme endomorphism of Closed subscheme of Projective Space of dimension
2 over Rational Field defined by:
  -y^2 + x*z
  Defn: Defined on coordinates by sending (x : y : z) to
        (x^4 : x^2*z^2 : z^4)
```

orbit (*P*, *N*, ***kwds*)

Returns the orbit of P by `self`. If n is an integer it returns $[P, self(P), \ldots, self^n(P)]$. If n is a list or tuple $n = [m, k]$ it returns $[self^m(P), \ldots, self^k(P)]$. Automatically normalize the points if `normalize=True`. Perform the checks on point initialize if `check=True`

INPUT:

- `P` – a point in `self.domain()`
- `n` – a non-negative integer or list or tuple of two non-negative integers

kwds:

- `check` – boolean (optional - default: `True`)
- `normalize` – boolean (optional - default: `False`)

OUTPUT:

- a list of points in `self.codomain()`

EXAMPLES:

```
sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2-z^2,2*z^2])
sage: f.orbit(P(1,2,1),3)
[(1 : 2 : 1), (5 : 3 : 2), (34 : 5 : 8), (1181 : -39 : 128)]
```

```
sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2-z^2,2*z^2])
sage: f.orbit(P(1,2,1),[2,4])
[(34 : 5 : 8), (1181 : -39 : 128), (1396282 : -14863 : 32768)]
```

```
sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: X = P.subscheme(x^2-y^2)
sage: H = Hom(X,X)
sage: f = H([x^2,y^2,x*z])
sage: f.orbit(X(2,2,3),3,normalize=True)
[(2 : 2 : 3), (2 : 2 : 3), (2 : 2 : 3), (2 : 2 : 3)]
```

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2])
sage: f.orbit(P.point([1,2],False),4,check = False)
[(1 : 2), (5 : 4), (41 : 16), (1937 : 256), (3817505 : 65536)]
```

```
sage: K.<c> = FunctionField(QQ)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = Hom(P,P)
sage: f = H([x^2+c*y^2,y^2])
sage: f.orbit(P(0,1),3)
[(0 : 1), (c : 1), (c^2 + c : 1), (c^4 + 2*c^3 + c^2 + c : 1)]
```

periodic_points(*n*, *minimal=True*)

Computes the periodic points of period `n` of `self`. For now, `self` must be a projective morphism over a number field.

INPUT:

- `n` - a positive integer
- **`minimal` - Boolean. True specifies to find only the periodic points of minimal period `n`.** False specifies to find all periodic points of period `n`. Default: True.

OUTPUT:

- a list of periodic points of `self`

EXAMPLES:

```
sage: set_verbose(None)
sage: P.<x,y> = ProjectiveSpace(QQbar,1)
sage: H = Hom(P,P)
sage: f = H([x^2-x*y+y^2,x^2-y^2+x*y])
sage: f.periodic_points(1)
[(-0.500000000000000? - 0.866025403784439?*I : 1), (-0.500000000000000? + 0.866025403784439?
(1 : 1)]
```

```
sage: P.<x,y,z> = ProjectiveSpace(QuadraticField(5,'t'),2)
sage: H = Hom(P,P)
sage: f = H([x^2 - 21/16*z^2,y^2-z^2,z^2])
```

```
sage: f.periodic_points(2)
[(-5/4 : -1 : 1), (-5/4 : -1/2*t + 1/2 : 1), (-5/4 : 0 : 1), (-5/4 : 1/2*t + 1/2 : 1), (-3/4
(-3/4 : 0 : 1), (1/4 : -1 : 1), (1/4 : -1/2*t + 1/2 : 1), (1/4 : 0 : 1), (1/4 : 1/2*t + 1/2
(7/4 : -1 : 1), (7/4 : 0 : 1)]
```

```
sage: w = QQ['w'].0
sage: K = NumberField(w^6 - 3*w^5 + 5*w^4 - 5*w^3 + 5*w^2 - 3*w + 1,'s')
sage: P.<x,y,z> = ProjectiveSpace(K,2)
sage: H = Hom(P,P)
sage: f = H([x^2+z^2,y^2+x^2,z^2+y^2])
sage: f.periodic_points(1)
[(-s^5 + 3*s^4 - 5*s^3 + 4*s^2 - 3*s + 1 : s^5 - 2*s^4 + 3*s^3 - 3*s^2 + 4*s - 1 : 1),
(2*s^5 - 6*s^4 + 9*s^3 - 8*s^2 + 7*s - 4 : 2*s^5 - 5*s^4 + 7*s^3 - 5*s^2 + 6*s - 2 : 1),
(-2*s^5 + 4*s^4 - 5*s^3 + 3*s^2 - 4*s : -2*s^5 + 5*s^4 - 7*s^3 + 6*s^2 - 7*s + 3 : 1),
(-s^5 + 3*s^4 - 4*s^3 + 4*s^2 - 4*s + 2 : -s^5 + 2*s^4 - 2*s^3 + s^2 - s : 1),
(s^5 - 2*s^4 + 2*s^3 + s : s^5 - 3*s^4 + 4*s^3 - 3*s^2 + 2*s - 1 : 1), (1 : 1 : 1),
(s^5 - 2*s^4 + 3*s^3 - 3*s^2 + 3*s - 1 : -s^5 + 3*s^4 - 5*s^3 + 4*s^2 - 4*s + 2 : 1)]
```

```
sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = Hom(P,P)
sage: f = H([x^2 - 21/16*z^2,y^2-2*z^2,z^2])
sage: f.periodic_points(2,False)
[(-5/4 : -1 : 1), (-5/4 : 2 : 1), (-3/4 : -1 : 1), (-3/4 : 2 : 1), (0 : 1 : 0), (1/4 : -1 :
(1/4 : 2 : 1), (1 : 0 : 0), (1 : 1 : 0), (7/4 : -1 : 1), (7/4 : 2 : 1)]
```

```
sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = Hom(P,P)
sage: f = H([x^2 - 21/16*z^2,y^2-2*z^2,z^2])
sage: f.periodic_points(2)
[(-5/4 : -1 : 1), (-5/4 : 2 : 1), (1/4 : -1 : 1), (1/4 : 2 : 1)]
```

possible_periods(***kwds*)

Returns the set of possible periods for rational periodic points of self. Must be defined over **Z** or **Q**.

ALGORITHM: Calls `self.possible_periods()` modulo all primes of good reduction in range `prime_bound`. Returns the intersection of those lists.

INPUT:

kwds:

- **`prime_bound` - a list or tuple of two positive integers. Or an integer for the upper bound. (optional)** default: [1,20].
- `bad_primes` - a list or tuple of integer primes, the primes of bad reduction. (optional)
- **`ncpus` - number of cpus to use in parallel. (optional)** default: all available cpus.

OUTPUT:

- a list of positive integers.

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2-29/16*y^2,y^2])
sage: f.possible_periods(ncpus=1)
[1, 3]
```

```
sage: PS.<x,y> = ProjectiveSpace(1,QQ)
sage: H = End(PS)
sage: f = H([5*x^3 - 53*x*y^2 + 24*y^3, 24*y^3])
sage: f.possible_periods(prime_bound=[1,5])
Traceback (most recent call last):
...
ValueError: No primes of good reduction in that range
sage: f.possible_periods(prime_bound=[1,10])
[1, 4, 12]
sage: f.possible_periods(prime_bound=[1,20])
[1, 4]
```

```
sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: H = End(P)
sage: f = H([2*x^3 - 50*x*z^2 + 24*z^3,5*y^3 - 53*y*z^2 + 24*z^3,24*z^3])
sage: f.possible_periods(prime_bound=10)
[1, 2, 6, 20, 42, 60, 140, 420]
sage: f.possible_periods(prime_bound=20) # long time
[1, 20]
```

primes_of_bad_reduction(*check=True*)

Determines the primes of bad reduction for a map $self : \mathbb{P}^N \to \mathbb{P}^N$ defined over number fields.

If `check` is `True`, each prime is verified to be of bad reduction.

ALGORITHM:

p is a prime of bad reduction if and only if the defining polynomials of self have a common zero. Or stated another way, p is a prime of bad reducion if and only if the radical of the ideal defined by the defining polynomials of self is not $(x_0, x_1, \ldots, x_N)$. This happens if and only if some power of each x_i is not in the ideal defined by the defining polynomials of self. This last condition is what is checked. The lcm of the coefficients of the monomials x_i in a groebner basis is computed. This may return extra primes.

INPUT:

- `check` – Boolean (optional - default: `True`)

OUTPUT:

- a list of integer primes.

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([1/3*x^2+1/2*y^2,y^2])
sage: print f.primes_of_bad_reduction()
[2, 3]
```

```
sage: P.<x,y,z,w> = ProjectiveSpace(QQ,3)
sage: H = Hom(P,P)
sage: f = H([12*x*z-7*y^2,31*x^2-y^2,26*z^2,3*w^2-z*w])
sage: f.primes_of_bad_reduction()
[2, 3, 7, 13, 31]
```

A number field example

```
sage: R.<z> = QQ[]
sage: K.<a> = NumberField(z^2 - 2)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = Hom(P,P)
```

```
sage: f = H([1/3*x^2+1/a*y^2,y^2])
sage: f.primes_of_bad_reduction()
[Fractional ideal (a), Fractional ideal (3)]
```

This is an example where check = False returns extra primes:

```
sage: P.<x,y,z> = ProjectiveSpace(ZZ,2)
sage: H = Hom(P,P)
sage: f = H([3*x*y^2 + 7*y^3 - 4*y^2*z + 5*z^3, -5*x^3 + x^2*y + y^3 + 2*x^2*z, -2*x^2*y + x
sage: f.primes_of_bad_reduction(False)
[2, 5, 37, 2239, 304432717]
sage: f.primes_of_bad_reduction()
[5, 37, 2239, 304432717]
```

resultant (*normalize=False*)

Computes the resultant of the defining polynomials of `self` if `self` is a map in $\mathbb{P}^n$

If `normalize` is `True`, then first normalize the coordinate functions with `normalize_coordinates()`.

INPUT:

- `normalize` – Boolean (optional - default: `False`)

OUTPUT:

- an element of `self.codomain().base_ring()`

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,6*y^2])
sage: f.resultant()
36

sage: R.<t> = PolynomialRing(GF(17))
sage: P.<x,y> = ProjectiveSpace(R,1)
sage: H = Hom(P,P)
sage: f = H([t*x^2+t*y^2,6*y^2])
sage: f.resultant()
2*t^2

sage: R.<t> = PolynomialRing(GF(17))
sage: P.<x,y,z> = ProjectiveSpace(R,2)
sage: H = Hom(P,P)
sage: f = H([t*x^2+t*y^2,6*y^2,2*t*z^2])
sage: f.resultant()
13*t^8

sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = Hom(P,P)
sage: F = H([x^2+y^2,6*y^2,10*x*z+z^2+y^2])
sage: F.resultant()
1296

sage: R.<t>=PolynomialRing(QQ)
sage: s = (t^3+t+1).roots(QQbar)[0][0]
sage: P.<x,y>=ProjectiveSpace(QQbar,1)
sage: H = Hom(P,P)
sage: f = H([s*x^3-13*y^3,y^3-15*y^3])
```

```
sage: f.resultant()
871.6925062959149?
```

scale_by(*t*)

Scales each coordinates by a factor of t.

A TypeError occurs if the point is not in the coordinate_ring of the parent after scaling.

INPUT:

- t – a ring element

OUTPUT:

- None.

EXAMPLES:

```
sage: A.<x,y> = ProjectiveSpace(QQ,1)
sage: H = Hom(A,A)
sage: f = H([x^3-2*x*y^2,x^2*y])
sage: f.scale_by(1/x)
sage: f
Scheme endomorphism of Projective Space of dimension 1 over Rational
Field
  Defn: Defined on coordinates by sending (x : y) to
        (x^2 - 2*y^2 : x*y)
```

```
sage: R.<t> = PolynomialRing(QQ)
sage: P.<x,y> = ProjectiveSpace(R,1)
sage: H = Hom(P,P)
sage: f = H([3/5*x^2,6*y^2])
sage: f.scale_by(5/3*t); f
Scheme endomorphism of Projective Space of dimension 1 over Univariate
Polynomial Ring in t over Rational Field
  Defn: Defined on coordinates by sending (x : y) to
        (t*x^2 : 10*t*y^2)
```

```
sage: P.<x,y,z> = ProjectiveSpace(GF(7),2)
sage: X = P.subscheme(x^2-y^2)
sage: H = Hom(X,X)
sage: f = H([x^2,y^2,z^2])
sage: f.scale_by(x-y);f
Scheme endomorphism of Closed subscheme of Projective Space of dimension
2 over Finite Field of size 7 defined by:
  x^2 - y^2
  Defn: Defined on coordinates by sending (x : y : z) to
        (x*y^2 - y^3 : x*y^2 - y^3 : x*z^2 - y*z^2)
```

sigma_invariants(*n*, *formal=True*)

Computes the values of the elementary symmetric polynomials of the formal n multilpier spectra of self. Can specify to instead compute the values corresponding to the elementary symmetric polynomials of the n multiplier spectra of self, which include the multipliers of all periodic points of period n of self. $self$‘ must be defined over projective space of dimension 1 over a number field.

INPUT:

- n - a positive integer, the period.
- **formal - a Boolean. True specifies to find the values of the elementary symmetric polynomials** corresponding to the formal n multiplier spectra of self. False specifies to instead find the

values corresponding to the n multiplier spectra of self, which includes the multipliers of all periodic points of period n of self. Default: True

OUTPUT:

- a list of QQbar elements

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([512*x^5 - 378128*x^4*y + 76594292*x^3*y^2 - 4570550136*x^2*y^3 - 2630045017*x*y
+ 28193217129*y^5,512*y^5])
sage: f.sigma_invariants(1)
[19575526074450617/1048576, -9078122048145044298567432325/2147483648,
-2622661114909099878224381377917540931367/1099511627776,
-2622661107937102104196133701280271632423/549755813888,
338523204830161116503153209450763500631714178825448006778305/72057594037927936, 0]

sage: set_verbose(None)
sage: z = QQ['z'].0
sage: K = NumberField(z^4 - 4*z^2 + 1,'z')
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = End(P)
sage: f = H([x^2 - 5/4*y^2,y^2])
sage: f.sigma_invariants(2,False)
[13.000000000000000?, 11.000000000000000?, -25.000000000000000?, 0]
```

wronskian_ideal()

Returns the ideal generated by the critical point locus. This is the vanishing of the maximal minors of the jacobian matrix. Not implemented for subvarieties.

OUTPUT: an ideal in self.domain().coordinate_ring()

Examples:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<w> = NumberField(x^2+11)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = End(P)
sage: f = H([x^2-w*y^2,w*y^2])
sage: f.wronskian_ideal()
Ideal ((4*w)*x*y) of Multivariate Polynomial Ring in x, y over Number
Field in w with defining polynomial x^2 + 11

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: P2.<u,v,t> = ProjectiveSpace(K,2)
sage: H = Hom(P,P2)
sage: f = H([x^2-2*y^2,y^2,x*y])
sage: f.wronskian_ideal()
Ideal (4*x*y, 2*x^2 + 4*y^2, -2*y^2) of Multivariate Polynomial Ring in
x, y over Rational Field
```

class sage.schemes.projective.projective_morphism.**SchemeMorphism_polynomial_projective_space_**

Bases: sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_s

The Python constructor.

See SchemeMorphism_polynomial for details.

EXAMPLES:

```
sage: P1.<x,y> = ProjectiveSpace(QQ,1)
sage: H = P1.Hom(P1)
sage: H([y,2*x])
Scheme endomorphism of Projective Space of dimension 1 over Rational Field
  Defn: Defined on coordinates by sending (x : y) to
        (y : 2*x)
```

all_rational_preimages(*points*)

Given a set of rational points in the domain of `self`, return all the rational pre-images of those points. In others words, all the rational points which have some iterate in the set points. This function repeatedly calls `rational_preimages`. If the degree is at least two, by Northocott, this is always a finite set. `self` must be defined over number fields and be an endomorphism.

INPUT:

- `points` - a list of rational points in the domain of `self`

OUTPUT:

- a list of rational points in the domain of `self`.

Examples:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([16*x^2 - 29*y^2,16*y^2])
sage: sorted(f.all_rational_preimages([P(-1,4)]))
[(-7/4 : 1), (-5/4 : 1), (-3/4 : 1), (-1/4 : 1), (1/4 : 1), (3/4 : 1),
(5/4 : 1), (7/4 : 1)]
```

```
sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = End(P)
sage: f = H([76*x^2 - 180*x*y + 45*y^2 + 14*x*z + 45*y*z - 90*z^2, 67*x^2 - 180*x*y - 157*x*
sage: sorted(f.all_rational_preimages([P(-9,-4,1)]))
[(-9 : -4 : 1), (0 : -1 : 1), (0 : 0 : 1), (0 : 1 : 1), (0 : 4 : 1), (1
: 0 : 1), (1 : 1 : 1), (1 : 2 : 1), (1 : 3 : 1)]
```

A non-periodic example

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2 + y^2,2*x*y])
sage: sorted(f.all_rational_preimages([P(17,15)]))
[(1/3 : 1), (3/5 : 1), (5/3 : 1), (3 : 1)]
```

A number field example.:

```
sage: z = QQ['z'].0
sage: K.<w> = NumberField(z^3 + (z^2)/4 - (41/16)*z + 23/64);
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = End(P)
sage: f = H([16*x^2 - 29*y^2,16*y^2])
sage: f.all_rational_preimages([P(16*w^2 - 29,16)])
[(-w^2 + 21/16 : 1), (-w^2 - w + 33/16 : 1), (w + 1/2 : 1), (-w^2 - w +
25/16 : 1), (w^2 - 29/16 : 1), (w^2 - 21/16 : 1), (w^2 + w - 25/16 : 1),
(-w - 1/2 : 1), (w : 1), (-w : 1), (-w^2 + 29/16 : 1), (w^2 + w - 33/16
: 1)]
```

connected_rational_component(*P*, *n=0*)

Computes the connected component of a rational preperiodic point P of `self`. Will work for non-preperiodic points if n is positive. Otherwise this will not terminate.

INPUT:

- `P` - A rational preperiodic point of `self`
- `n` - Maximum distance from P to branch out. A value of 0 indicates no bound. Default: 0

OUTPUT:

- a list of points connected to P up to the specified distance

Examples:

```
sage: R.<x>=PolynomialRing(QQ)
sage: K.<w>= NumberField(x^3+1/4*x^2-41/16*x+23/64)
sage: PS.<x,y> = ProjectiveSpace(1,K)
sage: H = End(PS)
sage: f = H([x^2 - 29/16*y^2,y^2])
sage: P = PS([w,1])
sage: f.connected_rational_component(P)
[(w : 1), (w^2 - 29/16 : 1), (-w^2 - w + 25/16 : 1), (w^2 + w - 25/16 : 1),
(-w : 1), (-w^2 + 29/16 : 1), (-w - 1/2 : 1), (w + 1/2 : 1), (w^2 - 21/16 : 1),
(-w^2 + 21/16 : 1), (-w^2 - w + 33/16 : 1), (w^2 + w - 33/16 : 1)]
```

```
sage: PS.<x,y,z> = ProjectiveSpace(2,QQ)
sage: H = End(PS)
sage: f = H([x^2 - 21/16*z^2,y^2-2*z^2,z^2])
sage: P = PS([17/16,7/4,1])
sage: f.connected_rational_component(P,3)
[(17/16 : 7/4 : 1), (-47/256 : 17/16 : 1), (-83807/65536 : -223/256 : 1), (17/16 : -7/4 : 1)
(-17/16 : -7/4 : 1), (-17/16 : 7/4 : 1), (1386468673/4294967296 : -81343/65536 : 1),
(47/256 : -17/16 : 1), (47/256 : 17/16 : 1), (-47/256 : -17/16 : 1), (-1/2 : -1/2 : 1),
(-1/2 : 1/2 : 1), (1/2 : 1/2 : 1), (1/2 : -1/2 : 1)]
```

lift_to_rational_periodic(*points_modp*, *B=None*)

Given a list of points in projective space over $GF(p)$, determine if they lift to $\mathbf{Q}$-rational periodic points. `self` must be an endomorphism of projective space defined over $\mathbf{Q}$

ALGORITHM: Use Hensel lifting to find a p-adic approximation for that rational point. The accuracy needed is determined by the height bound B. Then apply the the LLL algorithm to determine if the lift corresponds to a rational point.

If the point is a point of high multiplicity (multiplier 1) then procedure can be very slow.

INPUT:

- `points_modp` - a list or tuple of pairs containing a point in projective space over $GF(p)$ and the possible period.
- `B` - a positive integer - the height bound for a rational preperiodic point. (optional)

OUTPUT:

- a list of projective points.

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2 - y^2,y^2])
sage: f.lift_to_rational_periodic([[P(0,1).change_ring(GF(7)),4]])
[[(0 : 1), 2]]
```

```
There may be multiple points in the lift.
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([-5*x^2 + 4*y^2,4*x*y])
sage: f.lift_to_rational_periodic([[P(1,0).change_ring(GF(3)),1]]) # long time
[[(1 : 0), 1], [(2/3 : 1), 1], [(-2/3 : 1), 1]]
```

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([16*x^2 - 29*y^2,16*y^2])
sage: f.lift_to_rational_periodic([[P(3,1).change_ring(GF(13)), 3]])
[[(-1/4 : 1), 3]]
```

```
sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = End(P)
sage: f = H([76*x^2 - 180*x*y + 45*y^2 + 14*x*z + 45*y*z - 90*z^2, 67*x^2 - 180*x*y - 157*x*
sage: f.lift_to_rational_periodic([[P(14,19,1).change_ring(GF(23)), 9]]) # long time
[[(-9 : -4 : 1), 9]]
```

rational_periodic_points(***kwds*)

Determine the set of rational periodic points for self an endomorphism of projective space. Must be defined over $\mathbf{Q}$.

The default parameter values are typically good choices for $\mathbb{P}^1$. If you are having trouble getting a particular map to finish, try first computing the possible periods, then try various different `lifting_prime`.

ALGORITHM: Modulo each prime of good reduction p determine the set of periodic points modulo p. For each cycle modulo p compute the set of possible periods (mrp^e). Take the intersection of the list of possible periods modulo several primes of good reduction to get a possible list of minimal periods of rational periodic points. Take each point modulo p associated to each of these possible periods and try to lift it to a rational point with a combination of p-adic approximation and the LLL basis reducion algorithm.

See B. Hutz, Determination of all rational preperiodic points for morphisms of Pn, submitted, 2012.

INPUT:

kwds:

- **`prime_bound` - a pair (list or tuple) of positive integers that represent the** limits of primes to use in the reduction step. Or an integer that represents the upper bound. (optional) default: [1,20]
- **`lifting_prime` - a prime integer. (optional) argument that specifies modulo which prime to try and perform t** lifting. default: 23
- `periods` - a list of positive integers which is the list of possible periods. (optional)
- `bad_primes` - a list or tuple of integer primes, the primes of bad reduction. (optional)

OUTPUT:

- a list of rational points in projective space.

Examples:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2-3/4*y^2,y^2])
sage: sorted(f.rational_periodic_points(prime_bound=20,lifting_prime=7)) # long time
[(-1/2 : 1), (1 : 0), (3/2 : 1)]
```

```
sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = End(P)
sage: f = H([2*x^3 - 50*x*z^2 + 24*z^3,5*y^3 - 53*y*z^2 + 24*z^3,24*z^3])
sage: sorted(f.rational_periodic_points(prime_bound=[1,20])) # long time
[(-3 : -1 : 1), (-3 : 0 : 1), (-3 : 1 : 1), (-3 : 3 : 1), (-1 : -1 : 1),
(-1 : 0 : 1), (-1 : 1 : 1), (-1 : 3 : 1), (0 : 1 : 0), (1 : -1 : 1), (1
: 0 : 0), (1 : 0 : 1), (1 : 1 : 1), (1 : 3 : 1), (3 : -1 : 1), (3 : 0 :
1), (3 : 1 : 1), (3 : 3 : 1), (5 : -1 : 1), (5 : 0 : 1), (5 : 1 : 1), (5
: 3 : 1)]

sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([-5*x^2 + 4*y^2,4*x*y])
sage: sorted(f.rational_periodic_points()) # long time
[(-2 : 1), (-2/3 : 1), (2/3 : 1), (1 : 0), (2 : 1)]
```

Todo

- move some of this to Cython so that it is faster especially the possible periods mod p.
- have the last prime of good redution used also return the list of points instead of getting the information again for all_points.

rational_preimages(Q)

Given a rational point Q in the domain of `self`, return all the rational points P in the domain of `self` with $self(P) == Q$. In other words, the set of first pre-images of Q. `self` must be defined over number fields and be an endomorphism.

ALGORITHM: Use elimination via groebner bases to find the rational pre-images

INPUT:

- `Q` - a rational point in the domain of `self`.

OUTPUT:

- a list of rational points in the domain of `self`.

Examples:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([16*x^2 - 29*y^2,16*y^2])
sage: f.rational_preimages(P(-1,4))
[(5/4 : 1), (-5/4 : 1)]

sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: H = End(P)
sage: f = H([76*x^2 - 180*x*y + 45*y^2 + 14*x*z + 45*y*z - 90*z^2, 67*x^2 - 180*x*y - 157*x*
sage: f.rational_preimages(P(-9,-4,1))
[(0 : 4 : 1)]
```

A non-periodic example

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2 + y^2,2*x*y])
sage: f.rational_preimages(P(17,15))
[(5/3 : 1), (3/5 : 1)]
```

```
sage: P.<x,y,z,w> = ProjectiveSpace(QQ,3)
sage: H = End(P)
sage: f = H([x^2 - 2*y*w - 3*w^2, -2*x^2 + y^2 - 2*x*z + 4*y*w + 3*w^2, x^2 - y^2 + 2*x*z +
sage: f.rational_preimages(P(0,-1,0,1))
[]
```

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([x^2 + y^2,2*x*y])
sage: f.rational_preimages([CC.0,1])
Traceback (most recent call last):
...
TypeError: Point must be in codomain of self
```

A number field example

```
sage: z = QQ['z'].0
sage: K.<a> = NumberField(z^2 - 2);
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = End(P)
sage: f = H([x^2 + y^2,y^2])
sage: f.rational_preimages(P(3,1))
[(a : 1), (-a : 1)]
```

```
sage: z = QQ['z'].0
sage: K.<a> = NumberField(z^2 - 2);
sage: P.<x,y,z> = ProjectiveSpace(K,2)
sage: X = P.subscheme([x^2 - z^2])
sage: H = Hom(X,X)
sage: f= H([x^2 - z^2, a*y^2, z^2 - x^2])
sage: f.rational_preimages(X([1,2,-1]))
[]
```

```
sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: X = P.subscheme([x^2 - z^2])
sage: H = Hom(X,X)
sage: f= H([x^2-z^2, y^2, z^2-x^2])
sage: f.rational_preimages(X([0,1,0]))
Traceback (most recent call last):
...
NotImplementedError: Subschemes as Preimages not implemented
```

rational_preperiodic_graph(***kwds*)

Determine the directed graph of the rational preperiodic points for `self`. self must be defined over $\mathbf{Q}$ and be an endomorphism of projective space. If `self` is a polynomial endomorphism of $\mathbb{P}^1$, i.e. has a totally ramified fixed point, then the base ring can also be an absolute number field. This is done by passing to the Weil restriction.

ALGORITHM: - Determines the list of possible periods.

- Determines the rational periodic points from the possible periods.
- Determines the rational preperiodic points from the rational periodic points by determining rational preimages.

INPUT:

kwds:

- **prime_bound - a pair (list or tuple) of positive integers that represent the** limits of primes to use in the reduction step. Or an integer that represents the upper bound. (optional) default: [1,20]

- **lifting_prime - a prime integer. (optional) argument that specifies modulo which prime to try and perform the** lifting. default: 23
- periods - a list of positive integers which is the list of possible periods. (optional)
- bad_primes - a list or tuple of integer primes, the primes of bad reduction. (optional)

OUTPUT:

- a digraph representing the orbits of the rational preperiodic points in projective space.

Examples:

```
sage: PS.<x,y> = ProjectiveSpace(1,QQ)
sage: H = End(PS)
sage: f = H([7*x^2 - 28*y^2,24*x*y])
sage: f.rational_preperiodic_graph()
Looped digraph on 12 vertices
```

```
sage: PS.<x,y> = ProjectiveSpace(1,QQ)
sage: H = End(PS)
sage: f = H([-3/2*x^3 +19/6*x*y^2,y^3])
sage: f.rational_preperiodic_graph(prime_bound=[1,8])
Looped digraph on 12 vertices
```

```
sage: PS.<x,y,z> = ProjectiveSpace(2,QQ)
sage: H = End(PS)
sage: f = H([2*x^3 - 50*x*z^2 + 24*z^3,5*y^3 - 53*y*z^2 + 24*z^3,24*z^3])
sage: f.rational_preperiodic_graph(prime_bound=[1,11],lifting_prime=13) # long time
Looped digraph on 30 vertices
```

```
sage: K.<w> = QuadraticField(-3)
sage: P.<x,y> = ProjectiveSpace(K,1)
sage: H = End(P)
sage: f = H([x^2+y^2,y^2])
sage: f.rational_preperiodic_graph() # long time
Looped digraph on 5 vertices
```

rational_preperiodic_points(***kwds*)

Determine the set of rational preperiodic points for self. self must be defined over $\mathbf{Q}$ and be an endomorphism of projective space. If self is a polynomial endomorphism of $\mathbb{P}^1$, i.e. has a totally ramified fixed point, then the base ring can also be an absolute number field. This is done by passing to the Weil restriction.

The default parameter values are typically good choices for $\mathbb{P}^1$. If you are having trouble getting a particular map to finish, try first computing the possible periods, then try various different values for lifting_prime.

ALGORITHM:

- Determines the list of possible periods.
- Determines the rational periodic points from the possible periods.
- Determines the rational preperiodic points from the rational periodic points by determining rational preimages.

INPUT:

kwds:

- `prime_bound` - a pair (list or tuple) of positive integers that represent the limits of primes to use in the reduction step. Or an integer that represents the upper bound. (optional) default: [1,20]
- `lifting_prime` - a prime integer. (optional) argument that specifies modulo which prime to try and perform the lifting. default: 23
- `periods` - a list of positive integers which is the list of possible periods. (optional)
- `bad_primes` - a list or tuple of integer primes, the primes of bad reduction. (optional)

OUTPUT:

- a list of rational points in projective space.

Examples:

```
sage: PS.<x,y> = ProjectiveSpace(1,QQ)
sage: H = End(PS)
sage: f = H([x^2 -y^2,3*x*y])
sage: sorted(f.rational_preperiodic_points())
[(-2 : 1), (-1 : 1), (-1/2 : 1), (0 : 1), (1/2 : 1), (1 : 0), (1 : 1),
(2 : 1)]

sage: PS.<x,y> = ProjectiveSpace(1,QQ)
sage: H = End(PS)
sage: f = H([5*x^3 - 53*x*y^2 + 24*y^3, 24*y^3])
sage: sorted(f.rational_preperiodic_points(prime_bound=10))
[(-1 : 1), (0 : 1), (1 : 0), (1 : 1), (3 : 1)]

sage: PS.<x,y,z> = ProjectiveSpace(2,QQ)
sage: H = End(PS)
sage: f = H([x^2 - 21/16*z^2,y^2-2*z^2,z^2])
sage: sorted(f.rational_preperiodic_points(prime_bound=[1,8],lifting_prime=7,periods=[2])) #
[(-5/4 : -2 : 1), (-5/4 : -1 : 1), (-5/4 : 0 : 1), (-5/4 : 1 : 1), (-5/4
: 2 : 1), (-1/4 : -2 : 1), (-1/4 : -1 : 1), (-1/4 : 0 : 1), (-1/4 : 1 :
1), (-1/4 : 2 : 1), (1/4 : -2 : 1), (1/4 : -1 : 1), (1/4 : 0 : 1), (1/4
: 1 : 1), (1/4 : 2 : 1), (5/4 : -2 : 1), (5/4 : -1 : 1), (5/4 : 0 : 1),
(5/4 : 1 : 1), (5/4 : 2 : 1)]

sage: K.<w> = QuadraticField(33)
sage: PS.<x,y> = ProjectiveSpace(K,1)
sage: H = End(PS)
sage: f = H([x^2-71/48*y^2, y^2])
sage: sorted(f.rational_preperiodic_points()) # long time
[(-1/12*w - 1 : 1),
 (-1/6*w - 1/4 : 1),
 (-1/12*w - 1/2 : 1),
 (-1/6*w + 1/4 : 1),
 (1/12*w - 1 : 1),
 (1/12*w - 1/2 : 1),
 (-1/12*w + 1/2 : 1),
 (-1/12*w + 1 : 1),
 (1/6*w - 1/4 : 1),
 (1/12*w + 1/2 : 1),
 (1 : 0),
 (1/6*w + 1/4 : 1),
 (1/12*w + 1 : 1)]
```

class `sage.schemes.projective.projective_morphism.`**`SchemeMorphism_polynomial_projective_space_`**

Bases: sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_s

The Python constructor.

See SchemeMorphism_polynomial for details.

EXAMPLES:

```
sage: P1.<x,y> = ProjectiveSpace(QQ,1)
sage: H = P1.Hom(P1)
sage: H([y,2*x])
Scheme endomorphism of Projective Space of dimension 1 over Rational Field
  Defn: Defined on coordinates by sending (x : y) to
        (y : 2*x)
```

automorphism_group(***kwds*)

Given a homogenous rational function, this calculates the subsgroup of $PGL2$ that is the automorphism group of self, see [FMV] fir algorithm.

INPUT:

keywords:

- absolute– Boolean - True returns the absolute automorphism group and a field of definition. default: False (optional)
- iso_type – Boolean - True returns the isomorphism type of the automorphism group. default: False (optional)
- **return_functions– Boolean - True returns elements as linear fractional transformations.** False returns elements as $PGL2$ matrices. default: False. (optional)

OUTPUT:

- list - elements of automorphism group.

AUTHORS:

- Original algorithm written by Xander Faber, Michelle Manes, Bianca Viray
- Modified by Joao Alberto de Faria, Ben Hutz, Bianca Thompson

EXAMPLES:

```
sage: R.<x,y> = ProjectiveSpace(GF(7^3,'t'),1)
sage: H = End(R)
sage: f = H([x^2-y^2,x*y])
sage: f.automorphism_group()
[
[1 0]  [6 0]
[0 1], [0 1]
]

sage: R.<x,y> = ProjectiveSpace(GF(3^2,'t'),1)
sage: H = End(R)
sage: f = H([x^3,y^3])
sage: f.automorphism_group(return_functions=True,iso_type=True) # long time
([x, x/(x + 1), x/(2*x + 1), 2/(x + 2), (2*x + 1)/(2*x), (2*x + 2)/x,
1/(2*x + 2), x + 1, x + 2, x/(x + 2), 2*x/(x + 1), 2*x, 1/x, 2*x + 1,
2*x + 2, ((t + 2)*x + t + 2)/((2*t + 1)*x + t + 2), (t*x + 2*t)/(t*x +
t), 2/x, (x + 1)/(x + 2), (2*t*x + t)/(t*x), (2*t + 1)/((2*t + 1)*x +
2*t + 1), ((2*t + 1)*x + 2*t + 1)/((2*t + 1)*x), t/(t*x + 2*t), (2*x +
1)/(x + 1)], 'PGL(2,3)')
```

```
sage: R.<x,y> = ProjectiveSpace(GF(2^5,'t'),1)
sage: H = End(R)
sage: f=H([x^5,y^5])
sage: f.automorphism_group(return_functions=True,iso_type=True)
([x, 1/x], 'Cyclic of order 2')

::

sage: R.<x,y> = ProjectiveSpace(GF(3^4,'t'),1)
sage: H = End(R)
sage: f=H([x^2+25*x*y+y^2,x*y+3*y^2])
sage: f.automorphism_group(absolute=True)
[Univariate Polynomial Ring in w over Finite Field in b of size 3^4,
 [
[1 0]
[0 1]
]]
```

cyclegraph()

returns Digraph of all orbits of `self` mod p.

For subschemes, only points on the subscheme whose image are also on the subscheme are in the digraph.

OUTPUT:

- a digraph

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(GF(13),1)
sage: H = Hom(P,P)
sage: f = H([x^2-y^2,y^2])
sage: f.cyclegraph()
Looped digraph on 14 vertices

sage: P.<x,y,z> = ProjectiveSpace(GF(5^2,'t'),2)
sage: H = Hom(P,P)
sage: f = H([x^2+y^2,y^2,z^2+y*z])
sage: f.cyclegraph()
Looped digraph on 651 vertices

sage: P.<x,y,z> = ProjectiveSpace(GF(7),2)
sage: X = P.subscheme(x^2-y^2)
sage: H = Hom(X,X)
sage: f = H([x^2,y^2,z^2])
sage: f.cyclegraph()
Looped digraph on 15 vertices
```

orbit_structure(*P*)

Every point is preperiodic over a finite field. This function returns the pair $[m,n]$ where m is the preperiod and n is the period of the point `P` by `self`.

INPUT:

- `P` – a point in `self.domain()`

OUTPUT:

- a list $[m,n]$ of integers

EXAMPLES:

```
sage: P.<x,y,z> = ProjectiveSpace(GF(5),2)
sage: H = Hom(P,P)
sage: f = H([x^2 + y^2,y^2,z^2 + y*z])
sage: f.orbit_structure(P(2,1,2))
[0, 6]

sage: P.<x,y,z> = ProjectiveSpace(GF(7),2)
sage: X = P.subscheme(x^2-y^2)
sage: H = Hom(X,X)
sage: f = H([x^2,y^2,z^2])
sage: f.orbit_structure(X(1,1,2))
[0, 2]

sage: P.<x,y> = ProjectiveSpace(GF(13),1)
sage: H = Hom(P,P)
sage: f = H([x^2 - y^2,y^2])
sage: f.orbit_structure(P(3,4))
[2, 3]
```

possible_periods(*return_points=False*)

Returns the list of possible minimal periods of a periodic point over $\mathbf{Q}$ and (optionally) a point in each cycle.

ALGORITHM:

The list comes from: Hutz, Good reduction of periodic points, Illinois Journal of Mathematics 53 (Winter 2009), no. 4, 1109-1126.

INPUT:

- `return_points` - Boolean (optional) - a value of True returns the points as well as the possible periods.

OUTPUT:

- a list of positive integers, or a list of pairs of projective points and periods if `flag` is 1.

Examples:

```
sage: P.<x,y> = ProjectiveSpace(GF(23),1)
sage: H = End(P)
sage: f = H([x^2-2*y^2,y^2])
sage: f.possible_periods()
[1, 5, 11, 22, 110]

sage: P.<x,y> = ProjectiveSpace(GF(13),1)
sage: H = End(P)
sage: f = H([x^2-y^2,y^2])
sage: sorted(f.possible_periods(True))
[[(0 : 1), 2], [(1 : 0), 1], [(3 : 1), 3], [(3 : 1), 36]]

sage: PS.<x,y,z> = ProjectiveSpace(2,GF(7))
sage: H = End(PS)
sage: f = H([-360*x^3 + 760*x*z^2, y^3 - 604*y*z^2 + 240*z^3, 240*z^3])
sage: f.possible_periods()
[1, 2, 4, 6, 12, 14, 28, 42, 84]
```

Todo

- do not return duplicate points

•improve hash to reduce memory of pointtable

14.4 Morphisms on projective varieties (Cython helper)

This is the helper file providing functionality for projective_morphism.py.

AUTHORS:

- Dillon Rose (2014-01): Speed enhancements

14.5 Enumeration of rational points on projective schemes

Naive algorithms for enumerating rational points over **Q** or finite fields over for general schemes.

Warning: Incorrect results and infinite loops may occur if using a wrong function. (For instance using an affine function for a projective scheme or a finite field function for a scheme defined over an infinite field.)

EXAMPLES:

Projective, over **Q**:

```
sage: from sage.schemes.projective.projective_rational_point import enum_projective_rational_field
sage: P.<X,Y,Z> = ProjectiveSpace(2,QQ)
sage: C = P.subscheme([X+Y-Z])
sage: enum_projective_rational_field(C,3)
[(-2 : 3 : 1), (-1 : 1 : 0), (-1 : 2 : 1), (-1/2 : 3/2 : 1),
 (0 : 1 : 1), (1/3 : 2/3 : 1), (1/2 : 1/2 : 1), (2/3 : 1/3 : 1),
 (1 : 0 : 1), (3/2 : -1/2 : 1), (2 : -1 : 1), (3 : -2 : 1)]
```

Projective over a finite field:

```
sage: from sage.schemes.projective.projective_rational_point import enum_projective_finite_field
sage: E = EllipticCurve('72').change_ring(GF(19))
sage: enum_projective_finite_field(E)
[(0 : 1 : 0), (1 : 0 : 1), (3 : 0 : 1), (4 : 9 : 1), (4 : 10 : 1),
 (6 : 6 : 1), (6 : 13 : 1), (7 : 6 : 1), (7 : 13 : 1), (9 : 4 : 1),
 (9 : 15 : 1), (12 : 8 : 1), (12 : 11 : 1), (13 : 8 : 1), (13 : 11 : 1),
 (14 : 3 : 1), (14 : 16 : 1), (15 : 0 : 1), (16 : 9 : 1), (16 : 10 : 1),
 (17 : 7 : 1), (17 : 12 : 1), (18 : 9 : 1), (18 : 10 : 1)]
```

AUTHORS:

- David R. Kohel <kohel@maths.usyd.edu.au>: original version.
- John Cremona and Charlie Turner <charlotteturner@gmail.com> (06-2010): improvements to clarity and documentation.

`sage.schemes.projective.projective_rational_point.`**`enum_projective_finite_field`**(*X*)

Enumerates projective points on scheme `X` defined over a finite field.

INPUT:

•`X` - a scheme defined over a finite field or a set of abstract rational points of such a scheme.

OUTPUT:

- a list containing the projective points of X over the finite field, sorted.

EXAMPLES:

```
sage: F = GF(53)
sage: P.<X,Y,Z> = ProjectiveSpace(2,F)
sage: from sage.schemes.projective.projective_rational_point import enum_projective_finite_field
sage: len(enum_projective_finite_field(P(F)))
2863
sage: 53^2+53+1
2863
```

```
sage: F = GF(9,'a')
sage: P.<X,Y,Z> = ProjectiveSpace(2,F)
sage: C = Curve(X^3-Y^3+Z^2*Y)
sage: enum_projective_finite_field(C(F))
[(0 : 0 : 1), (0 : 1 : 1), (0 : 2 : 1), (1 : 1 : 0), (a + 1 : 2*a : 1),
(a + 1 : 2*a + 1 : 1), (a + 1 : 2*a + 2 : 1), (2*a + 2 : a : 1),
(2*a + 2 : a + 1 : 1), (2*a + 2 : a + 2 : 1)]
```

```
sage: F = GF(5)
sage: P2F.<X,Y,Z> = ProjectiveSpace(2,F)
sage: enum_projective_finite_field(P2F)
[(0 : 0 : 1), (0 : 1 : 0), (0 : 1 : 1), (0 : 2 : 1), (0 : 3 : 1), (0 : 4 : 1),
(1 : 0 : 0), (1 : 0 : 1), (1 : 1 : 0), (1 : 1 : 1), (1 : 2 : 1), (1 : 3 : 1),
(1 : 4 : 1), (2 : 0 : 1), (2 : 1 : 0), (2 : 1 : 1), (2 : 2 : 1), (2 : 3 : 1),
(2 : 4 : 1), (3 : 0 : 1), (3 : 1 : 0), (3 : 1 : 1), (3 : 2 : 1), (3 : 3 : 1),
(3 : 4 : 1), (4 : 0 : 1), (4 : 1 : 0), (4 : 1 : 1), (4 : 2 : 1), (4 : 3 : 1),
(4 : 4 : 1)]
```

ALGORITHM:

Checks all points in projective space to see if they lie on X.

> **Warning:** If X is defined over an infinite field, this code will not finish!

AUTHORS:

- John Cremona and Charlie Turner (06-2010).

sage.schemes.projective.projective_rational_point.**enum_projective_number_field**(*X*, *B*, *prec=53*)

Enumerates projective points on scheme X defined over a number field. Simply checks all of the points of absolute height of at most B and adds those that are on the scheme to the list.

INPUT:

- X - a scheme defined over a number field
- B - a real number
- prec - the precision to use for computing the elements of bounded height of number fields

OUTPUT:

- a list containing the projective points of X of absolute height up to B, sorted.

Warning: In the current implementation, the output of the [Doyle-Krumm] algorithm for elements of bounded height cannot be guaranteed to be correct due to the necessity of floating point computations. In some cases, the default 53-bit precision is considerably lower than would be required for the algorithm to generate correct output.

EXAMPLES:

```
sage: from sage.schemes.projective.projective_rational_point import enum_projective_number_field
sage: u = QQ['u'].0
sage: K = NumberField(u^3 - 5,'v')
sage: P.<x,y,z> = ProjectiveSpace(K, 2)
sage: X = P.subscheme([x - y])
sage: enum_projective_number_field(X(K), 5^(1/3), prec=2^10)
[(0 : 0 : 1), (-1 : -1 : 1), (1 : 1 : 1), (-1/5*v^2 : -1/5*v^2 : 1), (-v : -v : 1),
(1/5*v^2 : 1/5*v^2 : 1), (v : v : 1), (1 : 1 : 0)]
```

```
sage: u = QQ['u'].0
sage: K = NumberField(u^2 + 3, 'v')
sage: A.<x,y> = ProjectiveSpace(K,1)
sage: X=A.subscheme(x-y)
sage: from sage.schemes.projective.projective_rational_point import enum_projective_number_field
sage: enum_projective_number_field(X, 2)
[(1 : 1)]
```

`sage.schemes.projective.projective_rational_point.`**`enum_projective_rational_field`**(*X*, *B*)

Enumerates projective, rational points on scheme `X` of height up to bound `B`.

INPUT:

- `X` - a scheme or set of abstract rational points of a scheme;
- `B` - a positive integer bound.

OUTPUT:

- a list containing the projective points of `X` of height up to `B`, sorted.

EXAMPLES:

```
sage: P.<X,Y,Z> = ProjectiveSpace(2,QQ)
sage: C = P.subscheme([X+Y-Z])
sage: from sage.schemes.projective.projective_rational_point import enum_projective_rational_field
sage: enum_projective_rational_field(C(QQ),6)
[(-5 : 6 : 1), (-4 : 5 : 1), (-3 : 4 : 1), (-2 : 3 : 1),
 (-3/2 : 5/2 : 1), (-1 : 1 : 0), (-1 : 2 : 1), (-2/3 : 5/3 : 1),
 (-1/2 : 3/2 : 1), (-1/3 : 4/3 : 1), (-1/4 : 5/4 : 1),
 (-1/5 : 6/5 : 1), (0 : 1 : 1), (1/6 : 5/6 : 1), (1/5 : 4/5 : 1),
 (1/4 : 3/4 : 1), (1/3 : 2/3 : 1), (2/5 : 3/5 : 1), (1/2 : 1/2 : 1),
 (3/5 : 2/5 : 1), (2/3 : 1/3 : 1), (3/4 : 1/4 : 1), (4/5 : 1/5 : 1),
 (5/6 : 1/6 : 1), (1 : 0 : 1), (6/5 : -1/5 : 1), (5/4 : -1/4 : 1),
 (4/3 : -1/3 : 1), (3/2 : -1/2 : 1), (5/3 : -2/3 : 1), (2 : -1 : 1),
 (5/2 : -3/2 : 1), (3 : -2 : 1), (4 : -3 : 1), (5 : -4 : 1),
 (6 : -5 : 1)]
sage: enum_projective_rational_field(C,6) == enum_projective_rational_field(C(QQ),6)
True
```

```
sage: P3.<W,X,Y,Z> = ProjectiveSpace(3,QQ)
sage: enum_projective_rational_field(P3,1)
[(-1 : -1 : -1 : 1), (-1 : -1 : 0 : 1), (-1 : -1 : 1 : 0), (-1 : -1 : 1 : 1),
(-1 : 0 : -1 : 1), (-1 : 0 : 0 : 1), (-1 : 0 : 1 : 0), (-1 : 0 : 1 : 1),
```

```
(-1 : 1 : -1 : 1), (-1 : 1 : 0 : 0), (-1 : 1 : 0 : 1), (-1 : 1 : 1 : 0),
(-1 : 1 : 1 : 1), (0 : -1 : -1 : 1), (0 : -1 : 0 : 1), (0 : -1 : 1 : 0),
(0 : -1 : 1 : 1), (0 : 0 : -1 : 1), (0 : 0 : 0 : 1), (0 : 0 : 1 : 0),
(0 : 0 : 1 : 1), (0 : 1 : -1 : 1), (0 : 1 : 0 : 0), (0 : 1 : 0 : 1),
(0 : 1 : 1 : 0), (0 : 1 : 1 : 1), (1 : -1 : -1 : 1), (1 : -1 : 0 : 1),
(1 : -1 : 1 : 0), (1 : -1 : 1 : 1), (1 : 0 : -1 : 1), (1 : 0 : 0 : 0),
(1 : 0 : 0 : 1), (1 : 0 : 1 : 0), (1 : 0 : 1 : 1), (1 : 1 : -1 : 1),
(1 : 1 : 0 : 0), (1 : 1 : 0 : 1), (1 : 1 : 1 : 0), (1 : 1 : 1 : 1)]
```

ALGORITHM:

We just check all possible projective points in correct dimension of projective space to see if they lie on `X`.

AUTHORS:

- John Cremona and Charlie Turner (06-2010)

14.6 Set of homomorphisms between two projective schemes

For schemes X and Y, this module implements the set of morphisms $Hom(X, Y)$. This is done by `SchemeHomset_generic`.

As a special case, the Hom-sets can also represent the points of a scheme. Recall that the K-rational points of a scheme X over k can be identified with the set of morphisms $Spec(K) \to X$. In Sage the rational points are implemented by such scheme morphisms. This is done by `SchemeHomset_points` and its subclasses.

Note: You should not create the Hom-sets manually. Instead, use the `Hom()` method that is inherited by all schemes.

AUTHORS:

- William Stein (2006): initial version.
- Volker Braun (2011-08-11): significant improvement and refactoring.
- Ben Hutz (June 2012): added support for projective ring

class `sage.schemes.projective.projective_homset.`**`SchemeHomset_points_abelian_variety_field`**(*X*, *Y*, *cat-e-gory*, *chec* *base* *Ring*

Bases: `sage.schemes.projective.projective_homset.SchemeHomset_points_projective_field`

Set of rational points of an abelian variety.

INPUT:

See `SchemeHomset_generic`.

TESTS:

The bug reported at trac #1785 is fixed:

```
sage: K.<a> = NumberField(x^2 + x - (3^3-3))
sage: E = EllipticCurve('37a')
sage: X = E(K)
sage: X
```

```
Abelian group of points on Elliptic Curve defined by
y^2 + y = x^3 + (-1)*x over Number Field in a with
defining polynomial x^2 + x - 24
sage: P = X([3,a])
sage: P
(3 : a : 1)
sage: P in E
False
sage: P in E.base_extend(K)
True
sage: P in X.codomain()
False
sage: P in X.extended_codomain()
True
```

base_extend(*R*)

Extend the base ring.

This is currently not implemented except for the trivial case R==ZZ.

INPUT:

- R – a ring.

EXAMPLES:

```
sage: E = EllipticCurve('37a')
sage: Hom = E.point_homset();  Hom
Abelian group of points on Elliptic Curve defined
by y^2 + y = x^3 - x over Rational Field
sage: Hom.base_ring()
Integer Ring
sage: Hom.base_extend(QQ)
Traceback (most recent call last):
...
NotImplementedError: Abelian variety point sets are not
implemented as modules over rings other than ZZ.
```

class sage.schemes.projective.projective_homset.**SchemeHomset_points_projective_field**(*X*, *Y*, *category=None*, *check=True*, *base=Integer Ring*)

Bases: sage.schemes.generic.homset.SchemeHomset_points

Set of rational points of a projective variety over a field.

INPUT:

See SchemeHomset_generic.

EXAMPLES:

```
sage: from sage.schemes.projective.projective_homset import SchemeHomset_points_projective_field
sage: SchemeHomset_points_projective_field(Spec(QQ), ProjectiveSpace(QQ,2))
Set of rational points of Projective Space of dimension 2 over Rational Field
```

points (*B=0, prec=53*)

Return some or all rational points of a projective scheme.

INPUT:

- B - integer (optional, default=0). The bound for the coordinates.
- `prec` - he precision to use to compute the elements of bounded height for number fields

OUTPUT:

A list of points. Over a finite field, all points are returned. Over an infinite field, all points satisfying the bound are returned.

> **Warning:** In the current implementation, the output of the [Doyle-Krumm] algorithm cannot be guaranteed to be correct due to the necessity of floating point computations. In some cases, the default 53-bit precision is considerably lower than would be required for the algorithm to generate correct output.

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: P(QQ).points(4)
[(-4 : 1), (-3 : 1), (-2 : 1), (-3/2 : 1), (-4/3 : 1), (-1 : 1),
(-3/4 : 1), (-2/3 : 1), (-1/2 : 1), (-1/3 : 1), (-1/4 : 1), (0 : 1),
(1/4 : 1), (1/3 : 1), (1/2 : 1), (2/3 : 1), (3/4 : 1), (1 : 0), (1 : 1),
(4/3 : 1), (3/2 : 1), (2 : 1), (3 : 1), (4 : 1)]

sage: u = QQ['u'].0
sage: K.<v> = NumberField(u^2 + 3)
sage: P.<x,y,z> = ProjectiveSpace(K,2)
sage: len(P(K).points(1.8))
381

sage: P1 = ProjectiveSpace(GF(2),1)
sage: F.<a> = GF(4,'a')
sage: P1(F).points()
[(0 : 1), (1 : 0), (1 : 1), (a : 1), (a + 1 : 1)]

sage: P.<x,y,z> = ProjectiveSpace(QQ,2)
sage: E = P.subscheme([(y^3-y*z^2) - (x^3-x*z^2),(y^3-y*z^2) + (x^3-x*z^2)])
sage: E(P.base_ring()).points()
[(-1 : -1 : 1), (-1 : 0 : 1), (-1 : 1 : 1), (0 : -1 : 1), (0 : 0 : 1), (0 : 1 : 1),
(1 : -1 : 1), (1 : 0 : 1), (1 : 1 : 1)]
```

class `sage.schemes.projective.projective_homset.`**SchemeHomset_points_projective_ring** (*X, Y, category=None, check=True, base=Integer Ring*)

Bases: `sage.schemes.generic.homset.SchemeHomset_points`

Set of rational points of a projective variety over a commutative ring.

INPUT:

See `SchemeHomset_generic`.

EXAMPLES:

```
sage: from sage.schemes.projective.projective_homset import SchemeHomset_points_projective_ring
sage: SchemeHomset_points_projective_ring(Spec(ZZ), ProjectiveSpace(ZZ,2))
Set of rational points of Projective Space of dimension 2 over Integer Ring
```

points $(B=0)$

Return some or all rational points of a projective scheme.

INPUT:

- B – integer (optional, default=0). The bound for the coordinates.

EXAMPLES:

```
sage: from sage.schemes.projective.projective_homset import SchemeHomset_points_projective_r
sage: H = SchemeHomset_points_projective_ring(Spec(ZZ), ProjectiveSpace(ZZ,2))
sage: H.points(3)
[(0 : 0 : 1), (0 : 1 : -3), (0 : 1 : -2), (0 : 1 : -1), (0 : 1 : 0), (0
: 1 : 1), (0 : 1 : 2), (0 : 1 : 3), (0 : 2 : -3), (0 : 2 : -1), (0 : 2 :
1), (0 : 2 : 3), (0 : 3 : -2), (0 : 3 : -1), (0 : 3 : 1), (0 : 3 : 2),
(1 : -3 : -3), (1 : -3 : -2), (1 : -3 : -1), (1 : -3 : 0), (1 : -3 : 1),
(1 : -3 : 2), (1 : -3 : 3), (1 : -2 : -3), (1 : -2 : -2), (1 : -2 : -1),
(1 : -2 : 0), (1 : -2 : 1), (1 : -2 : 2), (1 : -2 : 3), (1 : -1 : -3),
(1 : -1 : -2), (1 : -1 : -1), (1 : -1 : 0), (1 : -1 : 1), (1 : -1 : 2),
(1 : -1 : 3), (1 : 0 : -3), (1 : 0 : -2), (1 : 0 : -1), (1 : 0 : 0), (1
: 0 : 1), (1 : 0 : 2), (1 : 0 : 3), (1 : 1 : -3), (1 : 1 : -2), (1 : 1 :
-1), (1 : 1 : 0), (1 : 1 : 1), (1 : 1 : 2), (1 : 1 : 3), (1 : 2 : -3),
(1 : 2 : -2), (1 : 2 : -1), (1 : 2 : 0), (1 : 2 : 1), (1 : 2 : 2), (1 :
2 : 3), (1 : 3 : -3), (1 : 3 : -2), (1 : 3 : -1), (1 : 3 : 0), (1 : 3 :
1), (1 : 3 : 2), (1 : 3 : 3), (2 : -3 : -3), (2 : -3 : -2), (2 : -3 :
-1), (2 : -3 : 0), (2 : -3 : 1), (2 : -3 : 2), (2 : -3 : 3), (2 : -2 :
-3), (2 : -2 : -1), (2 : -2 : 1), (2 : -2 : 3), (2 : -1 : -3), (2 : -1 :
-2), (2 : -1 : -1), (2 : -1 : 0), (2 : -1 : 1), (2 : -1 : 2), (2 : -1 :
3), (2 : 0 : -3), (2 : 0 : -1), (2 : 0 : 1), (2 : 0 : 3), (2 : 1 : -3),
(2 : 1 : -2), (2 : 1 : -1), (2 : 1 : 0), (2 : 1 : 1), (2 : 1 : 2), (2 :
1 : 3), (2 : 2 : -3), (2 : 2 : -1), (2 : 2 : 1), (2 : 2 : 3), (2 : 3 :
-3), (2 : 3 : -2), (2 : 3 : -1), (2 : 3 : 0), (2 : 3 : 1), (2 : 3 : 2),
(2 : 3 : 3), (3 : -3 : -2), (3 : -3 : -1), (3 : -3 : 1), (3 : -3 : 2),
(3 : -2 : -3), (3 : -2 : -2), (3 : -2 : -1), (3 : -2 : 0), (3 : -2 : 1),
(3 : -2 : 2), (3 : -2 : 3), (3 : -1 : -3), (3 : -1 : -2), (3 : -1 : -1),
(3 : -1 : 0), (3 : -1 : 1), (3 : -1 : 2), (3 : -1 : 3), (3 : 0 : -2), (3
: 0 : -1), (3 : 0 : 1), (3 : 0 : 2), (3 : 1 : -3), (3 : 1 : -2), (3 : 1
: -1), (3 : 1 : 0), (3 : 1 : 1), (3 : 1 : 2), (3 : 1 : 3), (3 : 2 : -3),
(3 : 2 : -2), (3 : 2 : -1), (3 : 2 : 0), (3 : 2 : 1), (3 : 2 : 2), (3 :
2 : 3), (3 : 3 : -2), (3 : 3 : -1), (3 : 3 : 1), (3 : 3 : 2)]
```

14.7 Automorphism groups of endomorphisms of the projective line

AUTHORS:

- Xander Faber, Michelle Manes, Bianca Viray: algorithm and original code "Computing Conjugating Sets and Automorphism Groups of Rational Functions" by Xander Faber, Michelle Manes, and Bianca Viray [FMV]
- Joao de Faria, Ben Hutz, Bianca Thompson (11-2013): adaption for inclusion in Sage

sage.schemes.projective.endPN_automorphism_group.**CRT_automorphisms**(*automorphisms*, *order_elts*, *degree*, *moduli*)

Given a list of automorphisms over various $Zmod(p^k)$, a list of the elements orders, an integer degree, and a list of the p^k values compute a maximal list of automorphisms over $Zmod(M)$, such that for every j in $len(moduli)$, each element reduces mod `moduli[j]` to one of the elements in $automorphisms[j]$ that has order = `degree`

INPUT:

- `automorphisms` – a list of lists of automorphisms over various $Zmod(p^k)$
- `order_elts` – a list of lists of the orders of the elements of `automorphisms`
- `degree` - a positive integer
- `moduli` – list of prime powers, i.e., p^k

OUTPUT:

- a list containing a list of automorphisms over $Zmod(M)$ and the product of the moduli

EXAMPLES:

```
sage: aut = [[matrix([[1,0],[0,1]]),matrix([[0,1],[1,0]])]]
sage: ords = [[1,2]]
sage: degree = 2
sage: mods = [5]
sage: from sage.schemes.projective.endPN_automorphism_group import CRT_automorphisms
sage: CRT_automorphisms(aut,ords,degree,mods)
([
[0 1]
[1 0]
], 5)
```

sage.schemes.projective.endPN_automorphism_group.**CRT_helper**(*automorphisms*, *moduli*)

Given a list of automorphisms over various $Zmod(p^k)$ find a list of automorphisms over $Zmod(M)$ where $M = \prod p^k$ that surjects onto every tuple of automorphisms from the various $Zmod(p^k)$.

INPUT:

- `automorphisms` – a list of lists of automorphisms over various $Zmod(p^k)$
- `moduli` – list of the various p^k

OUTPUT:

- a list of automorphisms over $Zmod(M)$.

EXAMPLES:

```
sage: from sage.schemes.projective.endPN_automorphism_group import CRT_helper
sage: CRT_helper([[matrix([[4,0],[0,1]]), matrix([[0,1],[1,0]])]],[5])
([
[4 0]  [0 1]
[0 1], [1 0]
], 5)
```

sage.schemes.projective.endPN_automorphism_group.**PGL_order**(*A*)

Find the multiplicative order of a linear fractional transformation that has a finite order as an element of $PGL_2(R)$. `A` can be represented either as a rational function or a 2x2 matrix

INPUT:

- A – a linear fractional transformation

OUTPUT:

- a positive integer

EXAMPLES:

```
sage: M = matrix([[0,2],[2,0]])
sage: from sage.schemes.projective.endPN_automorphism_group import PGL_order
sage: PGL_order(M)
2

sage: R.<x> = PolynomialRing(QQ)
sage: from sage.schemes.projective.endPN_automorphism_group import PGL_order
sage: PGL_order(-1/x)
2
```

sage.schemes.projective.endPN_automorphism_group.**PGL_repn**(*rational_function*)

Take a linear fraction transformation and represent it as a 2x2 matrix.

INPUT:

- rational_function – a linear fraction transformation

OUTPUT:

- a 2x2 matrix representing rational_function

EXAMPLES:

```
sage: R.<z> = PolynomialRing(QQ)
sage: f = ((2*z-1)/(3-z))
sage: from sage.schemes.projective.endPN_automorphism_group import PGL_repn
sage: PGL_repn(f)
[ 2 -1]
[-1  3]
```

sage.schemes.projective.endPN_automorphism_group.**automorphism_group_FF**(*rational_function*, *absolute=False*, *iso_type=False*, *return_functions=False*)

This function computes automorphism groups over finite fields.

ALGORITHM:

See Algorithm 4 in Faber-Manes-Viray [FMV]

INPUT:

- **rational_function – a rational function defined over the fraction field** of a polynomial ring in one variable with finite field coefficients.
- **absolute– Boolean - True returns the absolute automorphism group and a field of definition.** default: False (optional)
- **iso_type – Boolean - True returns the isomorphism type of the automorphism group.** default: False (optional)

- `return_functions` – Boolean, True returns linear fractional transformations False returns elements of $PGL(2)$. default: False (optional)

OUTPUT:

- List of automorphisms of `rational_function`

EXAMPLES:

```
sage: R.<x> = PolynomialRing(GF(5^2, 't'))
sage: from sage.schemes.projective.endPN_automorphism_group import automorphism_group_FF
sage: automorphism_group_FF((x^2+x+1)/(x+1))
[
[1 0]  [4 3]
[0 1], [0 1]
]
```

```
sage: R.<x> = PolynomialRing(GF(2^5, 't'))
sage: from sage.schemes.projective.endPN_automorphism_group import automorphism_group_FF
sage: automorphism_group_FF(x^(5), True, False, True)
[Univariate Polynomial Ring in w over Finite Field in b of size 2^5, [w, 1/w]]
```

```
sage: R.<x> = PolynomialRing(GF(2^5, 't'))
sage: from sage.schemes.projective.endPN_automorphism_group import automorphism_group_FF
sage: automorphism_group_FF(x^(5), False, False, True)
[x, 1/x]
```

`sage.schemes.projective.endPN_automorphism_group.`**`automorphism_group_FF_alg2`**(*rational_function*)

Implementation of algorithm for determining the absolute automorphism group over a finite field, given an invariant set., see [FMV].

INPUT:

- `rational_function`–a rational function defined over a finite field.

OUTPUT:

- absolute automorphism group of `rational_function` and a ring of definition.

EXAMPLES:

```
sage: R.<z> = PolynomialRing(GF(7^2,'t'))
sage: f = (3*z^3 - z^2)/(z-1)
sage: from sage.schemes.projective.endPN_automorphism_group import automorphism_group_FF_alg2
sage: automorphism_group_FF_alg2(f)
[Univariate Polynomial Ring in w over Finite Field in b of size 7^2, [w, (3*b + 2)/((2*b + 6)*w)
```

```
sage: R.<z> = PolynomialRing(GF(5^3,'t'))
sage: f = (3456*z^(4))
sage: from sage.schemes.projective.endPN_automorphism_group import automorphism_group_FF_alg2
sage: automorphism_group_FF_alg2(f)
[Univariate Polynomial Ring in w over Finite Field in b of size 5^6, [w,
(3*b^5 + 4*b^4 + 3*b^2 + 2*b + 1)*w, (2*b^5 + b^4 + 2*b^2 + 3*b + 3)*w,
(3*b^5 + 4*b^4 + 3*b^2 + 2*b)/((3*b^5 + 4*b^4 + 3*b^2 + 2*b)*w), (4*b^5
+ 2*b^4 + 4*b^2 + b + 2)/((3*b^5 + 4*b^4 + 3*b^2 + 2*b)*w), (3*b^5 +
4*b^4 + 3*b^2 + 2*b + 3)/((3*b^5 + 4*b^4 + 3*b^2 + 2*b)*w)]]
```

`sage.schemes.projective.endPN_automorphism_group.`**`automorphism_group_FF_alg3`**(*rational_function*)

Implementation of Algorithm 3 in the paper by Faber/Manes/Viray [FMV] for computing the automorphism group over a finite field.

INPUT:

•`rational_function`–a rational function defined over a finite field F.

OUTPUT:

•list of F-rational automorphisms of `rational_function`.

EXAMPLES:

```
sage: R.<z> = PolynomialRing(GF(5^3,'t'))
sage: f = (3456*z^4)
sage: from sage.schemes.projective.endPN_automorphism_group import automorphism_group_FF_alg3
sage: automorphism_group_FF_alg3(f)
[z, 3/(3*z)]
```

`sage.schemes.projective.endPN_automorphism_group.`**`automorphism_group_QQ_CRT`**(*rational_function*, *prime_lower_bound=4*, *return_functions=True*, *iso_type=False*)

Determines the complete group of rational automorphisms (under the conjugation action of $PGL(2, QQ)$) for a rational function of one variable, see [FMV] for details.

INPUT:

•`rational_function` - a rational function of a univariate polynomial ring over QQ

•prime_lower_bound‘‘ – a positive integer - a lower bound for the primes to use for the Chinese Remainder Theorem step. default: 4 (optional)

•`return_functions` – Boolean - True returns linear fractional transformations False returns elements of $PGL(2, QQ)$ default: True (optional).

•**`iso_type` – Boolean - True returns the isomorphism type of the automorphism group.** default: False (optional)

OUTPUT:

•a complete list of automorphisms of $rational_function$

EXAMPLES:

```
sage: R.<z> = PolynomialRing(QQ)
sage: f = (3*z^2 - 1)/(z^3 - 3*z)
sage: from sage.schemes.projective.endPN_automorphism_group import automorphism_group_QQ_CRT
sage: automorphism_group_QQ_CRT(f, 4,True)
[z, -z, 1/z, -1/z, (-z + 1)/(z + 1), (z + 1)/(z - 1), (z - 1)/(z + 1),
(-z - 1)/(z - 1)]

sage: R.<z> = PolynomialRing(QQ)
sage: f = (3*z^2 - 1)/(z^3 - 3*z)
sage: from sage.schemes.projective.endPN_automorphism_group import automorphism_group_QQ_CRT
sage: automorphism_group_QQ_CRT(f, 4,False)
[
[1 0]  [-1  0]  [0 1]  [ 0 -1]  [-1  1]  [ 1  1]  [ 1 -1]  [-1 -1]
[0 1], [ 0  1], [1 0], [ 1  0], [ 1  1], [ 1 -1], [ 1  1], [ 1 -1]
]
```

`sage.schemes.projective.endPN_automorphism_group.`**`automorphism_group_QQ_fixedpoints`**(*rational_fu* *return_functi* *iso_type=F*

This function will compute the automorphism group for `rational_function` via the method of fixed points

ALGORITHM:

See Algorithm 3 in Faber-Manes-Viray [FMV]

INPUT:

- `rational_function` - Rational Function defined over $\mathbb{Z}$ or $\mathbb{Q}$
- **`return_functions` - Boolean Value, True will return elements in the automorphism group** as linear fractional transformations. False will return elements as $PGL2$ matrices.
- **`iso_type` - Boolean - True will cause the classification of the finite automorphism** group to also be returned

OUTPUT:

- List of automorphisms that make up the Automorphism Group of `rational_function`.

EXAMPLES:

```
sage: F.<z> = PolynomialRing(QQ)
sage: rational_function = (z^2 - 2*z - 2)/(-2*z^2 - 2*z + 1)
sage: from sage.schemes.projective.endPN_automorphism_group import automorphism_group_QQ_fixedpo
sage: automorphism_group_QQ_fixedpoints(rational_function, True)
  [z, 2/(2*z), -z - 1, -2*z/(2*z + 2), (-z - 1)/z, -1/(z + 1)]

sage: F.<z> = PolynomialRing(QQ)
sage: rational_function = (z^2 + 2*z)/(-2*z - 1)
sage: from sage.schemes.projective.endPN_automorphism_group import automorphism_group_QQ_fixedpo
sage: automorphism_group_QQ_fixedpoints(rational_function)
  [
  [1 0]   [-1 -1]   [-2  0]   [0 2]   [-1 -1]   [ 0 -1]
  [0 1], [ 0  1], [ 2  2], [2 0], [ 1  0], [ 1  1]
  ]

sage: F.<z> = PolynomialRing(QQ)
sage: rational_function = (z^2 - 4*z -3)/(-3*z^2 - 2*z + 2)
sage: from sage.schemes.projective.endPN_automorphism_group import automorphism_group_QQ_fixedpo
sage: automorphism_group_QQ_fixedpoints(rational_function, True, True)
  ([z, (-z - 1)/z, -1/(z + 1)], 'Cyclic of order 3')
```

`sage.schemes.projective.endPN_automorphism_group.`**`automorphisms_fixing_pair`**(*rational_function, pair, quad*)

INPUT:

- `rational_function`– rational function defined over finite field E.
- `pair`– a pair of points of $\mathbb{P}^1(E)$.
- `quad`– Boolean: an indicator if this is a quadratic pair of points

OUTPUT:

- **set of automorphisms with order prime to characteristic defined over E that fix** the pair, excluding the identity.

EXAMPLES:

```
sage: R.<z> = PolynomialRing(GF(7^2,'t'))
sage: f = (z^2 + 5*z + 5)/(5*z^2 + 5*z + 1)
sage: L = [[4, 1], [2, 1]]
sage: from sage.schemes.projective.endPN_automorphism_group import automorphisms_fixing_pair
sage: automorphisms_fixing_pair(f,L,False)
[(6*z + 6)/z, 4/(3*z + 3)]
```

`sage.schemes.projective.endPN_automorphism_group.`**`field_descent`**(*sigma*, *y*)

Function for descending an element in a field E to a subfield F. Here F, E must be finite fields or number fields. This function determines the unique image of subfield which is `y` by the embedding `sigma` if it exists. Otherwise returns `None`. This functionality is necessary because Sage does not keep track of subfields.

INPUT:

- `sigma`– an embedding sigma: F -> E of fields.
- `y` –an element of the field E

OUTPUT:

- the unique element of the subfield if it exists, otherwise `None`

EXAMPLE:

```
sage: R = GF(11^2,'b')
sage: RR = GF(11)
sage: s = RR.Hom(R)[0]
sage: from sage.schemes.projective.endPN_automorphism_group import field_descent
sage: field_descent(s, R(1))
1
```

`sage.schemes.projective.endPN_automorphism_group.`**`height_bound`**(*polynomial*)

Compute the maximum height of the coefficients of an automorphism. This sets the termination criteria for the Chinese Remainder Theorem step.

Let f be a square-free polynomial with coefficients in K Let F be an automorphism of $\mathbb{P}^1_{Frac(R)}$ that permutes the roots of f This function returns a bound on the height of F, when viewed as an element of $\mathbb{P}^3$

In [FMV] it is proven that $ht(F) <= 6^{[K:Q]} * M$, where M is the Mahler measure of f M is bounded above by $H(f)$, so we return the floor of $6 * H(f)$ (since $ht(F)$ is an integer)

INPUT:

- `polynomial` – a univariate polynomial

OUTPUT:

- a positive integer

EXAMPLES:

```
sage: R.<z> = PolynomialRing(QQ)
sage: f = (z^3+2*z+6)
sage: from sage.schemes.projective.endPN_automorphism_group import height_bound
sage: height_bound(f)
413526
```

`sage.schemes.projective.endPN_automorphism_group.`**`order_p_automorphisms`**(*rational_function*, *pre_image*)

Determine the order-p automorphisms given the input data. This is algorithm 4 in Faber-Manes-Viray [FMV].

INPUT:

- `rational_function`–rational function defined over finite field F.
- **`pre_image`–set of triples** $[x, L, f]$**, where** x **is an** F**-rational** fixed point of `rational_function`, L is the list of F-rational pre-images of x (excluding x), and f is the polynomial defining the full set of pre-images of x (again excluding x itself).

OUTPUT:

- set of automorphisms of order p defined over F.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(GF(11))
sage: f = x^11
sage: L = [[[0, 1], [], 1], [[10, 1], [], 1], [[9, 1], [], 1],
....: [[8, 1], [],1], [[7, 1], [], 1], [[6, 1], [], 1], [[5, 1], [], 1],
....: [[4, 1], [], 1],[[3, 1], [], 1], [[2, 1], [], 1], [[1, 1], [], 1],
....: [[1, 0], [], 1]]
sage: from sage.schemes.projective.endPN_automorphism_group import order_p_automorphisms
sage: order_p_automorphisms(f,L)
[x/(x + 1), 6*x/(x + 6), 3*x/(x + 3), 7*x/(x + 7), 9*x/(x + 9), 10*x/(x
+ 10), 5*x/(x + 5), 8*x/(x + 8), 4*x/(x + 4), 2*x/(x + 2), 10/(x + 2),
(5*x + 10)/(x + 7), (2*x + 10)/(x + 4), (6*x + 10)/(x + 8), (8*x +
10)/(x + 10), (9*x + 10)/x, (4*x + 10)/(x + 6), (7*x + 10)/(x + 9), (3*x
+ 10)/(x + 5), (x + 10)/(x + 3), (10*x + 7)/(x + 3), (4*x + 7)/(x + 8),
(x + 7)/(x + 5), (5*x + 7)/(x + 9), (7*x + 7)/x, (8*x + 7)/(x + 1), (3*x
+ 7)/(x + 7), (6*x + 7)/(x + 10), (2*x + 7)/(x + 6), 7/(x + 4), (9*x +
2)/(x + 4), (3*x + 2)/(x + 9), 2/(x + 6), (4*x + 2)/(x + 10), (6*x +
2)/(x + 1), (7*x + 2)/(x + 2), (2*x + 2)/(x + 8), (5*x + 2)/x, (x +
2)/(x + 7), (10*x + 2)/(x + 5), (8*x + 6)/(x + 5), (2*x + 6)/(x + 10),
(10*x + 6)/(x + 7), (3*x + 6)/x, (5*x + 6)/(x + 2), (6*x + 6)/(x + 3),
(x + 6)/(x + 9), (4*x + 6)/(x + 1), 6/(x + 8), (9*x + 6)/(x + 6), (7*x +
8)/(x + 6), (x + 8)/x, (9*x + 8)/(x + 8), (2*x + 8)/(x + 1), (4*x +
8)/(x + 3), (5*x + 8)/(x + 4), 8/(x + 10), (3*x + 8)/(x + 2), (10*x +
8)/(x + 9), (8*x + 8)/(x + 7), (6*x + 8)/(x + 7), 8/(x + 1), (8*x +
8)/(x + 9), (x + 8)/(x + 2), (3*x + 8)/(x + 4), (4*x + 8)/(x + 5), (10*x
+ 8)/x, (2*x + 8)/(x + 3), (9*x + 8)/(x + 10), (7*x + 8)/(x + 8), (5*x +
6)/(x + 8), (10*x + 6)/(x + 2), (7*x + 6)/(x + 10), 6/(x + 3), (2*x +
6)/(x + 5), (3*x + 6)/(x + 6), (9*x + 6)/(x + 1), (x + 6)/(x + 4), (8*x
+ 6)/x, (6*x + 6)/(x + 9), (4*x + 2)/(x + 9), (9*x + 2)/(x + 3), (6*x +
2)/x, (10*x + 2)/(x + 4), (x + 2)/(x + 6), (2*x + 2)/(x + 7), (8*x +
2)/(x + 2), 2/(x + 5), (7*x + 2)/(x + 1), (5*x + 2)/(x + 10), (3*x +
7)/(x + 10), (8*x + 7)/(x + 4), (5*x + 7)/(x + 1), (9*x + 7)/(x + 5),
7/(x + 7), (x + 7)/(x + 8), (7*x + 7)/(x + 3), (10*x + 7)/(x + 6), (6*x
+ 7)/(x + 2), (4*x + 7)/x, (2*x + 10)/x, (7*x + 10)/(x + 5), (4*x +
10)/(x + 2), (8*x + 10)/(x + 6), (10*x + 10)/(x + 8), 10/(x + 9), (6*x +
10)/(x + 4), (9*x + 10)/(x + 7), (5*x + 10)/(x + 3), (3*x + 10)/(x + 1),
x + 1, x + 2, x + 4, x + 8, x + 5, x + 10, x + 9, x + 7, x + 3, x + 6]
```

`sage.schemes.projective.endPN_automorphism_group.`**`rational_function_coefficient_descent`**(*ratio*
sigm
*poly_*

Function for descending the coefficients of a rational function from field E to a subfield F. Here F, E must be finite fields or number fields. It determines the unique rational function in fraction field of `poly_ring` which is the image of `rational_function` by `ssigma`, if it exists, and otherwise returns `None`.

INPUT:

- `rational_function`–a rational function with coefficients in a field E,
- `sigma`– a field embedding sigma: F -> E.
- `poly_ring`– a polynomial ring R with coefficients in F.

OUTPUT:

- a rational function with coefficients in the fraction field of `poly_ring` if it exists, and otherwise `None`.

EXAMPLES:

```
sage: T.<z> = PolynomialRing(GF(11^2,'b'))
sage: S.<y> = PolynomialRing(GF(11))
sage: s = S.base_ring().hom(T.base_ring())
sage: f = (3*z^3 - z^2)/(z-1)
sage: from sage.schemes.projective.endPN_automorphism_group import rational_function_coefficient
sage: rational_function_coefficient_descent(f,s,S)
(3*y^3 + 10*y^2)/(y + 10)
```

sage.schemes.projective.endPN_automorphism_group.**rational_function_coerce**(*rational_function*, *sigma*, *S_polys*)

Function for coercing a rational function defined over a ring R to have coefficients in a second ring `S_polys`. The fraction field of polynomial ring `S_polys` will contain the new rational function.

INPUT:

- `rational_funtion` – rational function with coefficients in R.
- `sigma` – a ring homomorphism sigma: R -> `S_polys`.
- `S_polys` – a polynomial ring.

OUTPUT:

- a rational function with coefficients in `S_polys`.

EXAMPLES:

```
sage: R.<y> = PolynomialRing(QQ)
sage: S.<z> = PolynomialRing(ZZ)
sage: s = S.hom([z],R)
sage: f = (3*z^2 + 1)/(z^3-1)
sage: from sage.schemes.projective.endPN_automorphism_group import rational_function_coerce
sage: rational_function_coerce(f,s,R)
(3*y^2 + 1)/(y^3 - 1)
```

sage.schemes.projective.endPN_automorphism_group.**rational_function_reduce**(*rational_function*)

Force Sage to divide out common factors in numerator and denominator of rational function.

INPUT:

- `rational_function` – rational function $= F/G$ in univariate polynomial ring.

OUTPUT:

- rational function – $(F/gcd(F,G))/(G/gcd(F,G))$.

EXAMPLES:

```
sage: R.<z> = PolynomialRing(GF(7))
sage: f = ((z-1)*(z^2+z+1))/((z-1)*(z^3+1))
sage: from sage.schemes.projective.endPN_automorphism_group import rational_function_reduce
sage: rational_function_reduce(f)
(z^2 + z + 1)/(z^3 + 1)
```

sage.schemes.projective.endPN_automorphism_group.**remove_redundant_automorphisms**(*automorphisms, order_elts, moduli, integral_autos*)

If an element of Aut_{F_p} has been lifted to QQ remove that element from Aut_{F_p} so we don't attempt to lift that element again unnecessarily

INPUT:

- `automorphisms` – a list of lists of automorphisms
- `order_elts` – a list of lists of the orders of the elements of `automorphisms`
- `moduli` – a list of prime powers
- `integral_autos` – list of known automorphisms

OUTPUT:

- a list of automorphisms.

EXAMPLES:

```
sage: auts = [[matrix([[1,0],[0,1]]), matrix([[6,0],[0,1]]), matrix([[0,1],[1,0]]),
....: matrix([[6,1],[1,1]]), matrix([[1,1],[1,6]]), matrix([[0,6],[1,0]]),
....: matrix([[1,6],[1,1]]), matrix([[6,6],[1,6]])]]
sage: ord_elts = [[1, 2, 2, 2, 2, 2, 4, 4]]
sage: mods = [7]
sage: R.<x> = PolynomialRing(QQ)
sage: int_auts = [-1/x]
sage: from sage.schemes.projective.endPN_automorphism_group import remove_redundant_automorphisms
sage: remove_redundant_automorphisms(auts, ord_elts, mods, int_auts)
[[
[1 0]  [6 0]  [0 1]  [6 1]  [1 1]  [1 6]  [6 6]
[0 1], [0 1], [1 0], [1 1], [1 6], [1 1], [1 6]
]]
```

sage.schemes.projective.endPN_automorphism_group.**three_stable_points**(*rational_function, invariant_list*)

Implementation of Algorithm 1 for automorphism groups from Faber-Manes-Viray [FMV].

INPUT:

- `rational_function`–rational function phi defined over finite field E.
- `invariant_list`– a list of at least 3 points of $\mathbb{P}^1(E)$ that is stable under $Aut_{phi}(E)$.

OUTPUT:

- list of automorphisms

EXAMPLES:

```
sage: R.<z> = PolynomialRing(GF(5^2,'t'))
sage: f = z^3
sage: L = [[0,1],[4,1],[1,1],[1,0]]
sage: from sage.schemes.projective.endPN_automorphism_group import three_stable_points
sage: three_stable_points(f,L)
[z, 4*z, 2/(2*z), 3/(2*z)]
```

sage.schemes.projective.endPN_automorphism_group.**valid_automorphisms**(*automorphisms_CRT*, *rational_function*, *ht_bound*, *M*, *return_functions=False*)

Checks whether an element that is an automorphism of `rational_function` modulo p^k for various p s and k s can be lifted to an automorphism over ZZ. It uses the fact that every automorphism has height at most `ht_bound`

INPUT:

- `automorphisms` – a list of lists of automorphisms over various $Zmod(p^k)$
- `rational_function` – A one variable rational function
- `ht_bound` - a positive integer
- `M` – a positive integer, a product of prime powers
- `return_functions` – Boolean. default: False (optional)

OUTPUT:

- a list of automorphisms over ZZ.

EXAMPLES:

```
sage: R.<z> = PolynomialRing(QQ)
sage: F = z^2
sage: from sage.schemes.projective.endPN_automorphism_group import valid_automorphisms
sage: valid_automorphisms([matrix(GF(5),[[0,1],[1,0]])], F, 48, 5, True)
[1/z]
```

sage.schemes.projective.endPN_automorphism_group.**which_group**(*list_of_elements*)

Given a finite subgroup of $PGL2$ determine its isomorphism class. This function makes heavy use of the classification of finite subgroups of $PGL(2, K)$

INPUT:

- **`list_of_elements`– a finite list of elements of** $PGL(2, K)$ that we know a priori form a group

OUTPUT:

- String – the isomorphism type of the group.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(GF(7,'t'))
sage: G = [x, 6*x/(x + 1), 6*x + 6, 1/x, (6*x + 6)/x, 6/(x + 1)]
sage: from sage.schemes.projective.endPN_automorphism_group import which_group
sage: which_group(G)
'Dihedral of order 6'
```

14.8 Sage functions to compute minimal models of rational functions

under the conjugation action of $PGL_2(QQ)$.

AUTHORS:

- Alex Molnar (May 22, 2012)

- Brian Stout, Ben Hutz (Nov 2013): Modified code to use projective morphism functionality so that it can be included in Sage.

REFERENCES:

`sage.schemes.projective.endPN_minimal_model.`**`Min`**(*Fun*, *p*, *ubRes*, *conj*)

Local loop for Affine_minimal, where we check minimality at the prime p.

First we bound the possible k in our transformations A = zp^k + b. See Theorems 3.3.2 and 3.3.3 in [Molnar].

INPUT:

- `Fun` – a projective space morphisms
- `p` - a prime.
- `ubRes` – integer, the upper bound needed for Th. 3.3.3 in [Molnar].
- `conj` – a 2x2 matrix keeping track of the conjugation

OUTPUT:

- Boolean – `True` if `Fun` is minimal at `p`, `False` otherwise
- a projective morphism minimal at `p`

EXAMPLES:

```
sage: P.<x,y> = ProjectiveSpace(QQ,1)
sage: H = End(P)
sage: f = H([149*x^2 + 39*x*y + y^2, -8*x^2 + 137*x*y + 33*y^2])
sage: from sage.schemes.projective.endPN_minimal_model import Min
sage: Min(f, 3, -27000000, matrix(QQ,[[1, 0],[0, 1]]))
(
Scheme endomorphism of Projective Space of dimension 1 over Rational
Field
  Defn: Defined on coordinates by sending (x : y) to
        (181*x^2 + 313*x*y + 81*y^2 : -24*x^2 + 73*x*y + 151*y^2)
,
[3 4]
[0 1]
)
```

`sage.schemes.projective.endPN_minimal_model.`**`affine_minimal`**(*vp*, *return_transformation=False*, *D=None*, *quick=False*)

Given vp a scheme morphisms on the projective line over the rationals, this procedure determines if ϕ is minimal. In particular, it determines if the map is affine minimal, which is enough to decide if it is minimal or not. See Proposition 2.10 in [Bruin-Molnar].

INPUT:

- `vp` – scheme morphism on the projective line.
- **D – a list of primes, in case one only wants to check minimality** at those specific primes.
- `return_transformation` – a boolean value, default value True. This signals a return of the `PGL_2` transformation to conjugate `vp` to the calculated minimal model. default: False
- `quick` – a boolean value. If true the algorithm terminates once algorithm determines F/G is not minimal, otherwise algorithm only terminates once a minimal model has been found.

OUTPUT:

- newvp – scheme morphism on the projective line.
- conj – linear fractional transformation which conjugates vp to newvp

EXAMPLES:

```
sage: PS.<X,Y> = ProjectiveSpace(QQ,1)
sage: H = Hom(PS,PS)
sage: vp = H([X^2+9*Y^2,X*Y])
sage: from sage.schemes.projective.endPN_minimal_model import affine_minimal
sage: affine_minimal(vp,True)
(
Scheme endomorphism of Projective Space of dimension 1 over Rational
Field
  Defn: Defined on coordinates by sending (X : Y) to
        (X^2 + Y^2 : X*Y)
,
[3 0]
[0 1]
)
```

sage.schemes.projective.endPN_minimal_model.**bCheck**(*c*, *v*, *p*, *b*)

Compute a lower bound on the value of b needed, for a transformation $A(z) = z * p^k + b$ to satisfy $ord_p(Res(\phi^A)) < ord_p(Res(\phi))$ for a rational map ϕ. See Theorem 3.3.5 in [Molnar].

INPUT:

- c – a list of polynomials in b. See v for their use.
- **v – a list of rational numbers, where we are considering the inequalities** $ord_p(c[i]) > v[i]$.
- p – a prime.
- b – local variable.

OUTPUT:

- bval – Integer, lower bound in Theorem 3.3.5

EXAMPLES:

```
sage: R.<b> = PolynomialRing(QQ)
sage: from sage.schemes.projective.endPN_minimal_model import bCheck
sage: bCheck(11664*b^2 + 70227*b + 76059, 15/2, 3, b)
-1
```

sage.schemes.projective.endPN_minimal_model.**blift**(*LF*, *Li*, *p*, *S=None*)

Search for a solution to the given list of inequalities. If found, lift the solution to an appropriate valuation. See Lemma 3.3.6 in [Molnar]

INPUT:

- LF – a list of integer polynomials in one variable (the normalized coefficients)
- Li – an integer, the bound on coefficients
- p – a prime

OUTPUT:

- Boolean – whether or not the lift is successful
- integer – the lift

EXAMPLES:

```
sage: R.<b> = PolynomialRing(QQ)
sage: from sage.schemes.projective.endPN_minimal_model import blift
sage: blift([8*b^3 + 12*b^2 + 6*b + 1, 48*b^2 + 483*b + 117, 72*b + 1341, -24*b^2 + 411*b + 99,
(True, 4)
```

sage.schemes.projective.endPN_minimal_model.**scale**(c, v, p)

Given an integral polynomial c, we can write $c = p^i * c'$, where p does not divide c. Returns c' and $v - i$ where i is the smallest valuation of the coefficients of c.

INPUT:

- c – an integer polynomial
- v – an integer - the bound on the exponent from blift
- p – a prime

OUTPUT:

- Boolean – the new exponent bound is 0 or negative
- the scaled integer polynomial
- an integer the new exponent bound

EXAMPLES:

```
sage: R.<b> = PolynomialRing(QQ)
sage: from sage.schemes.projective.endPN_minimal_model import scale
sage: scale(24*b^3 + 108*b^2 + 162*b + 81, 1, 3)
[False, 8*b^3 + 36*b^2 + 54*b + 27, 0]
```

CHAPTER

FIFTEEN

PRODUCTS OF PROJECTIVE SPACES

15.1 Products of projective spaces

This class builds on the projective space class and its point and morphism classes.

Products of projective spaces of varying dimension are convenient ambient spaces for complete intersections. Group actions on them, and the interplay with representation theory, provide many interesting examples of algebraic varieties.

EXAMPLES:

We construct products projective spaces of various dimensions over the same ring.:

```
sage: P1 = ProjectiveSpace(ZZ,1,'x')
sage: P2 = ProjectiveSpace(ZZ,2,'y')
sage: ProductProjectiveSpaces([P1,P2])
Product of projective spaces P^1 x P^2 over Integer Ring
```

We can also construct the product by specifying the dimensions and the base ring:

```
sage: ProductProjectiveSpaces([1,2,3],QQ,'z')
Product of projective spaces P^1 x P^2 x P^3 over Rational Field

sage: P2xP2 = ProductProjectiveSpaces([2, 2], QQ, names=['x', 'y'])
sage: P2xP2.coordinate_ring().inject_variables()
Defining x0, x1, x2, y0, y1, y2
```

`sage.schemes.product_projective.space.`**`ProductProjectiveSpaces`**(*n*, *R=None*, *names='x'*)

Returns the cartesian product of projective spaces. Can input either a list of projective spaces over the same base ring or the list of dimensions, the base ring, and the variable names.

INPUT:

- `n` – a list of integers or a list of projective spaces
- `R` – a ring
- `names` – a string or list of strings

EXAMPLES:

```
sage: P1 = ProjectiveSpace(QQ,2,'x')
sage: P2 = ProjectiveSpace(QQ,3,'y')
sage: ProductProjectiveSpaces([P1,P2])
Product of projective spaces P^2 x P^3 over Rational Field

sage: ProductProjectiveSpaces([2,2],GF(7),'y')
Product of projective spaces P^2 x P^2 over Finite Field of size 7
```

```
sage: P1 = ProjectiveSpace(ZZ,2,'x')
sage: P2 = ProjectiveSpace(QQ,3,'y')
sage: ProductProjectiveSpaces([P1,P2])
Traceback (most recent call last):
...
AttributeError: Components must be over the same base ring
```

class sage.schemes.product_projective.space.**ProductProjectiveSpaces_ring**(*N*, *R=Rational Field*, *names=None*)

Bases: sage.schemes.generic.ambient_space.AmbientSpace

Cartesian product of projective spaces $\mathbb{P}^{n_1} \times \cdots \times \mathbb{P}^{n_r}$.

EXAMPLES:

```
sage: P.<x0,x1,x2,x3,x4> = ProductProjectiveSpaces([1,2],QQ); P
Product of projective spaces P^1 x P^2 over Rational Field
sage: P.coordinate_ring()
Multivariate Polynomial Ring in x0, x1, x2, x3, x4 over Rational Field
sage: P[0]
Projective Space of dimension 1 over Rational Field
sage: P[1]
Projective Space of dimension 2 over Rational Field
sage: Q = P(6,3,2,2,2); Q
(2 : 1 , 1 : 1 : 1)
sage: Q[0]
(2 : 1)
sage: H = Hom(P,P)
sage: f = H([x0^2*x3,x2*x1^2,x2^2,2*x3^2,x4^2])
sage: f(Q)
(4 : 1 , 1 : 2 : 1)
```

affine_patch(*I*, *return_embedding=False*)

Return the I^{th} affine patch of this projective space product where I is a multi-index.

INPUT:

- I – a list or tuple of positive integers
- return_embedding – Boolean, if true the projective embedding is also returned

OUTPUT:

- An affine space
- An embedding into a product of projective spaces (optional)

EXAMPLES:

```
sage: PP = ProductProjectiveSpaces([2,2,2], ZZ, 'x')
sage: phi = PP.affine_patch([0,1,2], True)
sage: phi.domain()
Affine Space of dimension 6 over Integer Ring
sage: phi
Scheme morphism:
      From: Affine Space of dimension 6 over Integer Ring
      To:   Product of projective spaces P^2 x P^2 x P^2 over Integer Ring
      Defn: Defined on coordinates by sending (x0, x1, x2, x3, x4, x5) to
            (1 : x0 : x1 , x2 : 1 : x3 , x4 : x5 : 1)
```

change_ring(*R*)
Return a product of projective spaces over a ring R and otherwise the same as `self`.

INPUT:

- `R` – commutative ring

OUTPUT:

product of projective spaces over `R`

Note: There is no need to have any relation between R and the base ring of `self`, if you want to have such a relation, use `self.base_extend(R)` instead.

EXAMPLES:

```
sage: T.<x,y,z,u,v,w> = ProductProjectiveSpaces([2,2],QQ)
sage: T.change_ring(GF(17))
Product of projective spaces P^2 x P^2 over Finite Field of size 17
```

dimension()
Return the absolute dimension of the product of projective spaces.

OUTPUT:

a positive integer.

EXAMPLES:

```
sage: T.<x,y,z,u,v,w> = ProductProjectiveSpaces([2,2],GF(17))
sage: T.dimension_absolute()
4
sage: T.dimension()
4
```

dimension_absolute()
Return the absolute dimension of the product of projective spaces.

OUTPUT:

a positive integer.

EXAMPLES:

```
sage: T.<x,y,z,u,v,w> = ProductProjectiveSpaces([2,2],GF(17))
sage: T.dimension_absolute()
4
sage: T.dimension()
4
```

dimension_absolute_components()
Return the absolute dimension of the product of projective spaces.

OUTPUT:

a list of positive integers.

EXAMPLES:

```
sage: T.<x,y,z,u,v,w> = ProductProjectiveSpaces([2,2],GF(17))
sage: T.dimension_absolute_components()
[2, 2]
```

```
sage: T.dimension_components()
[2, 2]
```

dimension_components()

Return the absolute dimension of the product of projective spaces.

OUTPUT:

a list of positive integers.

EXAMPLES:

```
sage: T.<x,y,z,u,v,w> = ProductProjectiveSpaces([2,2],GF(17))
sage: T.dimension_absolute_components()
[2, 2]
sage: T.dimension_components()
[2, 2]
```

dimension_relative()

Return the relative dimension of the product of projective spaces.

OUTPUT:

a positive integer.

EXAMPLES:

```
sage: T.<a,x,y,z,u,v,w> = ProductProjectiveSpaces([3,2],QQ)
sage: T.dimension_relative()
5
```

dimension_relative_components()

Return the relative dimension of the product of projective spaces.

OUTPUT:

a list of positive integers.

EXAMPLES:

```
sage: T.<a,x,y,z,u,v,w> = ProductProjectiveSpaces([3,2],QQ)
sage: T.dimension_relative_components()
[3, 2]
```

ngens()

Returns the number of generators of `self`, i.e., the number of variables in the coordinate ring of self

OUTPUT:

an integer.

EXAMPLES:

```
sage: T = ProductProjectiveSpaces([1,1,1],GF(5),'x')
sage: T.ngens()
6
```

num_components()

Returns the number of components of `self`.

OUTPUT:

an integer.

EXAMPLES:

```
sage: T = ProductProjectiveSpaces([1,1,1],GF(5),'x')
sage: T.num_components()
3
```

segre_embedding(*PP=None*, *var='u'*)

Return the Segre embedding of `self` into the appropriate projective space.

INPUT:

- **PP – (default: None) ambient image projective space;** this is constructed if it is not given.
- `var` – string, variable name of the image projective space, default u (optional)

OUTPUT:

Hom – from `self` to the appropriate subscheme of projective space

Todo

Cartesian products with more than two components

EXAMPLES:

```
sage: X.<y0,y1,y2,y3,y4,y5> = ProductProjectiveSpaces(ZZ,[2,2])
sage: phi = X.segre_embedding(); phi
Scheme morphism:
  From: Product of projective spaces P^2 x P^2 over Integer Ring
  To:   Closed subscheme of Projective Space of dimension 8 over Integer Ring defined by:
  -u5*u7 + u4*u8,
  -u5*u6 + u3*u8,
  -u4*u6 + u3*u7,
  -u2*u7 + u1*u8,
  -u2*u4 + u1*u5,
  -u2*u6 + u0*u8,
  -u1*u6 + u0*u7,
  -u2*u3 + u0*u5,
  -u1*u3 + u0*u4
  Defn: Defined by sending (y0 : y1 : y2 , y3 : y4 : y5) to
        (y0*y3 : y0*y4 : y0*y5 : y1*y3 : y1*y4 : y1*y5 : y2*y3 : y2*y4 : y2*y5).
```

::

```
sage: T = ProductProjectiveSpaces([1,2],CC,'z')
sage: T.segre_embedding()
Scheme morphism:
  From: Product of projective spaces P^1 x P^2 over Complex Field with 53 bits of precision
  To:   Closed subscheme of Projective Space of dimension 5 over Complex Field with 53 bits
  -u2*u4 + u1*u5,
  -u2*u3 + u0*u5,
  -u1*u3 + u0*u4
  Defn: Defined by sending (z0 : z1 , z2 : z3 : z4) to
        (z0*z2 : z0*z3 : z0*z4 : z1*z2 : z1*z3 : z1*z4).
```

subscheme(*X*)

Return the closed subscheme defined by `X`.

INPUT:

- `X` - a list or tuple of equations

OUTPUT:

`AlgebraicScheme_subscheme_projective_cartesian_product`

EXAMPLES:

```
sage: P.<x,y,z,w> = ProductProjectiveSpaces([1,1],GF(5))
sage: X = P.subscheme([x-y,z-w]);X
Closed subscheme of Product of projective spaces P^1 x P^1 over Finite Field of size 5 defin
      x - y,
      z - w
sage: X.defining_polynomials ()
[x - y, z - w]
sage: I = X.defining_ideal(); I
Ideal (x - y, z - w) of Multivariate Polynomial Ring in x, y, z, w over
Finite Field of size 5
sage: X.dimension()
0
sage: X.base_ring()
Finite Field of size 5
sage: X.base_scheme()
Spectrum of Finite Field of size 5
sage: X.structure_morphism()
Scheme morphism:
      From: Closed subscheme of Product of projective spaces P^1 x P^1 over Finite Field of
      x - y,
      z - w
      To:   Spectrum of Finite Field of size 5
      Defn: Structure map
```

`sage.schemes.product_projective.space.`**`is_ProductProjectiveSpaces`**`(x)`

Return True if x is a product of projective spaces, i.e., an ambient space $\mathbb{P}^n_R \times \cdots \times \mathbb{P}^m_R$, where R is a ring and $n, \ldots, m \geq 0$ are integers.

OUTPUT:

Boolean

EXAMPLES:

```
sage: is_ProductProjectiveSpaces(ProjectiveSpace(5, names='x'))
False
sage: is_ProductProjectiveSpaces(ProductProjectiveSpaces([1,2,3], ZZ, 'x'))
True
```

15.2 Set of homomorphisms

class `sage.schemes.product_projective.homset.`**`SchemeHomset_points_product_projective_spaces_ri`**

Bases: `sage.schemes.generic.homset.SchemeHomset_points`

Set of rational points of a product of projective spaces.

INPUT:

See SchemeHomset_generic.

EXAMPLES:

```
sage: from sage.schemes.product_projective.homset import SchemeHomset_points_product_projective_
sage: SchemeHomset_points_product_projective_spaces_ring(Spec(QQ), ProductProjectiveSpaces([1,1]
Set of rational points of Product of projective spaces P^1 x P^1 over Rational Field
```

15.3 Polynomial morphisms for products of projective spaces

This class builds on the projective space class and its point and morphism classes.

EXAMPLES:

```
sage: P1xP1.<x,y, u,v> = ProductProjectiveSpaces(QQ,[1,1])
sage: H = End(P1xP1)
sage: H([x^2*u, y^2*v, x*v^2, y*u^2])
Scheme endomorphism of Product of projective spaces P^1 x P^1 over Rational Field
  Defn: Defined by sending (x : y , u : v) to
        (x^2*u : y^2*v , x*v^2 : y*u^2).
```

class sage.schemes.product_projective.morphism.**ProductProjectiveSpaces_morphism_ring**(*parent*, *polys*, *check=True*)

Bases: sage.schemes.generic.morphism.SchemeMorphism_polynomial

The class of morphisms on products of projective spaces. The components are projective space morphisms.

EXAMPLES:

```
sage: T.<x,y,z,w,u> = ProductProjectiveSpaces([2,1],QQ)
sage: H = T.Hom(T)
sage: H([x^2,y^2,z^2,w^2,u^2])
Scheme endomorphism of Product of projective spaces P^2 x P^1 over Rational Field
  Defn: Defined by sending (x : y : z , w : u) to
        (x^2 : y^2 : z^2 , w^2 : u^2).
```

15.4 Points for products of projective spaces

This class builds on the projective space class and its point and morphism classes.

EXAMPLES:

We construct products projective spaces of various dimensions over the same ring.:

```
sage: P1xP1.<x,y, u,v> = ProductProjectiveSpaces(QQ, [1,1])
sage: P1xP1([2,1, 3,1])
(2 : 1 , 3 : 1)
```

class sage.schemes.product_projective.point.**ProductProjectiveSpaces_point_ring**(*parent*, *polys*, *check=True*)

Bases: sage.schemes.generic.morphism.SchemeMorphism_point

The class of points on products of projective spaces. The components are projective space points.

EXAMPLES:

```
sage: T.<x,y,z,w,u> = ProductProjectiveSpaces([2,1],QQ)
sage: T.point([1,2,3,4,5]);
(1/3 : 2/3 : 1 , 4/5 : 1)
```

change_ring (*R*, ***kwds*)

Returns a new `ProductProjectiveSpaces_point` which is `self` coerced to `R`.

If the keyword `check` is `True`, then the initialization checks are performed. The user may specify the embedding into `R` with a keyword.

INPUT:

•`R` – ring

kwds:

•`check` – Boolean

•`embedding` – field embedding from the base ring of `self` to `R`.

OUTPUT:

`ProductProjectiveSpaces_point`

EXAMPLES:

```
sage: T.<x,y,z,u,v,w> = ProductProjectiveSpaces([1,1,1],ZZ)
sage: P = T.point([5,3,15,4,2,6]);
sage: P.change_ring(GF(3))
(1 : 0 , 0 : 1 , 1 : 0)
```

normalize_coordinates ()

Removes common factors (componentwise) from the coordinates of `self` (including -1).

OUTPUT:

None.

EXAMPLES:

```
sage: T.<x,y,z,u,v,w> = ProductProjectiveSpaces([2,2],ZZ)
sage: P = T.point([5,10,15,4,2,6]);
sage: P.normalize_coordinates()
sage: P
(1 : 2 : 3 , 2 : 1 : 3)
```

scale_by (*t*)

Scale the coordinates of the point `self` by t, done componentwise.

A `TypeError` occurs if the point is not in the base ring of the codomain after scaling.

INPUT:

•`t` – a ring element

EXAMPLES:

```
sage: T.<x,y,z,u,v,w> = ProductProjectiveSpaces([1,1,1],ZZ)
sage: P = T.point([5,10,15,4,2,6]);
sage: P.scale_by([2,1,1])
sage: P
(10 : 20 , 15 : 4 , 2 : 6)
```

15.5 Wehler K3 Surfaces

AUTHORS:

- Ben Hutz (11-2012)
- Joao Alberto de Faria (10-2013)

  TODO:

  ```
  Riemann Zeta Function

  Picard Number

  Number Fields
  ```

  REFERENCES:

`sage.schemes.product_projective.wehlerK3.`**`WehlerK3Surface`**`(`*polys*`)`

Defines a K3 Surface over $\mathbb{P}^2 \times \mathbb{P}^2$ defined as the intersection of a bileaner and biquadratic form. [Wehl]

INPUT: Bilinear and Biquadratic polynomials as a tuple or list

OUTPUT: WehlerK3 Surface

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: L = x0*y0 + x1*y1 - x2*y2
sage: Q = x0*x1*y1^2 + x2^2*y0*y2
sage: WehlerK3Surface([L,Q])
Closed subscheme of Product of projective spaces P^2 x P^2 over Rational
Field defined by:
 x0*y0 + x1*y1 - x2*y2,
 x0*x1*y1^2 + x2^2*y0*y2
```

class `sage.schemes.product_projective.wehlerK3.`**`WehlerK3Surface_field`**`(`*polys*`)`

Bases: `sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring`

class `sage.schemes.product_projective.wehlerK3.`**`WehlerK3Surface_finite_field`**`(`*polys*`)`

Bases: `sage.schemes.product_projective.wehlerK3.WehlerK3Surface_field`

`cardinality``()`

Counts the total number of points on the K3 surface

ALGORITHM:

Enumerate points over $\mathbb{P}^2$, and then count the points on the fiber of each of those points

OUTPUT: Integer, total number of points on the surface

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],GF(7))
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 +\
3*x0*x1*y0*y1 -2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2\
- 4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2\
+ x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: X.cardinality()
55
```

class sage.schemes.product_projective.wehlerK3.**WehlerK3Surface_ring**(*polys*)

Bases: sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_product_projective

A K3 surface in $\mathbb{P}^2 \times \mathbb{P}^2$ defined as the intersection of a bileaner and biquadratic form. [Wehl]

EXAMPLES:

```
sage: R.<x,y,z,u,v,w> = PolynomialRing(QQ,6)
sage: L = x*u-y*v
sage: Q = x*y*v^2 + z^2*u*w
sage: WehlerK3Surface([L,Q])
Closed subscheme of Product of projective spaces P^2 x P^2 over Rational
Field defined by:
  x*u - y*v,
  x*y*v^2 + z^2*u*w
```

Gpoly(*component*, *k*)

Returns the G polynomials defined by $G_k^* = \left(L_j^*\right)^2 Q_{ii}^* - L_i^* L_j^* Q_{ij}^* + \left(L_i^*\right)^2 Q_{jj}^*$ where {i,j,k} is some permutation of (0,1,2) and * is either x (Component = 1) or y (Component = 0)

INPUT:

- component - Integer: 0 or 1
- k - Integer: 0, 1 or 2

OUTPUT: Polynomial in terms of either y (Component = 0) or x (Component = 1)

EXAMPLES:

```
sage: R.<x0,x1,x2,y0,y1,y2> = PolynomialRing(ZZ,6)
sage: Y = x0*y0 + x1*y1 - x2*y2
sage: Z = x0^2*y0*y1 + x0^2*y2^2 - x0*x1*y1*y2 + x1^2*y2*y1 +\
x2^2*y2^2 + x2^2*y1^2 + x1^2*y2^2
sage: X = WehlerK3Surface([Z,Y])
sage: X.Gpoly(1,0)
x0^2*x1^2 + x1^4 - x0*x1^2*x2 + x1^3*x2 + x1^2*x2^2 + x2^4
```

Hpoly(*component*, *i*, *j*)

Returns the H polynomials defined by $H_{ij}^* = 2L_i^* L_j^* Q_{kk}^* - L_i^* L_k^* Q_{jk}^* - L_j^* L_k^* Q_{ik}^* + \left(L_k^*\right)^2 Q_{ij}^*$ where {i,j,k} is some permutation of (0,1,2) and * is either y (Component = 0) or x (Component = 1)

INPUT:

- component - Integer: 0 or 1
- i - Integer: 0, 1 or 2
- j - Integer: 0, 1 or 2

OUTPUT: Polynomial in terms of either y (Component = 0) or x (Component = 1)

EXAMPLES:

```
sage: R.<x0,x1,x2,y0,y1,y2> = PolynomialRing(ZZ,6)
sage: Y = x0*y0 + x1*y1 - x2*y2
sage: Z = x0^2*y0*y1 + x0^2*y2^2 - x0*x1*y1*y2 + x1^2*y2*y1 +\
 x2^2*y2^2 + x2^2*y1^2 + x1^2*y2^2
sage: X = WehlerK3Surface([Z,Y])
sage: X.Hpoly(0,1,0)
 2*y0*y1^3 + 2*y0*y1*y2^2 - y1*y2^3
```

Lxa(*a*)

Function will return the L polynomial defining the fiber, given by:

$L_a^x = \{(a,y) \in \mathbb{P}^2 \times \mathbb{P}^2 : L(a,y) = 0\}$

Notation and definition from: [CaSi]

INPUT: `a` - Point in $\mathbb{P}^2$

OUTPUT: A polynomial representing the fiber

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1 -2*x2^2*y0*y1
 x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*
  + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP(1,1,0,1,0,0);
sage: X.Lxa(T[0])
y0 + y1
```

Lyb(b)

Function will return a fiber defined by:

$L_b^y = \{(x,b) \in \mathbb{P}^2 \times \mathbb{P}^2 : L(x,b) = 0\}$

Notation and definition from: [CaSi]

INPUT: `b` - Point in Projective Space

OUTPUT: A polynomial representing the fiber

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1\
- 2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2\
- 4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP(1,1,0,1,0,0);
sage: X.Lyb(T[1])
  x0
```

Qxa(a)

Function will return the Q polynomial defining a fiber. given by:

$Q_a^x = \{(a,y) \in \mathbb{P}^2 \times \mathbb{P}^2 : Q(a,y) = 0\}$

Notation and definition from: [CaSi]

INPUT: `a` - Point in $\mathbb{P}^2$

OUTPUT: A polynomial representing the fiber

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1\
-2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2\
-4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP(1,1,0,1,0,0);
sage: X.Qxa(T[0])
5*y0^2 + 7*y0*y1 + y1^2 + 11*y1*y2 + y2^2
```

Qyb (b)

Function will return a fiber defined by:

$Q^y_b = \{(x, b) \in \mathbb{P}^2 \times \mathbb{P}^2 : Q(x, b) = 0\}$

Notation and definition from: [CaSi]

INPUT: `b` - Point in Projective Space

OUTPUT: A polynomial representing the fiber

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + \
3*x0*x1*y0*y1 -2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2\
- 4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2\
+ x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP(1,1,0,1,0,0);
sage: X.Qyb(T[1])
x0^2 + 3*x0*x1 + x1^2
```

Ramification_poly (i)

Function will return the Ramification polynomial defined by: $g^* = \frac{(H^*_{ij})^2 - 4G^*_iG^*_j}{(L^*_k)^2}$ The roots of this polynomial will either be degenerate fibers or fixed points of the involutions σ_x or σ_y for more information, see [CaSi]

INPUT: `i` - Integer, either 0 (polynomial in y) or 1 (polynomial in x)

OUTPUT: polynomial in the coordinate ring of the ambient space

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1\
- 2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2\
- 4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: X.Ramification_poly(0)
8*y0^5*y1 - 24*y0^4*y1^2 + 48*y0^2*y1^4 - 16*y0*y1^5 + y1^6 + 84*y0^3*y1^2*y2
+ 46*y0^2*y1^3*y2 - 20*y0*y1^4*y2 + 16*y1^5*y2 + 53*y0^4*y2^2 + 56*y0^3*y1*y2^2
- 32*y0^2*y1^2*y2^2 - 80*y0*y1^3*y2^2 - 92*y1^4*y2^2 - 12*y0^2*y1*y2^3
- 168*y0*y1^2*y2^3 - 122*y1^3*y2^3 + 14*y0^2*y2^4 + 8*y0*y1*y2^4 - 112*y1^2*y2^4 + y2^6
```

Sxa (a)

Function will return fiber defined by:

$S^x_a = L^x_a \cap Q^x_a$

Notation and definition from: [CaSi]

INPUT: `a` - Point in $\mathbb{P}^2$

OUTPUT: A subscheme representing the fiber

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1\
- 2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2\
- 4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP(1,1,0,1,0,0);
sage: X.Sxa(T[0])
  Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
   y0 + y1,
   5*y0^2 + 7*y0*y1 + y1^2 + 11*y1*y2 + y2^2
```

Syb (*b*)

Function will return fiber defined by:

$S^y_b = L^y_b \cap Q^y_b$

Notation and definition from: [CaSi]

INPUT: `b` - Point in $\mathbb{P}^2$

OUTPUT: A subscheme representing the fiber

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + \
3*x0*x1*y0*y1 -2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2\
- 4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2\
+ x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP(1,1,0,1,0,0);
sage: X.Syb(T[1])
 Closed subscheme of Projective Space of dimension 2 over Rational Field defined by:
   x0,
   x0^2 + 3*x0*x1 + x1^2
```

canonical_height (*P*, *N*, *badprimes=None*, *prec=100*)

Evaluates the canonical height for `P` with `N` terms of the series of the local heights.

ALGORITHM:

The sum of the canonical height minus and canonical height plus, for more info see section 4 of [CaSi]

INPUT:

- `P` - a surface point
- `N` - positive integer. number of terms of the series to use
- `badprimes` - list of integer primes (where the surface is degenerate) (optional)
- `prec` - float point or p-adic precision, default: 100

OUTPUT: a real number

EXAMPLES:

```
sage: set_verbose(None)
sage: R.<x0,x1,x2,y0,y1,y2> = PolynomialRing(QQ,6)
sage: L =  (-y0 - y1)*x0 + (-y0*x1 - y2*x2)
sage: Q = (-y2*y0 - y1^2)*x0^2 + ((-y0^2 - y2*y0 + (-y2*y1 - y2^2))*x1 +\
(-y0^2 - y2*y1)*x2)*x0 + ((-y0^2 - y2*y0 - y2^2)*x1^2 + (-y2*y0 - y1^2)*x2*x1\
```

```
+ (-y0^2 + (-y1 - y2)*y0)*x2^2)
sage: X = WehlerK3Surface([L,Q])
sage: P = X([1,0,-1,1,-1,0]) #order 16
sage: X.canonical_height(P,5)  # long time
0.00000000000000000000000000000
```

Call-Silverman example:

```
sage: set_verbose(None)
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1 -\
2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2\
-4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: P = X(0,1,0,0,0,1)
sage: X.canonical_height(P,4)
0.69826458668659859569990618895
```

canonical_height_minus(*P*, *N*, *badprimes=None*, *prec=100*)

Evaluates the canonical height minus function of Call-Silverman for P with N terms of the series of the local heights. Must be over $\mathbf{Z}$ or $\mathbf{Q}$.

ALGORITHM:

Sum over the lambda minus heights (local heights) in a convergent series, for more detail see section 7 of [CaSi]

INPUT:

- P - a surface point
- N - positive integer. number of terms of the series to use
- badprimes - list of integer primes (where the surface is degenerate) (optional)
- prec - float point or p-adic precision, default: 100

OUTPUT: a real number

EXAMPLES:

```
sage: set_verbose(None)
sage: R.<x0,x1,x2,y0,y1,y2> = PolynomialRing(QQ,6)
sage: L =  (-y0 - y1)*x0 + (-y0*x1 - y2*x2)
sage: Q = (-y2*y0 - y1^2)*x0^2 + ((-y0^2 - y2*y0 + (-y2*y1 - y2^2))*x1\
 + (-y0^2 - y2*y1)*x2)*x0 + ((-y0^2 - y2*y0 - y2^2)*x1^2 + (-y2*y0 - y1^2)*x2*x1\
  + (-y0^2 + (-y1 - y2)*y0)*x2^2)
sage: X = WehlerK3Surface([L,Q])
sage: P = X([1,0,-1,1,-1,0]) #order 16
sage: X.canonical_height_minus(P,5)  # long time
0.00000000000000000000000000000
```

Call-Silverman example:

```
sage: set_verbose(None)
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 +\
 3*x0*x1*y0*y1 -2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -\
 4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 +\
  x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
```

```
sage: X = WehlerK3Surface([Z,Y])
sage: P = X([0,1,0,0,0,1])
sage: X.canonical_height_minus(P,4) # long time
0.55073705369676788175590206734
```

canonical_height_plus (*P*, *N*, *badprimes=None*, *prec=100*)

Evaluates the canonical height plus function of Call-Silverman for P with N terms of the series of the local heights. Must be over $\mathbf{Z}$ or $\mathbf{Q}$.

ALGORITHM:

Sum over the lambda plus heights (local heights) in a convergent series, for more detail see section 7 of [CaSi]

INPUT:

- P - a surface point
- N - positive integer. number of terms of the series to use
- badprimes - list of integer primes (where the surface is degenerate) (optional)
- prec - float point or p-adic precision, default: 100

OUTPUT: a real number

EXAMPLES:

```
sage: set_verbose(None)
sage: R.<x0,x1,x2,y0,y1,y2> = PolynomialRing(QQ,6)
sage: L =  (-y0 - y1)*x0 + (-y0*x1 - y2*x2)
sage: Q = (-y2*y0 - y1^2)*x0^2 + ((-y0^2 - y2*y0 + (-y2*y1 - y2^2))*x1 + \
(-y0^2 - y2*y1)*x2)*x0 + ((-y0^2 - y2*y0 - y2^2)*x1^2 + (-y2*y0 - y1^2)*x2*x1\
+ (-y0^2 + (-y1 - y2)*y0)*x2^2)
sage: X = WehlerK3Surface([L,Q])
sage: P = X([1,0,-1,1,-1,0]) #order 16
sage: X.canonical_height_plus(P,5)  # long time
0.00000000000000000000000000000
```

Call-Silverman Example:

```
sage: set_verbose(None)
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 +\
3*x0*x1*y0*y1 - 2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2\
- 4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2\
+ x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: P = X([0,1,0,0,0,1])
sage: X.canonical_height_plus(P,4) # long time
0.14752753298983071394400412161
```

change_ring (*R*)

Changes the base ring on which the Wehler K3 Surface is defined

INPUT: R - ring

OUTPUT: WehlerK3 Surface defined over input ring

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],GF(3))
sage: L = x0*y0 + x1*y1 - x2*y2
sage: Q = x0*x1*y1^2 + x2^2*y0*y2
sage: W = WehlerK3Surface([L,Q])
sage: W.base_ring()
Finite Field of size 3
sage: T = W.change_ring(GF(7))
sage: T.base_ring()
Finite Field of size 7
```

degenerate_fibers()

Function will return the (rational) degenerate fibers of the surface defined over the base ring, or the fraction field of the base ring if it is not a field.

ALGORITHM:

The criteria for degeneracy by the common vanishing of the polynomials `self.Gpoly(1,0)`, `self.Gpoly(1,1)`, `self.Gpoly(1,2)`, `self.Hpoly(1,0,1)`, `self.Hpoly(1,0,2)`, `self.Hpoly(1,1,2)` (for the first component), is from Proposition 1.4 in the following article: [CaSi]. This function finds the common solution through elimination via Groebner bases by using the .variety() function on the three affine charts in each component.

OUTPUT: The output is a list of lists where the elements of lists are points in the appropriate projective space.
The first list is the points whose pullback by the projection to the first component (projective space) is dimension greater than 0. The second list is points in the second component.

EXAMPLES:

```
sage: R.<x0,x1,x2,y0,y1,y2> = PolynomialRing(ZZ,6)
sage: Y = x0*y0 + x1*y1 - x2*y2
sage: Z = x0^2*y0*y1 + x0^2*y2^2 - x0*x1*y1*y2 + x1^2*y2*y1 + x2^2*y2^2\
+ x2^2*y1^2 + x1^2*y2^2
sage: X = WehlerK3Surface([Z,Y])
sage: X.degenerate_fibers()
[[], [(1 : 0 : 0)]]

sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1\
- 2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2\
- 4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: X.degenerate_fibers()
[[], []]

sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: R = PP.coordinate_ring()
sage: l = y0*x0 + y1*x1 + (y0 - y1)*x2
sage: q = (y1*y0 + y2^2)*x0^2 + ((y0^2 - y2*y1)*x1 + (y0^2 + (y1^2 - y2^2))*x2)*x0 \
+ (y2*y0 + y1^2)*x1^2 + (y0^2 + (-y1^2 + y2^2))*x2*x1
sage: X = WehlerK3Surface([l,q])
sage: X.degenerate_fibers()
[[(-1 : 1 : 1), (0 : 0 : 1)], [(-1 : -1 : 1), (0 : 0 : 1)]]
```

degenerate_primes(*check=True*)

Determine which primes p self has degenerate fibers over $GF(p)$. If check is False, then may return primes that do not have degenerate fibers. Raises an error if the surface is degenerate. Works only for `ZZ` or `QQ`.

INPUT: `check` - Boolean (Default: True) then the primes are verified

ALGORITHM:

p is a prime of bad reduction if and only if the defining polynomials of self plus the G and H polynomials have a common zero. Or stated another way, p is a prime of bad reducion if and only if the radical of the ideal defined by the defining polynomials of self plus the G and H polynomials is not $(x_0, x_1, \ldots, x_N)$. This happens if and only if some power of each x_i is not in the ideal defined by the defining polynomials of self (with G and H). This last condition is what is checked. The lcm of the coefficients of the monomials x_i in a groebner basis is computed. This may return extra primes.

OUTPUT: list of primes.

EXAMPLES:

```
sage: R.<x0,x1,x2,y0,y1,y2> = PolynomialRing(QQ,6)
sage: L =  y0*x0 + (y1*x1 + y2*x2)
sage: Q = (2*y0^2 + y2*y0 + (2*y1^2 + y2^2))*x0^2 + ((y0^2 + y1*y0 +\
(y1^2 + 2*y2*y1 + y2^2))*x1 + (2*y1^2 + y2*y1 + y2^2)*x2)*x0 + ((2*y0^2\
+ (y1 + 2*y2)*y0 + (2*y1^2 + y2*y1))*x1^2 + ((2*y1 + 2*y2)*y0 + (y1^2 +\
y2*y1 + 2*y2^2))*x2*x1 + (2*y0^2 + y1*y0 + (2*y1^2 + y2^2))*x2^2)
sage: X = WehlerK3Surface([L,Q])
sage: X.degenerate_primes()
[2, 3, 5, 11, 23, 47, 48747691, 111301831]
```

fiber (*p*, *component*)

Returns the fibers [y (component = 1) or x (Component = 0)] of a point on a K3 Surface, will work for nondegenerate fibers only. For algorithm, see [Hutzthesis]

INPUT: p - a point in $\mathbb{P}^2$

OUTPUT: The corresponding fiber (as a list)

EXAMPLES:

```
sage: R.<x0,x1,x2,y0,y1,y2> = PolynomialRing(ZZ,6)
sage: Y = x0*y0 + x1*y1 - x2*y2
sage: Z = y0^2*x0*x1 + y0^2*x2^2 - y0*y1*x1*x2 + y1^2*x2*x1 + y2^2*x2^2 +\
y2^2*x1^2 + y1^2*x2^2
sage: X = WehlerK3Surface([Z,Y])
sage: Proj = ProjectiveSpace(QQ,2)
sage: P = Proj([1,0,0])
sage: X.fiber(P, 1)
Traceback (most recent call last):
...
TypeError: Fiber is degenerate

sage: P.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1 - \
2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2 - \
4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: Proj = P[0]
sage: T = Proj([0,0,1])
sage: X.fiber(T,1)
[(0 : 0 : 1 , 0 : 1 : 0), (0 : 0 : 1 , 2 : 0 : 0)]

sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],GF(7))
sage: L = x0*y0 + x1*y1 - 1*x2*y2
sage: Q=(2*x0^2 + x2*x0 + (2*x1^2 + x2^2))*y0^2 + ((x0^2 + x1*x0 +(x1^2 + 2*x2*x1 + x2^2))*y
(2*x1^2 + x2*x1 + x2^2)*y2)*y0 + ((2*x0^2+ (x1 + 2*x2)*x0 + (2*x1^2 + x2*x1))*y1^2 + ((2*x1
(x1^2 +x2*x1 + 2*x2^2))*y2*y1 + (2*x0^2 + x1*x0 + (2*x1^2 + x2^2))*y2^2)
```

```
sage: W = WehlerK3Surface([L,Q])
sage: W.fiber([4,0,1],0)
[(0 : 1 : 0 , 4 : 0 : 1), (4 : 0 : 2 , 4 : 0 : 1)]
```

is_degenerate()

Function will return True if there is a fiber (over the algebraic closure of the base ring) of dimension greater than 0 and False otherwise.

OUTPUT: Boolean value of True or False

EXAMPLES:

```
sage: R.<x0,x1,x2,y0,y1,y2> = PolynomialRing(ZZ,6)
sage: Y = x0*y0 + x1*y1 - x2*y2
sage: Z = x0^2*y0*y1 + x0^2*y2^2 - x0*x1*y1*y2 + x1^2*y2*y1 + x2^2*y2^2 + \
x2^2*y1^2 + x1^2*y2^2
sage: X = WehlerK3Surface([Z,Y])
sage: X.is_degenerate()
True

sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1 - \
2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2 - \
4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: X.is_degenerate()
False

sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],GF(3))
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1 - \
2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2 - \
4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: X.is_degenerate()
True
```

is_isomorphic(*right*)

Checks to see if two K3 surfaces have the same defining ideal

INPUT: `right` - the K3 surface to compare to the original

OUTPUT: Boolean

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 +\
3*x0*x1*y0*y1 -2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2\
-4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2\
+ x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: W = WehlerK3Surface([Z + Y^2,Y])
sage: X.is_isomorphic(W)
True

sage: R.<x,y,z,u,v,w> = PolynomialRing(QQ,6)
sage: L = x*u-y*v
sage: Q = x*y*v^2 + z^2*u*w
```

```
sage: W1 = WehlerK3Surface([L,Q])
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: L = x0*y0 + x1*y1 + x2*y2
sage: Q = x1^2*y0^2 + 2*x2^2*y0*y1 + x0^2*y1^2 -x0*x1*y2^2
sage: W2 = WehlerK3Surface([L,Q])
sage: W1.is_isomorphic(W2)
False
```

is_smooth()

Function will return the status of the smoothness of the surface

ALGORITHM:

Checks to confirm that all of the 2x2 minors of the jacobian generated from the Biquadratic and Bilinear forms have
no common vanishing points.

OUTPUT: Boolean

EXAMPLES:

```
sage: R.<x0,x1,x2,y0,y1,y2> = PolynomialRing(ZZ,6)
sage: Y = x0*y0 + x1*y1 - x2*y2
sage: Z = x0^2*y0*y1 + x0^2*y2^2 - x0*x1*y1*y2 + x1^2*y2*y1 +\
 x2^2*y2^2 + x2^2*y1^2 + x1^2*y2^2
sage: X = WehlerK3Surface([Z,Y])
sage: X.is_smooth()
False

sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 +\
3*x0*x1*y0*y1 -2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2\
- 4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2\
+ x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: X.is_smooth()
True
```

is_symmetric_orbit(*orbit*)

Checks to see if the orbit is symmetric (i.e. if one of the points on the orbit is fixed by 'sigma_x' or 'sigma_y')

INPUT: `orbit` a periodic cycle of either psi or phi.

OUTPUT: Boolean

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],GF(7))
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1\
-2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2\
-4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP([0,0,1,1,0,0])
sage: orbit = X.orbit_psi(T,4)
sage: X.is_symmetric_orbit(orbit)
True

sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: L = x0*y0 + x1*y1 + x2*y2
```

```
sage: Q = x1^2*y0^2 + 2*x2^2*y0*y1 + x0^2*y1^2 - x0*x1*y2^2
sage: W = WehlerK3Surface([L,Q])
sage: T = W([-1,-1,1,1,0,1])
sage: Orb = W.orbit_phi(T,7)
sage: W.is_symmetric_orbit(Orb)
False
```

lambda_minus (*P*, *v*, *N*, *m*, *n*, *prec=100*)

Evaluates the local canonical height minus function of Call-Silverman at the place `v` for `P` with `N` terms of the series. Use `v = 0` for the archimedean place. Must be over **Z** or **Q**.

ALGORITHM:

Sum over local heights using convergent series, for more details, see section 4 of [CaSi]

INPUT:

- `P` - a projective point
- `N` - positive integer. number of terms of the series to use
- `v` - non-negative integer. a place, use v = 0 for the archimedean place
- **`m,n` - positive integers, We compute the local height for the divisor E^+_{mn}.** These must be indices of non-zero coordinates of the point `P`.
- `prec` - float point or p-adic precision, default: 100

OUTPUT: a real number

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1 \
- 2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2\
- 4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: P = X([0,0,1,1,0,0])
sage: X.lambda_minus(P,2,20,2,0,200)
-0.18573351672047135037172805779671791488351056677474271893705
```

lambda_plus (*P*, *v*, *N*, *m*, *n*, *prec=100*)

Evaluates the local canonical height plus function of Call-Silverman at the place `v` for `P` with `N` terms of the series. Use `v = 0` for the archimedean place. Must be over **Z** or **Q**.

ALGORITHM:

Sum over local heights using convergent series, for more details, see section 4 of [CaSi]

INPUT:

- `P` - a surface point
- `N` - positive integer. number of terms of the series to use
- `v` - non-negative integer. a place, use v = 0 for the archimedean place
- **`m,n` - positive integers, We compute the local height for the divisor E^+_{mn}.** These must be indices of non-zero coordinates of the point `P`.
- `prec` - float point or p-adic precision, default: 100

OUTPUT: a real number

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + 3*x0*x1*y0*y1\
- 2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -4*x1*x2*y1^2 + 5*x0*x2*y0*y2\
- 4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: P = X([0,0,1,1,0,0])
sage: X.lambda_plus(P,0,10,2,0)
0.89230705169161608922595928129
```

nth_iterate_phi(*P*, *n*, ***kwds*)

Computes the nth iterate for the phi function

INPUT:

- P – - a point in $\mathbb{P}^2 \times \mathbb{P}^2$
- n – an integer.

kwds:

- check - Boolean (optional - default: True) checks to see if point is on the surface
- normalize – boolean (optional - default: False) normalizes the point

OUTPUT:

The nth iterate of the point given the phi function (if n is positive), or the psi function (if n is negative)

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: L = x0*y0 + x1*y1 + x2*y2
sage: Q = x1^2*y0^2 + 2*x2^2*y0*y1 + x0^2*y1^2 - x0*x1*y2^2
sage: W = WehlerK3Surface([L,Q])
sage: T = W([-1,-1,1,1,0,1])
sage: W.nth_iterate_phi(T,7)
(-1 : 0 : 1 , 1 : -2 : 1)
```

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: L = x0*y0 + x1*y1 + x2*y2
sage: Q = x1^2*y0^2 + 2*x2^2*y0*y1 + x0^2*y1^2 - x0*x1*y2^2
sage: W = WehlerK3Surface([L,Q])
sage: T = W([-1,-1,1,1,0,1])
sage: W.nth_iterate_phi(T,-7)
(1 : 0 : 1 , -1 : 2 : 1)
```

```
sage: R.<x0,x1,x2,y0,y1,y2>=PolynomialRing(QQ,6)
sage: L = (-y0 - y1)*x0 + (-y0*x1 - y2*x2)
sage: Q = (-y2*y0 - y1^2)*x0^2 + ((-y0^2 - y2*y0 + (-y2*y1 - y2^2))*x1 + (-y0^2 - y2*y1)*x2)
+ ((-y0^2 - y2*y0 - y2^2)*x1^2 + (-y2*y0 - y1^2)*x2*x1 + (-y0^2 + (-y1 - y2)*y0)*x2^2)
sage: X = WehlerK3Surface([L,Q])
sage: P = X([1,0,-1,1,-1,0])
sage: X.nth_iterate_phi(P,8) == X.nth_iterate_psi(P,8)
True
```

nth_iterate_psi(*P*, *n*, ***kwds*)

Computes the nth iterate for the psi function

INPUT:

- P – - a point in $\mathbb{P}^2 \times \mathbb{P}^2$
- n – an integer.

kwds:

- check - Boolean (optional - default: True) checks to see if point is on the surface
- normalize – boolean (optional - default: False) normalizes the point

OUTPUT:

The nth iterate of the point given the psi function (if n is positive), or the phi function (if n is negative)

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: L = x0*y0 + x1*y1 + x2*y2
sage: Q = x1^2*y0^2 + 2*x2^2*y0*y1 + x0^2*y1^2 - x0*x1*y2^2
sage: W = WehlerK3Surface([L,Q])
sage: T = W([-1,-1,1,1,0,1])
sage: W.nth_iterate_psi(T,-7)
(-1 : 0 : 1 , 1 : -2 : 1)
```

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: L = x0*y0 + x1*y1 + x2*y2
sage: Q = x1^2*y0^2 + 2*x2^2*y0*y1 + x0^2*y1^2 - x0*x1*y2^2
sage: W = WehlerK3Surface([L,Q])
sage: T = W([-1,-1,1,1,0,1])
sage: W.nth_iterate_psi(T,7)
(1 : 0 : 1 , -1 : 2 : 1)
```

orbit_phi (*P*, *N*, ***kwds*)

Returns the orbit of the ϕ function defined by $\phi = \sigma_y \circ \sigma_x$ Function is defined in [CaSi]

INPUT:

- P - Point on the K3 surface
- N - a non-negative integer or list or tuple of two non-negative integers

kwds:

- check - Boolean (optional - default: True) checks to see if point is on the surface
- normalize – boolean (optional - default: False) normalizes the point

OUTPUT: List of points in the orbit

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 +\
3*x0*x1*y0*y1 -2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -\
4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 +\
x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP(0,0,1,1,0,0)
sage: X.orbit_phi(T,2, normalize = True)
[(0 : 0 : 1 , 1 : 0 : 0), (-1 : 0 : 1 , 0 : 1 : 0), (-12816/6659 : 55413/6659 : 1 , 1 : 1/9
sage: X.orbit_phi(T,[2,3], normalize = True)
[(-12816/6659 : 55413/6659 : 1 , 1 : 1/9 : 1),
(7481279673854775690938629732119966552954626693713001783595660989241/18550615454277582153932
```

```
: 3992260691327218828582255586014718568398539828275296031491644987908/185506154542775821539
1 , -117756062505511/54767410965117 : -23134047983794359/37466994368025041 : 1)]
```

orbit_psi (*P*, *N*, ***kwds*)

Returns the orbit of the ψ function defined by $\psi = \sigma_x \circ \sigma_y$

Function is defined in [CaSi]

INPUT:

- `P` - Point on the K3 surface
- `N` - a non-negative integer or list or tuple of two non-negative integers

kwds:

- `check` - Boolean (optional - default: `True`) checks to see if point is on the surface
- `normalize` – boolean (optional - default: `False`) normalizes the point

OUTPUT: List of points in the orbit

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 +\
 3*x0*x1*y0*y1 -2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -\
 4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2 +\
  x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP(0,0,1,1,0,0)
sage: X.orbit_psi(T,2, normalize = True)
[(0 : 0 : 1 , 1 : 0 : 0), (0 : 0 : 1 , 0 : 1 : 0), (-1 : 0 : 1 , 1 : 1/9 : 1)]
sage: X.orbit_psi(T,[2,3], normalize = True)
[(-1 : 0 : 1 , 1 : 1/9 : 1),
(-12816/6659 : 55413/6659 : 1 , -117756062505511/54767410965117 : -23134047983794359/3746699
```

phi (*a*, ***kwds*)

Evaluates the function $\phi = \sigma_y \circ \sigma_x$

ALGORITHM:

Refer to Section 6: "An algorithm to compute σ_x, σ_y, ϕ, and ψ" in [CaSi]

For the degenerate case refer to [FaHu]

INPUT:

- `a` - Point in $\mathbb{P}^2 \times \mathbb{P}^2$

kwds:

- `check` - Boolean (optional - default: `True`) checks to see if point is on the surface
- `normalize` – boolean (optional - default: `True`) normalizes the point

OUTPUT: a point on `self`

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + \
3*x0*x1*y0*y1 -2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2\
- 4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2\
+ x0*x1*y2^2 + 3*x2^2*y2^2
```

```
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP([0,0,1,1,0,0])
sage: X.phi(T)
(-1 : 0 : 1 , 0 : 1 : 0)
```

psi (*a*, ***kwds*)

Evaluates the function $\psi = \sigma_x \circ \sigma_y$

ALGORITHM:

Refer to Section 6: "An algorithm to compute σ_x, σ_y, ϕ, and ψ" in

[CaSi]

For the degenerate case refer to [FaHu]

INPUT:

- `a` - Point in $\mathbb{P}^2 \times \mathbb{P}^2$

kwds:

- `check` - Boolean (optional - default: `True`) checks to see if point is on the surface
- `normalize` – boolean (optional - default: `True`) normalizes the point

OUTPUT: a point on `self`

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 + \
3*x0*x1*y0*y1 -2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2\
- 4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2\
+ x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP([0,0,1,1,0,0])
sage: X.psi(T)
(0 : 0 : 1 , 0 : 1 : 0)
```

sigmaX (*P*, ***kwds*)

Function returns the involution on the surface `self` induced by the double covers. In particular, it fixes the projection to the first coordinate and swaps the two points in the fiber, i.e. $(x, y) \to (x, y')$. Note that in the degenerate case, while we can split fiber into pairs of points, it is not always possible to distinguish them, using this algorithm.

ALGORITHM:

Refer to Section 6: "An algorithm to compute σ_x, σ_y, ϕ, and ψ" in [CaSi] For the degenerate case refer to [FaHu]

INPUT:

- `P` - a point in $\mathbb{P}^2 \times \mathbb{P}^2$

kwds:

- `check` - Boolean (optional - default: `True`) checks to see if point is on the surface
- `normalize` – boolean (optional - default: `True`) normalizes the point

OUTPUT: a point on `self`

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 +\
3*x0*x1*y0*y1 -2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2 -\
4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 +\
4*x1^2*y1*y2 + x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP(0,0,1,1,0,0)
sage: X.sigmaX(T)
(0 : 0 : 1 , 0 : 1 : 0)
```

degenerate examples:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: l = y0*x0 + y1*x1 + (y0 - y1)*x2
sage: q = (y1*y0)*x0^2 + ((y0^2)*x1 + (y0^2 + (y1^2 - y2^2))*x2)*x0\
+ (y2*y0 + y1^2)*x1^2 + (y0^2 + (-y1^2 + y2^2))*x2*x1
sage: X = WehlerK3Surface([l,q])
sage: X.sigmaX(X([1,0,0,0,1,-2]))
(1 : 0 : 0 , 0 : 1/2 : 1)
sage: X.sigmaX(X([1,0,0,0,0,1]))
(1 : 0 : 0 , 0 : 0 : 1)
sage: X.sigmaX(X([-1,1,1,-1,-1,1]))
(-1 : 1 : 1 , 2 : 2 : 1)
sage: X.sigmaX(X([0,0,1,1,1,0]))
(0 : 0 : 1 , 1 : 1 : 0)
sage: X.sigmaX(X([0,0,1,1,1,1]))
(0 : 0 : 1 , -1 : -1 : 1)
```

Case where we cannot distinguish the two points:

```
sage: PP.<y0,y1,y2,x0,x1,x2>=ProductProjectiveSpaces([2,2],GF(3))
sage: l = x0*y0 + x1*y1 + x2*y2
sage: q=-3*x0^2*y0^2 + 4*x0*x1*y0^2 - 3*x0*x2*y0^2 - 5*x0^2*y0*y1 - \
190*x0*x1*y0*y1- 5*x1^2*y0*y1 + 5*x0*x2*y0*y1 + 14*x1*x2*y0*y1 + \
5*x2^2*y0*y1 - x0^2*y1^2 - 6*x0*x1*y1^2- 2*x1^2*y1^2 + 2*x0*x2*y1^2 - \
4*x2^2*y1^2 + 4*x0^2*y0*y2 - x1^2*y0*y2 + 3*x0*x2*y0*y2+ 6*x1*x2*y0*y2 - \
6*x0^2*y1*y2 - 4*x0*x1*y1*y2 - x1^2*y1*y2 + 51*x0*x2*y1*y2 - 7*x1*x2*y1*y2 - \
9*x2^2*y1*y2 - x0^2*y2^2 - 4*x0*x1*y2^2 + 4*x1^2*y2^2 - x0*x2*y2^2 + 13*x1*x2*y2^2 - x2^2*y2
sage: X = WehlerK3Surface([l,q])
sage: P = X([1,0,0,0,1,1])
sage: X.sigmaX(X.sigmaX(P))
Traceback (most recent call last):
...
ValueError: Cannot distinguish points in the degenerate fiber
```

sigmaY(*P*, ***kwds*)

Function returns the involution on the surface `self` induced by the double covers. In particular, it fixes the projection to the second coordinate and swaps the two points in the fiber, i.e. $(x, y) \to (x', y)$. Note that in the degenerate case, while we can split the fiber into two points, it is not always possible to distinguish them, using this algorithm.

ALGORITHM:

Refer to Section 6: "An algorithm to compute σ_x, σ_y, ϕ, and ψ" in [CaSi] For the degenerate case refer to [FaHu]

INPUT:

- P - a point in $\mathbb{P}^2 \times \mathbb{P}^2$

kwds:

- check - Boolean (optional - default: True) checks to see if point is on the surface
- normalize – boolean (optional - default: True) normalizes the point

OUTPUT: a point on self

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: Z = x0^2*y0^2 + 3*x0*x1*y0^2 + x1^2*y0^2 + 4*x0^2*y0*y1 +\
3*x0*x1*y0*y1 -2*x2^2*y0*y1 - x0^2*y1^2 + 2*x1^2*y1^2 - x0*x2*y1^2\
- 4*x1*x2*y1^2 + 5*x0*x2*y0*y2 -4*x1*x2*y0*y2 + 7*x0^2*y1*y2 + 4*x1^2*y1*y2\
+ x0*x1*y2^2 + 3*x2^2*y2^2
sage: Y = x0*y0 + x1*y1 + x2*y2
sage: X = WehlerK3Surface([Z,Y])
sage: T = PP(0,0,1,1,0,0)
sage: X.sigmaY(T)
(0 : 0 : 1 , 1 : 0 : 0)
```

degenerate examples:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],QQ)
sage: l = y0*x0 + y1*x1 + (y0 - y1)*x2
sage: q = (y1*y0)*x0^2 + ((y0^2)*x1 + (y0^2 + (y1^2 - y2^2))*x2)*x0 +\
 (y2*y0 + y1^2)*x1^2 + (y0^2 + (-y1^2 + y2^2))*x2*x1
sage: X = WehlerK3Surface([l,q])
sage: X.sigmaY(X([1,-1,0,-1,-1,1]))
(1/10 : -1/10 : 1 , -1 : -1 : 1)
sage: X.sigmaY(X([0,0,1,-1,-1,1]))
(-4 : 4 : 1 , -1 : -1 : 1)
sage: X.sigmaY(X([1,2,0,0,0,1]))
(-3 : -3 : 1 , 0 : 0 : 1)
sage: X.sigmaY(X([1,1,1,0,0,1]))
(1 : 0 : 0 , 0 : 0 : 1)
```

Case where we cannot distinguish the two points:

```
sage: PP.<x0,x1,x2,y0,y1,y2>=ProductProjectiveSpaces([2,2],GF(3))
sage: l = x0*y0 + x1*y1 + x2*y2
sage: q=-3*x0^2*y0^2 + 4*x0*x1*y0^2 - 3*x0*x2*y0^2 - 5*x0^2*y0*y1 - 190*x0*x1*y0*y1\
- 5*x1^2*y0*y1 + 5*x0*x2*y0*y1 + 14*x1*x2*y0*y1 + 5*x2^2*y0*y1 - x0^2*y1^2 - 6*x0*x1*y1^2\
- 2*x1^2*y1^2 + 2*x0*x2*y1^2 - 4*x2^2*y1^2 + 4*x0^2*y0*y2 - x1^2*y0*y2 + 3*x0*x2*y0*y2\
+ 6*x1*x2*y0*y2 - 6*x0^2*y1*y2 - 4*x0*x1*y1*y2 - x1^2*y1*y2 + 51*x0*x2*y1*y2 - 7*x1*x2*y1*y2
- 9*x2^2*y1*y2 - x0^2*y2^2 - 4*x0*x1*y2^2 + 4*x1^2*y2^2 - x0*x2*y2^2 + 13*x1*x2*y2^2 - x2^2*
sage: X = WehlerK3Surface([l,q])
sage: P = X([0,1,1,1,0,0])
sage: X.sigmaY(X.sigmaY(P))
Traceback (most recent call last):
...
ValueError: Cannot distinguish points in the degenerate fiber
```

sage.schemes.product_projective.wehlerK3.**random_WehlerK3Surface**(*PP*)

Produces a random K3 surface in $\mathbb{P}^2 \times \mathbb{P}^2$ defined as the intersection of a bileaner and biquadratic form. [Wehl]

INPUT: Projective Space Cartesian Product

OUTPUT: WehlerK3 Surface

EXAMPLES:

```
sage: PP.<x0,x1,x2,y0,y1,y2> = ProductProjectiveSpaces([2,2],GF(3))
sage: random_WehlerK3Surface(PP)
Closed subscheme of Product of projective spaces P^2 x P^2 over Finite Field
 of size 3 defined by:
  x0*y0 + x1*y1 + x2*y2,
  -x1^2*y0^2 - x2^2*y0^2 + x0^2*y0*y1 - x0*x1*y0*y1 - x1^2*y0*y1 + x1*x2*y0*y1
  + x0^2*y1^2 + x0*x1*y1^2 - x1^2*y1^2 + x0*x2*y1^2 - x0^2*y0*y2 - x0*x1*y0*y2
  + x0*x2*y0*y2 + x1*x2*y0*y2 + x0*x1*y1*y2 - x1^2*y1*y2 - x1*x2*y1*y2 - x0^2*y2^2
  + x0*x1*y2^2 - x1^2*y2^2 - x0*x2*y2^2
```

CHAPTER

SIXTEEN

TORIC VARIETIES

16.1 Toric varieties

This module provides support for (normal) toric varieties, corresponding to `rational polyhedral fans`. See also `fano_variety` for a more restrictive class of (weak) Fano toric varieties.

An **excellent reference on toric varieties** is the book "Toric Varieties" by David A. Cox, John B. Little, and Hal Schenck [CLS].

The interface to this module is provided through functions `AffineToricVariety()` and `ToricVariety()`, although you may also be interested in `normalize_names()`.

Note: We do NOT build "general toric varieties" from affine toric varieties. Instead, we are using the quotient representation of toric varieties with the homogeneous coordinate ring (a.k.a. Cox's ring or the total coordinate ring). This description works best for simplicial fans of the full dimension.

REFERENCES:

AUTHORS:

- Andrey Novoseltsev (2010-05-17): initial version.
- Volker Braun (2010-07-24): Cohomology and characteristic classes added.

EXAMPLES:

We start with constructing the affine plane as an affine toric variety. First, we need to have a corresponding cone:

```
sage: quadrant = Cone([(1,0), (0,1)])
```

If you don't care about variable names and the base field, that's all we need for now:

```
sage: A2 = AffineToricVariety(quadrant)
sage: A2
2-d affine toric variety
sage: origin = A2(0,0)
sage: origin
[0 : 0]
```

Only affine toric varieties have points whose (homogeneous) coordinates are all zero.

```
sage: parent(origin)
Set of rational points of 2-d affine toric variety
```

As you can see, by default toric varieties live over the field of rational numbers:

```
sage: A2.base_ring()
Rational Field
```

While usually toric varieties are considered over the field of complex numbers, for computational purposes it is more convenient to work with fields that have exact representation on computers. You can also always do

```
sage: C2 = AffineToricVariety(quadrant, base_field=CC)
sage: C2.base_ring()
Complex Field with 53 bits of precision
sage: C2(1,2+i)
[1.00000000000000 : 2.00000000000000 + 1.00000000000000*I]
```

or even

```
sage: F = CC["a, b"].fraction_field()
sage: F.inject_variables()
Defining a, b
sage: A2 = AffineToricVariety(quadrant, base_field=F)
sage: A2(a,b)
[a : b]
```

OK, if you need to work only with affine spaces, `AffineSpace()` may be a better way to construct them. Our next example is the product of two projective lines realized as the toric variety associated to the `face fan` of the "diamond":

```
sage: diamond = lattice_polytope.cross_polytope(2)
sage: diamond.vertices_pc()
M( 1,  0),
M( 0,  1),
M(-1,  0),
M( 0, -1)
in 2-d lattice M
sage: fan = FaceFan(diamond)
sage: P1xP1 = ToricVariety(fan)
sage: P1xP1
2-d toric variety covered by 4 affine patches
sage: P1xP1.fan().rays()
M( 1,  0),
M( 0,  1),
M(-1,  0),
M( 0, -1)
in 2-d lattice M
sage: P1xP1.gens()
(z0, z1, z2, z3)
```

We got four coordinates - two for each of the projective lines, but their names are perhaps not very well chosen. Let's make (x, y) to be coordinates on the first line and (s, t) on the second one:

```
sage: P1xP1 = ToricVariety(fan, coordinate_names="x s y t")
sage: P1xP1.gens()
(x, s, y, t)
```

Now, if we want to define subschemes of this variety, the defining polynomials must be homogeneous in each of these pairs:

```
sage: P1xP1.inject_variables()
Defining x, s, y, t
sage: P1xP1.subscheme(x)
```

```
Closed subscheme of 2-d toric variety
covered by 4 affine patches defined by:
  x
sage: P1xP1.subscheme(x^2 + y^2)
Closed subscheme of 2-d toric variety
covered by 4 affine patches defined by:
  x^2 + y^2
sage: P1xP1.subscheme(x^2 + s^2)
Traceback (most recent call last):
...
ValueError: x^2 + s^2 is not homogeneous
on 2-d toric variety covered by 4 affine patches!
sage: P1xP1.subscheme([x^2*s^2 + x*y*t^2 +y^2*t^2, s^3 + t^3])
Closed subscheme of 2-d toric variety
covered by 4 affine patches defined by:
  x^2*s^2 + x*y*t^2 + y^2*t^2,
  s^3 + t^3
```

While we don't build toric varieties from affine toric varieties, we still can access the "building pieces":

```
sage: patch = P1xP1.affine_patch(2)
sage: patch
2-d affine toric variety
sage: patch.fan().rays()
M(1, 0),
M(0, 1)
in 2-d lattice M
sage: patch.embedding_morphism()
Scheme morphism:
  From: 2-d affine toric variety
  To:   2-d toric variety covered by 4 affine patches
  Defn: Defined on coordinates by sending [x : s] to
        [x : s : 1 : 1]
```

The patch above was specifically chosen to coincide with our representation of the affine plane before, but you can get the other three patches as well. (While any cone of a fan will correspond to an affine toric variety, the main interest is usually in the generating fans as "the biggest" affine subvarieties, and these are precisely the patches that you can get from `affine_patch()`.)

All two-dimensional toric varieties are "quite nice" because any two-dimensional cone is generated by exactly two rays. From the point of view of the corresponding toric varieties, this means that they have at worst quotient singularities:

```
sage: P1xP1.is_orbifold()
True
sage: P1xP1.is_smooth()
True
sage: TV = ToricVariety(NormalFan(diamond))
sage: TV.fan().rays()
N(-1,  1),
N( 1,  1),
N(-1, -1),
N( 1, -1)
in 2-d lattice N
sage: TV.is_orbifold()
True
sage: TV.is_smooth()
False
```

In higher dimensions worse things can happen:

```
sage: TV3 = ToricVariety(NormalFan(lattice_polytope.cross_polytope(3)))
sage: TV3.fan().rays()
N(-1, -1,  1),
N( 1, -1,  1),
N(-1,  1,  1),
N( 1,  1,  1),
N(-1, -1, -1),
N( 1, -1, -1),
N(-1,  1, -1),
N( 1,  1, -1)
in 3-d lattice N
sage: TV3.is_orbifold()
False
```

Fortunately, we can perform a (partial) resolution:

```
sage: TV3_res = TV3.resolve_to_orbifold()
sage: TV3_res.is_orbifold()
True
sage: TV3_res.fan().ngenerating_cones()
12
sage: TV3.fan().ngenerating_cones()
6
```

In this example we had to double the number of affine patches. The result is still singular:

```
sage: TV3_res.is_smooth()
False
```

You can resolve it further using `resolve()` method, but (at least for now) you will have to specify which rays should be inserted into the fan. See also `CPRFanoToricVariety()`, which can construct some other "nice partial resolutions."

The intersection theory on toric varieties is very well understood, and there are explicit algorithms to compute many quantities of interest. The most important tools are the `cohomology ring` and the `Chow group`. For d-dimensional compact toric varieties with at most orbifold singularities, the rational cohomology ring $H^*(X, \mathbf{Q})$ and the rational Chow ring $A^*(X, \mathbf{Q}) = A_{d-*}(X) \otimes \mathbf{Q}$ are isomorphic except for a doubling in degree. More precisely, the Chow group has the same rank

$$A_{d-k}(X) \otimes \mathbf{Q} \simeq H^{2k}(X, \mathbf{Q})$$

and the intersection in of Chow cycles matches the cup product in cohomology.

In this case, you should work with the cohomology ring description because it is much faster. For example, here is a weighted projective space with a curve of $\mathbf{Z}_3$-orbifold singularities:

```
sage: P4_11133 = toric_varieties.P4_11133()
sage: P4_11133.is_smooth(), P4_11133.is_orbifold()
(False, True)
sage: cone = P4_11133.fan(3)[8]
sage: cone.is_smooth(), cone.is_simplicial()
(False, True)
sage: HH = P4_11133.cohomology_ring();  HH
Rational cohomology ring of a 4-d CPR-Fano toric variety covered by 5 affine patches
sage: P4_11133.cohomology_basis()
(([1],), ([z4],), ([z4^2],), ([z4^3],), ([z4^4],))
```

Every cone defines a torus orbit closure, and hence a (co)homology class:

```
sage: HH.gens()
([3*z4], [3*z4], [z4], [z4], [z4])
sage: map(HH, P4_11133.fan(1))
[[3*z4], [3*z4], [z4], [z4], [z4]]
sage: map(HH, P4_11133.fan(4) )
[[9*z4^4], [9*z4^4], [9*z4^4], [9*z4^4], [9*z4^4]]
sage: HH(cone)
[3*z4^3]
```

We can compute intersection numbers by integrating top-dimensional cohomology classes:

```
sage: D = P4_11133.divisor(0)
sage: HH(D)
[3*z4]
sage: P4_11133.integrate( HH(D)^4 )
9
sage: P4_11133.integrate( HH(D) * HH(cone) )
1
```

Although computationally less efficient, we can do the same computations with the rational Chow group:

```
sage: AA = P4_11133.Chow_group(QQ)
sage: map(AA, P4_11133.fan(1))  # long time (5s on sage.math, 2012)
[( 0 | 0 | 0 | 3 | 0 ), ( 0 | 0 | 0 | 3 | 0 ), ( 0 | 0 | 0 | 1 | 0 ), ( 0 | 0 | 0 | 1 | 0 ), ( 0 | 0
sage: map(AA, P4_11133.fan(4))  # long time (5s on sage.math, 2012)
[( 1 | 0 | 0 | 0 | 0 ), ( 1 | 0 | 0 | 0 | 0 ), ( 1 | 0 | 0 | 0 | 0 ), ( 1 | 0 | 0 | 0 | 0 ), ( 1 | 0
sage: AA(cone).intersection_with_divisor(D)  # long time (4s on sage.math, 2013)
( 1 | 0 | 0 | 0 | 0 )
sage: AA(cone).intersection_with_divisor(D).count_points()  # long time
1
```

The real advantage of the Chow group is that

- it works just as well over $\mathbf{Z}$, so torsion information is also easily available, and
- its combinatorial description also works over worse-than-orbifold singularities. By contrast, the cohomology groups can become very complicated to compute in this case, and one usually only has a spectral sequence but no toric algorithm.

Below you will find detailed descriptions of available functions. If you are familiar with toric geometry, you will likely see that many important objects and operations are unavailable. However, this module is under active development and hopefully will improve in future releases of Sage. If there are some particular features that you would like to see implemented ASAP, please consider reporting them to the Sage Development Team or even implementing them on your own as a patch for inclusion!

`sage.schemes.toric.variety.`**`AffineToricVariety`**(*cone*, **args*, ***kwds*)

Construct an affine toric variety.

INPUT:

- `cone` – strictly convex rational polyhedral cone.

This cone will be used to construct a `rational polyhedral fan`, which will be passed to `ToricVariety()` with the rest of positional and keyword arguments.

OUTPUT:

- `toric variety`.

Note: The generating rays of the fan of this variety are guaranteed to be listed in the same order as the rays of the original cone.

EXAMPLES:

We will create the affine plane as an affine toric variety:

```
sage: quadrant = Cone([(1,0), (0,1)])
sage: A2 = AffineToricVariety(quadrant)
sage: origin = A2(0,0)
sage: origin
[0 : 0]
sage: parent(origin)
Set of rational points of 2-d affine toric variety
```

Only affine toric varieties have points whose (homogeneous) coordinates are all zero.

class `sage.schemes.toric.variety.`**`CohomologyClass`**(*cohomology_ring*, *representative*)

Bases: `sage.rings.quotient_ring_element.QuotientRingElement`

An element of the `CohomologyRing`.

Warning: You should not create instances of this class manually. The generators of the cohomology ring as well as the cohomology classes associated to cones of the fan can be obtained from `ToricVariety_field.cohomology_ring()`.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2.cohomology_ring().gen(0)
[z]
sage: HH = P2.cohomology_ring()
sage: HH.gen(0)
[z]
sage: cone = P2.fan(1)[0];  HH(cone)
[z]
```

deg()

The degree of the cohomology class.

OUTPUT:

An integer d such that the cohomology class is in degree $2d$. If the cohomology class is of mixed degree, the highest degree is returned.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2.cohomology_ring().gen(0).deg()
1
sage: P2.cohomology_ring().zero().deg()
-1
```

exp()

Exponentiate `self`.

Note: The exponential $\exp(x)$ of a rational number x is usually not rational. Therefore, the cohomology class must not have a constant (degree zero) part. The coefficients in the Taylor series of $\exp$ are rational, so any cohomology class without constant term can be exponentiated.

OUTPUT

The cohomology class exp(`self`) if the constant part vanishes, otherwise a `ValueError` is raised.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: H_class = P2.cohomology_ring().gen(0)
sage: H_class
[z]
sage: H_class.exp()
[1/2*z^2 + z + 1]
```

part_of_degree (*d*)

Project the (mixed-degree) cohomology class to the given degree.

$$pr_d: \; H^\bullet(X_\Delta, \mathbf{Q}) \to H^{2d}(X_\Delta, \mathbf{Q})$$

INPUT:

- An integer `d`

OUTPUT:

- The degree-`2d` part of the cohomology class.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: t = P1xP1.cohomology_ring().gen(0)
sage: y = P1xP1.cohomology_ring().gen(2)
sage: 3*t+4*t^2*y+y+t*y+t+1
[t*y + 4*t + y + 1]
sage: (3*t+4*t^2*y+y+t*y+t+1).part_of_degree(1)
[4*t + y]
```

class `sage.schemes.toric.variety.`**CohomologyRing** (*variety*)

Bases: `sage.rings.quotient_ring.QuotientRing_generic`, `sage.structure.unique_representation.UniqueRepresentation`

The (even) cohomology ring of a toric variety.

Irregardles of the variety's base ring, we always work with the variety over $\mathbf{C}$ and its topology.

The cohomology is always the singular cohomology with $\mathbf{Q}$-coefficients. Note, however, that the cohomology of smooth toric varieties is torsion-free, so there is no loss of information in that case.

Currently, the toric variety must not be "too singular". See `ToricVariety_field.cohomology_ring()` for a detailed description of which toric varieties are admissible. For such varieties the odd-dimensional cohomology groups vanish.

Warning: You should not create instances of this class manually. Use `ToricVariety_field.cohomology_ring()` to generate the cohomology ring.

INPUT:

- `variety` – a toric variety. Currently, the toric variety must be at least an orbifold. See `ToricVariety_field.cohomology_ring()` for a detailed description of which toric varieties are admissible.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2.cohomology_ring()
Rational cohomology ring of a 2-d CPR-Fano toric variety covered by 3 affine patches
```

This is equivalent to:

```
sage: from sage.schemes.toric.variety import CohomologyRing
sage: CohomologyRing(P2)
Rational cohomology ring of a 2-d CPR-Fano toric variety covered by 3 affine patches
```

gen(*i*)

Return the generators of the cohomology ring.

INPUT:

- `i` – integer.

OUTPUT:

The `i`-th generator of the cohomology ring. If we denote the toric variety by `X`, then this generator is associated to the ray `X.fan().ray(i)`, which spans the one-cone `X.fan(1)[i]`

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2.cohomology_ring().gen(2)
[z]
```

gens()

Return the generators of the cohomology ring.

OUTPUT:

A tuple of generators, one for each toric divisor of the toric variety `X`. The order is the same as the ordering of the rays of the fan `X.fan().rays()`, which is also the same as the ordering of the one-cones in `X.fan(1)`

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2.cohomology_ring().gens()
([z], [z], [z])
```

`sage.schemes.toric.variety.`**ToricVariety**(*fan, coordinate_names=None, names=None, coordinate_indices=None, base_ring=Rational Field, base_field=None*)

Construct a toric variety.

INPUT:

- `fan` – `rational polyhedral fan`;
- `coordinate_names` – names of variables for the coordinate ring, see `normalize_names()` for acceptable formats. If not given, indexed variable names will be created automatically;
- `names` – an alias of `coordinate_names` for internal use. You may specify either `names` or `coordinate_names`, but not both;
- `coordinate_indices` – list of integers, indices for indexed variables. If not given, the index of each variable will coincide with the index of the corresponding ray of the fan;
- `base_ring` – base ring of the toric variety (default: $\mathbf{Q}$). Must be a field.

- `base_field` – alias for `base_ring`. Takes precedence if both are specified.

OUTPUT:

- `toric variety`.

EXAMPLES:

We will create the product of two projective lines:

```
sage: fan = FaceFan(lattice_polytope.cross_polytope(2))
sage: fan.rays()
M( 1,  0),
M( 0,  1),
M(-1,  0),
M( 0, -1)
in 2-d lattice M
sage: P1xP1 = ToricVariety(fan)
sage: P1xP1.gens()
(z0, z1, z2, z3)
```

Let's create some points:

```
sage: P1xP1(1,1,1,1)
[1 : 1 : 1 : 1]
sage: P1xP1(0,1,1,1)
[0 : 1 : 1 : 1]
sage: P1xP1(0,1,0,1)
Traceback (most recent call last):
...
TypeError: coordinates (0, 1, 0, 1)
are in the exceptional set!
```

We cannot set to zero both coordinates of the same projective line!

Let's change the names of the variables. We have to re-create our toric variety:

```
sage: P1xP1 = ToricVariety(fan, "x s y t")
sage: P1xP1.gens()
(x, s, y, t)
```

Now (x, y) correspond to one line and (s, t) to the other one.

```
sage: P1xP1.inject_variables()
Defining x, s, y, t
sage: P1xP1.subscheme(x*s-y*t)
Closed subscheme of 2-d toric variety
covered by 4 affine patches defined by:
  x*s - y*t
```

Here is a shorthand for defining the toric variety and homogeneous coordinates in one go:

```
sage: P1xP1.<a,b,c,d> = ToricVariety(fan)
sage: (a^2+b^2) * (c+d)
a^2*c + b^2*c + a^2*d + b^2*d
```

class `sage.schemes.toric.variety.`**`ToricVariety_field`**(*fan*, *coordinate_names*, *coordinate_indices*, *base_field*)

Bases: `sage.misc.cachefunc.ClearCacheOnPickle`, `sage.schemes.generic.ambient_space.AmbientS`

Construct a toric variety associated to a rational polyhedral fan.

Warning: This class does not perform any checks of correctness of input. Use `ToricVariety()` and `AffineToricVariety()` to construct toric varieties.

INPUT:

- `fan` – `rational polyhedral fan`;
- `coordinate_names` – names of variables, see `normalize_names()` for acceptable formats. If `None`, indexed variable names will be created automatically;
- `coordinate_indices` – list of integers, indices for indexed variables. If `None`, the index of each variable will coincide with the index of the corresponding ray of the fan;
- `base_field` – base field of the toric variety.

OUTPUT:

- `toric variety`.

TESTS:

```
sage: fan = FaceFan(lattice_polytope.cross_polytope(2))
sage: P1xP1 = ToricVariety(fan)
```

Aut_dimension()

Return the dimension of the automorphism group

There are three kinds of symmetries of toric varieties:

- Toric automorphisms (rescaling of homogeneous coordinates)
- Demazure roots. These are translations $x_i \to x_i + \epsilon x^m$ of a homogeneous coordinate x_i by a monomial x^m of the same homogeneous degree.
- Symmetries of the fan. These yield discrete subgroups.

OUTPUT:

An integer. The dimension of the automorphism group. Equals the dimension of the M-lattice plus the number of Demazure roots.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2.Aut_dimension()
8
```

TESTS:

```
sage: toric_varieties.A1().Aut_dimension()
Traceback (most recent call last):
...
NotImplementedError: Aut_dimension() is only implemented for complete toric varieties.
```

Chern_character(*deg=None*)

Return the Chern character (of the tangent bundle) of the toric variety.

INPUT:

- `deg` – integer (optional). The degree of the Chern character.

OUTPUT:

- If the degree is specified, the degree-`deg` part of the Chern character.

•If no degree is specified, the total Chern character.

REFERENCES:

http://en.wikipedia.org/wiki/Chern_character#The_Chern_character

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: dP6.Chern_character()
[3*w^2 + y + 2*v + 2*z + w + 2]
sage: dP6.ch()
[3*w^2 + y + 2*v + 2*z + w + 2]
sage: dP6.ch(1) == dP6.c(1)
True
```

Chern_class(*deg=None*)

Return Chern classes of the (tangent bundle of the) toric variety.

INPUT:

•`deg` – integer (optional). The degree of the Chern class.

OUTPUT:

•If the degree is specified, the `deg`-th Chern class.

•If no degree is specified, the total Chern class.

REFERENCES:

http://en.wikipedia.org/wiki/Chern_class

EXAMPLES:

```
sage: X = toric_varieties.dP6()
sage: X.Chern_class()
[-6*w^2 + y + 2*v + 2*z + w + 1]
sage: X.c()
[-6*w^2 + y + 2*v + 2*z + w + 1]
sage: X.c(1)
[y + 2*v + 2*z + w]
sage: X.c(2)
[-6*w^2]
sage: X.integrate( X.c(2) )
6
sage: X.integrate( X.c(2) ) == X.Euler_number()
True
```

Chow_group(*base_ring=Integer Ring*)

Return the toric Chow group.

INPUT:

•`base_ring` – either `ZZ` (default) or `QQ`. The coefficient ring of the Chow group.

OUTPUT:

A `sage.schemes.toric.chow_group.ChowGroup_class`

EXAMPLES:

```
sage: A = toric_varieties.P2().Chow_group(); A
Chow group of 2-d CPR-Fano toric variety covered by 3 affine patches
sage: A.gens()
(( 1 | 0 | 0 ), ( 0 | 1 | 0 ), ( 0 | 0 | 1 ))
```

Demazure_roots()

Return the Demazure roots.

OUTPUT:

The roots as points of the M-lattice.

REFERENCES:

EXAMPLE:

```
sage: P2 = toric_varieties.P2()
sage: P2.Demazure_roots()
(M(-1, 0), M(-1, 1), M(0, -1), M(0, 1), M(1, -1), M(1, 0))
```

Here are the remaining three examples listed in [Bazhov], Example 2.1 and 2.3:

```
sage: s = 3
sage: cones = [(0,1),(1,2),(2,3),(3,0)]
sage: Hs = ToricVariety(Fan(rays=[(1,0),(0,-1),(-1,s),(0,1)], cones=cones))
sage: Hs.Demazure_roots()
(M(-1, 0), M(1, 0), M(0, 1), M(1, 1), M(2, 1), M(3, 1))

sage: P11s = ToricVariety(Fan(rays=[(1,0),(0,-1),(-1,s)], cones=[(0,1),(1,2),(2,0)]))
sage: P11s.Demazure_roots()
(M(-1, 0), M(1, 0), M(0, 1), M(1, 1), M(2, 1), M(3, 1))
sage: P11s.Demazure_roots() == Hs.Demazure_roots()
True

sage: Bs = ToricVariety(Fan(rays=[(s,1),(s,-1),(-s,-1),(-s,1)], cones=cones))
sage: Bs.Demazure_roots()
()
```

TESTS:

```
sage: toric_varieties.A1().Demazure_roots()
Traceback (most recent call last):
...
NotImplementedError: Demazure_roots() is only implemented for complete toric varieties.
```

Euler_number()

Return the topological Euler number of the toric variety.

Sometimes, this is also called the Euler characteristic. `chi()` is a synonym for `Euler_number()`.

REFERENCES:

http://en.wikipedia.org/wiki/Euler_characteristic

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1.Euler_number()
4
sage: P1xP1.chi()
4
```

K()

Returns the canonical divisor of the toric variety.

EXAMPLES:

Lets test that the del Pezzo surface dP_6 has degree 6, as its name implies:

```
sage: dP6 = toric_varieties.dP6()
sage: HH = dP6.cohomology_ring()
sage: dP6.K()
-V(x) - V(u) - V(y) - V(v) - V(z) - V(w)
sage: dP6.integrate( HH(dP6.K())^2 )
6
```

Kaehler_cone()

Return the closure of the Kähler cone of `self`.

OUTPUT:

- `cone`.

Note: This cone sits in the rational divisor class group of `self` and the choice of coordinates agrees with `rational_class_group()`.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: Kc = P1xP1.Kaehler_cone()
sage: Kc
2-d cone in 2-d lattice
sage: Kc.rays()
Divisor class [0, 1],
Divisor class [1, 0]
in Basis lattice of The toric rational divisor class group
of a 2-d CPR-Fano toric variety covered by 4 affine patches
sage: [ divisor_class.lift() for divisor_class in Kc.rays() ]
[V(x), V(s)]
sage: Kc.lattice()
Basis lattice of The toric rational divisor class group of a
2-d CPR-Fano toric variety covered by 4 affine patches
```

Mori_cone()

Returns the Mori cone of `self`.

OUTPUT:

- `cone`.

Note:

- The Mori cone is dual to the Kähler cone.
- We think of the Mori cone as living inside the row span of the Gale transform matrix (computed by `self.fan().Gale_transform()`).
- The points in the Mori cone are the effective curves in the variety.
- The `i`-th entry in each Mori vector is the intersection number of the curve corresponding to the generator of the `i`-th ray of the fan with the corresponding divisor class. The very last entry is associated to the orgin of the fan lattice.
- The Mori vectors are also known as the gauged linear sigma model charge vectors.

EXAMPLES:

```
sage: P4_11169 = toric_varieties.P4_11169_resolved()
sage: P4_11169.Mori_cone()
```

```
2-d cone in 7-d lattice
sage: P4_11169.Mori_cone().rays()
(3, 2, 0, 0, 0,  1, -6),
(0, 0, 1, 1, 1, -3,  0)
in Ambient free module of rank 7
over the principal ideal domain Integer Ring
```

Spec (*cone=None*, *names=None*)

Return the spectrum associated to the dual cone.

Let $\sigma \in N_{\mathbf{R}}$ be a cone and $\sigma^\vee \cap M$ the associated semigroup of lattice points in the dual cone. Then

$$S = \mathbf{C}[\sigma^\vee \cap M]$$

is a $\mathbf{C}$-algebra. It is spanned over $\mathbf{C}$ by the points of $\sigma \cap N$, addition is formal linear combination of lattice points, and multiplication of lattice points is the semigroup law (that is, addition of lattice points). The $\mathbf{C}$-algebra S then defines a scheme $Spec(S)$.

For example, if $\sigma = \{(x, y)|x \geq 0, y \geq 0\}$ is the first quadrant then S is the polynomial ring in two variables. The associated scheme is $Spec(S) = \mathbf{C}^2$.

The same construction works over any base field, this introduction only used $\mathbf{C}$ for simplicity.

INPUT:

- `cone` – a `Cone`. Can be omitted for an affine toric variety, in which case the (unique) generating cone is used.
- `names` – (optional). Names of variables for the semigroup ring, see `normalize_names()` for acceptable formats. If not given, indexed variable names will be created automatically.

Output:

The spectrum of the semigroup ring $\mathbf{C}[\sigma^\vee \cap M]$.

EXAMPLES:

```
sage: quadrant = Cone([(1,0),(0,1)])
sage: AffineToricVariety(quadrant).Spec()
Spectrum of Multivariate Polynomial Ring in z0, z1 over Rational Field
```

A more interesting example:

```
sage: A2Z2 = Cone([(0,1),(2,1)])
sage: AffineToricVariety(A2Z2).Spec(names='u,v,t')
Spectrum of Quotient of Multivariate Polynomial Ring
in u, v, t over Rational Field by the ideal (-u*v + t^2)
```

Stanley_Reisner_ideal ()

Return the Stanley-Reisner ideal.

OUTPUT:

- The Stanley-Reisner ideal in the polynomial ring over $\mathbf{Q}$ generated by the homogeneous coordinates.

EXAMPLES:

```
sage: fan = Fan([[0,1,3],[3,4],[2,0],[1,2,4]], [(-3, -2, 1), (0, 0, 1), (3, -2, 1), (-1, -1,
sage: X = ToricVariety(fan, coordinate_names='A B C D E', base_field=GF(5))
sage: SR = X.Stanley_Reisner_ideal(); SR
Ideal (A*E, C*D, A*B*C, B*D*E) of Multivariate Polynomial Ring in A, B, C, D, E over Rationa
```

Td (*deg=None*)

Return the Todd class (of the tangent bundle) of the toric variety.

INPUT:

- deg – integer (optional). The desired degree part.

OUTPUT:

- If the degree is specified, the degree-deg part of the Todd class.
- If no degree is specified, the total Todd class.

REFERENCES:

http://en.wikipedia.org/wiki/Todd_class

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: dP6.Todd_class()
[-w^2 + 1/2*y + v + z + 1/2*w + 1]
sage: dP6.Td()
[-w^2 + 1/2*y + v + z + 1/2*w + 1]
sage: dP6.integrate( dP6.Td() )
1
```

Todd_class (*deg=None*)

Return the Todd class (of the tangent bundle) of the toric variety.

INPUT:

- deg – integer (optional). The desired degree part.

OUTPUT:

- If the degree is specified, the degree-deg part of the Todd class.
- If no degree is specified, the total Todd class.

REFERENCES:

http://en.wikipedia.org/wiki/Todd_class

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: dP6.Todd_class()
[-w^2 + 1/2*y + v + z + 1/2*w + 1]
sage: dP6.Td()
[-w^2 + 1/2*y + v + z + 1/2*w + 1]
sage: dP6.integrate( dP6.Td() )
1
```

affine_algebraic_patch (*cone=None, names=None*)

Return the patch corresponding to cone as an affine algebraic subvariety.

INPUT:

- cone – a Cone σ of the fan. It can be omitted for an affine toric variety, in which case the single generating cone is used.

OUTPUT:

A affine algebraic subscheme corresponding to the patch $Spec(\sigma^{\vee} \cap M)$ associated to the cone σ.

See also `affine_patch()`, which expresses the patches as subvarieties of affine toric varieties instead.

EXAMPLES:

```
sage: cone = Cone([(0,1),(2,1)])
sage: A2Z2 = AffineToricVariety(cone)
sage: A2Z2.affine_algebraic_patch()
Closed subscheme of Affine Space of dimension 3 over Rational Field defined by:
  -z0*z1 + z2^2
sage: A2Z2.affine_algebraic_patch(Cone([(0,1)]), names='x, y, t')
Closed subscheme of Affine Space of dimension 3 over Rational Field defined by:
  1
```

affine_patch(*i*)

Return the `i`-th affine patch of `self`.

INPUT:

- `i` – integer, index of a generating cone of the fan of `self`.

OUTPUT:

- affine `toric variety` corresponding to the `i`-th generating cone of the fan of `self`.

The result is cached, so the `i`-th patch is always the same object in memory.

See also `affine_algebraic_patch()`, which expresses the patches as subvarieties of affine space instead.

EXAMPLES:

```
sage: fan = FaceFan(lattice_polytope.cross_polytope(2))
sage: P1xP1 = ToricVariety(fan, "x s y t")
sage: patch0 = P1xP1.affine_patch(0)
sage: patch0
2-d affine toric variety
sage: patch0.embedding_morphism()
Scheme morphism:
  From: 2-d affine toric variety
  To:   2-d toric variety covered by 4 affine patches
  Defn: Defined on coordinates by sending [x : t] to
        [x : 1 : 1 : t]
sage: patch1 = P1xP1.affine_patch(1)
sage: patch1.embedding_morphism()
Scheme morphism:
  From: 2-d affine toric variety
  To:   2-d toric variety covered by 4 affine patches
  Defn: Defined on coordinates by sending [y : t] to
        [1 : 1 : y : t]
sage: patch1 is P1xP1.affine_patch(1)
True
```

c(*deg=None*)

Return Chern classes of the (tangent bundle of the) toric variety.

INPUT:

- `deg` – integer (optional). The degree of the Chern class.

OUTPUT:

- If the degree is specified, the `deg`-th Chern class.
- If no degree is specified, the total Chern class.

REFERENCES:

http://en.wikipedia.org/wiki/Chern_class

EXAMPLES:

```
sage: X = toric_varieties.dP6()
sage: X.Chern_class()
[-6*w^2 + y + 2*v + 2*z + w + 1]
sage: X.c()
[-6*w^2 + y + 2*v + 2*z + w + 1]
sage: X.c(1)
[y + 2*v + 2*z + w]
sage: X.c(2)
[-6*w^2]
sage: X.integrate( X.c(2) )
6
sage: X.integrate( X.c(2) ) == X.Euler_number()
True
```

cartesian_product (*other*, *coordinate_names=None*, *coordinate_indices=None*)

Return the Cartesian product of `self` with `other`.

INPUT:

- `other` – a `toric variety`;
- `coordinate_names` – names of variables for the coordinate ring, see `normalize_names()` for acceptable formats. If not given, indexed variable names will be created automatically;
- `coordinate_indices` – list of integers, indices for indexed variables. If not given, the index of each variable will coincide with the index of the corresponding ray of the fan.

OUTPUT:

– a `toric variety`.

EXAMPLES:

```
sage: P1 = ToricVariety(Fan([Cone([(1,)]), Cone([(-1,)])]))
sage: P1xP1 = P1.cartesian_product(P1); P1xP1
2-d toric variety covered by 4 affine patches
sage: P1xP1.fan().rays()
N+N(-1,  0),
N+N( 1,  0),
N+N( 0, -1),
N+N( 0,  1)
in 2-d lattice N+N
```

ch (*deg=None*)

Return the Chern character (of the tangent bundle) of the toric variety.

INPUT:

- `deg` – integer (optional). The degree of the Chern character.

OUTPUT:

- If the degree is specified, the degree-`deg` part of the Chern character.
- If no degree is specified, the total Chern character.

REFERENCES:

http://en.wikipedia.org/wiki/Chern_character#The_Chern_character

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: dP6.Chern_character()
[3*w^2 + y + 2*v + 2*z + w + 2]
sage: dP6.ch()
[3*w^2 + y + 2*v + 2*z + w + 2]
sage: dP6.ch(1) == dP6.c(1)
True
```

change_ring(*F*)

Return a toric variety over `F` and otherwise the same as `self`.

INPUT:

- `F` – field.

OUTPUT:

- `toric variety` over `F`.

Note: There is no need to have any relation between `F` and the base field of `self`. If you do want to have such a relation, use `base_extend()` instead.

EXAMPLES:

```
sage: P1xA1 = toric_varieties.P1xA1()
sage: P1xA1.base_ring()
Rational Field
sage: P1xA1_RR = P1xA1.change_ring(RR)
sage: P1xA1_RR.base_ring()
Real Field with 53 bits of precision
sage: P1xA1_QQ = P1xA1_RR.change_ring(QQ)
sage: P1xA1_QQ.base_ring()
Rational Field
sage: P1xA1_RR.base_extend(QQ)
Traceback (most recent call last):
...
ValueError: no natural map from the base ring
(=Real Field with 53 bits of precision)
to R (=Rational Field)!
sage: R = PolynomialRing(QQ, 2, 'a')
sage: P1xA1.change_ring(R)
Traceback (most recent call last):
...
TypeError: need a field to construct a toric variety!
 Got Multivariate Polynomial Ring in a0, a1 over Rational Field
```

chi()

Return the topological Euler number of the toric variety.

Sometimes, this is also called the Euler characteristic. `chi()` is a synonym for `Euler_number()`.

REFERENCES:

http://en.wikipedia.org/wiki/Euler_characteristic

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1.Euler_number()
4
```

```
sage: P1xP1.chi()
4
```

cohomology_basis(*d=None*)

Return a basis for the cohomology of the toric variety.

INPUT:

- d (optional) – integer.

OUTPUT:

- Without the optional argument, a list whose d-th entry is a basis for $H^{2d}(X, \mathbf{Q})$
- If the argument is an integer d, returns basis for $H^{2d}(X, \mathbf{Q})$

EXAMPLES:

```
sage: X = toric_varieties.dP8()
sage: X.cohomology_basis()
(([1],), ([z], [y]), ([y*z],))
sage: X.cohomology_basis(1)
([z], [y])
sage: X.cohomology_basis(dimension(X))[0] == X.volume_class()
True
```

cohomology_ring()

Return the cohomology ring of the toric variety.

OUTPUT:

- If the toric variety is is over $\mathbf{C}$ and has at most finite orbifold singularities: $H^\bullet(X, \mathbf{Q})$ as a polynomial quotient ring.
- Other cases are not handled yet.

Note:

- Toric varieties over any field of characteristic 0 are treated as if they were varieties over $\mathbf{C}$.
- The integral cohomology of smooth toric varieties is torsion-free, so in this case there is no loss of information when going to rational coefficients.
- self.cohomology_ring().gen(i) is the divisor class corresponding to the i-th ray of the fan.

EXAMPLES:

```
sage: X = toric_varieties.dP6()
sage: X.cohomology_ring()
Rational cohomology ring of a 2-d CPR-Fano toric variety covered by 6 affine patches
sage: X.cohomology_ring().defining_ideal()
Ideal (-u - y + z + w, x - y - v + w, x*y, x*v, x*z, u*v, u*z, u*w, y*z, y*w, v*w) of Multiv
sage: X.cohomology_ring().defining_ideal().ring()
Multivariate Polynomial Ring in x, u, y, v, z, w over Rational Field
sage: X.variable_names()
('x', 'u', 'y', 'v', 'z', 'w')
sage: X.cohomology_ring().gens()
([y + v - w], [-y + z + w], [y], [v], [z], [w])
```

TESTS:

The cohomology ring is a circular reference that is potentially troublesome on unpickling, see trac ticket #15050 and trac ticket #15149

```
sage: variety = toric_varieties.P(1)
sage: a = [variety.cohomology_ring(), variety.cohomology_basis(), variety.volume_class()]
sage: b = [variety.Todd_class(), variety.Chern_class(), variety.Chern_character(), variety.K
sage: loads(dumps(variety)) == variety
True
```

coordinate_ring()

Return the coordinate ring of self.

For toric varieties this is the homogeneous coordinate ring (a.k.a. Cox's ring and total ring).

OUTPUT:

- polynomial ring.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1.coordinate_ring()
Multivariate Polynomial Ring in s, t, x, y
over Rational Field
```

TESTS:

```
sage: R = toric_varieties.A1().coordinate_ring();  R
Multivariate Polynomial Ring in z over Rational Field
sage: type(R)
<type 'sage.rings.polynomial.multi_polynomial_libsingular.MPolynomialRing_libsingular'>
```

count_points()

Return the number of points of self.

This is an alias for point_set().cardinality(), see cardinality() for details.

EXAMPLES:

```
sage: o = lattice_polytope.cross_polytope(3)
sage: V = ToricVariety(FaceFan(o))
sage: V2 = V.change_ring(GF(2))
sage: V2.point_set().cardinality()
27
sage: V2.count_points()
27
```

dimension_singularities()

Return the dimension of the singular set.

OUTPUT:

Integer. The dimension of the singular set of the toric variety. Often the singular set is a reducible subvariety, and this method will return the dimension of the largest-dimensional component.

Returns -1 if the toric variety is smooth.

EXAMPLES:

```
sage: toric_varieties.P4_11169().dimension_singularities()
1
sage: toric_varieties.Conifold().dimension_singularities()
0
```

```
sage: toric_varieties.P2().dimension_singularities()
-1
```

divisor (*arg*, *base_ring=None*, *check=True*, *reduce=True*)

Return a divisor.

INPUT:

The arguments are the same as in `sage.schemes.toric.divisor.ToricDivisor()`, with the exception of defining a divisor with a single integer: this method considers it to be the index of a ray of the `fan()` of `self`.

OUTPUT:

- A `sage.schemes.toric.divisor.ToricDivisor_generic`

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: dP6.coordinate_ring()
Multivariate Polynomial Ring in x, u, y, v, z, w
over Rational Field
sage: dP6.divisor(range(6))
V(u) + 2*V(y) + 3*V(v) + 4*V(z) + 5*V(w)
sage: dP6.inject_variables()
Defining x, u, y, v, z, w
sage: dP6.divisor(x*u^3)
V(x) + 3*V(u)
```

You can also construct divisors based on ray indices:

```
sage: dP6.divisor(0)
V(x)
sage: for i in range(0,dP6.fan().nrays()):
...       print dP6.divisor(i),
...       print ': generated by ray',
...       dP6.fan().ray(i)
V(x) : generated by ray N(0, 1)
V(u) : generated by ray N(-1, 0)
V(y) : generated by ray N(-1, -1)
V(v) : generated by ray N(0, -1)
V(z) : generated by ray N(1, 0)
V(w) : generated by ray N(1, 1)
```

TESTS:

We check that the issue trac ticket #12812 is resolved:

```
sage: sum(dP6.divisor(i) for i in range(3))
V(x) + V(u) + V(y)
```

divisor_group (*base_ring=Integer Ring*)

Return the group of Weil divisors.

INPUT:

- `base_ring` – the coefficient ring, usually `ZZ` (default) or `QQ`.

OUTPUT:

The (free abelian) group of Cartier divisors, that is, formal linear combinations of polynomial equations over the coefficient ring `base_ring`.

These need not be toric (=defined by monomials), but allow general polynomials. The output will be an instance of `sage.schemes.generic.divisor_group.DivisorGroup_generic`.

> **Warning:** You almost certainly want the group of toric divisors, see `toric_divisor_group()`. The toric divisor group is generated by the rays of the fan. The general divisor group has no toric functionality implemented.

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: Div = dP6.divisor_group(); Div
Group of ZZ-Divisors on 2-d CPR-Fano toric variety
covered by 6 affine patches
sage: Div(x)
V(x)
```

embedding_morphism()

Return the default embedding morphism of `self`.

Such a morphism is always defined for an affine patch of a toric variety (which is also a toric varieties itself).

OUTPUT:

- `scheme morphism` if the default embedding morphism was defined for `self`, otherwise a `ValueError` exception is raised.

EXAMPLES:

```
sage: fan = FaceFan(lattice_polytope.cross_polytope(2))
sage: P1xP1 = ToricVariety(fan, "x s y t")
sage: P1xP1.embedding_morphism()
Traceback (most recent call last):
...
ValueError: no default embedding was
defined for this toric variety!
sage: patch = P1xP1.affine_patch(0)
sage: patch
2-d affine toric variety
sage: patch.embedding_morphism()
Scheme morphism:
  From: 2-d affine toric variety
  To:   2-d toric variety covered by 4 affine patches
  Defn: Defined on coordinates by sending [x : t] to
        [x : 1 : 1 : t]
```

fan(*dim=None*, *codim=None*)

Return the underlying fan of `self` or its cones.

INPUT:

- `dim` – dimension of the requested cones;
- `codim` – codimension of the requested cones.

OUTPUT:

- `rational polyhedral fan` if no parameters were given, `tuple` of `cones` otherwise.

EXAMPLES:

```
sage: fan = FaceFan(lattice_polytope.cross_polytope(2))
sage: P1xP1 = ToricVariety(fan)
sage: P1xP1.fan()
Rational polyhedral fan in 2-d lattice M
sage: P1xP1.fan() is fan
True
sage: P1xP1.fan(1)[0]
1-d cone of Rational polyhedral fan in 2-d lattice M
```

inject_coefficients(*scope=None*, *verbose=True*)

Inject generators of the base field of `self` into `scope`.

This function is useful if the base field is the field of rational functions.

INPUT:

- `scope` – namespace (default: global, not just the scope from which this function was called);
- `verbose` – if `True` (default), names of injected generators will be printed.

OUTPUT:

- none.

EXAMPLES:

```
sage: fan = FaceFan(lattice_polytope.cross_polytope(2))
sage: F = QQ["a, b"].fraction_field()
sage: P1xP1 = ToricVariety(fan, base_field=F)
sage: P1xP1.inject_coefficients()
Defining a, b
```

We check that we can use names `a` and `b`, Trac #10498 is fixed:

```
sage: a + b
a + b
sage: a + b in P1xP1.coordinate_ring()
True
```

integrate(*cohomology_class*)

Integrate a cohomology class over the toric variety.

INPUT:

- `cohomology_class` – A cohomology class given as a polynomial in `self.cohomology_ring()`

OUTPUT:

The integral of the cohomology class over the variety. The volume normalization is given by `volume_class()`, that is, `self.integrate(self.volume_class())` is always one (if the volume class exists).

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: HH = dP6.cohomology_ring()
sage: D = [ HH(c) for c in dP6.fan(dim=1) ]
sage: matrix([ [ D[i]*D[j] for i in range(0,6) ] for j in range(0,6) ])
[ [w^2] [-w^2]    [0]    [0]    [0] [-w^2]]
[[-w^2]  [w^2] [-w^2]    [0]    [0]    [0]]
[   [0] [-w^2]  [w^2] [-w^2]    [0]    [0]]
[   [0]    [0] [-w^2]  [w^2] [-w^2]    [0]]
```

```
[    [0]     [0]     [0] [-w^2]   [w^2] [-w^2]]
[[-w^2]     [0]     [0]     [0] [-w^2]   [w^2]]
sage: matrix([ [ dP6.integrate(D[i]*D[j]) for i in range(0,6) ] for j in range(0,6) ])
[-1  1  0  0  0  1]
[ 1 -1  1  0  0  0]
[ 0  1 -1  1  0  0]
[ 0  0  1 -1  1  0]
[ 0  0  0  1 -1  1]
[ 1  0  0  0  1 -1]
```

If the toric variety is an orbifold, the intersection numbers are usually fractional:

```
sage: P2_123 = toric_varieties.P2_123()
sage: HH = P2_123.cohomology_ring()
sage: D = [ HH(c) for c in P2_123.fan(dim=1) ]
sage: matrix([ [ P2_123.integrate(D[i]*D[j]) for i in range(0,3) ] for j in range(0,3) ])
[2/3   1 1/3]
[  1 3/2 1/2]
[1/3 1/2 1/6]
sage: A = P2_123.Chow_group(QQ)
sage: matrix([ [ A(P2_123.divisor(i))
...              .intersection_with_divisor(P2_123.divisor(j))
...              .count_points() for i in range(0,3) ] for j in range(0,3) ])
[2/3   1 1/3]
[  1 3/2 1/2]
[1/3 1/2 1/6]
```

is_affine()

Check if `self` is an affine toric variety.

An affine toric variety is a toric variety whose fan is the face lattice of a single cone. See also `AffineToricVariety()`.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: toric_varieties.A2().is_affine()
True
sage: toric_varieties.P1xA1().is_affine()
False
```

is_complete()

Check if `self` is complete.

OUTPUT:

- `True` if `self` is complete and `False` otherwise.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1.is_complete()
True
sage: P1xP1.affine_patch(0).is_complete()
False
```

is_homogeneous(*polynomial*)

Check if `polynomial` is homogeneous.

The coordinate ring of a toric variety is multigraded by relations between generating rays of the underlying fan.

INPUT:

•`polynomial` – polynomial in the coordinate ring of `self` or its quotient.

OUTPUT:

•`True` if `polynomial` is homogeneous and `False` otherwise.

EXAMPLES:

We will use the product of two projective lines with coordinates (x, y) for one and (s, t) for the other:

```
sage: P1xP1.<x,y,s,t> = toric_varieties.P1xP1()
sage: P1xP1.is_homogeneous(x - y)
True
sage: P1xP1.is_homogeneous(x*s + y*t)
True
sage: P1xP1.is_homogeneous(x - t)
False
sage: P1xP1.is_homogeneous(1)
True
```

Note that by homogeneous, we mean well-defined with respect to the homogeneous rescalings of `self`. So a polynomial that you would usually not call homogeneous can be homogeneous if there are no homogeneous rescalings, for example:

```
sage: A1.<z> = toric_varieties.A1()
sage: A1.is_homogeneous(z^3+z^7)
True
```

Finally, the degree group is really the Chow group $A_{d-1}(X)$ and can contain torsion. For example, take $\mathbf{C}^2/\mathbf{Z}_2$. Here, the Chow group is $A_{d-1}(\mathbf{C}^2/\mathbf{Z}_2) = \mathbf{Z}_2$ and distinguishes even-degree homogeneous polynomials from odd-degree homogeneous polynomials:

```
sage: A2_Z2.<x,y> = toric_varieties.A2_Z2()
sage: A2_Z2.is_homogeneous(x+y+x^3+y^5+x^3*y^4)
True
sage: A2_Z2.is_homogeneous(x^2+x*y+y^4+(x*y)^5+x^4*y^4)
True
sage: A2_Z2.is_homogeneous(x+y^2)
False
```

is_isomorphic(*another*)

Check if `self` is isomorphic to `another`.

INPUT:

•`another` - `toric variety`.

OUTPUT:

•`True` if `self` and `another` are isomorphic, `False` otherwise.

EXAMPLES:

```
sage: TV1 = toric_varieties.P1xA1()
sage: TV2 = toric_varieties.P1xP1()
```

Only the most trivial case is implemented so far:

```
sage: TV1.is_isomorphic(TV1)
True
sage: TV1.is_isomorphic(TV2)
Traceback (most recent call last):
...
NotImplementedError:
isomorphism check is not yet implemented!
```

is_orbifold()

Check if `self` has only quotient singularities.

A toric variety with at most orbifold singularities (in this sense) is often called a simplicial toric variety. In this package, we generally try to avoid this term since it mixes up differential geometry and cone terminology.

OUTPUT:

- `True` if `self` has at most quotient singularities by finite groups, `False` otherwise.

EXAMPLES:

```
sage: fan1 = FaceFan(lattice_polytope.cross_polytope(2))
sage: P1xP1 = ToricVariety(fan1)
sage: P1xP1.is_orbifold()
True
sage: fan2 = NormalFan(lattice_polytope.cross_polytope(3))
sage: TV = ToricVariety(fan2)
sage: TV.is_orbifold()
False
```

is_smooth()

Check if `self` is smooth.

OUTPUT:

- `True` if `self` is smooth and `False` otherwise.

EXAMPLES:

```
sage: fan1 = FaceFan(lattice_polytope.cross_polytope(2))
sage: P1xP1 = ToricVariety(fan1)
sage: P1xP1.is_smooth()
True
sage: fan2 = NormalFan(lattice_polytope.cross_polytope(2))
sage: TV = ToricVariety(fan2)
sage: TV.is_smooth()
False
```

linear_equivalence_ideal()

Return the ideal generated by linear relations

OUTPUT:

- The ideal generated by the linear relations of the rays in the polynomial ring over $\mathbf{Q}$ generated by the homogeneous coordinates.

EXAMPLES:

```
sage: fan = Fan([[0,1,3],[3,4],[2,0],[1,2,4]], [(-3, -2, 1), (0, 0, 1), (3, -2, 1), (-1, -1,
sage: X = ToricVariety(fan, coordinate_names='A B C D E', base_field=GF(5))
sage: lin = X.linear_equivalence_ideal(); lin
Ideal (-3*A + 3*C - D + E, -2*A - 2*C - D - E, A + B + C + D + E) of Multivariate Polynomial
```

orbit_closure (*cone*)

Return the orbit closure of `cone`.

The cones σ of a fan Σ are in one-to-one correspondence with the torus orbits $O(\sigma)$ of the corresponding toric variety X_Σ. Each orbit is isomorphic to a lower dimensional torus (of dimension equal to the codimension of σ). Just like the toric variety X_Σ itself, these orbits are (partially) compactified by lower-dimensional orbits. In particular, one can define the closure $V(\sigma)$ of the torus orbit $O(\sigma)$ in the ambient toric variety X_Σ, which is again a toric variety.

See Proposition 3.2.7 of [CLS] for more details.

INPUT:

- `cone` – a `cone` of the fan.

OUTPUT:

- a torus orbit closure associated to `cone` as a `toric variety`.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: H = P1xP1.fan(1)[0]
sage: V = P1xP1.orbit_closure(H);  V
1-d toric variety covered by 2 affine patches
sage: V.embedding_morphism()
Scheme morphism:
  From: 1-d toric variety covered by 2 affine patches
  To:   2-d CPR-Fano toric variety covered by 4 affine patches
  Defn: Defined by embedding the torus closure associated to the 1-d
        cone of Rational polyhedral fan in 2-d lattice N.
sage: V.embedding_morphism().as_polynomial_map()
Scheme morphism:
  From: 1-d toric variety covered by 2 affine patches
  To:   2-d CPR-Fano toric variety covered by 4 affine patches
  Defn: Defined on coordinates by sending [z0 : z1] to
        [0 : 1 : z1 : z0]
```

TESTS:

```
sage: A2 = toric_varieties.A2()
sage: A2.orbit_closure(A2.fan(2)[0])
0-d affine toric variety
```

plot (***options*)

Plot `self`, i.e. the corresponding fan.

INPUT:

- any options for toric plots (see `toric_plotter.options`), none are mandatory.

OUTPUT:

- a plot.

Note: The difference between `X.plot()` and `X.fan().plot()` is that in the first case default ray labels correspond to variables of `X`.

EXAMPLES:

```
sage: X = toric_varieties.Cube_deformation(4)
sage: X.plot()
Graphics3d Object
```

rational_class_group()

Return the rational divisor class group of self.

Let X be a toric variety.

The **Weil divisor class group** $Cl(X)$ is a finitely generated abelian group and can contain torsion. Its rank equals the number of rays in the fan of X minus the dimension of X.

The **rational divisor class group** is $Cl(X) \otimes_{\mathbf{Z}} \mathbf{Q}$ and never includes torsion. If X is *smooth*, this equals the **Picard group** of X, whose elements are the isomorphism classes of line bundles on X. The group law (which we write as addition) is the tensor product of the line bundles. The Picard group of a toric variety is always torsion-free.

OUTPUT:

- rational divisor class group.

Note:

- Coordinates correspond to the rows of self.fan().gale_transform().
- Kaehler_cone() yields a cone in this group.

EXAMPLES:

```
sage: P1xA1 = toric_varieties.P1xA1()
sage: P1xA1.rational_class_group()
The toric rational divisor class group
of a 2-d toric variety covered by 2 affine patches
```

resolve(***kwds*)

Construct a toric variety whose fan subdivides the fan of self.

The name of this function reflects the fact that usually such subdivisions are done for resolving singularities of the original variety.

INPUT:

This function accepts only keyword arguments, none of which are mandatory.

- coordinate_names – names for coordinates of the new variety. If not given, will be constructed from the coordinate names of self and necessary indexed ones. See normalize_names() for the description of acceptable formats;
- coordinate_indices – coordinate indices which should be used for indexed variables of the new variety;
- all other arguments will be passed to subdivide() method of the underlying rational polyhedral fan, see its documentation for the available options.

OUTPUT:

- toric variety.

EXAMPLES:

First we will "manually" resolve a simple orbifold singularity:

```
sage: cone = Cone([(1,1), (-1,1)])
sage: fan = Fan([cone])
sage: TV = ToricVariety(fan)
sage: TV.is_smooth()
```

```
False
sage: TV_res = TV.resolve(new_rays=[(0,1)])
sage: TV_res.is_smooth()
True
sage: TV_res.fan().rays()
N( 1, 1),
N(-1, 1),
N( 0, 1)
in 2-d lattice N
sage: [cone.ambient_ray_indices() for cone in TV_res.fan()]
[(0, 2), (1, 2)]
```

Now let's "automatically" partially resolve a more complicated fan:

```
sage: fan = NormalFan(lattice_polytope.cross_polytope(3))
sage: TV = ToricVariety(fan)
sage: TV.is_smooth()
False
sage: TV.is_orbifold()
False
sage: TV.fan().nrays()
8
sage: TV.fan().ngenerating_cones()
6
sage: TV_res = TV.resolve(make_simplicial=True)
sage: TV_res.is_smooth()
False
sage: TV_res.is_orbifold()
True
sage: TV_res.fan().nrays()
8
sage: TV_res.fan().ngenerating_cones()
12
sage: TV.gens()
(z0, z1, z2, z3, z4, z5, z6, z7)
sage: TV_res.gens()
(z0, z1, z2, z3, z4, z5, z6, z7)
sage: TV_res = TV.resolve(coordinate_names="x+",
...                       make_simplicial=True)
sage: TV_res.gens()
(x0, x1, x2, x3, x4, x5, x6, x7)
```

resolve_to_orbifold(***kwds*)

Construct an orbifold whose fan subdivides the fan of `self`.

It is a synonym for `resolve()` with `make_simplicial=True` option.

INPUT:

- this function accepts only keyword arguments. See `resolve()` for documentation.

OUTPUT:

- `toric variety`.

EXAMPLES:

```
sage: fan = NormalFan(lattice_polytope.cross_polytope(3))
sage: TV = ToricVariety(fan)
sage: TV.is_orbifold()
False
```

```
sage: TV.fan().nrays()
8
sage: TV.fan().ngenerating_cones()
6
sage: TV_res = TV.resolve_to_orbifold()
sage: TV_res.is_orbifold()
True
sage: TV_res.fan().nrays()
8
sage: TV_res.fan().ngenerating_cones()
12
```

subscheme (*polynomials*)

Return the subscheme of `self` defined by `polynomials`.

INPUT:

- `polynomials` – list of polynomials in the coordinate ring of `self`.

OUTPUT:

- `subscheme of a toric variety`.

EXAMPLES:

We will construct a subscheme of the product of two projective lines with coordinates (x, y) for one and (s, t) for the other:

```
sage: P1xP1.<x,y,s,t> = toric_varieties.P1xP1()
sage: X = P1xP1.subscheme([x*s + y*t, x^3+y^3])
sage: X
Closed subscheme of 2-d CPR-Fano toric variety
covered by 4 affine patches defined by:
  x*s + y*t,
  x^3 + y^3
sage: X.defining_polynomials()
(x*s + y*t, x^3 + y^3)
sage: X.defining_ideal()
Ideal (x*s + y*t, x^3 + y^3)
of Multivariate Polynomial Ring in x, y, s, t
over Rational Field
sage: X.base_ring()
Rational Field
sage: X.base_scheme()
Spectrum of Rational Field
sage: X.structure_morphism()
Scheme morphism:
  From: Closed subscheme of 2-d CPR-Fano toric variety
        covered by 4 affine patches defined by:
  x*s + y*t,
  x^3 + y^3
  To:   Spectrum of Rational Field
  Defn: Structure map
```

toric_divisor_group (*base_ring=Integer Ring*)

Return the group of toric (T-Weil) divisors.

INPUT:

- `base_ring` – the coefficient ring, usually `ZZ` (default) or `QQ`.

OUTPUT:

The free Abelian agroup of toric Weil divisors, that is, formal base_ring-linear combinations of codimension-one toric subvarieties. The output will be an instance of sage.schemes.toric.divisor.ToricDivisorGroup.

The i-th generator of the divisor group is the divisor where the i-th homogeneous coordinate vanishes, $\{z_i = 0\}$.

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: TDiv = dP6.toric_divisor_group(); TDiv
Group of toric ZZ-Weil divisors on 2-d CPR-Fano toric variety
covered by 6 affine patches
sage: TDiv == dP6.toric_divisor_group()
True
sage: TDiv.gens()
(V(x), V(u), V(y), V(v), V(z), V(w))
sage: dP6.coordinate_ring()
Multivariate Polynomial Ring in x, u, y, v, z, w over Rational Field
```

volume_class()

Return the cohomology class of the volume form on the toric variety.

Note that we are using cohomology with compact supports. If the variety is non-compact this is dual to homology without any support condition. In particular, for non-compact varieties the volume form $\mathrm{dVol} = \wedge_i (dx_i \wedge dy_i)$ does not define a (non-zero) cohomology class.

OUTPUT:

A CohomologyClass. If it exists, it is the class of the (properly normalized) volume form, that is, it is the Poincare dual of a single point. If it does not exist, a ValueError is raised.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2.volume_class()
[z^2]

sage: A2_Z2 = toric_varieties.A2_Z2()
sage: A2_Z2.volume_class()
Traceback (most recent call last):
...
ValueError: Volume class does not exist.
```

If none of the maximal cones is smooth things get more tricky. In this case no torus-fixed point is smooth. If we want to count an ordinary point as 1, then a G-orbifold point needs to count as $\frac{1}{|G|}$. For example, take $\mathbb{P}^1 \times \mathbb{P}^1$ with inhomogeneous coordinates (t, y). Take the quotient by the action $(t, y) \mapsto (-t, -y)$. The $\mathbf{Z}_2$-invariant Weil divisors $\{t = 0\}$ and $\{y = 0\}$ intersect in a $\mathbf{Z}_2$-fixed point, so they ought to have intersection number $\frac{1}{2}$. This means that the cohomology class $[t] \cap [y]$ should be $\frac{1}{2}$ times the volume class. Note that this is different from the volume normalization chosen in [Schubert]:

```
sage: P1xP1_Z2 = toric_varieties.P1xP1_Z2()
sage: Dt = P1xP1_Z2.divisor(1);  Dt
V(t)
sage: Dy = P1xP1_Z2.divisor(3);  Dy
V(y)
sage: P1xP1_Z2.volume_class()
[2*t*y]
```

```
sage: HH = P1xP1_Z2.cohomology_ring()
sage: HH(Dt) * HH(Dy) == 1/2 * P1xP1_Z2.volume_class()
True
```

The fractional coefficients are also necessary to match the normalization in the rational Chow group for simplicial toric varieties:

```
sage: A = P1xP1_Z2.Chow_group(QQ)
sage: A(Dt).intersection_with_divisor(Dy).count_points()
1/2
```

REFERENCES:

`sage.schemes.toric.variety.`**`certify_names`**(*names*)

Make sure that `names` are valid in Python.

INPUT:

- `names` – list of strings.

OUTPUT:

- none, but a `ValueError` exception is raised if `names` are invalid.

Each name must satisfy the following requirements:

- Be non-empty.
- Contain only (Latin) letters, digits, and underscores (“_”).
- Start with a letter.

In addition, all names must be distinct.

EXAMPLES:

```
sage: from sage.schemes.toric.variety import certify_names
sage: certify_names([])
sage: certify_names(["a", "x0", "x_45"])
sage: certify_names(["", "x0", "x_45"])
Traceback (most recent call last):
...
ValueError: name must be nonempty!
sage: certify_names(["a", "0", "x_45"])
Traceback (most recent call last):
...
ValueError: name must start with a letter! Got 0
sage: certify_names(["a", "x0", "@_45"])
Traceback (most recent call last):
...
ValueError: name must be alphanumeric! Got @_45
sage: certify_names(["a", "x0", "x0"])
Traceback (most recent call last):
...
ValueError: names must be distinct! Got: ['a', 'x0', 'x0']
```

`sage.schemes.toric.variety.`**`is_CohomologyClass`**(*x*)

Check whether `x` is a cohomology class of a toric variety.

INPUT:

- `x` – anything.

OUTPUT:

True or False depending on whether x is an instance of CohomologyClass

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: HH = P2.cohomology_ring()
sage: from sage.schemes.toric.variety import is_CohomologyClass
sage: is_CohomologyClass( HH.one() )
True
sage: is_CohomologyClass( HH(P2.fan(1)[0]) )
True
sage: is_CohomologyClass('z')
False
```

sage.schemes.toric.variety.**is_ToricVariety**(*x*)

Check if x is a toric variety.

INPUT:

- x – anything.

OUTPUT:

- True if x is a toric variety and False otherwise.

Note: While projective spaces are toric varieties mathematically, they are not toric varieties in Sage due to efficiency considerations, so this function will return False.

EXAMPLES:

```
sage: from sage.schemes.toric.variety import is_ToricVariety
sage: is_ToricVariety(1)
False
sage: fan = FaceFan(lattice_polytope.cross_polytope(2))
sage: P = ToricVariety(fan)
sage: P
2-d toric variety covered by 4 affine patches
sage: is_ToricVariety(P)
True
sage: is_ToricVariety(ProjectiveSpace(2))
False
```

sage.schemes.toric.variety.**normalize_names**(*names=None*, *ngens=None*, *prefix=None*, *indices=None*, *return_prefix=False*)

Return a list of names in the standard form.

INPUT:

All input parameters are optional.

- names – names given either as a single string (with individual names separated by commas or spaces) or a list of strings with each string specifying a name. If the last name ends with the plus sign, "+", this name will be used as prefix (even if prefix was given explicitly);
- ngens – number of names to be returned;
- prefix – prefix for the indexed names given as a string;
- indices – list of integers (default: range(ngens)) used as indices for names with prefix. If given, must be of length ngens;

- `return_prefix` – if `True`, the last element of the returned list will contain the prefix determined from `names` or given as the parameter `prefix`. This is useful if you may need more names in the future.

OUTPUT:

- list of names given as strings.

These names are constructed in the following way:

1. If necessary, split `names` into separate names.
2. If the last name ends with "+", put it into `prefix`.
3. If `ngens` was given, add to the names obtained so far as many indexed names as necessary to get this number. If the `k`-th name of the *total* list of names is indexed, it is `prefix + str(indices[k])`. If there were already more names than `ngens`, discard "extra" ones.
4. Check if constructed names are valid. See `certify_names()` for details.
5. If the option `return_prefix=True` was given, add `prefix` to the end of the list.

EXAMPLES:

As promised, all parameters are optional:

```
sage: from sage.schemes.toric.variety import normalize_names
sage: normalize_names()
[]
```

One of the most common uses is probably this one:

```
sage: normalize_names("x+", 4)
['x0', 'x1', 'x2', 'x3']
```

Now suppose that you want to enumerate your variables starting with one instead of zero:

```
sage: normalize_names("x+", 4, indices=range(1,5))
['x1', 'x2', 'x3', 'x4']
```

You may actually have an arbitrary enumeration scheme:

```
sage: normalize_names("x+", 4, indices=[1, 10, 100, 1000])
['x1', 'x10', 'x100', 'x1000']
```

Now let's add some "explicit" names:

```
sage: normalize_names("x y z t+", 4)
['x', 'y', 'z', 't3']
```

Note that the "automatic" name is `t3` instead of `t0`. This may seem weird, but the reason for this behaviour is that the fourth name in this list will be the same no matter how many explicit names were given:

```
sage: normalize_names("x y t+", 4)
['x', 'y', 't2', 't3']
```

This is especially useful if you get `names` from a user but want to specify all default names:

```
sage: normalize_names("x, y", 4, prefix="t")
['x', 'y', 't2', 't3']
```

In this format, the user can easily override your choice for automatic names:

```
sage: normalize_names("x y s+", 4, prefix="t")
['x', 'y', 's2', 's3']
```

Let's now use all parameters at once:

```
sage: normalize_names("x, y, s+", 4, prefix="t",
...        indices=range(1,5), return_prefix=True)
['x', 'y', 's3', 's4', 's']
```

Note that you still need to give indices for all names, even if some of the first ones will be "wasted" because of the explicit names. The reason is the same as before - this ensures consistency of automatically generated names, no matter how many explicit names were given.

The prefix is discarded if `ngens` was not given:

```
sage: normalize_names("alpha, beta, gamma, zeta+")
['alpha', 'beta', 'gamma']
```

Finally, let's take a look at some possible mistakes:

```
sage: normalize_names("123")
Traceback (most recent call last):
...
ValueError: name must start with a letter! Got 123
```

A more subtle one:

```
sage: normalize_names("x1", 4, prefix="x")
Traceback (most recent call last):
...
ValueError: names must be distinct! Got: ['x1', 'x1', 'x2', 'x3']
```

16.2 Fano toric varieties

This module provides support for (Crepant Partial Resolutions of) Fano toric varieties, corresponding to crepant subdivisions of face fans of reflexive `lattice polytopes`. The interface is provided via `CPRFanoToricVariety()`.

A careful exposition of different flavours of Fano varieties can be found in the paper by Benjamin Nill [Nill2005]. The main goal of this module is to support work with **Gorenstein weak Fano toric varieties**. Such a variety corresponds to a **coherent crepant refinement of the normal fan of a reflexive polytope** Δ, where crepant means that primitive generators of the refining rays lie on the facets of the polar polytope Δ° and coherent (a.k.a. regular or projective) means that there exists a strictly upper convex piecewise linear function whose domains of linearity are precisely the maximal cones of the subdivision. These varieties are important for string theory in physics, as they serve as ambient spaces for mirror pairs of Calabi-Yau manifolds via constructions due to Victor V. Batyrev [Batyrev1994] and Lev A. Borisov [Borisov1993].

From the combinatorial point of view "crepant" requirement is much more simple and natural to work with than "coherent." For this reason, the code in this module will allow work with arbitrary crepant subdivisions without checking whether they are coherent or not. We refer to corresponding toric varieties as **CPR-Fano toric varieties**.

REFERENCES:

AUTHORS:

- Andrey Novoseltsev (2010-05-18): initial version.

EXAMPLES:

Most of the functions available for Fano toric varieties are the same as for general toric varieties, so here we will concentrate only on Calabi-Yau subvarieties, which were the primary goal for creating this module.

For our first example we realize the projective plane as a Fano toric variety:

```
sage: simplex = LatticePolytope([(1,0), (0,1), (-1,-1)])
sage: P2 = CPRFanoToricVariety(Delta_polar=simplex)
```

Its anticanonical "hypersurface" is a one-dimensional Calabi-Yau manifold:

```
sage: P2.anticanonical_hypersurface(
...           monomial_points="all")
Closed subscheme of 2-d CPR-Fano toric variety
covered by 3 affine patches defined by:
  a0*z0^3 + a9*z0^2*z1 + a7*z0*z1^2
+ a1*z1^3 + a8*z0^2*z2 + a6*z0*z1*z2
+ a4*z1^2*z2 + a5*z0*z2^2
+ a3*z1*z2^2 + a2*z2^3
```

In many cases it is sufficient to work with the "simplified polynomial moduli space" of anticanonical hypersurfaces:

```
sage: P2.anticanonical_hypersurface(
...           monomial_points="simplified")
Closed subscheme of 2-d CPR-Fano toric variety
covered by 3 affine patches defined by:
  a0*z0^3 + a1*z1^3 + a6*z0*z1*z2 + a2*z2^3
```

The mirror family to these hypersurfaces lives inside the Fano toric variety obtained using `simplex` as `Delta` instead of `Delta_polar`:

```
sage: FTV = CPRFanoToricVariety(Delta=simplex,
...           coordinate_points="all")
sage: FTV.anticanonical_hypersurface(
...           monomial_points="simplified")
Closed subscheme of 2-d CPR-Fano toric variety
covered by 9 affine patches defined by:
  a2*z2^3*z3^2*z4*z5^2*z8
+ a1*z1^3*z3*z4^2*z7^2*z9
+ a3*z0*z1*z2*z3*z4*z5*z7*z8*z9
+ a0*z0^3*z5*z7*z8^2*z9^2
```

Here we have taken the resolved version of the ambient space for the mirror family, but in fact we don't have to resolve singularities corresponding to the interior points of facets - they are singular points which do not lie on a generic anticanonical hypersurface:

```
sage: FTV = CPRFanoToricVariety(Delta=simplex,
...           coordinate_points="all but facets")
sage: FTV.anticanonical_hypersurface(
...           monomial_points="simplified")
Closed subscheme of 2-d CPR-Fano toric variety
covered by 3 affine patches defined by:
  a0*z0^3 + a1*z1^3 + a3*z0*z1*z2 + a2*z2^3
```

This looks very similar to our second version of the anticanonical hypersurface of the projective plane, as expected, since all one-dimensional Calabi-Yau manifolds are elliptic curves!

Now let's take a look at a toric realization of M-polarized K3 surfaces studied by Adrian Clingher and Charles F. Doran in [CD2007]:

```
sage: p4318 = ReflexivePolytope(3, 4318)  # long time
sage: FTV = CPRFanoToricVariety(Delta_polar=p4318)  # long time
sage: FTV.anticanonical_hypersurface()  # long time
Closed subscheme of 3-d CPR-Fano toric variety
```

```
covered by 4 affine patches defined by:
  a3*z2^12 + a4*z2^6*z3^6 + a2*z3^12
+ a8*z0*z1*z2*z3 + a0*z1^3 + a1*z0^2
```

Below you will find detailed descriptions of available functions. Current functionality of this module is very basic, but it is under active development and hopefully will improve in future releases of Sage. If there are some particular features that you would like to see implemented ASAP, please consider reporting them to the Sage Development Team or even implementing them on your own as a patch for inclusion!

class `sage.schemes.toric.fano_variety.`**`AnticanonicalHypersurface`**(*P_Delta*, *monomial_points=None*, *coefficient_names=None*, *coefficient_name_indices=None*, *coefficients=None*)

Bases: `sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_toric`

Construct an anticanonical hypersurface of a CPR-Fano toric variety.

INPUT:

- `P_Delta` – `CPR-Fano toric variety` associated to a reflexive polytope Δ;
- see `CPRFanoToricVariety_field.anticanonical_hypersurface()` for documentation on all other acceptable parameters.

OUTPUT:

- `anticanonical hypersurface` of `P_Delta` (with the extended base field, if necessary).

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: import sage.schemes.toric.fano_variety as ftv
sage: ftv.AnticanonicalHypersurface(P1xP1)
Closed subscheme of 2-d CPR-Fano toric variety
covered by 4 affine patches defined by:
  a1*s^2*x^2 + a0*t^2*x^2 + a6*s*t*x*y + a3*s^2*y^2 + a2*t^2*y^2
```

See `anticanonical_hypersurface()` for a more elaborate example.

`sage.schemes.toric.fano_variety.`**`CPRFanoToricVariety`**(*Delta=None*, *Delta_polar=None*, *coordinate_points=None*, *charts=None*, *coordinate_names=None*, *names=None*, *coordinate_name_indices=None*, *make_simplicial=False*, *base_ring=None*, *base_field=None*, *check=True*)

Construct a CPR-Fano toric variety.

Note: See documentation of the module `fano_variety` for the used definitions and supported varieties.

Due to the large number of available options, it is recommended to always use keyword parameters.

INPUT:

- `Delta` – reflexive `lattice polytope`. The fan of the constructed CPR-Fano toric variety will be a crepant subdivision of the *normal fan* of `Delta`. Either `Delta` or `Delta_polar` must be given, but not both at the same time, since one is completely determined by another via `polar` method;
- `Delta_polar` – reflexive `lattice polytope`. The fan of the constructed CPR-Fano toric variety will be a crepant subdivision of the *face fan* of `Delta_polar`. Either `Delta` or `Delta_polar` must be given, but not both at the same time, since one is completely determined by another via `polar` method;
- `coordinate_points` – list of integers or string. A list will be interpreted as indices of (boundary) points of `Delta_polar` which should be used as rays of the underlying fan. It must include all vertices of `Delta_polar` and no repetitions are allowed. A string must be one of the following descriptions of points of `Delta_polar`:
  - "vertices" (default),
  - "all" (will not include the origin),
  - "all but facets" (will not include points in the relative interior of facets);
- `charts` – list of lists of elements from `coordinate_points`. Each of these lists must define a generating cone of a fan subdividing the normal fan of `Delta`. Default `charts` correspond to the normal fan of `Delta` without subdivision. The fan specified by `charts` will be subdivided to include all of the requested `coordinate_points`;
- `coordinate_names` – names of variables for the coordinate ring, see `normalize_names()` for acceptable formats. If not given, indexed variable names will be created automatically;
- `names` – an alias of `coordinate_names` for internal use. You may specify either `names` or `coordinate_names`, but not both;
- `coordinate_name_indices` – list of integers, indices for indexed variables. If not given, the index of each variable will coincide with the index of the corresponding point of `Delta_polar`;
- `make_simplicial` – if `True`, the underlying fan will be made simplicial (default: `False`);
- `base_ring` – base field of the CPR-Fano toric variety (default: $\mathbf{Q}$);
- `base_field` – alias for `base_ring`. Takes precedence if both are specified.
- `check` – by default the input data will be checked for correctness (e.g. that `charts` do form a subdivision of the normal fan of `Delta`). If you know for sure that the input is valid, you may significantly decrease construction time using `check=False` option.

OUTPUT:

- `CPR-Fano toric variety`.

EXAMPLES:

We start with the product of two projective lines:

```
sage: diamond = lattice_polytope.cross_polytope(2)
sage: diamond.vertices_pc()
M( 1,  0),
M( 0,  1),
M(-1,  0),
M( 0, -1)
in 2-d lattice M
sage: P1xP1 = CPRFanoToricVariety(Delta_polar=diamond)
sage: P1xP1
2-d CPR-Fano toric variety covered by 4 affine patches
sage: P1xP1.fan()
Rational polyhedral fan in 2-d lattice M
sage: P1xP1.fan().rays()
```

```
M( 1,  0),
M( 0,  1),
M(-1,  0),
M( 0, -1)
in 2-d lattice M
```

"Unfortunately," this variety is smooth to start with and we cannot perform any subdivisions of the underlying fan without leaving the category of CPR-Fano toric varieties. Our next example starts with a square:

```
sage: square = diamond.polar()
sage: square.vertices_pc()
N(-1,  1),
N( 1,  1),
N(-1, -1),
N( 1, -1)
in 2-d lattice N
sage: square.points_pc()
N(-1,  1),
N( 1,  1),
N(-1, -1),
N( 1, -1),
N(-1,  0),
N( 0, -1),
N( 0,  0),
N( 0,  1),
N( 1,  0)
in 2-d lattice N
```

We will construct several varieties associated to it:

```
sage: FTV = CPRFanoToricVariety(Delta_polar=square)
sage: FTV.fan().rays()
N(-1,  1),
N( 1,  1),
N(-1, -1),
N( 1, -1)
in 2-d lattice N
sage: FTV.gens()
(z0, z1, z2, z3)

sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...         coordinate_points=[0,1,2,3,8])
sage: FTV.fan().rays()
N(-1,  1),
N( 1,  1),
N(-1, -1),
N( 1, -1),
N( 1,  0)
in 2-d lattice N
sage: FTV.gens()
(z0, z1, z2, z3, z8)

sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...         coordinate_points=[8,0,2,1,3],
...         coordinate_names="x+")
sage: FTV.fan().rays()
N( 1,  0),
N(-1,  1),
N(-1, -1),
```

```
N( 1,  1),
N( 1, -1)
in 2-d lattice N
sage: FTV.gens()
(x8, x0, x2, x1, x3)

sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points="all",
...           coordinate_names="x y Z+")
sage: FTV.fan().rays()
N(-1,  1),
N( 1,  1),
N(-1, -1),
N( 1, -1),
N(-1,  0),
N( 0, -1),
N( 0,  1),
N( 1,  0)
in 2-d lattice N
sage: FTV.gens()
(x, y, Z2, Z3, Z4, Z5, Z7, Z8)
```

Note that `Z6` is "missing". This is due to the fact that the 6-th point of `square` is the origin, and all automatically created names have the same indices as corresponding points of `Delta_polar()`. This is usually very convenient, especially if you have to work with several partial resolutions of the same Fano toric variety. However, you can change it, if you want:

```
sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points="all",
...           coordinate_names="x y Z+",
...           coordinate_name_indices=range(8))
sage: FTV.gens()
(x, y, Z2, Z3, Z4, Z5, Z6, Z7)
```

Note that you have to provide indices for *all* variables, including those that have "completely custom" names. Again, this is usually convenient, because you can add or remove "custom" variables without disturbing too much "automatic" ones:

```
sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points="all",
...           coordinate_names="x Z+",
...           coordinate_name_indices=range(8))
sage: FTV.gens()
(x, Z1, Z2, Z3, Z4, Z5, Z6, Z7)
```

If you prefer to always start from zero, you will have to shift indices accordingly:

```
sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points="all",
...           coordinate_names="x Z+",
...           coordinate_name_indices=[0] + range(7))
sage: FTV.gens()
(x, Z0, Z1, Z2, Z3, Z4, Z5, Z6)

sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points="all",
...           coordinate_names="x y Z+",
...           coordinate_name_indices=[0]*2 + range(6))
sage: FTV.gens()
```

```
(x, y, Z0, Z1, Z2, Z3, Z4, Z5)
```

So you always can get any names you want, somewhat complicated default behaviour was designed with the hope that in most cases you will have no desire to provide different names.

Now we will use the possibility to specify initial charts:

```
sage: charts = [(0,1), (1,3), (3,2), (2,0)]
```

(these charts actually form exactly the face fan of our square)

```
sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points=[0,1,2,3,4],
...           charts=charts)
sage: FTV.fan().rays()
N(-1,  1),
N( 1,  1),
N(-1, -1),
N( 1, -1),
N(-1,  0)
in 2-d lattice N
sage: [cone.ambient_ray_indices() for cone in FTV.fan()]
[(0, 1), (1, 3), (2, 3), (0, 4), (2, 4)]
```

If charts are wrong, it should be detected:

```
sage: bad_charts = charts + [(2,0)]
sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points=[0,1,2,3,4],
...           charts=bad_charts)
Traceback (most recent call last):
...
ValueError: you have provided 5 cones, but only 4 of them are maximal!
Use discard_faces=True if you indeed need to construct a fan from
these cones.
```

These charts are technically correct, they just happened to list one of them twice, but it is assumed that such a situation will not happen. It is especially important when you try to speed up your code:

```
sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points=[0,1,2,3,4],
...           charts=bad_charts,
...           check=False)
Traceback (most recent call last):
...
IndexError: list assignment index out of range
```

In this case you still get an error message, but it is harder to figure out what is going on. It may also happen that "everything will still work" in the sense of not crashing, but work with such an invalid variety may lead to mathematically wrong results, so use `check=False` carefully!

Here are some other possible mistakes:

```
sage: bad_charts = charts + [(0,3)]
sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points=[0,1,2,3,4],
...           charts=bad_charts)
Traceback (most recent call last):
...
ValueError: (0, 3) does not form a chart of a subdivision of
```

```
the face fan of 2-d reflexive polytope #14 in 2-d lattice N!

sage: bad_charts = charts[:-1]
sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points=[0,1,2,3,4],
...           charts=bad_charts)
Traceback (most recent call last):
...
ValueError: given charts do not form a complete fan!

sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points=[1,2,3,4])
Traceback (most recent call last):
...
ValueError: all 4 vertices of Delta_polar
must be used for coordinates!
Got: [1, 2, 3, 4]

sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points=[0,0,1,2,3,4])
Traceback (most recent call last):
...
ValueError: no repetitions are
allowed for coordinate points!
Got: [0, 0, 1, 2, 3, 4]

sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...           coordinate_points=[0,1,2,3,6])
Traceback (most recent call last):
...
ValueError: the origin (point #6)
cannot be used for a coordinate!
Got: [0, 1, 2, 3, 6]
```

Here is a shorthand for defining the toric variety and homogeneous coordinates in one go:

```
sage: P1xP1.<a,b,c,d> = CPRFanoToricVariety(Delta_polar=diamond)
sage: (a^2+b^2) * (c+d)
a^2*c + b^2*c + a^2*d + b^2*d
```

class `sage.schemes.toric.fano_variety.`**`CPRFanoToricVariety_field`**(*Delta_polar*, *fan*, *coordinate_points*, *point_to_ray*, *coordinate_names*, *coordinate_name_indices*, *base_field*)

Bases: `sage.schemes.toric.variety.ToricVariety_field`

Construct a CPR-Fano toric variety associated to a reflexive polytope.

> **Warning:** This class does not perform any checks of correctness of input and it does assume that the internal structure of the given parameters is coordinated in a certain way. Use `CPRFanoToricVariety()` to construct CPR-Fano toric varieties.

Note: See documentation of the module `fano_variety` for the used definitions and supported varieties.

INPUT:

- `Delta_polar` – reflexive polytope;
- `fan` – rational polyhedral fan subdividing the face fan of `Delta_polar`;
- `coordinate_points` – list of indices of points of `Delta_polar` used for rays of `fan`;
- `point_to_ray` – dictionary mapping the index of a coordinate point to the index of the corresponding ray;
- `coordinate_names` – names of the variables of the coordinate ring in the format accepted by `normalize_names()`;
- `coordinate_name_indices` – indices for indexed variables, if `None`, will be equal to `coordinate_points`;
- `base_field` – base field of the CPR-Fano toric variety.

OUTPUT:

- `CPR-Fano toric variety`.

TESTS:

```
sage: P1xP1 = CPRFanoToricVariety(
...       Delta_polar=lattice_polytope.cross_polytope(2))
sage: P1xP1
2-d CPR-Fano toric variety covered by 4 affine patches
```

Delta()

Return the reflexive polytope associated to `self`.

OUTPUT:

- reflexive `lattice polytope`. The underlying fan of `self` is a coherent subdivision of the *normal fan* of this polytope.

EXAMPLES:

```
sage: diamond = lattice_polytope.cross_polytope(2)
sage: P1xP1 = CPRFanoToricVariety(Delta_polar=diamond)
sage: P1xP1.Delta()
2-d reflexive polytope #14 in 2-d lattice N
sage: P1xP1.Delta() is diamond.polar()
True
```

Delta_polar()

Return polar of `Delta()`.

OUTPUT:

- reflexive `lattice polytope`. The underlying fan of `self` is a coherent subdivision of the *face fan* of this polytope.

EXAMPLES:

```
sage: diamond = lattice_polytope.cross_polytope(2)
sage: P1xP1 = CPRFanoToricVariety(Delta_polar=diamond)
sage: P1xP1.Delta_polar()
2-d reflexive polytope #3 in 2-d lattice M
sage: P1xP1.Delta_polar() is diamond
True
```

```
sage: P1xP1.Delta_polar() is P1xP1.Delta().polar()
True
```

anticanonical_hypersurface(***kwds*)

Return an anticanonical hypersurface of `self`.

Note: The returned hypersurface may be actually a subscheme of **another** CPR-Fano toric variety: if the base field of `self` does not include all of the required names for generic monomial coefficients, it will be automatically extended.

Below Δ is the reflexive polytope corresponding to `self`, i.e. the fan of `self` is a refinement of the normal fan of Δ. This function accepts only keyword parameters.

INPUT:

- `monomial points` – a list of integers or a string. A list will be interpreted as indices of points of Δ which should be used for monomials of this hypersurface. A string must be one of the following descriptions of points of Δ:
  - "vertices",
  - "vertices+origin",
  - "all",
  - "simplified" (default) – all points of Δ except for the interior points of facets, this choice corresponds to working with the "simplified polynomial moduli space" of anticanonical hypersurfaces;
- `coefficient_names` – names for the monomial coefficients, see `normalize_names()` for acceptable formats. If not given, indexed coefficient names will be created automatically;
- `coefficient_name_indices` – a list of integers, indices for indexed coefficients. If not given, the index of each coefficient will coincide with the index of the corresponding point of Δ;
- `coefficients` – as an alternative to specifying coefficient names and/or indices, you can give the coefficients themselves as arbitrary expressions and/or strings. Using strings allows you to easily add "parameters": the base field of `self` will be extended to include all necessary names.

OUTPUT:

- an `anticanonical hypersurface` of `self` (with the extended base field, if necessary).

EXAMPLES:

We realize the projective plane as a Fano toric variety:

```
sage: simplex = LatticePolytope([(1,0), (0,1), (-1,-1)])
sage: P2 = CPRFanoToricVariety(Delta_polar=simplex)
```

Its anticanonical "hypersurface" is a one-dimensional Calabi-Yau manifold:

```
sage: P2.anticanonical_hypersurface(
...         monomial_points="all")
Closed subscheme of 2-d CPR-Fano toric variety
covered by 3 affine patches defined by:
  a0*z0^3 + a9*z0^2*z1 + a7*z0*z1^2
+ a1*z1^3 + a8*z0^2*z2 + a6*z0*z1*z2
+ a4*z1^2*z2 + a5*z0*z2^2
+ a3*z1*z2^2 + a2*z2^3
```

In many cases it is sufficient to work with the "simplified polynomial moduli space" of anticanonical hypersurfaces:

```
sage: P2.anticanonical_hypersurface(
...           monomial_points="simplified")
Closed subscheme of 2-d CPR-Fano toric variety
covered by 3 affine patches defined by:
  a0*z0^3 + a1*z1^3 + a6*z0*z1*z2 + a2*z2^3
```

The mirror family to these hypersurfaces lives inside the Fano toric variety obtained using `simplex` as `Delta` instead of `Delta_polar`:

```
sage: FTV = CPRFanoToricVariety(Delta=simplex,
...           coordinate_points="all")
sage: FTV.anticanonical_hypersurface(
...           monomial_points="simplified")
Closed subscheme of 2-d CPR-Fano toric variety
covered by 9 affine patches defined by:
  a2*z2^3*z3^2*z4*z5^2*z8
+ a1*z1^3*z3*z4^2*z7^2*z9
+ a3*z0*z1*z2*z3*z4*z5*z7*z8*z9
+ a0*z0^3*z5*z7*z8^2*z9^2
```

Here we have taken the resolved version of the ambient space for the mirror family, but in fact we don't have to resolve singularities corresponding to the interior points of facets - they are singular points which do not lie on a generic anticanonical hypersurface:

```
sage: FTV = CPRFanoToricVariety(Delta=simplex,
...           coordinate_points="all but facets")
sage: FTV.anticanonical_hypersurface(
...           monomial_points="simplified")
Closed subscheme of 2-d CPR-Fano toric variety
covered by 3 affine patches defined by:
  a0*z0^3 + a1*z1^3 + a3*z0*z1*z2 + a2*z2^3
```

This looks very similar to our second anticanonical hypersurface of the projective plane, as expected, since all one-dimensional Calabi-Yau manifolds are elliptic curves!

All anticanonical hypersurfaces constructed above were generic with automatically generated coefficients. If you want, you can specify your own names

```
sage: FTV.anticanonical_hypersurface(
...           coefficient_names="a b c d")
Closed subscheme of 2-d CPR-Fano toric variety
covered by 3 affine patches defined by:
  a*z0^3 + b*z1^3 + d*z0*z1*z2 + c*z2^3
```

or give concrete coefficients

```
sage: FTV.anticanonical_hypersurface(
...           coefficients=[1, 2, 3, 4])
Closed subscheme of 2-d CPR-Fano toric variety
covered by 3 affine patches defined by:
  z0^3 + 2*z1^3 + 4*z0*z1*z2 + 3*z2^3
```

or even mix numerical coefficients with some expressions

```
sage: H = FTV.anticanonical_hypersurface(
...       coefficients=[0, "t", "1/t", "psi/(psi^2 + phi)"])
sage: H
Closed subscheme of 2-d CPR-Fano toric variety
covered by 3 affine patches defined by:
  t*z1^3 + (psi/(psi^2 + phi))*z0*z1*z2 + 1/t*z2^3
```

```
sage: R = H.ambient_space().base_ring()
sage: R
Fraction Field of
Multivariate Polynomial Ring in phi, psi, t
over Rational Field
```

cartesian_product(*other*, *coordinate_names=None*, *coordinate_indices=None*)

Return the Cartesian product of `self` with `other`.

INPUT:

- `other` – a (possibly `CPR-Fano`) `toric variety`;
- `coordinate_names` – names of variables for the coordinate ring, see `normalize_names()` for acceptable formats. If not given, indexed variable names will be created automatically;
- `coordinate_indices` – list of integers, indices for indexed variables. If not given, the index of each variable will coincide with the index of the corresponding ray of the fan.

OUTPUT:

- a `toric variety`, which is `CPR-Fano` if `other` was.

EXAMPLES:

```
sage: P1 = toric_varieties.P1()
sage: P2 = toric_varieties.P2()
sage: P1xP2 = P1.cartesian_product(P2); P1xP2
3-d CPR-Fano toric variety covered by 6 affine patches
sage: P1xP2.fan().rays()
N+N( 1,  0,  0),
N+N(-1,  0,  0),
N+N( 0,  1,  0),
N+N( 0,  0,  1),
N+N( 0, -1, -1)
in 3-d lattice N+N
sage: P1xP2.Delta_polar()
3-d reflexive polytope in 3-d lattice N+N
```

change_ring(*F*)

Return a CPR-Fano toric variety over field `F`, otherwise the same as `self`.

INPUT:

- `F` – field.

OUTPUT:

- `CPR-Fano toric variety` over `F`.

Note: There is no need to have any relation between `F` and the base field of `self`. If you do want to have such a relation, use `base_extend()` instead.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1.base_ring()
Rational Field
sage: P1xP1_RR = P1xP1.change_ring(RR)
sage: P1xP1_RR.base_ring()
Real Field with 53 bits of precision
sage: P1xP1_QQ = P1xP1_RR.change_ring(QQ)
```

```
sage: P1xP1_QQ.base_ring()
Rational Field
sage: P1xP1_RR.base_extend(QQ)
Traceback (most recent call last):
...
ValueError: no natural map from the base ring
(=Real Field with 53 bits of precision)
to R (=Rational Field)!
sage: R = PolynomialRing(QQ, 2, 'a')
sage: P1xP1.change_ring(R)
Traceback (most recent call last):
...
TypeError: need a field to construct a Fano toric variety!
 Got Multivariate Polynomial Ring in a0, a1 over Rational Field
```

coordinate_point_to_coordinate(*point*)

Return the variable of the coordinate ring corresponding to `point`.

INPUT:

- `point` – integer from the list of `coordinate_points()`.

OUTPUT:

- the corresponding generator of the coordinate ring of `self`.

EXAMPLES:

```
sage: diamond = lattice_polytope.cross_polytope(2)
sage: FTV = CPRFanoToricVariety(diamond,
...             coordinate_points=[0,1,2,3,8])
sage: FTV.coordinate_points()
(0, 1, 2, 3, 8)
sage: FTV.gens()
(z0, z1, z2, z3, z8)
sage: FTV.coordinate_point_to_coordinate(8)
z8
```

coordinate_points()

Return indices of points of `Delta_polar()` used for coordinates.

OUTPUT:

- `tuple` of integers.

EXAMPLES:

```
sage: diamond = lattice_polytope.cross_polytope(2)
sage: square = diamond.polar()
sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...             coordinate_points=[0,1,2,3,8])
sage: FTV.coordinate_points()
(0, 1, 2, 3, 8)
sage: FTV.gens()
(z0, z1, z2, z3, z8)

sage: FTV = CPRFanoToricVariety(Delta_polar=square,
...             coordinate_points="all")
sage: FTV.coordinate_points()
(0, 1, 2, 3, 4, 5, 7, 8)
sage: FTV.gens()
(z0, z1, z2, z3, z4, z5, z7, z8)
```

Note that one point is missing, namely

```
sage: square.origin()
6
```

nef_complete_intersection(*nef_partition*, ***kwds*)

Return a nef complete intersection in `self`.

Note: The returned complete intersection may be actually a subscheme of **another** CPR-Fano toric variety: if the base field of `self` does not include all of the required names for monomial coefficients, it will be automatically extended.

Below Δ is the reflexive polytope corresponding to `self`, i.e. the fan of `self` is a refinement of the normal fan of Δ. Other polytopes are described in the documentation of `nef-partitions` of `reflexive polytopes`.

Except for the first argument, `nef_partition`, this method accepts only keyword parameters.

INPUT:

- `nef_partition` – a k-part `nef-partition` of Δ°, all other parameters (if given) must be lists of length k;
- `monomial_points` – the i-th element of this list is either a list of integers or a string. A list will be interpreted as indices of points of Δ_i which should be used for monomials of the i-th polynomial of this complete intersection. A string must be one of the following descriptions of points of Δ_i:
  - "vertices",
  - "vertices+origin",
  - "all" (default),

  when using this description, it is also OK to pass a single string as `monomial_points` instead of repeating it k times;
- `coefficient_names` – the i-th element of this list specifies names for the monomial coefficients of the i-th polynomial, see `normalize_names()` for acceptable formats. If not given, indexed coefficient names will be created automatically;
- `coefficient_name_indices` – the i-th element of this list specifies indices for indexed coefficients of the i-th polynomial. If not given, the index of each coefficient will coincide with the index of the corresponding point of Δ_i;
- `coefficients` – as an alternative to specifying coefficient names and/or indices, you can give the coefficients themselves as arbitrary expressions and/or strings. Using strings allows you to easily add "parameters": the base field of `self` will be extended to include all necessary names.

OUTPUT:

- a `nef complete intersection` of `self` (with the extended base field, if necessary).

EXAMPLES:

We construct several complete intersections associated to the same nef-partition of the 3-dimensional reflexive polytope #2254:

```
sage: p = ReflexivePolytope(3, 2254)   # long time (7s on sage.math, 2011)
sage: np = p.nef_partitions()[1]       # long time
sage: np  # long time
Nef-partition {2, 3, 4, 7, 8} U {0, 1, 5, 6}
```

```
sage: X = CPRFanoToricVariety(Delta_polar=p)  # long time
sage: X.nef_complete_intersection(np)  # long time
Closed subscheme of 3-d CPR-Fano toric variety
covered by 10 affine patches defined by:
  a2*z1*z4^2*z5^2*z7^3 + a1*z2*z4*z5*z6*z7^2*z8^2
  + a3*z2*z3*z4*z7*z8 + a0*z0*z2,
  b2*z1*z4*z5^2*z6^2*z7^2*z8^2 + b0*z2*z5*z6^3*z7*z8^4
  + b5*z1*z3*z4*z5*z6*z7*z8 + b3*z2*z3*z6^2*z8^3
  + b1*z1*z3^2*z4 + b4*z0*z1*z5*z6
```

Now we include only monomials associated to vertices of Δ_i:

```
sage: X.nef_complete_intersection(np, monomial_points="vertices")  # long time
Closed subscheme of 3-d CPR-Fano toric variety
covered by 10 affine patches defined by:
  a2*z1*z4^2*z5^2*z7^3 + a1*z2*z4*z5*z6*z7^2*z8^2
  + a3*z2*z3*z4*z7*z8 + a0*z0*z2,
  b2*z1*z4*z5^2*z6^2*z7^2*z8^2 + b0*z2*z5*z6^3*z7*z8^4
  + b3*z2*z3*z6^2*z8^3 + b1*z1*z3^2*z4 + b4*z0*z1*z5*z6
```

(effectively, we set `b5=0`). Next we provide coefficients explicitly instead of using default generic names:

```
sage: X.nef_complete_intersection(np,  # long time
...         monomial_points="vertices",
...         coefficients=[("a", "a^2", "a/e", "c_i"), range(1,6)])
Closed subscheme of 3-d CPR-Fano toric variety
covered by 10 affine patches defined by:
  a/e*z1*z4^2*z5^2*z7^3 + a^2*z2*z4*z5*z6*z7^2*z8^2
  + c_i*z2*z3*z4*z7*z8 + a*z0*z2,
  3*z1*z4*z5^2*z6^2*z7^2*z8^2 + z2*z5*z6^3*z7*z8^4
  + 4*z2*z3*z6^2*z8^3 + 2*z1*z3^2*z4 + 5*z0*z1*z5*z6
```

Finally, we take a look at the generic representative of these complete intersections in a completely resolved ambient toric variety:

```
sage: X = CPRFanoToricVariety(Delta_polar=p,  # long time
...                          coordinate_points="all")
sage: X.nef_complete_intersection(np)  # long time
Closed subscheme of 3-d CPR-Fano toric variety
covered by 22 affine patches defined by:
  a1*z2*z4*z5*z6*z7^2*z8^2*z9^2*z10^2*z11*z12*z13
  + a2*z1*z4^2*z5^2*z7^3*z9*z10^2*z12*z13
  + a3*z2*z3*z4*z7*z8*z9*z10*z11*z12 + a0*z0*z2,
  b0*z2*z5*z6^3*z7*z8^4*z9^3*z10^2*z11^2*z12*z13^2
  + b2*z1*z4*z5^2*z6^2*z7^2*z8^2*z9^2*z10^2*z11*z12*z13^2
  + b3*z2*z3*z6^2*z8^3*z9^2*z10*z11^2*z12*z13
  + b5*z1*z3*z4*z5*z6*z7*z8*z9*z10*z11*z12*z13
  + b1*z1*z3^2*z4*z11*z12 + b4*z0*z1*z5*z6*z13
```

resolve(***kwds*)

Construct a toric variety whose fan subdivides the fan of `self`.

This function accepts only keyword arguments, none of which are mandatory.

INPUT:

- `new_points` – list of integers, indices of boundary points of `Delta_polar()`, which should be added as rays to the subdividing fan;
- all other arguments will be passed to `resolve()` method of (general) toric varieties, see its docu-

mentation for details.

OUTPUT:

•CPR-Fano toric variety if there was no new_rays argument and toric variety otherwise.

EXAMPLES:

```
sage: diamond = lattice_polytope.cross_polytope(2)
sage: FTV = CPRFanoToricVariety(Delta=diamond)
sage: FTV.coordinate_points()
(0, 1, 2, 3)
sage: FTV.gens()
(z0, z1, z2, z3)
sage: FTV_res = FTV.resolve(new_points=[6,8])
Traceback (most recent call last):
...
ValueError: the origin (point #6)
cannot be used for subdivision!
sage: FTV_res = FTV.resolve(new_points=[8,5])
sage: FTV_res
2-d CPR-Fano toric variety covered by 6 affine patches
sage: FTV_res.coordinate_points()
(0, 1, 2, 3, 8, 5)
sage: FTV_res.gens()
(z0, z1, z2, z3, z8, z5)

sage: TV_res = FTV.resolve(new_rays=[(1,2)])
sage: TV_res
2-d toric variety covered by 5 affine patches
sage: TV_res.gens()
(z0, z1, z2, z3, z4)
```

class sage.schemes.toric.fano_variety.**NefCompleteIntersection**(*P_Delta*, *nef_partition*, *monomial_points='all'*, *coefficient_names=None*, *coefficient_name_indices=None*, *coefficients=None*)

Bases: sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_toric

Construct a nef complete intersection in a CPR-Fano toric variety.

INPUT:

•P_Delta – a CPR-Fano toric variety associated to a reflexive polytope Δ;

•see CPRFanoToricVariety_field.nef_complete_intersection() for documentation on all other acceptable parameters.

OUTPUT:

•a nef complete intersection of P_Delta (with the extended base field, if necessary).

EXAMPLES:

```
sage: o = lattice_polytope.cross_polytope(3)
sage: np = o.nef_partitions()[0]
sage: np
Nef-partition {0, 1, 3} U {2, 4, 5}
```

```
sage: X = CPRFanoToricVariety(Delta_polar=o)
sage: X.nef_complete_intersection(np)
Closed subscheme of 3-d CPR-Fano toric variety
covered by 8 affine patches defined by:
  a1*z0^2*z1 + a4*z0*z1*z3 + a3*z1*z3^2
  + a0*z0^2*z4 + a5*z0*z3*z4 + a2*z3^2*z4,
  b0*z1*z2^2 + b1*z2^2*z4 + b4*z1*z2*z5
  + b5*z2*z4*z5 + b3*z1*z5^2 + b2*z4*z5^2
```

See `CPRFanoToricVariety_field.nef_complete_intersection()` for a more elaborate example.

nef_partition()

Return the nef-partition associated to `self`.

OUTPUT:

- a `nef-partition`.

EXAMPLES:

```
sage: o = lattice_polytope.cross_polytope(3)
sage: np = o.nef_partitions()[0]
sage: np
Nef-partition {0, 1, 3} U {2, 4, 5}
sage: X = CPRFanoToricVariety(Delta_polar=o)
sage: CI = X.nef_complete_intersection(np)
sage: CI
Closed subscheme of 3-d CPR-Fano toric variety
covered by 8 affine patches defined by:
  a1*z0^2*z1 + a4*z0*z1*z3 + a3*z1*z3^2
  + a0*z0^2*z4 + a5*z0*z3*z4 + a2*z3^2*z4,
  b0*z1*z2^2 + b1*z2^2*z4 + b4*z1*z2*z5
  + b5*z2*z4*z5 + b3*z1*z5^2 + b2*z4*z5^2
sage: CI.nef_partition()
Nef-partition {0, 1, 3} U {2, 4, 5}
sage: CI.nef_partition() is np
True
```

`sage.schemes.toric.fano_variety.`**add_variables**(*field*, *variables*)

Extend `field` to include all `variables`.

INPUT:

- `field` - a field;
- `variables` - a list of strings.

OUTPUT:

- a fraction field extending the original `field`, which has all `variables` among its generators.

EXAMPLES:

We start with the rational field and slowly add more variables:

```
sage: from sage.schemes.toric.fano_variety import *
sage: F = add_variables(QQ, []); F      # No extension
Rational Field
sage: F = add_variables(QQ, ["a"]); F
Fraction Field of Univariate Polynomial Ring
in a over Rational Field
sage: F = add_variables(F, ["a"]); F
```

```
Fraction Field of Univariate Polynomial Ring
in a over Rational Field
sage: F = add_variables(F, ["b", "c"]); F
Fraction Field of Multivariate Polynomial Ring
in a, b, c over Rational Field
sage: F = add_variables(F, ["c", "d", "b", "c", "d"]); F
Fraction Field of Multivariate Polynomial Ring
in a, b, c, d over Rational Field
```

sage.schemes.toric.fano_variety.**is_CPRFanoToricVariety**(*x*)

Check if x is a CPR-Fano toric variety.

INPUT:

- x – anything.

OUTPUT:

- True if x is a CPR-Fano toric variety and False otherwise.

Note: While projective spaces are Fano toric varieties mathematically, they are not toric varieties in Sage due to efficiency considerations, so this function will return False.

EXAMPLES:

```
sage: from sage.schemes.toric.fano_variety import (
...       is_CPRFanoToricVariety)
sage: is_CPRFanoToricVariety(1)
False
sage: FTV = toric_varieties.P2()
sage: FTV
2-d CPR-Fano toric variety covered by 3 affine patches
sage: is_CPRFanoToricVariety(FTV)
True
sage: is_CPRFanoToricVariety(ProjectiveSpace(2))
False
```

16.3 Library of toric varieties

This module provides a simple way to construct often-used toric varieties. Please see the help for the individual methods of toric_varieties for a more detailed description of which varieties can be constructed.

AUTHORS:

- Volker Braun (2010-07-02): initial version

EXAMPLES:

```
sage: toric_varieties.dP6()
2-d CPR-Fano toric variety covered by 6 affine patches
```

You can assign the homogeneous coordinates to Sage variables either with inject_variables() or immediately during assignment like this:

```
sage: P2.<x,y,z> = toric_varieties.P2()
sage: x^2 + y^2 + z^2
x^2 + y^2 + z^2
```

```
sage: P2.coordinate_ring()
Multivariate Polynomial Ring in x, y, z over Rational Field
```

class sage.schemes.toric.library.**ToricVarietyFactory**

Bases: sage.structure.sage_object.SageObject

The methods of this class construct toric varieties.

> **Warning:** You need not create instances of this class. Use the already-provided object toric_varieties instead.

A (*n*, *names='z+'*, *base_ring=Rational Field*)

Construct the n-dimensional affine space.

INPUT:

- n – positive integer. The dimension of the affine space.
- names – string. Names for the homogeneous coordinates. See normalize_names() for acceptable formats.
- base_ring – a ring (default: **Q**). The base ring for the toric variety.

OUTPUT:

A toric variety.

EXAMPLES:

```
sage: A3 = toric_varieties.A(3)
sage: A3
3-d affine toric variety
sage: A3.fan().rays()
N(1, 0, 0),
N(0, 1, 0),
N(0, 0, 1)
in 3-d lattice N
sage: A3.gens()
(z0, z1, z2)
```

A1 (*names='z'*, *base_ring=Rational Field*)

Construct the affine line $\mathbb{A}^1$ as a toric variety.

INPUT:

- names – string. Names for the homogeneous coordinates. See normalize_names() for acceptable formats.
- base_ring – a ring (default: **Q**). The base ring for the toric variety.

OUTPUT:

A toric variety.

EXAMPLES:

```
sage: A1 = toric_varieties.A1()
sage: A1
1-d affine toric variety
sage: A1.fan().rays()
N(1)
in 1-d lattice N
```

```
sage: A1.gens()
(z,)
```

A2 (*names='x y'*, *base_ring=Rational Field*)

Construct the affine plane $\mathbb{A}^2$ as a toric variety.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `toric variety`.

EXAMPLES:

```
sage: A2 = toric_varieties.A2()
sage: A2
2-d affine toric variety
sage: A2.fan().rays()
N(1, 0),
N(0, 1)
in 2-d lattice N
sage: A2.gens()
(x, y)
```

A2_Z2 (*names='x y'*, *base_ring=Rational Field*)

Construct the orbifold $\mathbb{A}^2/\mathbf{Z}_2$ as a toric variety.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `toric variety`.

EXAMPLES:

```
sage: A2_Z2 = toric_varieties.A2_Z2()
sage: A2_Z2
2-d affine toric variety
sage: A2_Z2.fan().rays()
N(1, 0),
N(1, 2)
in 2-d lattice N
sage: A2_Z2.gens()
(x, y)
```

BCdlOG (*names='v1 v2 c1 c2 v4 v5 b e1 e2 e3 f g v6'*, *base_ring=Rational Field*)

Construct the 5-dimensional toric variety studied in [BCdlOG], [HLY]

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.

•base_ring – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A CPR-Fano toric variety.

EXAMPLES:

```
sage: X = toric_varieties.BCdlOG()
sage: X
5-d CPR-Fano toric variety covered by 54 affine patches
sage: X.fan().rays()
N(-1,  0,  0,  2,  3),
N( 0, -1,  0,  2,  3),
N( 0,  0, -1,  2,  3),
N( 0,  0, -1,  1,  2),
N( 0,  0,  0, -1,  0),
N( 0,  0,  0,  0, -1),
N( 0,  0,  0,  2,  3),
N( 0,  0,  1,  2,  3),
N( 0,  0,  2,  2,  3),
N( 0,  0,  1,  1,  1),
N( 0,  1,  2,  2,  3),
N( 0,  1,  3,  2,  3),
N( 1,  0,  4,  2,  3)
in 5-d lattice N
sage: X.gens()
(v1, v2, c1, c2, v4, v5, b, e1, e2, e3, f, g, v6)
```

REFERENCES:

BCdlOG_base (*names='d4 d3 r2 r1 d2 u d1'*, *base_ring=Rational Field*)

Construct the base of the $\mathbb{P}^2(1,2,3)$ fibration BCdlOG().

INPUT:

•names – string. Names for the homogeneous coordinates. See normalize_names() for acceptable formats.

•base_ring – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A toric variety.

EXAMPLES:

```
sage: base = toric_varieties.BCdlOG_base()
sage: base
3-d toric variety covered by 10 affine patches
sage: base.fan().rays()
N(-1,  0,  0),
N( 0, -1,  0),
N( 0,  0, -1),
N( 0,  0,  1),
N( 0,  1,  2),
N( 0,  1,  3),
N( 1,  0,  4)
in 3-d lattice N
sage: base.gens()
(d4, d3, r2, r1, d2, u, d1)
```

Conifold (*names='u x y v'*, *base_ring=Rational Field*)

Construct the conifold as a toric variety.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `toric variety`.

EXAMPLES:

```
sage: Conifold = toric_varieties.Conifold()
sage: Conifold
3-d affine toric variety
sage: Conifold.fan().rays()
N(0, 0, 1),
N(0, 1, 1),
N(1, 0, 1),
N(1, 1, 1)
in 3-d lattice N
sage: Conifold.gens()
(u, x, y, v)
```

Cube_deformation (*k*, *names=None*, *base_ring=Rational Field*)

Construct, for each $k \in \mathbf{Z}_{\geq 0}$, a toric variety with $\mathbf{Z}_k$-torsion in the Chow group.

The fans of this sequence of toric varieties all equal the face fan of a unit cube topologically, but the `(1,1,1)`-vertex is moved to `(1,1,2k+1)`. This example was studied in [FS].

INPUT:

- `k` – integer. The case `k=0` is the same as `Cube_face_fan()`.
- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `toric variety` X_k. Its Chow group is $A_1(X_k) = \mathbf{Z}_k$.

EXAMPLES:

```
sage: X_2 = toric_varieties.Cube_deformation(2)
sage: X_2
3-d toric variety covered by 6 affine patches
sage: X_2.fan().rays()
N( 1,  1,  5),
N( 1, -1,  1),
N(-1,  1,  1),
N(-1, -1,  1),
N(-1, -1, -1),
N(-1,  1, -1),
N( 1, -1, -1),
N( 1,  1, -1)
in 3-d lattice N
sage: X_2.gens()
(z0, z1, z2, z3, z4, z5, z6, z7)
```

REFERENCES:

Cube_face_fan (*names='z+', base_ring=Rational Field*)

Construct the toric variety given by the face fan of the 3-dimensional unit lattice cube.

This variety has 6 conifold singularities but the fan is still polyhedral.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `CPR-Fano toric variety`.

EXAMPLES:

```
sage: Cube_face_fan = toric_varieties.Cube_face_fan()
sage: Cube_face_fan
3-d CPR-Fano toric variety covered by 6 affine patches
sage: Cube_face_fan.fan().rays()
N( 1,  1,  1),
N( 1, -1,  1),
N(-1,  1,  1),
N(-1, -1,  1),
N(-1, -1, -1),
N(-1,  1, -1),
N( 1, -1, -1),
N( 1,  1, -1)
in 3-d lattice N
sage: Cube_face_fan.gens()
(z0, z1, z2, z3, z4, z5, z6, z7)
```

Cube_nonpolyhedral (*names='z+', base_ring=Rational Field*)

Construct the toric variety defined by a fan that is not the face fan of a polyhedron.

This toric variety is defined by a fan that is topologically like the face fan of a 3-dimensional cube, but with a different N-lattice structure.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `toric variety`.

NOTES:

- This is an example of an non-polyhedral fan.
- Its Chow group has torsion: $A_2(X) = \mathbf{Z}^5 \oplus \mathbf{Z}_2$

EXAMPLES:

```
sage: Cube_nonpolyhedral = toric_varieties.Cube_nonpolyhedral()
sage: Cube_nonpolyhedral
3-d toric variety covered by 6 affine patches
sage: Cube_nonpolyhedral.fan().rays()
N( 1,  2,  3),
```

```
N( 1, -1,  1),
N(-1,  1,  1),
N(-1, -1,  1),
N(-1, -1, -1),
N(-1,  1, -1),
N( 1, -1, -1),
N( 1,  1, -1)
in 3-d lattice N
sage: Cube_nonpolyhedral.gens()
(z0, z1, z2, z3, z4, z5, z6, z7)
```

Cube_sublattice (*names='z+'*, *base_ring=Rational Field*)

Construct the toric variety defined by a face fan over a 3-dimensional cube, but not the unit cube in the N-lattice. See [FultonP65].

Its Chow group is $A_2(X) = \mathbb{Z}^5$, which distinguishes it from the face fan of the unit cube.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `CPR-Fano toric variety`.

EXAMPLES:

```
sage: Cube_sublattice = toric_varieties.Cube_sublattice()
sage: Cube_sublattice
3-d CPR-Fano toric variety covered by 6 affine patches
sage: Cube_sublattice.fan().rays()
N( 1,  0,  0),
N( 0,  1,  0),
N( 0,  0,  1),
N(-1,  1,  1),
N(-1,  0,  0),
N( 0, -1,  0),
N( 0,  0, -1),
N( 1, -1, -1)
in 3-d lattice N
sage: Cube_sublattice.gens()
(z0, z1, z2, z3, z4, z5, z6, z7)
```

REFERENCES:

P (*n*, *names='z+'*, *base_ring=Rational Field*)

Construct the `n`-dimensional projective space $\mathbb{P}^n$.

INPUT:

- `n` – positive integer. The dimension of the projective space.
- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `CPR-Fano toric variety`.

EXAMPLES:

```
sage: P3 = toric_varieties.P(3)
sage: P3
3-d CPR-Fano toric variety covered by 4 affine patches
sage: P3.fan().rays()
N( 1,  0,  0),
N( 0,  1,  0),
N( 0,  0,  1),
N(-1, -1, -1)
in 3-d lattice N
sage: P3.gens()
(z0, z1, z2, z3)
```

P1 (*names='s t'*, *base_ring=Rational Field*)

Construct the projective line $\mathbb{P}^1$ as a toric variety.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `CPR-Fano toric variety`.

EXAMPLES:

```
sage: P1 = toric_varieties.P1()
sage: P1
1-d CPR-Fano toric variety covered by 2 affine patches
sage: P1.fan().rays()
N( 1),
N(-1)
in 1-d lattice N
sage: P1.gens()
(s, t)
```

P1xA1 (*names='s t z'*, *base_ring=Rational Field*)

Construct the cartesian product $\mathbb{P}^1 \times \mathbb{A}^1$ as a toric variety.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `toric variety`.

EXAMPLES:

```
sage: P1xA1 = toric_varieties.P1xA1()
sage: P1xA1
2-d toric variety covered by 2 affine patches
sage: P1xA1.fan().rays()
N( 1, 0),
N(-1, 0),
N( 0, 1)
in 2-d lattice N
```

```
sage: P1xA1.gens()
(s, t, z)
```

P1xP1 (*names='s t x y'*, *base_ring=Rational Field*)
Construct the del Pezzo surface $\mathbb{P}^1 \times \mathbb{P}^1$ as a toric variety.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `CPR-Fano toric variety`.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1
2-d CPR-Fano toric variety covered by 4 affine patches
sage: P1xP1.fan().rays()
N( 1,  0),
N(-1,  0),
N( 0,  1),
N( 0, -1)
in 2-d lattice N
sage: P1xP1.gens()
(s, t, x, y)
```

P1xP1_Z2 (*names='s t x y'*, *base_ring=Rational Field*)
Construct the toric $\mathbb{Z}_2$-orbifold of the del Pezzo surface $\mathbb{P}^1 \times \mathbb{P}^1$ as a toric variety.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `CPR-Fano toric variety`.

EXAMPLES:

```
sage: P1xP1_Z2 = toric_varieties.P1xP1_Z2()
sage: P1xP1_Z2
2-d CPR-Fano toric variety covered by 4 affine patches
sage: P1xP1_Z2.fan().rays()
N( 1,  1),
N(-1, -1),
N(-1,  1),
N( 1, -1)
in 2-d lattice N
sage: P1xP1_Z2.gens()
(s, t, x, y)
sage: P1xP1_Z2.Chow_group().degree(1)
C2 x Z^2
```

P2 (*names='x y z'*, *base_ring=Rational Field*)
Construct the projective plane $\mathbb{P}^2$ as a toric variety.

INPUT:

- names – string. Names for the homogeneous coordinates. See normalize_names() for acceptable formats.
- base_ring – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A CPR-Fano toric variety.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2
2-d CPR-Fano toric variety covered by 3 affine patches
sage: P2.fan().rays()
N( 1,  0),
N( 0,  1),
N(-1, -1)
in 2-d lattice N
sage: P2.gens()
(x, y, z)
```

P2_112 (*names='z+'*, *base_ring=Rational Field*)

Construct the weighted projective space $\mathbb{P}^2(1,1,2)$.

INPUT:

- names – string. Names for the homogeneous coordinates. See normalize_names() for acceptable formats.
- base_ring – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A CPR-Fano toric variety.

EXAMPLES:

```
sage: P2_112 = toric_varieties.P2_112()
sage: P2_112
2-d CPR-Fano toric variety covered by 3 affine patches
sage: P2_112.fan().rays()
N( 1,  0),
N( 0,  1),
N(-1, -2)
in 2-d lattice N
sage: P2_112.gens()
(z0, z1, z2)
```

P2_123 (*names='z+'*, *base_ring=Rational Field*)

Construct the weighted projective space $\mathbb{P}^2(1,2,3)$.

INPUT:

- names – string. Names for the homogeneous coordinates. See normalize_names() for acceptable formats.
- base_ring – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A CPR-Fano toric variety.

EXAMPLES:

```
sage: P2_123 = toric_varieties.P2_123()
sage: P2_123
2-d CPR-Fano toric variety covered by 3 affine patches
sage: P2_123.fan().rays()
N( 1,  0),
N( 0,  1),
N(-2, -3)
in 2-d lattice N
sage: P2_123.gens()
(z0, z1, z2)
```

P4_11133(*names='z+'*, *base_ring=Rational Field*)

Construct the weighted projective space $\mathbb{P}^4(1,1,1,3,3)$.

INPUT:

- names – string. Names for the homogeneous coordinates. See normalize_names() for acceptable formats.
- base_ring – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A CPR-Fano toric variety.

EXAMPLES:

```
sage: P4_11133 = toric_varieties.P4_11133()
sage: P4_11133
4-d CPR-Fano toric variety covered by 5 affine patches
sage: P4_11133.fan().rays()
N( 1,  0,  0,  0),
N( 0,  1,  0,  0),
N( 0,  0,  1,  0),
N( 0,  0,  0,  1),
N(-3, -3, -1, -1)
in 4-d lattice N
sage: P4_11133.gens()
(z0, z1, z2, z3, z4)
```

P4_11133_resolved(*names='z+'*, *base_ring=Rational Field*)

Construct the weighted projective space $\mathbb{P}^4(1,1,1,3,3)$.

INPUT:

- names – string. Names for the homogeneous coordinates. See normalize_names() for acceptable formats.
- base_ring – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A CPR-Fano toric variety.

EXAMPLES:

```
sage: P4_11133_resolved = toric_varieties.P4_11133_resolved()
sage: P4_11133_resolved
4-d CPR-Fano toric variety covered by 9 affine patches
sage: P4_11133_resolved.fan().rays()
N( 1,  0,  0,  0),
N( 0,  1,  0,  0),
```

```
N( 0,  0,  1,  0),
N( 0,  0,  0,  1),
N(-3, -3, -1, -1),
N(-1, -1,  0,  0)
in 4-d lattice N
sage: P4_11133_resolved.gens()
(z0, z1, z2, z3, z4, z5)
```

P4_11169 (*names='z+'*, *base_ring=Rational Field*)

Construct the weighted projective space $\mathbb{P}^4(1,1,1,6,9)$.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `CPR-Fano toric variety`.

EXAMPLES:

```
sage: P4_11169 = toric_varieties.P4_11169()
sage: P4_11169
4-d CPR-Fano toric variety covered by 5 affine patches
sage: P4_11169.fan().rays()
N( 1,  0,  0,  0),
N( 0,  1,  0,  0),
N( 0,  0,  1,  0),
N( 0,  0,  0,  1),
N(-9, -6, -1, -1)
in 4-d lattice N
sage: P4_11169.gens()
(z0, z1, z2, z3, z4)
```

P4_11169_resolved (*names='z+'*, *base_ring=Rational Field*)

Construct the blow-up of the weighted projective space $\mathbb{P}^4(1,1,1,6,9)$ at its curve of $\mathbf{Z}_3$ orbifold fixed points.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `CPR-Fano toric variety`.

EXAMPLES:

```
sage: P4_11169_resolved = toric_varieties.P4_11169_resolved()
sage: P4_11169_resolved
4-d CPR-Fano toric variety covered by 9 affine patches
sage: P4_11169_resolved.fan().rays()
N( 1,  0,  0,  0),
N( 0,  1,  0,  0),
N( 0,  0,  1,  0),
N( 0,  0,  0,  1),
N(-9, -6, -1, -1),
```

```
N(-3, -2,  0,  0)
in 4-d lattice N
sage: P4_11169_resolved.gens()
(z0, z1, z2, z3, z4, z5)
```

WP (**q*, ***kw*)

Construct weighted projective n-space over a field.

INPUT:

- `q` – a sequence of positive integers relatively prime to one another. The weights `q` can be given either as a list or tuple, or as positional arguments.

Two keyword arguments:

- `base_ring` – a field (default: $\mathbf{Q}$).
- `names` – string or list (tuple) of strings (default 'z+'). See `normalize_names()` for acceptable formats.

OUTPUT:

- A `toric variety`. If $q = (q_0, \ldots, q_n)$, then the output is the weighted projective space $\mathbb{P}(q_0, \ldots, q_n)$ over `base_ring`. `names` are the names of the generators of the homogeneous coordinate ring.

EXAMPLES:

A hyperelliptic curve C of genus 2 as a subscheme of the weighted projective plane $\mathbb{P}(1, 3, 1)$:

```
sage: X = toric_varieties.WP([1,3,1], names='x y z')
sage: X.inject_variables()
Defining x, y, z
sage: g = y^2-(x^6-z^6)
sage: C = X.subscheme([g]); C
Closed subscheme of 2-d toric variety covered by 3 affine patches defined by:
  -x^6 + z^6 + y^2
```

dP6 (*names='x u y v z w'*, *base_ring=Rational Field*)

Construct the del Pezzo surface of degree 6 ($\mathbb{P}^2$ blown up at 3 points) as a toric variety.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `CPR-Fano toric variety`.

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: dP6
2-d CPR-Fano toric variety covered by 6 affine patches
sage: dP6.fan().rays()
N( 0,  1),
N(-1,  0),
N(-1, -1),
N( 0, -1),
N( 1,  0),
N( 1,  1)
```

```
in 2-d lattice N
sage: dP6.gens()
(x, u, y, v, z, w)
```

dP6xdP6 (*names='x0 x1 x2 x3 x4 x5 y0 y1 y2 y3 y4 y5'*, *base_ring=Rational Field*)

Construct the product of two del Pezzo surfaces of degree 6 ($\mathbb{P}^2$ blown up at 3 points) as a toric variety.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `CPR-Fano toric variety`.

EXAMPLES:

```
sage: dP6xdP6 = toric_varieties.dP6xdP6()
sage: dP6xdP6
4-d CPR-Fano toric variety covered by 36 affine patches
sage: dP6xdP6.fan().rays()
N( 0,  1,  0,  0),
N(-1,  0,  0,  0),
N(-1, -1,  0,  0),
N( 0, -1,  0,  0),
N( 1,  0,  0,  0),
N( 1,  1,  0,  0),
N( 0,  0,  0,  1),
N( 0,  0, -1,  0),
N( 0,  0, -1, -1),
N( 0,  0,  0, -1),
N( 0,  0,  1,  0),
N( 0,  0,  1,  1)
in 4-d lattice N
sage: dP6xdP6.gens()
(x0, x1, x2, x3, x4, x5, y0, y1, y2, y3, y4, y5)
```

dP7 (*names='x u y v z'*, *base_ring=Rational Field*)

Construct the del Pezzo surface of degree 7 ($\mathbb{P}^2$ blown up at 2 points) as a toric variety.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `CPR-Fano toric variety`.

EXAMPLES:

```
sage: dP7 = toric_varieties.dP7()
sage: dP7
2-d CPR-Fano toric variety covered by 5 affine patches
sage: dP7.fan().rays()
N( 0,  1),
N(-1,  0),
N(-1, -1),
```

```
N( 0, -1),
N( 1,  0)
in 2-d lattice N
sage: dP7.gens()
(x, u, y, v, z)
```

dP8 (*names='t x y z'*, *base_ring=Rational Field*)

Construct the del Pezzo surface of degree 8 ($\mathbb{P}^2$ blown up at 1 point) as a toric variety.

INPUT:

- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `CPR-Fano toric variety`.

EXAMPLES:

```
sage: dP8 = toric_varieties.dP8()
sage: dP8
2-d CPR-Fano toric variety covered by 4 affine patches
sage: dP8.fan().rays()
N( 1,  1),
N( 0,  1),
N(-1, -1),
N( 1,  0)
in 2-d lattice N
sage: dP8.gens()
(t, x, y, z)
```

torus (*n*, *names='z+'*, *base_ring=Rational Field*)

Construct the `n`-dimensional algebraic torus $(\mathbb{F}^{\times})^n$.

INPUT:

- `n` – non-negative integer. The dimension of the algebraic torus.
- `names` – string. Names for the homogeneous coordinates. See `normalize_names()` for acceptable formats.
- `base_ring` – a ring (default: $\mathbf{Q}$). The base ring for the toric variety.

OUTPUT:

A `toric variety`.

EXAMPLES:

```
sage: T3 = toric_varieties.torus(3);  T3
3-d affine toric variety
sage: T3.fan().rays()
Empty collection
in 3-d lattice N
sage: T3.fan().virtual_rays()
N(1, 0, 0),
N(0, 1, 0),
N(0, 0, 1)
in 3-d lattice N
sage: T3.gens()
```

```
(z0, z1, z2)
sage: sorted(T3.change_ring(GF(3)).point_set().list())
[[1 : 1 : 1], [1 : 1 : 2], [1 : 2 : 1], [1 : 2 : 2],
 [2 : 1 : 1], [2 : 1 : 2], [2 : 2 : 1], [2 : 2 : 2]]
```

16.4 Toric divisors and divisor classes

Let X be a `toric variety` corresponding to a `rational polyhedral fan` Σ. A `toric divisor` D is a T-Weil divisor over a given coefficient ring (usually $\mathbf{Z}$ or $\mathbf{Q}$), i.e. a formal linear combination of torus-invariant subvarieties of X of codimension one. In homogeneous coordinates $[z_0 : \cdots : z_k]$, these are the subvarieties $\{z_i = 0\}$. Note that there is a finite number of such subvarieties, one for each ray of Σ. We generally identify

- Toric divisor D,
- Sheaf $\mathcal{O}(D)$ (if D is Cartier, it is a line bundle),
- Support function ϕ_D (if D is $\mathbf{Q}$-Cartier, it is a function linear on each cone of Σ).

EXAMPLES:

We start with an illustration of basic divisor arithmetic:

```
sage: dP6 = toric_varieties.dP6()
sage: Dx,Du,Dy,Dv,Dz,Dw = dP6.toric_divisor_group().gens()
sage: Dx
V(x)
sage: -Dx
-V(x)
sage: 2*Dx
2*V(x)
sage: Dx*2
2*V(x)
sage: (1/2)*Dx + Dy/3 - Dz
1/2*V(x) + 1/3*V(y) - V(z)
sage: Dx.parent()
Group of toric ZZ-Weil divisors
on 2-d CPR-Fano toric variety covered by 6 affine patches
sage: (Dx/2).parent()
Group of toric QQ-Weil divisors
on 2-d CPR-Fano toric variety covered by 6 affine patches
```

Now we create a more complicated variety to demonstrate divisors of different types:

```
sage: F = Fan(cones=[(0,1,2,3), (0,1,4)],
...       rays=[(1,1,1), (1,-1,1), (1,-1,-1), (1,1,-1), (0,0,1)])
sage: X = ToricVariety(F)
sage: QQ_Cartier = X.divisor([2,2,1,1,1])
sage: Cartier = 2 * QQ_Cartier
sage: Weil = X.divisor([1,1,1,0,0])
sage: QQ_Weil = 1/2 * Weil
sage: [QQ_Weil.is_QQ_Weil(),
...    QQ_Weil.is_Weil(),
...    QQ_Weil.is_QQ_Cartier(),
...    QQ_Weil.is_Cartier()]
[True, False, False, False]
sage: [Weil.is_QQ_Weil(),
...    Weil.is_Weil(),
```

```
...     Weil.is_QQ_Cartier(),
...     Weil.is_Cartier()]
[True, True, False, False]
sage: [QQ_Cartier.is_QQ_Weil(),
...     QQ_Cartier.is_Weil(),
...     QQ_Cartier.is_QQ_Cartier(),
...     QQ_Cartier.is_Cartier()]
[True, True, True, False]
sage: [Cartier.is_QQ_Weil(),
...     Cartier.is_Weil(),
...     Cartier.is_QQ_Cartier(),
...     Cartier.is_Cartier()]
[True, True, True, True]
```

The toric ($\mathbf{Q}$-Weil) divisors on a toric variety X modulo linear equivalence generate the divisor **class group** $\mathrm{Cl}(X)$, implemented by `ToricRationalDivisorClassGroup`. If X is smooth, this equals the **Picard group** $\mathrm{Pic}(X)$. We continue using del Pezzo surface of degree 6 introduced above:

```
sage: Cl = dP6.rational_class_group(); Cl
The toric rational divisor class group
of a 2-d CPR-Fano toric variety covered by 6 affine patches
sage: Cl.ngens()
4
sage: c0,c1,c2,c3 = Cl.gens()
sage: c = c0 + 2*c1 - c3; c
Divisor class [1, 2, 0, -1]
```

Divisors are mapped to their classes and lifted via:

```
sage: Dx.divisor_class()
Divisor class [1, 0, 0, 0]
sage: Dx.divisor_class() in Cl
True
sage: (-Dw+Dv+Dy).divisor_class()
Divisor class [1, 0, 0, 0]
sage: c0
Divisor class [1, 0, 0, 0]
sage: c0.lift()
V(x)
```

The (rational) divisor class group is where the Kaehler cone lives:

```
sage: Kc = dP6.Kaehler_cone(); Kc
4-d cone in 4-d lattice
sage: Kc.rays()
Divisor class [0, 1, 1, 0],
Divisor class [0, 0, 1, 1],
Divisor class [1, 1, 0, 0],
Divisor class [1, 1, 1, 0],
Divisor class [0, 1, 1, 1]
in Basis lattice of The toric rational divisor class group
of a 2-d CPR-Fano toric variety covered by 6 affine patches
sage: Kc.ray(1).lift()
V(y) + V(v)
```

Given a divisor D, we have an associated line bundle (or a reflexive sheaf, if D is not Cartier) $\mathcal{O}(D)$. Its sections are:

```
sage: P2 = toric_varieties.P2()
sage: H = P2.divisor(0); H
V(x)
sage: H.sections()
(M(-1, 0), M(-1, 1), M(0, 0))
sage: H.sections_monomials()
(z, y, x)
```

Note that the space of sections is always spanned by monomials. Therefore, we can grade the sections (as homogeneous monomials) by their weight under rescaling individual coordinates. This weight data amounts to a point of the dual lattice.

In the same way, we can grade cohomology groups by their cohomological degree and a weight:

```
sage: M = P2.fan().lattice().dual()
sage: H.cohomology(deg=0, weight=M(-1,0))
Vector space of dimension 1 over Rational Field
sage: _.dimension()
1
```

Here is a more complicated example with $h^1(dP_6, \mathcal{O}(D)) = 4$

```
sage: D = dP6.divisor([0, 0, -1, 0, 2, -1])
sage: D.cohomology()
{0: Vector space of dimension 0 over Rational Field,
 1: Vector space of dimension 4 over Rational Field,
 2: Vector space of dimension 0 over Rational Field}
sage: D.cohomology(dim=True)
(0, 4, 0)
```

AUTHORS:

- Volker Braun, Andrey Novoseltsev (2010-09-07): initial version.

sage.schemes.toric.divisor.**ToricDivisor**(*toric_variety*, *arg=None*, *ring=None*, *check=True*, *reduce=True*)

Construct a divisor of `toric_variety`.

INPUT:

- `toric_variety` – a `toric variety`;
- `arg` – one of the following description of the toric divisor to be constructed:
  - `None` or 0 (the trivial divisor);
  - monomial in the homogeneous coordinates;
  - one-dimensional cone of the fan of `toric_variety` or a lattice point generating such a cone;
  - sequence of rational numbers, specifying multiplicities for each of the toric divisors.
- `ring` – usually either **Z** or **Q**. The base ring of the divisor group. If `ring` is not specified, a coefficient ring suitable for `arg` is derived.
- `check` – bool (default: True). Whether to coerce coefficients into base ring. Setting it to `False` can speed up construction.
- `reduce` – reduce (default: True). Whether to combine common terms. Setting it to `False` can speed up construction.

Warning: The coefficients of the divisor must be in the base ring and the terms must be reduced. If you set `check=False` and/or `reduce=False` it is your responsibility to pass valid input data `arg`.

OUTPUT:

- A `sage.schemes.toric.divisor.ToricDivisor_generic`

EXAMPLES:

```
sage: from sage.schemes.toric.divisor import ToricDivisor
sage: dP6 = toric_varieties.dP6()
sage: ToricDivisor(dP6, [(1,dP6.gen(2)), (1,dP6.gen(1))])
V(u) + V(y)
sage: ToricDivisor(dP6, (0,1,1,0,0,0), ring=QQ)
V(u) + V(y)
sage: dP6.inject_variables()
Defining x, u, y, v, z, w
sage: ToricDivisor(dP6, u+y)
Traceback (most recent call last):
...
ValueError: u + y is not a monomial!
sage: ToricDivisor(dP6, u*y)
V(u) + V(y)
sage: ToricDivisor(dP6, dP6.fan(dim=1)[2] )
V(y)
sage: cone = Cone(dP6.fan(dim=1)[2])
sage: ToricDivisor(dP6, cone)
V(y)
sage: N = dP6.fan().lattice()
sage: ToricDivisor(dP6, N(1,1) )
V(w)
```

We attempt to guess the correct base ring:

```
sage: ToricDivisor(dP6, [(1/2,u)])
1/2*V(u)
sage: _.parent()
Group of toric QQ-Weil divisors on
2-d CPR-Fano toric variety covered by 6 affine patches
sage: ToricDivisor(dP6, [(1/2,u), (1/2,u)])
V(u)
sage: _.parent()
Group of toric ZZ-Weil divisors on
2-d CPR-Fano toric variety covered by 6 affine patches
sage: ToricDivisor(dP6, [(u,u)])
Traceback (most recent call last):
...
TypeError: Cannot deduce coefficient ring for [(u, u)]!
```

class `sage.schemes.toric.divisor.`**`ToricDivisorGroup`**(*toric_variety*, *base_ring*)

Bases: `sage.schemes.generic.divisor_group.DivisorGroup_generic`

The group of (**Q**-T-Weil) divisors on a toric variety.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2.toric_divisor_group()
Group of toric ZZ-Weil divisors
on 2-d CPR-Fano toric variety covered by 3 affine patches
```

class **Element** (*v*, *parent*, *check=True*, *reduce=True*)

Bases: `sage.schemes.generic.divisor.Divisor_generic`

Construct a `Divisor_generic`.

INPUT:

INPUT:

- `v` – object. Usually a list of pairs `(coefficient,divisor)`.
- `parent` – FormalSums(R) module (default: FormalSums(ZZ))
- `check` – bool (default: True). Whether to coerce coefficients into base ring. Setting it to `False` can speed up construction.
- `reduce` – reduce (default: True). Whether to combine common terms. Setting it to `False` can speed up construction.

Warning: The coefficients of the divisor must be in the base ring and the terms must be reduced. If you set `check=False` and/or `reduce=False` it is your responsibility to pass a valid object `v`.

EXAMPLES:

```
sage: from sage.schemes.generic.divisor import Divisor_generic
sage: from sage.schemes.generic.divisor_group import DivisorGroup
sage: Divisor_generic( [(4,5)], DivisorGroup(Spec(ZZ)), False, False)
4*V(5)
```

`ToricDivisorGroup.`**base_extend**(*R*)

Extend the scalars of `self` to `R`.

INPUT:

- `R` – ring.

OUTPUT:

- toric divisor group.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: DivZZ = P2.toric_divisor_group()
sage: DivQQ = P2.toric_divisor_group(base_ring=QQ)
sage: DivZZ.base_extend(QQ) is DivQQ
True
```

`ToricDivisorGroup.`**gen**(*i*)

Return the `i`-th generator of the divisor group.

INPUT:

- `i` – integer.

OUTPUT:

The divisor $z_i = 0$, where z_i is the i-th homogeneous coordinate.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: TDiv = P2.toric_divisor_group()
sage: TDiv.gen(2)
V(z)
```

ToricDivisorGroup.**gens**()

Return the generators of the divisor group.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: TDiv = P2.toric_divisor_group()
sage: TDiv.gens()
(V(x), V(y), V(z))
```

ToricDivisorGroup.**ngens**()

Return the number of generators.

OUTPUT:

The number of generators of `self`, which equals the number of rays in the fan of the toric variety.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: TDiv = P2.toric_divisor_group()
sage: TDiv.ngens()
3
```

class sage.schemes.toric.divisor.**ToricDivisor_generic**(*v*, *parent*, *check=True*, *reduce=True*)

Bases: `sage.schemes.generic.divisor.Divisor_generic`

Construct a `(toric Weil) divisor` on the given toric variety.

INPUT:

- `v` – a list of tuples (multiplicity, coordinate).
- `parent` – `ToricDivisorGroup`. The parent divisor group.
- `check` – boolean. Type-check the entries of `v`, see `sage.schemes.generic.divisor_group.DivisorGroup_g`
- `reduce` – boolean. Combine coefficients in `v`, see `sage.schemes.generic.divisor_group.DivisorGroup_g`

> **Warning:** Do not construct `ToricDivisor_generic` objects manually. Instead, use either the function `ToricDivisor()` or the method `divisor()` of toric varieties.

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: ray = dP6.fan().ray(0)
sage: ray
N(0, 1)
sage: D = dP6.divisor(ray); D
V(x)
sage: D.parent()
Group of toric ZZ-Weil divisors
on 2-d CPR-Fano toric variety covered by 6 affine patches
```

Chern_character()

Return the Chern character of the sheaf $\mathcal{O}(D)$ defined by the divisor D.

You can also use a shortcut `ch()`.

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: N = dP6.fan().lattice()
```

```
sage: D3 = dP6.divisor(dP6.fan().cone_containing( N(0,1)   ))
sage: D5 = dP6.divisor(dP6.fan().cone_containing( N(-1,-1) ))
sage: D6 = dP6.divisor(dP6.fan().cone_containing( N(0,-1)  ))
sage: D = -D3 + 2*D5 - D6
sage: D.Chern_character()
[5*w^2 + y - 2*v + w + 1]
sage: dP6.integrate( D.ch() * dP6.Td() )
-4
```

Chow_cycle(*ring=Integer Ring*)

Returns the Chow homology class of the divisor.

INPUT:

- `ring` – Either `ZZ` (default) or `QQ`. The base ring of the Chow group.

OUTPUT:

The `ChowCycle` represented by the divisor.

EXAMPLES:

sage: dP6 = toric_varieties.dP6() sage: cone = dP6.fan(1)[0] sage: D = dP6.divisor(cone); D V(x) sage: D.Chow_cycle() (0 | -1, 0, 1, 1 | 0) sage: dP6.Chow_group()(cone) (0 | -1, 0, 1, 1 | 0)

Kodaira_map(*names='z'*)

Return the Kodaira map.

The Kodaira map is the rational map $X_\Sigma \to \mathbb{P}^{n-1}$, where n equals the number of sections. It is defined by the monomial sections of the line bundle.

If the divisor is ample and the toric variety smooth or of dimension 2, then this is an embedding.

INPUT:

- `names` – string (optional; default `'z'`). The variable names for the destination projective space.

EXAMPLES:

```
sage: P1.<u,v> = toric_varieties.P1()
sage: D = -P1.K()
sage: D.Kodaira_map()
Scheme morphism:
  From: 1-d CPR-Fano toric variety covered by 2 affine patches
  To:   Closed subscheme of Projective Space of dimension 2
        over Rational Field defined by:
  -z1^2 + z0*z2
  Defn: Defined on coordinates by sending [u : v] to
        (v^2 : u*v : u^2)

sage: dP6 = toric_varieties.dP6()
sage: D = -dP6.K()
sage: D.Kodaira_map(names='x')
Scheme morphism:
  From: 2-d CPR-Fano toric variety covered by 6 affine patches
  To:   Closed subscheme of Projective Space of dimension 6
        over Rational Field defined by:
  -x1*x5 + x0*x6,
  -x2*x3 + x0*x5,
  -x1*x3 + x0*x4,
  x4*x5 - x3*x6,
```

```
-x1*x2 + x0*x3,
x3*x5 - x2*x6,
x3*x4 - x1*x6,
x3^2 - x1*x5,
x2*x4 - x1*x5,
-x1*x5^2 + x2*x3*x6,
-x1*x5^3 + x2^2*x6^2
Defn: Defined on coordinates by sending [x : u : y : v : z : w] to
      (x*u^2*y^2*v : x^2*u^2*y*w : u*y^2*v^2*z : x*u*y*v*z*w :
       x^2*u*z*w^2 : y*v^2*z^2*w : x*v*z^2*w^2)
```

ch()

Return the Chern character of the sheaf $\mathcal{O}(D)$ defined by the divisor D.

You can also use a shortcut `ch()`.

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: N = dP6.fan().lattice()
sage: D3 = dP6.divisor(dP6.fan().cone_containing( N(0,1)   ))
sage: D5 = dP6.divisor(dP6.fan().cone_containing( N(-1,-1) ))
sage: D6 = dP6.divisor(dP6.fan().cone_containing( N(0,-1)  ))
sage: D = -D3 + 2*D5 - D6
sage: D.Chern_character()
[5*w^2 + y - 2*v + w + 1]
sage: dP6.integrate( D.ch() * dP6.Td() )
-4
```

coefficient(*x*)

Return the coefficient of `x`.

INPUT:

- `x` – one of the homogeneous coordinates, either given by the variable or its index.

OUTPUT:

The coefficient of `x`.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: D = P2.divisor((11,12,13)); D
11*V(x) + 12*V(y) + 13*V(z)
sage: D.coefficient(1)
12
sage: P2.inject_variables()
Defining x, y, z
sage: D.coefficient(y)
12
```

cohomology(*weight=None*, *deg=None*, *dim=False*)

Return the cohomology of the line bundle associated to the Cartier divisor or reflexive sheaf associated to the Weil divisor.

Note: The cohomology of a toric line bundle/reflexive sheaf is graded by the usual degree as well as by the M-lattice.

INPUT:

- `weight` – (optional) a point of the M-lattice.
- `deg` – (optional) the degree of the cohomology group.
- `dim` – boolean. If `False` (default), the cohomology groups are returned as vector spaces. If `True`, only the dimension of the vector space(s) is returned.

OUTPUT:

The vector space $H^{\deg}(X, \mathcal{O}(D))$ (if `deg` is specified) or a dictionary `{degree:cohomology(degree)}` of all degrees between 0 and the dimension of the variety.

If `weight` is specified, return only the subspace $H^{\deg}(X, \mathcal{O}(D))_{\text{weight}}$ of the cohomology of the given weight.

If `dim==True`, the dimension of the cohomology vector space is returned instead of actual vector space. Moreover, if `deg` was not specified, a vector whose entries are the dimensions is returned instead of a dictionary.

ALGORITHM:

Roughly, Chech cohomology is used to compute the cohomology. For toric divisors, the local sections can be chosen to be monomials (instead of general homogeneous polynomials), this is the reason for the extra grading by $m \in M$. General refrences would be [Fulton], [CLS]. Here are some salient features of our implementation:

- First, a finite set of M-lattice points is identified that supports the cohomology. The toric divisor determines a (polyhedral) chamber decomposition of $M_{\mathbf{R}}$, see Section 9.1 and Figure 4 of [CLS]. The cohomology vanishes on the non-compact chambers. Hence, the convex hull of the vertices of the chamber decomposition contains all non-vanishing cohomology groups. This is returned by the private method `_sheaf_cohomology_support()`.

  It would be more efficient, but more difficult to implement, to keep track of all of the individual chambers. We leave this for future work.

- For each point $m \in M$, the weight-m part of the cohomology can be rewritten as the cohomology of a simplicial complex, see Exercise 9.1.10 of [CLS], [Perling]. This is returned by the private method `_sheaf_complex()`.

  The simplicial complex is the same for all points in a chamber, but we currently do not make use of this and compute each point $m \in M$ separately.

- Finally, the cohomology (over $\mathbf{Q}$) of this simplicial complex is computed in the private method `_sheaf_cohomology()`. Summing over the supporting points $m \in M$ yields the cohomology of the sheaf‘.

REFERENCES:

EXAMPLES:

Example 9.1.7 of Cox, Little, Schenck: “Toric Varieties” [CLS]:

```
sage: F = Fan(cones=[(0,1), (1,2), (2,3), (3,4), (4,5), (5,0)],
...           rays=[(1,0), (1,1), (0,1), (-1,0), (-1,-1), (0,-1)])
sage: dP6 = ToricVariety(F)
sage: D3 = dP6.divisor(2)
sage: D5 = dP6.divisor(4)
sage: D6 = dP6.divisor(5)
sage: D = -D3 + 2*D5 - D6
sage: D.cohomology()
{0: Vector space of dimension 0 over Rational Field,
 1: Vector space of dimension 4 over Rational Field,
 2: Vector space of dimension 0 over Rational Field}
sage: D.cohomology(deg=1)
```

```
Vector space of dimension 4 over Rational Field
sage: M = F.dual_lattice()
sage: D.cohomology( M(0,0) )
{0: Vector space of dimension 0 over Rational Field,
 1: Vector space of dimension 1 over Rational Field,
 2: Vector space of dimension 0 over Rational Field}
sage: D.cohomology( weight=M(0,0), deg=1 )
Vector space of dimension 1 over Rational Field
sage: dP6.integrate( D.ch() * dP6.Td() )
-4
```

Note the different output options:

```
sage: D.cohomology()
{0: Vector space of dimension 0 over Rational Field,
 1: Vector space of dimension 4 over Rational Field,
 2: Vector space of dimension 0 over Rational Field}
sage: D.cohomology(dim=True)
(0, 4, 0)
sage: D.cohomology(weight=M(0,0))
{0: Vector space of dimension 0 over Rational Field,
 1: Vector space of dimension 1 over Rational Field,
 2: Vector space of dimension 0 over Rational Field}
sage: D.cohomology(weight=M(0,0), dim=True)
(0, 1, 0)
sage: D.cohomology(deg=1)
Vector space of dimension 4 over Rational Field
sage: D.cohomology(deg=1, dim=True)
4
sage: D.cohomology(weight=M(0,0), deg=1)
Vector space of dimension 1 over Rational Field
sage: D.cohomology(weight=M(0,0), deg=1, dim=True)
1
```

Here is a Weil (non-Cartier) divisor example:

```
sage: K = toric_varieties.Cube_nonpolyhedral().K()
sage: K.is_Weil()
True
sage: K.is_QQ_Cartier()
False
sage: K.cohomology(dim=True)
(0, 0, 0, 1)
```

cohomology_class()

Return the degree-2 cohomology class associated to the divisor.

OUTPUT:

Returns the corresponding cohomology class as an instance of `CohomologyClass`. The cohomology class is the first Chern class of the associated line bundle $\mathcal{O}(D)$.

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: D = dP6.divisor(dP6.fan().ray(0) )
sage: D.cohomology_class()
[y + v - w]
```

cohomology_support()

Return the weights for which the cohomology groups do not vanish.

OUTPUT:

A tuple of dual lattice points. `self.cohomology(weight=m)` does not vanish if and only if `m` is in the output.

Note: This method is provided for educational purposes and it is not an efficient way of computing the cohomology groups.

EXAMPLES:

```
sage: F = Fan(cones=[(0,1), (1,2), (2,3), (3,4), (4,5), (5,0)],
...           rays=[(1,0), (1,1), (0,1), (-1,0), (-1,-1), (0,-1)])
sage: dP6 = ToricVariety(F)
sage: D3 = dP6.divisor(2)
sage: D5 = dP6.divisor(4)
sage: D6 = dP6.divisor(5)
sage: D = -D3 + 2*D5 - D6
sage: D.cohomology_support()
(M(0, 0), M(1, 0), M(2, 0), M(1, 1))
```

divisor_class()

Return the linear equivalence class of the divisor.

OUTPUT:

Returns the class of the divisor in $Cl(X) \otimes_{\mathbf{Z}} \mathbf{Q}$ as an instance of `ToricRationalDivisorClassGroup`.

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: D = dP6.divisor(0)
sage: D.divisor_class()
Divisor class [1, 0, 0, 0]
```

function_value(*point*)

Return the value of the support function at `point`.

Let X be the ambient toric variety of `self`, Σ the fan associated to X, and N the ambient lattice of Σ.

INPUT:

- `point` – either an integer, interpreted as the index of a ray of Σ, or a point of the lattice N.

OUTPUT:

- an interger or a rational number.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: D = P2.divisor([11,22,44])   # total degree 77
sage: D.function_value(0)
11
sage: N = P2.fan().lattice()
sage: D.function_value( N(1,1) )
33
sage: D.function_value( P2.fan().ray(0) )
11
```

is_Cartier()

Return whether the divisor is a Cartier-divisor.

Note: The sheaf $\mathcal{O}(D)$ associated to the given divisor D is a line bundle if and only if the divisor is Cartier.

EXAMPLES:

```
sage: X = toric_varieties.P4_11169()
sage: D = X.divisor(3)
sage: D.is_Cartier()
False
sage: D.is_QQ_Cartier()
True
```

is_QQ_Cartier()

Return whether the divisor is a $\mathbf{Q}$-Cartier divisor.

A $\mathbf{Q}$-Cartier divisor is a divisor such that some multiple of it is Cartier.

EXAMPLES:

```
sage: X = toric_varieties.P4_11169()
sage: D = X.divisor(3)
sage: D.is_QQ_Cartier()
True

sage: X = toric_varieties.Cube_face_fan()
sage: D = X.divisor(3)
sage: D.is_QQ_Cartier()
False
```

is_QQ_Weil()

Return whether the divisor is a $\mathbf{Q}$-Weil-divisor.

Note: This function returns always `True` since `ToricDivisor` can only describe $\mathbf{Q}$-Weil divisors.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: D = P2.divisor([1,2,3])
sage: D.is_QQ_Weil()
True
sage: (D/2).is_QQ_Weil()
True
```

is_Weil()

Return whether the divisor is a Weil-divisor.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: D = P2.divisor([1,2,3])
sage: D.is_Weil()
True
sage: (D/2).is_Weil()
False
```

is_ample()

Return whether a $\mathbf{Q}$-Cartier divisor is ample.

OUTPUT:

- True if the divisor is in the ample cone, False otherwise.

Note:

- For a QQ-Cartier divisor, some positive integral multiple is Cartier. We return wheher this associtated divisor is ample, i.e. corresponds to an ample line bundle.
- In the orbifold case, the ample cone is an open and full-dimensional cone in the rational divisor class group ToricRationalDivisorClassGroup.
- If the variety has worse than orbifold singularities, the ample cone is a full-dimensional cone within the (not full-dimensional) subspace spanned by the Cartier divisors inside the rational (Weil) divisor class group, that is, ToricRationalDivisorClassGroup. The ample cone is then relative open (open in this subspace).
- See also is_nef().
- A toric divisor is ample if and only if its support function is strictly convex.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: K = P2.K()
sage: (+K).is_ample()
False
sage: (0*K).is_ample()
False
sage: (-K).is_ample()
True
```

Example 6.1.3, 6.1.11, 6.1.17 of [CLS]:

```
sage: fan = Fan(cones=[(0,1), (1,2), (2,3), (3,0)],
...             rays=[(-1,2), (0,1), (1,0), (0,-1)])
sage: F2 = ToricVariety(fan,'u1, u2, u3, u4')
sage: def D(a,b): return a*F2.divisor(2) + b*F2.divisor(3)
...
sage: [ (a,b) for a,b in CartesianProduct(range(-3,3),range(-3,3))
...             if D(a,b).is_ample() ]
[(1, 1), (1, 2), (2, 1), (2, 2)]
sage: [ (a,b) for a,b in CartesianProduct(range(-3,3),range(-3,3))
...             if D(a,b).is_nef() ]
[(0, 0), (0, 1), (0, 2), (1, 0),
 (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]
```

A (worse than orbifold) singular Fano threefold:

```
sage: points = [(1,0,0),(0,1,0),(0,0,1),(-2,0,-1),(-2,-1,0),(-3,-1,-1),(1,1,1)]
sage: facets = [[0,1,3],[0,1,6],[0,2,4],[0,2,6],[0,3,5],[0,4,5],[1,2,3,4,5,6]]
sage: X = ToricVariety(Fan(cones=facets, rays=points))
sage: X.rational_class_group().dimension()
4
sage: X.Kaehler_cone().rays()
Divisor class [1, 0, 0, 0]
in Basis lattice of The toric rational divisor class group
of a 3-d toric variety covered by 7 affine patches
sage: antiK = -X.K()
sage: antiK.divisor_class()
Divisor class [2, 0, 0, 0]
```

```
sage: antiK.is_ample()
True
```

is_integral()

Return whether the coefficients of the divisor are all integral.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: DZZ = P2.toric_divisor_group(base_ring=ZZ).gen(0); DZZ
V(x)
sage: DQQ = P2.toric_divisor_group(base_ring=QQ).gen(0); DQQ
V(x)
sage: DZZ.is_integral()
True
sage: DQQ.is_integral()
True
```

is_nef()

Return whether a **Q**-Cartier divisor is nef.

OUTPUT:

- `True` if the divisor is in the closure of the ample cone, `False` otherwise.

Note:

- For a **Q**-Cartier divisor, some positive integral multiple is Cartier. We return wheher this associtated divisor is nef.
- The nef cone is the closure of the ample cone.
- See also `is_ample()`.
- A toric divisor is nef if and only if its support function is convex (but not necessarily strictly convex).
- A toric Cartier divisor is nef if and only if its linear system is basepoint free.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: K = P2.K()
sage: (+K).is_nef()
False
sage: (0*K).is_nef()
True
sage: (-K).is_nef()
True
```

Example 6.1.3, 6.1.11, 6.1.17 of [CLS]:

```
sage: fan = Fan(cones=[(0,1), (1,2), (2,3), (3,0)],
...             rays=[(-1,2), (0,1), (1,0), (0,-1)])
sage: F2 = ToricVariety(fan,'u1, u2, u3, u4')
sage: def D(a,b): return a*F2.divisor(2) + b*F2.divisor(3)
...
sage: [ (a,b) for a,b in CartesianProduct(range(-3,3),range(-3,3))
...           if D(a,b).is_ample() ]
[(1, 1), (1, 2), (2, 1), (2, 2)]
sage: [ (a,b) for a,b in CartesianProduct(range(-3,3),range(-3,3))
...           if D(a,b).is_nef() ]
```

```
[(0, 0), (0, 1), (0, 2), (1, 0),
 (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]
```

m(*cone*)

Return m_σ representing ϕ_D on `cone`.

Let X be the ambient toric variety of this divisor D associated to the fan Σ in lattice N. Let M be the lattice dual to N. Given the cone $\sigma = \langle v_1, \ldots, v_k \rangle$ in Σ, this method searches for a vector $m_\sigma \in M_{\mathbf{Q}}$ such that $\phi_D(v_i) = \langle m_\sigma, v_i \rangle$ for all $i = 1, \ldots, k$, where ϕ_D is the support function of D.

INPUT:

- `cone` – A cone in the fan of the toric variety.

OUTPUT:

- If possible, a point of lattice M.
- If the dual vector cannot be chosen integral, a rational vector is returned.
- If there is no such vector (i.e. `self` is not even a **Q**-Cartier divisor), a `ValueError` is raised.

EXAMPLES:

```
sage: F = Fan(cones=[(0,1,2,3), (0,1,4)],
...         rays=[(1,1,1), (1,-1,1), (1,-1,-1), (1,1,-1), (0,0,1)])
sage: X = ToricVariety(F)
sage: square_cone = X.fan().cone_containing(0,1,2,3)
sage: triangle_cone = X.fan().cone_containing(0,1,4)
sage: ray = X.fan().cone_containing(0)
sage: QQ_Cartier = X.divisor([2,2,1,1,1])
sage: QQ_Cartier.m(ray)
M(0, 2, 0)
sage: QQ_Cartier.m(square_cone)
(3/2, 0, 1/2)
sage: QQ_Cartier.m(triangle_cone)
M(1, 0, 1)
sage: QQ_Cartier.m(Cone(triangle_cone))
M(1, 0, 1)
sage: Weil = X.divisor([1,1,1,0,0])
sage: Weil.m(square_cone)
Traceback (most recent call last):
...
ValueError: V(z0) + V(z1) + V(z2) is not QQ-Cartier,
cannot choose a dual vector on 3-d cone
of Rational polyhedral fan in 3-d lattice N!
sage: Weil.m(triangle_cone)
M(1, 0, 0)
```

monomial(*point*)

Return the monomial in the homogeneous coordinate ring associated to the `point` in the dual lattice.

INPUT:

- `point` – a point in `self.variety().fan().dual_lattice()`.

OUTPUT:

For a fixed divisor `D`, the sections are generated by monomials in `ToricVariety.coordinate_ring`. Alternatively, the monomials can be described as M-lattice points in the polyhedron `D.polyhedron()`. This method converts the points $m \in M$ into homogeneous polynomials.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: O3_P2 = -P2.K()
sage: M = P2.fan().dual_lattice()
sage: O3_P2.monomial( M(0,0) )
x*y*z
```

move_away_from(*cone*)

Move the divisor away from the orbit closure of `cone`.

INPUT:

- A `cone` of the fan of the toric variety.

OUTPUT:

A (rationally equivalent) divisor that is moved off the orbit closure of the given cone.

Note: A divisor that is Weil but not Cartier might be impossible to move away. In this case, a `ValueError` is raised.

EXAMPLES:

```
sage: F = Fan(cones=[(0,1,2,3), (0,1,4)],
...           rays=[(1,1,1), (1,-1,1), (1,-1,-1), (1,1,-1), (0,0,1)])
sage: X = ToricVariety(F)
sage: square_cone = X.fan().cone_containing(0,1,2,3)
sage: triangle_cone = X.fan().cone_containing(0,1,4)
sage: line_cone = square_cone.intersection(triangle_cone)
sage: Cartier = X.divisor([2,2,1,1,1])
sage: Cartier
2*V(z0) + 2*V(z1) + V(z2) + V(z3) + V(z4)
sage: Cartier.move_away_from(line_cone)
-V(z2) - V(z3) + V(z4)
sage: QQ_Weil = X.divisor([1,0,1,1,0])
sage: QQ_Weil.move_away_from(line_cone)
V(z2)
```

polyhedron()

Return the polyhedron $P_D \subset M$ associated to a toric divisor D.

OUTPUT:

P_D as an instance of `Polyhedron_base`.

EXAMPLES:

```
sage: dP7 = toric_varieties.dP7()
sage: D = dP7.divisor(2)
sage: P_D = D.polyhedron(); P_D
A 0-dimensional polyhedron in QQ^2 defined as the convex hull of 1 vertex
sage: P_D.Vrepresentation()
(A vertex at (0, 0),)
sage: D.is_nef()
False
sage: dP7.integrate( D.ch() * dP7.Td() )
1
sage: P_antiK = (-dP7.K()).polyhedron(); P_antiK
A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 5 vertices
sage: P_antiK.Vrepresentation()
(A vertex at (1, -1), A vertex at (0, 1), A vertex at (1, 0),
```

```
 A vertex at (-1, 1), A vertex at (-1, -1))
sage: P_antiK.integral_points()
((-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0))
```

Example 6.1.3, 6.1.11, 6.1.17 of [CLS]:

```
sage: fan = Fan(cones=[(0,1), (1,2), (2,3), (3,0)],
...             rays=[(-1,2), (0,1), (1,0), (0,-1)])
sage: F2 = ToricVariety(fan,'u1, u2, u3, u4')
sage: D = F2.divisor(3)
sage: D.polyhedron().Vrepresentation()
(A vertex at (0, 0), A vertex at (2, 1), A vertex at (0, 1))
sage: Dprime = F2.divisor(1) + D
sage: Dprime.polyhedron().Vrepresentation()
(A vertex at (2, 1), A vertex at (0, 1), A vertex at (0, 0))
sage: D.is_ample()
False
sage: D.is_nef()
True
sage: Dprime.is_nef()
False
```

A more complicated example where P_D is not a lattice polytope:

```
sage: X = toric_varieties.BCdlOG_base()
sage: antiK = -X.K()
sage: P_D = antiK.polyhedron()
sage: P_D
A 3-dimensional polyhedron in QQ^3 defined as the convex hull of 8 vertices
sage: P_D.Vrepresentation()
(A vertex at (1, -1, 0), A vertex at (1, -3, 1),
 A vertex at (1, 1, 1), A vertex at (-5, 1, 1),
 A vertex at (1, 1, -1/2), A vertex at (1, 1/2, -1/2),
 A vertex at (-1, -1, 0), A vertex at (-5, -3, 1))
sage: P_D.Hrepresentation()
(An inequality (-1, 0, 0) x + 1 >= 0, An inequality (0, -1, 0) x + 1 >= 0,
 An inequality (0, 0, -1) x + 1 >= 0, An inequality (1, 0, 4) x + 1 >= 0,
 An inequality (0, 1, 3) x + 1 >= 0, An inequality (0, 1, 2) x + 1 >= 0)
sage: P_D.integral_points()
((-1, -1, 0), (0, -1, 0), (1, -1, 0), (-1, 0, 0), (0, 0, 0),
 (1, 0, 0), (-1, 1, 0), (0, 1, 0), (1, 1, 0), (-5, -3, 1),
 (-4, -3, 1), (-3, -3, 1), (-2, -3, 1), (-1, -3, 1), (0, -3, 1),
 (1, -3, 1), (-5, -2, 1), (-4, -2, 1), (-3, -2, 1), (-2, -2, 1),
 (-1, -2, 1), (0, -2, 1), (1, -2, 1), (-5, -1, 1), (-4, -1, 1),
 (-3, -1, 1), (-2, -1, 1), (-1, -1, 1), (0, -1, 1), (1, -1, 1),
 (-5, 0, 1), (-4, 0, 1), (-3, 0, 1), (-2, 0, 1), (-1, 0, 1),
 (0, 0, 1), (1, 0, 1), (-5, 1, 1), (-4, 1, 1), (-3, 1, 1),
 (-2, 1, 1), (-1, 1, 1), (0, 1, 1), (1, 1, 1))
```

sections()

Return the global sections (as points of the M-lattice) of the line bundle (or reflexive sheaf) associated to the divisor.

OUTPUT:

- `tuple` of points of lattice M.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2.fan().nrays()
3
sage: P2.divisor(0).sections()
(M(-1, 0), M(-1, 1), M(0, 0))
sage: P2.divisor(1).sections()
(M(0, -1), M(0, 0), M(1, -1))
sage: P2.divisor(2).sections()
(M(0, 0), M(0, 1), M(1, 0))
```

The divisor can be non-nef yet still have sections:

```
sage: rays = [(1,0,0),(0,1,0),(0,0,1),(-2,0,-1),(-2,-1,0),(-3,-1,-1),(1,1,1),(-1,0,0)]
sage: cones = [[0,1,3],[0,1,6],[0,2,4],[0,2,6],[0,3,5],[0,4,5],[1,3,7],[1,6,7],[2,4,7],[2,6,
sage: X = ToricVariety(Fan(rays=rays,cones=cones))
sage: D = X.divisor(2); D
V(z2)
sage: D.is_nef()
False
sage: D.sections()
(M(0, 0, 0),)
sage: D.cohomology(dim=True)
(1, 0, 0, 0)
```

sections_monomials()

Return the global sections of the line bundle associated to the Cartier divisor.

The sections are described as monomials in the generalized homogeneous coordinates.

OUTPUT:

- tuple of monomials in the coordinate ring of `self`.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2.fan().nrays()
3
sage: P2.divisor(0).sections_monomials()
(z, y, x)
sage: P2.divisor(1).sections_monomials()
(z, y, x)
sage: P2.divisor(2).sections_monomials()
(z, y, x)
```

From [CoxTutorial] page 38:

```
sage: lp = LatticePolytope([(1,0),(1,1),(0,1),(-1,0),(0,-1)])
sage: lp
2-d reflexive polytope #5 in 2-d lattice M
sage: dP7 = ToricVariety( FaceFan(lp), 'x1, x2, x3, x4, x5')
sage: AK = -dP7.K()
sage: AK.sections()
(N(-1, 0), N(-1, 1), N(0, -1), N(0, 0),
 N(0, 1), N(1, -1), N(1, 0), N(1, 1))
sage: AK.sections_monomials()
(x3*x4^2*x5, x2*x3^2*x4^2, x1*x4*x5^2, x1*x2*x3*x4*x5,
 x1*x2^2*x3^2*x4, x1^2*x2*x5^2, x1^2*x2^2*x3*x5, x1^2*x2^3*x3^2)
```

REFERENCES:

class sage.schemes.toric.divisor.**ToricRationalDivisorClassGroup**(*toric_variety*)
Bases: sage.modules.free_module.FreeModule_ambient_field, sage.structure.unique_representation.UniqueRepresentation

The rational divisor class group of a toric variety.

The **T-Weil divisor class group** $Cl(X)$ of a toric variety X is a finitely generated abelian group and can contain torsion. Its rank equals the number of rays in the fan of X minus the dimension of X.

The **rational divisor class group** is $Cl(X) \otimes_{\mathbf{Z}} \mathbf{Q}$ and never includes torsion. If X is *smooth*, this equals the **Picard group** $\mathrm{Pic}(X)$, whose elements are the isomorphism classes of line bundles on X. The group law (which we write as addition) is the tensor product of the line bundles. The Picard group of a toric variety is always torsion-free.

Warning: Do not instantiate this class yourself. Use rational_class_group() method of toric varieties if you need the divisor class group. Or you can obtain it as the parent of any divisor class constructed, for example, via ToricDivisor_generic.divisor_class().

INPUT:

- toric_variety – toric variety <sage.schemes.toric.variety.ToricVariety_field.

OUTPUT:

- rational divisor class group of a toric variety.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2.rational_class_group()
The toric rational divisor class group of a 2-d CPR-Fano
toric variety covered by 3 affine patches
sage: D = P2.divisor(0); D
V(x)
sage: Dclass = D.divisor_class(); Dclass
Divisor class [1]
sage: Dclass.lift()
V(y)
sage: Dclass.parent()
The toric rational divisor class group of a 2-d CPR-Fano
toric variety covered by 3 affine patches
```

Element
alias of ToricRationalDivisorClass

class sage.schemes.toric.divisor.**ToricRationalDivisorClassGroup_basis_lattice**(*group*)
Bases: sage.modules.free_module.FreeModule_ambient_pid

Construct the basis lattice of the group.

INPUT:

- group – toric rational divisor class group.

OUTPUT:

- the basis lattice of group.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: L = P1xP1.Kaehler_cone().lattice()
sage: L
```

```
Basis lattice of The toric rational divisor class group of a
2-d CPR-Fano toric variety covered by 4 affine patches
sage: L.basis()
[
Divisor class [1, 0],
Divisor class [0, 1]
]
```

Element
alias of `ToricRationalDivisorClass`

`sage.schemes.toric.divisor.`**`is_ToricDivisor`**`(x)`
Test whether `x` is a toric divisor.

INPUT:

- `x` – anything.

OUTPUT:

- `True` if `x` is an instance of `ToricDivisor_generic` and `False` otherwise.

EXAMPLES:

```
sage: from sage.schemes.toric.divisor import is_ToricDivisor
sage: is_ToricDivisor(1)
False
sage: P2 = toric_varieties.P2()
sage: D = P2.divisor(0); D
V(x)
sage: is_ToricDivisor(D)
True
```

16.5 Toric rational divisor classes

This module is a part of the framework for `toric varieties`.

AUTHORS:

- Volker Braun and Andrey Novoseltsev (2010-09-05): initial version.

TESTS:

Toric rational divisor clases are elements of the rational class group of a toric variety, represented as rational vectors in some basis:

```
sage: dP6 = toric_varieties.dP6()
sage: Cl = dP6.rational_class_group()
sage: D = Cl([1, -2, 3, -4])
sage: D
Divisor class [1, -2, 3, -4]
sage: E = Cl([1/2, -2/3, 3/4, -4/5])
sage: E
Divisor class [1/2, -2/3, 3/4, -4/5]
```

They behave much like ordinary vectors:

```
sage: D + E
Divisor class [3/2, -8/3, 15/4, -24/5]
sage: 2 * D
Divisor class [2, -4, 6, -8]
sage: E / 10
Divisor class [1/20, -1/15, 3/40, -2/25]
sage: D * E
Traceback (most recent call last):
...
TypeError: cannot multiply two divisor classes!
```

The only special method is `lift()` to get a divisor representing a divisor class:

```
sage: D.lift()
V(x) - 2*V(u) + 3*V(y) - 4*V(v)
sage: E.lift()
1/2*V(x) - 2/3*V(u) + 3/4*V(y) - 4/5*V(v)
```

class `sage.schemes.toric.divisor_class.`**`ToricRationalDivisorClass`**

Bases: `sage.modules.vector_rational_dense.Vector_rational_dense`

Create a toric rational divisor class.

> **Warning:** You probably should not construct divisor classes explicitly.

INPUT:

- same as for `Vector_rational_dense`.

OUTPUT:

- toric rational divisor class.

TESTS:

```
sage: dP6 = toric_varieties.dP6()
sage: Cl = dP6.rational_class_group()
sage: D = dP6.divisor(2)
sage: Cl(D)
Divisor class [0, 0, 1, 0]
```

`lift`()

Return a divisor representing this divisor class.

OUTPUT:

An instance of `ToricDivisor` representing `self`.

EXAMPLES:

```
sage: X = toric_varieties.Cube_nonpolyhedral()
sage: D = X.divisor([0,1,2,3,4,5,6,7]); D
V(z1) + 2*V(z2) + 3*V(z3) + 4*V(z4) + 5*V(z5) + 6*V(z6) + 7*V(z7)
sage: D.divisor_class()
Divisor class [29, 6, 8, 10, 0]
sage: Dequiv = D.divisor_class().lift(); Dequiv
6*V(z1) - 17*V(z2) - 22*V(z3) - 7*V(z4) + 25*V(z6) + 32*V(z7)
sage: Dequiv == D
False
sage: Dequiv.divisor_class() == D.divisor_class()
True
```

sage.schemes.toric.divisor_class.**is_ToricRationalDivisorClass**(*x*)

Check if x is a toric rational divisor class.

INPUT:

- x – anything.

OUTPUT:

- True if x is a toric rational divisor class, False otherwise.

EXAMPLES:

```
sage: from sage.schemes.toric.divisor_class import (
...       is_ToricRationalDivisorClass)
sage: is_ToricRationalDivisorClass(1)
False
sage: dP6 = toric_varieties.dP6()
sage: D = dP6.rational_class_group().gen(0)
sage: D
Divisor class [1, 0, 0, 0]
sage: is_ToricRationalDivisorClass(D)
True
```

16.6 The Chow group of a toric variety

In general, the Chow group is an algebraic version of a homology theory. That is, the objects are formal linear combinations of submanifolds modulo relations. In particular, the objects of the Chow group are formal linear combinations of algebraic subvarieties and the equivalence relation is rational equivalence. There is no relative version of the Chow group, so it is not a generalized homology theory.

The Chow groups of smooth or mildly singular toric varieties are almost the same as the homology groups:

- For smooth toric varieties, $A_k(X) = H_{2k}(X, \mathbf{Z})$. While they are the same, using the cohomology ring instead of the Chow group will be much faster! The cohomology ring does not try to keep track of torsion and uses Groebner bases to encode the cup product.
- For simplicial toric varieties, $A_k(X) \otimes \mathbf{Q} = H_{2k}(X, \mathbf{Q})$.

Note that in these cases the odd-dimensional (co)homology groups vanish. But for sufficiently singular toric varieties the Chow group differs from the homology groups (and the odd-dimensional homology groups no longer vanish). For singular varieties the Chow group is much easier to compute than the (co)homology groups.

The toric Chow group of a toric variety is the Chow group generated by the toric subvarieties, that is, closures of orbits under the torus action. These are in one-to-one correspondence with the cones of the fan and, therefore, the toric Chow group is a quotient of the free Abelian group generated by the cones. In particular, the toric Chow group has finite rank. One can show [FMSS1] that the toric Chow groups equal the "full" Chow group of a toric variety, so there is no need to distinguish these in the following.

AUTHORS:

- Volker Braun (2010-08-09): Initial version

REFERENCES:

EXAMPLES:

```
sage: X = toric_varieties.Cube_deformation(7)
sage: X.is_smooth()
False
sage: X.is_orbifold()
```

```
False
sage: A = X.Chow_group()
sage: A.degree()
(Z, C7, C2 x C2 x Z^5, Z)
sage: A.degree(2).ngens()
7
sage: a = sum( A.gen(i) * (i+1) for i in range(0,A.ngens()) )   # an element of A
sage: a  # long time (2s on sage.math, 2011)
( 3 | 1 mod 7 | 0 mod 2, 1 mod 2, 4, 5, 6, 7, 8 | 9 )
```

The Chow group elements are printed as `( a0 | a1 mod 7 | a2 mod 2, a3 mod 2, a4, a5, a6, a7, a8 | a9 )`, which denotes the element of the Chow group in the same basis as `A.degree()`. The `|` separates individual degrees, so the example means:

- The degree-0 part is $3 \in \mathbf{Z}$.
- The degree-1 part is $1 \in \mathbf{Z}_7$.
- The torsion of the degree-2 Chow group is $(0, 1) \in \mathbf{Z}_2 \oplus \mathbf{Z}_2$.
- The free part of the degree-2 Chow group is $(4, 5, 6, 7, 8) \in \mathbf{Z}^5$.
- The degree-3 part is $9 \in \mathbf{Z}$.

Note that the generators `A.gens()` are not sorted in any way. In fact, they may be of mixed degree. Use `A.gens(degree=d)` to obtain the generators in a fixed degree `d`. See `ChowGroup_class.gens()` for more details.

Cones of toric varieties can determine their own Chow cycle:

```
sage: A = X.Chow_group(); A
Chow group of 3-d toric variety covered by 6 affine patches
sage: cone = X.fan(dim=2)[3]; cone
2-d cone of Rational polyhedral fan in 3-d lattice N
sage: A_cone = A(cone); A_cone
( 0 | 1 mod 7 | 0 mod 2, 0 mod 2, 0, 0, 0, 0, 0 | 0 )
sage: A_cone.degree()
1
sage: 2 * A_cone
( 0 | 2 mod 7 | 0 mod 2, 0 mod 2, 0, 0, 0, 0, 0 | 0 )
sage: A_cone + A.gen(0)
( 0 | 1 mod 7 | 0 mod 2, 1 mod 2, 0, 0, 0, 0, 0 | 0 )
```

Chow cycles can be of mixed degrees:

```
sage: mixed = sum(A.gens()); mixed
( 1 | 4 mod 7 | 1 mod 2, 1 mod 2, 1, 1, 1, 1, 1 | 1 )
sage: mixed.project_to_degree(1)
( 0 | 4 mod 7 | 0 mod 2, 0 mod 2, 0, 0, 0, 0, 0 | 0 )
sage: sum( mixed.project_to_degree(i) for i in range(0,X.dimension()+1) ) == mixed
True
```

class `sage.schemes.toric.chow_group.`**`ChowCycle`**(*parent*, *v*, *check=True*)

Bases: `sage.modules.fg_pid.fgp_element.FGP_Element`

The elements of the Chow group.

> **Warning:** Do not construct `ChowCycle` objects manually. Instead, use the parent `ChowGroup` to obtain generators or Chow cycles correspondig to cones of the fan.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: A = P2.Chow_group()
sage: A.gens()
(( 1 | 0 | 0 ), ( 0 | 1 | 0 ), ( 0 | 0 | 1 ))
sage: cone = P2.fan(1)[0]
sage: A(cone)
( 0 | 1 | 0 )
sage: A( Cone([(1,0)]) )
( 0 | 1 | 0 )
```

cohomology_class()

Return the (Poincare-dual) cohomology class.

Consider a simplicial cone of the fan, that is, a d-dimensional cone spanned by d rays. Take the product of the corresponding d homogeneous coordinates. This monomial represents a cohomology classes of the toric variety X, see `cohomology_ring()`. Its cohomological degree is $2d$, which is the same degree as the Poincare-dual of the (real) $\dim(X) - 2d$-dimensional torus orbit associated to the simplicial cone. By linearity, we can associate a cohomology class to each Chow cycle of a simplicial toric variety.

If the toric variety is compact and smooth, the associated cohomology class actually is the Poincare dual (over the integers) of the Chow cycle. In particular, integrals of dual cohomology classes perform intersection computations.

If the toric variety is compact and has at most orbifold singularities, the torsion parts in cohomology and the Chow group can differ. But they are still isomorphic as rings over the rationals. Moreover, the normalization of integration (`volume_class`) and `count_points()` are chosen to agree.

OUTPUT:

The `CohomologyClass` which is associated to the Chow cycle.

If the toric variety is not simplicial, that is, has worse than orbifold singularities, there is no way to associate a cohomology class of the correct degree. In this case, `cohomology_class()` rasies a `ValueError`.

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: cone = dP6.fan().cone_containing(2,3)
sage: HH = dP6.cohomology_ring()
sage: A = dP6.Chow_group()
sage: HH(cone)
[-w^2]
sage: A(cone)
( 1 | 0, 0, 0, 0 | 0 )
sage: A(cone).cohomology_class()
[-w^2]
```

Here is an example of a toric variety with orbifold singularities, where we can also use the isomorphism with the rational cohomology ring:

```
sage: WP4 = toric_varieties.P4_11169()
sage: A = WP4.Chow_group()
sage: HH = WP4.cohomology_ring()
sage: cone3d = Cone([(0,0,1,0), (0,0,0,1), (-9,-6,-1,-1)])
sage: A(cone3d)
( 0 | 1 | 0 | 0 | 0 )
sage: HH(cone3d)
[3*z4^3]

sage: D = -WP4.K()  # the anticanonical divisor
```

```
sage: A(D)
( 0 | 0 | 0 | 18 | 0 )
sage: HH(D)
[18*z4]

sage: WP4.integrate( A(cone3d).cohomology_class() * D.cohomology_class() )
1
sage: WP4.integrate( HH(cone3d) * D.cohomology_class() )
1
sage: A(cone3d).intersection_with_divisor(D).count_points()
1
```

count_points()

Return the number of points in the Chow cycle.

OUTPUT:

An element of `self.base_ring()`, which is usually $\mathbf{Z}$. The number of points in the Chow cycle.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: A = P2.Chow_group()
sage: a = 5*A.gen(0) + 7*A.gen(1); a
( 5 | 7 | 0 )
sage: a.count_points()
5
```

In the case of a smooth complete toric variety, the Chow (homology) groups are Poincare dual to the integral cohomology groups. Here is such a smooth example:

```
sage: D = P2.divisor(1)
sage: a = D.Chow_cycle()
sage: aD = a.intersection_with_divisor(D)
sage: aD.count_points()
1
sage: P2.integrate( aD.cohomology_class() )
1
```

For toric varieties with at most orbifold singularities, the isomorphism only holds over $\mathbf{Q}$. But the normalization of the integral is still chosen such that the intersection numbers (which are potentially rational) computed both ways agree:

```
sage: P1xP1_Z2 = toric_varieties.P1xP1_Z2()
sage: Dt = P1xP1_Z2.divisor(1);  Dt
V(t)
sage: Dy = P1xP1_Z2.divisor(3);  Dy
V(y)
sage: Dt.Chow_cycle(QQ).intersection_with_divisor(Dy).count_points()
1/2
sage: P1xP1_Z2.integrate( Dt.cohomology_class() * Dy.cohomology_class() )
1/2
```

degree()

The degree of the Chow cycle.

OUTPUT:

Integer. The complex dimension of the subvariety representing the Chow cycle. Raises a `ValueError` if the Chow cycle is a sum of mixed degree cycles.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: A = P2.Chow_group()
sage: [ a.degree() for a in A.gens() ]
[0, 1, 2]
```

intersection_with_divisor(*divisor*)

Intersect the Chow cycle with `divisor`.

See [FultonChow] for a description of the toric algorithm.

INPUT:

- `divisor` – a `ToricDivisor` that can be moved away from the Chow cycle. For example, any Cartier divisor. See also `ToricDivisor.move_away_from`.

OUTPUT:

A new `ChowCycle`. If the divisor is not Cartier then this method potentially raises a `ValueError`, indicating that the divisor cannot be made transversal to the Chow cycle.

EXAMPLES:

```
sage: dP6 = toric_varieties.dP6()
sage: cone = dP6.fan().cone_containing(2); cone
1-d cone of Rational polyhedral fan in 2-d lattice N
sage: D = dP6.divisor(cone); D
V(y)
sage: A = dP6.Chow_group()
sage: A(cone)
( 0 | 0, 0, 0, 1 | 0 )
sage: intersection = A(cone).intersection_with_divisor(D); intersection
( -1 | 0, 0, 0, 0 | 0 )
sage: intersection.count_points()
-1
```

You can do the same computation over the rational Chow group since there is no torsion in this case:

```
sage: A_QQ = dP6.Chow_group(base_ring=QQ)
sage: A_QQ(cone)
( 0 | 0, 0, 0, 1 | 0 )
sage: intersection_QQ = A_QQ(cone).intersection_with_divisor(D); intersection
( -1 | 0, 0, 0, 0 | 0 )
sage: intersection_QQ.count_points()
-1
sage: type(intersection_QQ.count_points())
<type 'sage.rings.rational.Rational'>
sage: type(intersection.count_points())
<type 'sage.rings.integer.Integer'>
```

TESTS:

The relations are the Chow cycles rationally equivalent to the zero cycle. Their intersection with any divisor must be the zero cycle:

```
sage: [ r.intersection_with_divisor(D) for r in dP6.Chow_group().relation_gens() ]
[( 0 | 0, 0, 0, 0 | 0 ), ( 0 | 0, 0, 0, 0 | 0 ),
 ( 0 | 0, 0, 0, 0 | 0 ), ( 0 | 0, 0, 0, 0 | 0 ),
 ( 0 | 0, 0, 0, 0 | 0 ), ( 0 | 0, 0, 0, 0 | 0 ),
 ( 0 | 0, 0, 0, 0 | 0 )]
sage: [ r.intersection_with_divisor(D).lift() for r in dP6.Chow_group().relation_gens() ]
[(0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 0, 0),
```

```
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)]
```

project_to_degree (*degree*)

Project a (mixed-degree) Chow cycle to the given `degree`.

INPUT:

- `degree` – integer. The degree to project to.

OUTPUT:

The projection of the Chow class to the given degree as a new `ChowCycle` of the same Chow group.

EXAMPLES:

```
sage: A = toric_varieties.P2().Chow_group()
sage: cycle = 10*A.gen(0) + 11*A.gen(1) + 12*A.gen(2)
sage: cycle
( 10 | 11 | 12 )
sage: cycle.project_to_degree(2)
( 0 | 0 | 12 )
```

class `sage.schemes.toric.chow_group.`**ChowGroupFactory**

Bases: `sage.structure.factory.UniqueFactory`

Factory for `ChowGroup_class`.

create_key_and_extra_args (*toric_variety*, *base_ring=Integer Ring*, *check=True*)

Create a key that uniquely determines the `ChowGroup_class`.

INPUT:

- `toric_variety` – a toric variety.
- `base_ring` – either $\mathbf{Z}$ (default) or $\mathbf{Q}$. The coefficient ring of the Chow group.
- `check` – boolean (default: `True`).

EXAMPLES:

```
sage: from sage.schemes.toric.chow_group import *
sage: P2 = toric_varieties.P2()
sage: ChowGroup(P2, ZZ, check=True) == ChowGroup(P2, ZZ, check=False)   # indirect doctest
True
```

create_object (*version*, *key*, ***extra_args*)

Create a `ChowGroup_class`.

INPUT:

- `version` – object version. Currently not used.
- `key` – a key created by `create_key_and_extra_args()`.
- `**extra_args` – Currently not used.

EXAMPLES:

```
sage: from sage.schemes.toric.chow_group import *
sage: P2 = toric_varieties.P2()
sage: ChowGroup(P2)     # indirect doctest
Chow group of 2-d CPR-Fano toric variety covered by 3 affine patches
```

class sage.schemes.toric.chow_group.**ChowGroup_class**(*toric_variety*, *base_ring*, *check*)

Bases: sage.modules.fg_pid.fgp_module.FGP_Module_class

The Chow group of a toric variety.

EXAMPLES:

```
sage: P2=toric_varieties.P2()
sage: from sage.schemes.toric.chow_group import ChowGroup_class
sage: A = ChowGroup_class(P2,ZZ,True);   A
Chow group of 2-d CPR-Fano toric variety covered by 3 affine patches
sage: A.an_element()
( 1 | 0 | 0 )
```

Element

alias of ChowCycle

coordinate_vector(*chow_cycle*, *degree=None*, *reduce=True*)

Return the coordinate vector of the chow_cycle.

INPUT:

- chow_cycle – a ChowCycle.
- degree – None (default) or an integer.
- reduce – boolean (default: True). Whether to reduce modulo the invariants.

OUTPUT:

- If degree is None (default), the coordinate vector relative to the basis self.gens() is returned.
- If some integer degree=d is specified, the chow cycle is projected to the given degree and the coordinate vector relative to the basis self.gens(degree=d) is returned.

EXAMPLES:

```
sage: A = toric_varieties.P2().Chow_group()
sage: a = A.gen(0) + 2*A.gen(1) + 3*A.gen(2)
sage: A.coordinate_vector(a)
(1, 2, 3)
sage: A.coordinate_vector(a, degree=1)
(2)
```

degree(*k=None*)

Return the degree-k Chow group.

INPUT:

- k – an integer or None (default). The degree of the Chow group.

OUTPUT:

- if k was specified, the Chow group A_k as an Abelian group.
- if k was not specified, a tuple containing the Chow groups in all degrees.

Note:

- For a smooth toric variety, this is the same as the Poincare-dual cohomology group $H^{d-2k}(X, \mathbf{Z})$.
- For a simplicial toric variety ("orbifold"), $A_k(X) \otimes \mathbf{Q} = H^{d-2k}(X, \mathbf{Q})$.

EXAMPLES:

Four exercises from page 65 of [FultonP65]. First, an example with $A_1(X) = \mathbf{Z} \oplus \mathbf{Z}/3\mathbf{Z}$:

```
sage: X = ToricVariety(Fan(cones=[[0,1],[1,2],[2,0]],
...                         rays=[[2,-1],[-1,2],[-1,-1]]))
sage: A = X.Chow_group()
sage: A.degree(1)
C3 x Z
```

Second, an example with $A_2(X) = \mathbf{Z}^2$:

```
sage: points = [[1,0,0],[0,1,0],[0,0,1],[1,-1,1],[-1,0,-1]]
sage: l = LatticePolytope(points)
sage: l.show3d()
sage: X = ToricVariety(FaceFan(l))
sage: A = X.Chow_group()
sage: A.degree(2)
Z^2
```

Third, an example with $A_2(X) = \mathbf{Z}^5$:

```
sage: cube = [[ 1,0,0],[0, 1,0],[0,0, 1],[-1, 1, 1],
...           [-1,0,0],[0,-1,0],[0,0,-1],[ 1,-1,-1]]
sage: lat_cube = LatticePolytope(cube)
sage: X = ToricVariety(FaceFan((LatticePolytope(lat_cube))))
sage: X.Chow_group().degree(2)
Z^5
```

Fourth, a fan that is not the fan over a polytope. Combinatorially, the fan is the same in the third example, only the coordinates of the first point are different. But the resulting fan is not the face fan of a cube, so the variety is "more singular". Its Chow group has torsion, $A_2(X) = \mathbf{Z}^5 \oplus \mathbf{Z}/2$:

```
sage: rays = [[ 1, 2, 3],[ 1,-1, 1],[-1, 1, 1],[-1,-1, 1],
...           [-1,-1,-1],[-1, 1,-1],[ 1,-1,-1],[ 1, 1,-1]]
sage: cones = [[0,1,2,3],[4,5,6,7],[0,1,7,6],
...            [4,5,3,2],[0,2,5,7],[4,6,1,3]]
sage: X = ToricVariety(Fan(cones, rays))
sage: X.Chow_group().degree(2)  # long time (2s on sage.math, 2011)
C2 x Z^5
```

Finally, Example 1.3 of [FS]:

```
sage: points_mod = lambda k: matrix([[ 1, 1, 2*k+1],[ 1,-1, 1],
...                                  [-1, 1, 1],[-1,-1, 1],[-1,-1,-1],
...                                  [-1, 1,-1],[ 1,-1,-1],[ 1, 1,-1]])
sage: rays = lambda k: matrix([[1,1,1],[1,-1,1],[-1,1,1]]
...                           ).solve_left(points_mod(k)).rows()
sage: cones = [[0,1,2,3],[4,5,6,7],[0,1,7,6],
...            [4,5,3,2],[0,2,5,7],[4,6,1,3]]
sage: X_Delta = lambda k: ToricVariety(Fan(cones=cones, rays=rays(k)))
sage: X_Delta(0).Chow_group().degree()  # long time (3s on sage.math, 2011)
(Z, Z, Z^5, Z)
sage: X_Delta(1).Chow_group().degree()  # long time (3s on sage.math, 2011)
(Z, 0, Z^5, Z)
sage: X_Delta(2).Chow_group().degree()  # long time (3s on sage.math, 2011)
(Z, C2, Z^5, Z)
```

```
sage: X_Delta(2).Chow_group(base_ring=QQ).degree()   # long time (4s on sage.math, 2011)
(Q, 0, Q^5, Q)
```

gens (*degree=None*)

Return the generators of the Chow group.

INPUT:

- `degree` – integer (optional). The degree of the Chow group.

OUTPUT:

- if no degree is specified, the generators of the whole Chow group. The chosen generators may be of mixed degree.
- if `degree=` k was specified, the generators of the degree-k part A_k of the Chow group.

EXAMPLES:

```
sage: A = toric_varieties.P2().Chow_group()
sage: A.gens()
(( 1 | 0 | 0 ), ( 0 | 1 | 0 ), ( 0 | 0 | 1 ))
sage: A.gens(degree=1)
(( 0 | 1 | 0 ),)
```

relation_gens ()

Return the Chow cycles equivalent to zero.

For each $d-k-1$-dimensional cone $\rho \in \Sigma^{(d-k-1)}$, the relations in $A_k(X)$, that is the cycles equivalent to zero, are generated by

$$0 \stackrel{!}{=} \mathrm{div}(u) = \sum_{\rho<\sigma\in\Sigma^{(n-p)}} \langle u, n_{\rho,\sigma}\rangle V(\sigma), \qquad u \in M(\rho)$$

where $n_{\rho,\sigma}$ is a (randomly chosen) lift of the generator of $N_\sigma/N_\rho \simeq \mathbf{Z}$. See also Exercise 12.5.7 of [CLS].

See also `relations()` to obtain the relations as submodule of the free module generated by the cones. Or use `self.relations().gens()` to list the relations in the free module.

OUTPUT:

A tuple of Chow cycles, each rationally equivalent to zero, that generates the rational equivalence.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: A = P2.Chow_group()
sage: first = A.relation_gens()[0]
sage: first
( 0 | 0 | 0 )
sage: first.is_zero()
True
sage: first.lift()
(0, 1, 0, -1, 0, 0, 0)
```

scheme ()

Return the underlying toric variety.

OUTPUT:

A `ToricVariety`.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: A = P2.Chow_group()
sage: A.scheme()
2-d CPR-Fano toric variety covered by 3 affine patches
sage: A.scheme() is P2
True
```

class sage.schemes.toric.chow_group.**ChowGroup_degree_class**(A, d)

Bases: sage.structure.sage_object.SageObject

A fixed-degree subgroup of the Chow group of a toric variety.

WARNING ..

Use degree() to construct ChowGroup_degree_class instances.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: A = P2.Chow_group()
sage: A
Chow group of 2-d CPR-Fano toric variety covered by 3 affine patches
sage: A.degree()
(Z, Z, Z)
sage: A.degree(2)
Z
sage: type(_)
<class 'sage.schemes.toric.chow_group.ChowGroup_degree_class'>
```

gen(i)

Return the i-th generator of the Chow group of fixed degree.

INPUT:

- i – integer. The index of the generator to be returned.

OUTPUT:

A tuple of Chow cycles of fixed degree generating module().

EXAMPLES:

```
sage: projective_plane = toric_varieties.P2()
sage: A2 = projective_plane.Chow_group().degree(2)
sage: A2.gen(0)
( 0 | 0 | 1 )
```

gens()

Return the generators of the Chow group of fixed degree.

OUTPUT:

A tuple of Chow cycles of fixed degree generating module().

EXAMPLES:

```
sage: projective_plane = toric_varieties.P2()
sage: A2 = projective_plane.Chow_group().degree(2)
sage: A2.gens()
(( 0 | 0 | 1 ),)
```

module()

Return the submodule of the toric Chow group generated.

OUTPUT:

A `sage.modules.fg_pid.fgp_module.FGP_Module_class`

EXAMPLES:

```
sage: projective_plane = toric_varieties.P2()
sage: A2 = projective_plane.Chow_group().degree(2)
sage: A2.module()
Finitely generated module V/W over Integer Ring with invariants (0)
```

ngens()

Return the number of generators.

OUTPUT:

An integer.

EXAMPLES:

```
sage: projective_plane = toric_varieties.P2()
sage: A2 = projective_plane.Chow_group().degree(2)
sage: A2.ngens()
1
```

`sage.schemes.toric.chow_group.`**is_ChowCycle**(*x*)

Return whether `x` is a `ChowGroup_class`

INPUT:

- `x` – anything.

OUTPUT:

`True` or `False`.

EXAMPLES:

```
sage: P2=toric_varieties.P2()
sage: A = P2.Chow_group()
sage: from sage.schemes.toric.chow_group import *
sage: is_ChowCycle(A)
False
sage: is_ChowCycle(A.an_element())
True
sage: is_ChowCycle('Victoria')
False
```

`sage.schemes.toric.chow_group.`**is_ChowGroup**(*x*)

Return whether `x` is a `ChowGroup_class`

INPUT:

- `x` – anything.

OUTPUT:

`True` or `False`.

EXAMPLES:

```
sage: P2=toric_varieties.P2()
sage: A = P2.Chow_group()
sage: from sage.schemes.toric.chow_group import is_ChowGroup
sage: is_ChowGroup(A)
True
```

```
sage: is_ChowGroup('Victoria')
False
```

16.7 Toric ideals

A toric ideal (associated to an integer matrix A) is an ideal of the form

$$I_A = \left\langle x^u - x^v : u, v \in \mathbf{Z}_{\geq}^n, u - v \in \ker(A) \right\rangle$$

In other words, it is an ideal generated by irreducible "binomials", that is, differences of monomials without a common factor. Since the Buchberger algorithm preserves this property, any Groebner basis is then also generated by binomials.

EXAMPLES:

```
sage: A = matrix([[1,1,1],[0,1,2]])
sage: IA = ToricIdeal(A)
sage: IA.ker()
Free module of degree 3 and rank 1 over Integer Ring
User basis matrix:
[-1  2 -1]
sage: IA
Ideal (-z1^2 + z0*z2) of Multivariate Polynomial
Ring in z0, z1, z2 over Rational Field
```

Here, the "naive" ideal generated by $z_0 z_2 - z_1^2$ does already equal the toric ideal. But that is not true in general! For example, this toric ideal ([ProcSympPureMath62], Example 1.2) is the twisted cubic and cannot be generated by $2 = \dim \ker(A)$ polynomials:

```
sage: A = matrix([[3,2,1,0],[0,1,2,3]])
sage: IA = ToricIdeal(A)
sage: IA.ker()
Free module of degree 4 and rank 2 over Integer Ring
User basis matrix:
[-1  1  1 -1]
[-1  2 -1  0]
sage: IA
Ideal (-z1*z2 + z0*z3, -z1^2 + z0*z2, z2^2 - z1*z3) of
Multivariate Polynomial Ring in z0, z1, z2, z3 over Rational Field
```

The following family of toric ideals is from Example 4.4 of [ProcSympPureMath62]. One can show that I_d is generated by one quadric and d binomials of degree d:

```
sage: I = lambda d: ToricIdeal(matrix([[1,1,1,1,1],[0,1,1,0,0],[0,0,1,1,d]]))
sage: I(2)
Ideal (-z3^2 + z0*z4,
       z0*z2 - z1*z3,
       z2*z3 - z1*z4) of
Multivariate Polynomial Ring in z0, z1, z2, z3, z4 over Rational Field
sage: I(3)
Ideal (-z3^3 + z0^2*z4,
       z0*z2 - z1*z3,
       z2*z3^2 - z0*z1*z4,
       z2^2*z3 - z1^2*z4) of
Multivariate Polynomial Ring in z0, z1, z2, z3, z4 over Rational Field
sage: I(4)
Ideal (-z3^4 + z0^3*z4,
```

```
        z0*z2 - z1*z3,
        z2*z3^3 - z0^2*z1*z4,
        z2^2*z3^2 - z0*z1^2*z4,
        z2^3*z3 - z1^3*z4) of
Multivariate Polynomial Ring in z0, z1, z2, z3, z4 over Rational Field
```

Finally, the example in [GRIN]

```
sage: A = matrix(ZZ, [ [15,  4, 14, 19,  2,  1, 10, 17],
...                    [18, 11, 13,  5, 16, 16,  8, 19],
...                    [11,  7,  8, 19, 15, 18, 14,  6],
...                    [17, 10, 13, 17, 16, 14, 15, 18] ])
sage: IA = ToricIdeal(A)      # long time
sage: IA.ngens()              # long time
213
```

TESTS:

```
sage: A = matrix(ZZ, [[1, 1, 0, 0, -1,  0,  0, -1],
...                   [0, 0, 1, 1,  0, -1, -1,  0],
...                   [1, 0, 0, 1,  1,  1,  0,  0],
...                   [1, 0, 0, 1,  0,  0, -1, -1]])
sage: IA = ToricIdeal(A)
sage: R = IA.ring()
sage: R.inject_variables()
Defining z0, z1, z2, z3, z4, z5, z6, z7
sage: IA == R.ideal([z4*z6-z5*z7, z2*z5-z3*z6, -z3*z7+z2*z4,
...       -z2*z6+z1*z7, z1*z4-z3*z6, z0*z7-z3*z6, -z1*z5+z0*z6, -z3*z5+z0*z4,
...       z0*z2-z1*z3])  # Computed with Maple 12
True
```

The next example first appeared in Example 12.7 in [GBCP]. It is also used by the Maple 12 help system as example:

```
sage: A = matrix(ZZ, [[1, 2, 3, 4, 0, 1, 4, 5],
...                   [2, 3, 4, 1, 1, 4, 5, 0],
...                   [3, 4, 1, 2, 4, 5, 0, 1],
...                   [4, 1, 2, 3, 5, 0, 1, 4]])
sage: IA = ToricIdeal(A, 'z1, z2, z3, z4, z5, z6, z7, z8')
sage: R = IA.ring()
sage: R.inject_variables()
Defining z1, z2, z3, z4, z5, z6, z7, z8
sage: IA == R.ideal([z4^4-z6*z8^3, z3^4-z5*z7^3, -z4^3+z2*z8^2,
...       z2*z4-z6*z8, -z4^2*z6+z2^2*z8, -z4*z6^2+z2^3, -z3^3+z1*z7^2,
...       z1*z3-z5*z7, -z3^2*z5+z1^2*z7, z1^3-z3*z5^2])
True
```

REFERENCES:

AUTHORS:

- Volker Braun (2011-01-03): Initial version

class `sage.schemes.toric.ideal.`**`ToricIdeal`**(*A*, *names='z'*, *base_ring=Rational Field*, *polynomial_ring=None*, *algorithm='HostenSturmfels'*)

Bases: `sage.rings.polynomial.multi_polynomial_ideal.MPolynomialIdeal`

This class represents a toric ideal defined by an integral matrix.

INPUT:

- `A` – integer matrix. The defining matrix of the toric ideal.

- names – string (optional). Names for the variables. By default, this is `'z'` and the variables will be named `z0`, `z1`, ...
- base_ring – a ring (optional). Default: $\mathbf{Q}$. The base ring of the ideal. A toric ideal uses only coefficents ± 1.
- polynomial_ring – a polynomial ring (optional). The polynomial ring to construct the ideal in.

  You may specify the ambient polynomial ring via the `polynomial_ring` parameter or via the `names` and `base_ring` parameter. A `ValueError` is raised if you specify both.
- algorithm – string (optional). The algorithm to use. For now, must be `'HostenSturmfels'` which is the algorithm proposed by Hosten and Sturmfels in [GRIN].

EXAMPLES:

```
sage: A = matrix([[1,1,1],[0,1,2]])
sage: ToricIdeal(A)
Ideal (-z1^2 + z0*z2) of Multivariate Polynomial Ring
in z0, z1, z2 over Rational Field
```

First way of specifying the polynomial ring:

```
sage: ToricIdeal(A, names='x,y,z', base_ring=ZZ)
Ideal (-y^2 + x*z) of Multivariate Polynomial Ring
in x, y, z over Integer Ring
```

Second way of specifying the polynomial ring:

```
sage: R.<x,y,z> = ZZ[]
sage: ToricIdeal(A, polynomial_ring=R)
Ideal (-y^2 + x*z) of Multivariate Polynomial Ring
in x, y, z over Integer Ring
```

It is an error to specify both:

```
sage: ToricIdeal(A, names='x,y,z', polynomial_ring=R)
Traceback (most recent call last):
...
ValueError: You must not specify both variable names and a polynomial ring.
```

A()

Return the defining matrix.

OUTPUT:

An integer matrix.

EXAMPLES:

```
sage: A = matrix([[1,1,1],[0,1,2]])
sage: IA = ToricIdeal(A)
sage: IA.A()
[1 1 1]
[0 1 2]
```

ker()

Return the kernel of the defining matrix.

OUTPUT:

The kernel of `self.A()`.

EXAMPLES:

```
sage: A = matrix([[1,1,1],[0,1,2]])
sage: IA = ToricIdeal(A)
sage: IA.ker()
Free module of degree 3 and rank 1 over Integer Ring
User basis matrix:
[-1  2 -1]
```

nvariables()

Return the number of variables of the ambient polynomial ring.

OUTPUT:

Integer. The number of columns of the defining matrix `A()`.

EXAMPLES:

```
sage: A = matrix([[1,1,1],[0,1,2]])
sage: IA = ToricIdeal(A)
sage: IA.nvariables()
3
```

16.8 Morphisms of Toric Varieties

There are three "obvious" ways to map toric varieties to toric varieties:

1. Polynomial maps in local coordinates, the usual morphisms in algebraic geometry.
2. Polynomial maps in the (global) homogeneous coordinates.
3. Toric morphisms, that is, algebraic morphisms equivariant with respect to the torus action on the toric variety.

Both 2 and 3 are special cases of 1, which is just to say that we always remain within the realm of algebraic geometry. But apart from that, none is included in one of the other cases. In the examples below, we will explore some algebraic maps that can or can not be written as a toric morphism. Often a toric morphism can be written with polynomial maps in homogeneous coordinates, but sometimes it cannot.

The toric morphisms are perhaps the most mysterious at the beginning. Let us quickly review their definition (See Definition 3.3.3 of [CLS]). Let Σ_1 be a fan in $N_{1,\mathbf{R}}$ and Σ_2 be a fan in $N_{2,\mathbf{R}}$. A morphism $\phi : X_{\Sigma_1} \to X_{\Sigma_2}$ of the associated toric varieties is toric if ϕ maps the maximal torus $T_{N_1} \subseteq X_{\Sigma_1}$ into $T_{N_2} \subseteq X_{\Sigma_2}$ and $\phi|_{T_N}$ is a group homomorphism.

The data defining a toric morphism is precisely what defines a fan morphism (see `fan_morphism`), extending the more familiar dictionary between toric varieties and fans. Toric geometry is a functor from the category of fans and fan morphisms to the category of toric varieties and toric morphisms.

Note: Do not create the toric morphisms (or any morphism of schemes) directly from the the `SchemeMorphism...` classes. Instead, use the `hom()` method common to all algebraic schemes to create new homomorphisms.

EXAMPLES:

First, consider the following embedding of $\mathbb{P}^1$ into $\mathbb{P}^2$

```
sage: P2.<x,y,z> = toric_varieties.P2()
sage: P1.<u,v> = toric_varieties.P1()
sage: P1.hom([0,u^2+v^2,u*v], P2)
Scheme morphism:
  From: 1-d CPR-Fano toric variety covered by 2 affine patches
```

```
  To:   2-d CPR-Fano toric variety covered by 3 affine patches
  Defn: Defined on coordinates by sending [u : v] to
        [0 : u^2 + v^2 : u*v]
```

This is a well-defined morphism of algebraic varieties because homogeneously rescaled coordinates of a point of $\mathbb{P}^1$ map to the same point in $\mathbb{P}^2$ up to its homogeneous rescalings. It is not equivariant with respect to the torus actions

$$\mathbf{C}^\times \times \mathbb{P}^1, (\mu, [u : v]) \mapsto [u : \mu v] \quad \text{and} \quad \left(\mathbf{C}^\times\right)^2 \times \mathbb{P}^2, ((\alpha, \beta), [x : y : z]) \mapsto [x : \alpha y : \beta z],$$

hence it is not a toric morphism. Clearly, the problem is that the map in homogeneous coordinates contains summands that transform differently under the torus action. However, this is not the only difficulty. For example, consider

```
sage: phi = P1.hom([0,u,v], P2);  phi
Scheme morphism:
  From: 1-d CPR-Fano toric variety covered by 2 affine patches
  To:   2-d CPR-Fano toric variety covered by 3 affine patches
  Defn: Defined on coordinates by sending [u : v] to
        [0 : u : v]
```

This map is actually the embedding of the `orbit_closure()` associated to one of the rays of the fan of $\mathbb{P}^2$. Now the morphism is equivariant with respect to **some** map $\mathbf{C}^\times \to (\mathbf{C}^\times)^2$ of the maximal tori of $\mathbb{P}^1$ and $\mathbb{P}^2$. But this map of the maximal tori cannot be the same as `phi` defined above. Indeed, the image of `phi` completely misses the maximal torus $T_{\mathbb{P}^2} = \{[x : y : z] | x \neq 0, y \neq 0, z \neq 0\}$ of $\mathbb{P}^2$.

Consider instead the following morphism of fans:

```
sage: fm = FanMorphism( matrix(ZZ,[[1,0]]), P1.fan(), P2.fan() );  fm
Fan morphism defined by the matrix
[1 0]
Domain fan: Rational polyhedral fan in 1-d lattice N
Codomain fan: Rational polyhedral fan in 2-d lattice N
```

which also defines a morphism of toric varieties:

```
sage: P1.hom(fm, P2)
Scheme morphism:
  From: 1-d CPR-Fano toric variety covered by 2 affine patches
  To:   2-d CPR-Fano toric variety covered by 3 affine patches
  Defn: Defined by sending Rational polyhedral fan in 1-d lattice N
        to Rational polyhedral fan in 2-d lattice N.
```

The fan morphism map is equivalent to the following polynomial map:

```
sage: _.as_polynomial_map()
Scheme morphism:
  From: 1-d CPR-Fano toric variety covered by 2 affine patches
  To:   2-d CPR-Fano toric variety covered by 3 affine patches
  Defn: Defined on coordinates by sending [u : v] to
        [u : v : v]
```

Finally, here is an example of a fan morphism that cannot be written using homogeneous polynomials. Consider the blowup $O_{\mathbb{P}^1}(2) \to \mathbf{C}^2/\mathbf{Z}_2$. In terms of toric data, this blowup is:

```
sage: A2_Z2 = toric_varieties.A2_Z2()
sage: A2_Z2.fan().rays()
N(1, 0),
N(1, 2)
in 2-d lattice N
```

```
sage: O2_P1 = A2_Z2.resolve(new_rays=[(1,1)])
sage: blowup = O2_P1.hom(identity_matrix(2), A2_Z2)
sage: blowup.as_polynomial_map()
Traceback (most recent call last):
...
TypeError: The fan morphism cannot be written in homogeneous polynomials.
```

If we denote the homogeneous coordinates of $O_{\mathbb{P}^1}(2)$ by x, t, y corresponding to the rays $(1, 2)$, $(1, 1)$, and $(1, 0)$ then the blow-up map is [BB]:

$$f : O_{\mathbb{P}^1}(2) \to \mathbf{C}^2/\mathbf{Z}_2, \quad (x, t, y) \mapsto \left(x\sqrt{t}, y\sqrt{t}\right)$$

which requires square roots.

16.8.1 Fibrations

If a toric morphism is `dominant`, then all fibers over a fixed torus orbit in the base are isomorphic. Hence, studying the fibers is again a combinatorial question and Sage implements additional methods to study such fibrations that are not available otherwise (however, note that you can always `factor()` to pick out the part that is dominant over the image or its closure).

For example, consider the blow-up restricted to one of the two coordinate charts of $O_{\mathbb{P}^1}(2)$

```
sage: O2_P1_chart = ToricVariety(Fan([O2_P1.fan().generating_cones()[0]]))
sage: single_chart = O2_P1_chart.hom(identity_matrix(2), A2_Z2)
sage: single_chart.is_dominant()
True
sage: single_chart.is_surjective()
False

sage: fiber = single_chart.fiber_generic();  fiber
(0-d affine toric variety, 1)
sage: fiber[0].embedding_morphism().as_polynomial_map()
Scheme morphism:
  From: 0-d affine toric variety
  To:   2-d affine toric variety
  Defn: Defined on coordinates by sending [] to
        [1 : 1]
```

The fibers are labeled by torus orbits in the base, that is, cones of the codomain fan. In this case, the fibers over lower-dimensional torus orbits are:

```
sage: A2_Z2_cones = flatten(A2_Z2.fan().cones())
sage: table([('cone', 'dim')] +
....:       [(cone.ambient_ray_indices(), single_chart.fiber_dimension(cone))
....:        for cone in A2_Z2_cones], header_row=True)
  cone     dim
+--------+-----+
  ()       0
  (0,)     0
  (1,)     -1
  (0, 1)   1
```

Lets look closer at the one-dimensional fiber. Although not the case in this example, connected components of fibers over higher-dimensional cones (corresponding to lower-dimensional torus orbits) of the base are often not irreducible. The irreducible components are labeled by the `primitive_preimage_cones()`, which are certain cones of the domain fan that map to the cone in the base that defines the torus orbit:

```
sage: table([('base cone', 'primitive preimage cones')] +
....:       [(cone.ambient_ray_indices(),
....:         single_chart.fan_morphism().primitive_preimage_cones(cone))
....:        for cone in A2_Z2_cones], header_row=True)
  base cone   primitive preimage cones
+-----------+-------------------------------------------------------+
  ()          (0-d cone of Rational polyhedral fan in 2-d lattice N,)
  (0,)        (1-d cone of Rational polyhedral fan in 2-d lattice N,)
  (1,)        ()
  (0, 1)      (1-d cone of Rational polyhedral fan in 2-d lattice N,)
```

The fiber over the trivial cone is the generic fiber that we have already encountered. The interesting fiber is the one over the 2-dimensional cone, which represents the exceptional set of the blow-up in this single coordinate chart. Lets investigate further:

```
sage: exceptional_cones = single_chart.fan_morphism().primitive_preimage_cones(A2_Z2.fan(2)[0])
sage: exceptional_set = single_chart.fiber_component(exceptional_cones[0])
sage: exceptional_set
1-d affine toric variety
sage: exceptional_set.embedding_morphism().as_polynomial_map()
Scheme morphism:
  From: 1-d affine toric variety
  To:   2-d affine toric variety
  Defn: Defined on coordinates by sending [z0] to
        [z0 : 0]
```

So we see that the fiber over this point is an affine line. Together with another affine line in the other coordinate patch, this covers the exceptional $\mathbb{P}^1$ of the blowup $O_{\mathbb{P}^1}(2) \to \mathbf{C}^2/\mathbf{Z}_2$.

Here is an example with higher dimensional varieties involved:

```
sage: A3 = toric_varieties.A(3)
sage: P3 = toric_varieties.P(3)
sage: m = matrix([(2,0,0), (1,1,0), (3, 1, 0)])
sage: phi = A3.hom(m, P3)
sage: phi.as_polynomial_map()
Scheme morphism:
  From: 3-d affine toric variety
  To:   3-d CPR-Fano toric variety covered by 4 affine patches
  Defn: Defined on coordinates by sending [z0 : z1 : z2] to
        [z0^2*z1*z2^3 : z1*z2 : 1 : 1]
sage: phi.fiber_generic()
Traceback (most recent call last):
...
AttributeError: 'SchemeMorphism_fan_toric_variety' object
has no attribute 'fiber_generic'
```

Let's use factorization mentioned above:

```
sage: phi_i, phi_b, phi_s = phi.factor()
```

It is possible to study fibers of the last two morphisms or their composition:

```
sage: phi_d = phi_b * phi_s
sage: phi_d
Scheme morphism:
  From: 3-d affine toric variety
  To:   2-d toric variety covered by 3 affine patches
```

```
  Defn: Defined by sending Rational polyhedral fan in 3-d lattice N to
        Rational polyhedral fan in Sublattice <N(1, 0, 0), N(0, 1, 0)>.
sage: phi_d.as_polynomial_map()
Scheme morphism:
  From: 3-d affine toric variety
  To:   2-d toric variety covered by 3 affine patches
  Defn: Defined on coordinates by sending [z0 : z1 : z2] to
        [z0^2*z1*z2^3 : z1*z2 : 1]
sage: phi_d.codomain().fan().rays()
N( 1,  0, 0),
N( 0,  1, 0),
N(-1, -1, 0)
in Sublattice <N(1, 0, 0), N(0, 1, 0)>
sage: for c in phi_d.codomain().fan():
...       c.ambient_ray_indices()
(1, 2)
(0, 2)
(0, 1)
```

We see that codomain fan of this morphism is a projective plane, which can be verified by

```
sage: phi_d.codomain().fan().is_isomorphic(toric_varieties.P2().fan()) # known bug
True
```

(Unfortunately it cannot be verified correctly until trac ticket #16012 is fixed.)

We now have access to fiber methods:

```
sage: fiber = phi_d.fiber_generic()
sage: fiber
(1-d affine toric variety, 2)
sage: fiber[0].embedding_morphism()
Scheme morphism:
  From: 1-d affine toric variety
  To:   3-d affine toric variety
  Defn: Defined by sending
        Rational polyhedral fan in Sublattice <N(1, 1, -1)> to
        Rational polyhedral fan in 3-d lattice N.
sage: fiber[0].embedding_morphism().as_polynomial_map()
Traceback (most recent call last):
...
NotImplementedError: polynomial representations for
fans with virtual rays are not implemented yet
sage: fiber[0].fan().rays()
Empty collection
in Sublattice <N(1, 1, -1)>
```

We see that generic fibers of this morphism consist of 2 one-dimensional tori each. To see what happens over boundary points we can look at fiber components corresponding to the cones of the domain fan:

```
sage: fm = phi_d.fan_morphism()
sage: for c in flatten(phi_d.domain().fan().cones()):
...       fc, m = phi_d.fiber_component(c, multiplicity=True)
...       print "{} |-> {} ({} rays, multiplicity {}) over {}".format(
...         c.ambient_ray_indices(), fc, fc.fan().nrays(),
...         m, fm.image_cone(c).ambient_ray_indices())
() |-> 1-d affine toric variety (0 rays, multiplicity 2) over ()
(0,) |-> 1-d affine toric variety (0 rays, multiplicity 1) over (0,)
(1,) |-> 2-d affine toric variety (2 rays, multiplicity 1) over (0, 1)
```

```
(2,) |-> 2-d affine toric variety (2 rays, multiplicity 1) over (0, 1)
(0, 1) |-> 1-d affine toric variety (1 rays, multiplicity 1) over (0, 1)
(1, 2) |-> 1-d affine toric variety (1 rays, multiplicity 1) over (0, 1)
(0, 2) |-> 1-d affine toric variety (1 rays, multiplicity 1) over (0, 1)
(0, 1, 2) |-> 0-d affine toric variety (0 rays, multiplicity 1) over (0, 1)
```

Now we see that over one of the coordinate lines of the projective plane we also have one-dimensional tori (but only one in each fiber), while over one of the points fixed by torus action we have two affine planes intersecting along an affine line. An alternative perspective is provided by cones of the codomain fan:

```
sage: for c in flatten(phi_d.codomain().fan().cones()):
...       print "{} connected components over {}, each with {} irreducible components.".format(
...         fm.index(c), c.ambient_ray_indices(),
...         len(fm.primitive_preimage_cones(c)))
2 connected components over (), each with 1 irreducible components.
1 connected components over (0,), each with 1 irreducible components.
None connected components over (1,), each with 0 irreducible components.
None connected components over (2,), each with 0 irreducible components.
None connected components over (1, 2), each with 0 irreducible components.
None connected components over (0, 2), each with 0 irreducible components.
1 connected components over (0, 1), each with 2 irreducible components.
```

REFERENCES:

class `sage.schemes.toric.morphism.`**`SchemeMorphism_fan_fiber_component_toric_variety`**(*toric_morphism*, *defining_cone*)

Bases: `sage.schemes.generic.morphism.SchemeMorphism`

The embedding of a fiber component of a toric morphism.

Note that the embedding map of a fiber component of a toric morphism is itself not a toric morphism!

INPUT:

- `toric_morphism` – a toric morphism. The toric morphism whose fiber component we are describing.
- `defining_cone` – a cone of the fan of the domain of `toric_morphism`. See `fiber_component()` for details.

EXAMPLES:

```
sage: polytope = Polyhedron(
...       [(-3,0,-1,-1),(-1,2,-1,-1),(0,-1,0,0),(0,0,0,1),(0,0,1,0),
...        (0,1,0,0),(0,2,-1,-1),(1,0,0,0),(2,0,-1,-1)])
sage: coarse_fan = FaceFan(polytope, lattice=ToricLattice(4))
sage: P2 = toric_varieties.P2()
sage: proj24 = matrix([[0,0],[1,0],[0,0],[0,1]])
sage: fm = FanMorphism(proj24, coarse_fan, P2.fan(), subdivide=True)
sage: fibration = ToricVariety(fm.domain_fan()).hom(fm, P2)
sage: primitive_cones = fibration.fan_morphism().primitive_preimage_cones(P2.fan(1)[0])
sage: primitive_cone = primitive_cones[0]
sage: fiber_component = fibration.fiber_component(primitive_cone)
sage: fiber_component
2-d toric variety covered by 4 affine patches
sage: fiber_component.embedding_morphism()
Scheme morphism:
  From: 2-d toric variety covered by 4 affine patches
  To:   4-d toric variety covered by 23 affine patches
  Defn: Defined by embedding a fiber component corresponding to
        1-d cone of Rational polyhedral fan in 4-d lattice N.
```

```
sage: fiber_component.embedding_morphism().as_polynomial_map()
Scheme morphism:
  From: 2-d toric variety covered by 4 affine patches
  To:   4-d toric variety covered by 23 affine patches
  Defn: Defined on coordinates by sending [z0 : z1 : z2 : z3] to
        [1 : 1 : 1 : 1 : z1 : 0 : 1 : z0 : 1 : 1 : 1 : z2 : z3 : 1 : 1]
sage: type(fiber_component.embedding_morphism())
<class 'sage.schemes.toric.morphism.SchemeMorphism_fan_fiber_component_toric_variety'>
```

as_polynomial_map()

Express the embedding morphism via homogeneous polynomials.

OUTPUT:

A `SchemeMorphism_polynomial_toric_variety`. Raises a `ValueError` if the morphism cannot be written in terms of homogeneous polynomials.

EXAMPLES:

```
sage: polytope = Polyhedron(
...          [(-3,0,-1,-1),(-1,2,-1,-1),(0,-1,0,0),(0,0,0,1),(0,0,1,0),
...           (0,1,0,0),(0,2,-1,-1),(1,0,0,0),(2,0,-1,-1)])
sage: coarse_fan = FaceFan(polytope, lattice=ToricLattice(4))
sage: P2 = toric_varieties.P2()
sage: proj24 = matrix([[0,0],[1,0],[0,0],[0,1]])
sage: fm = FanMorphism(proj24, coarse_fan, P2.fan(), subdivide=True)
sage: fibration = ToricVariety(fm.domain_fan()).hom(fm, P2)

sage: primitive_cone = Cone([(0, 1, 0, 0)])
sage: f = fibration.fiber_component(primitive_cone).embedding_morphism()
sage: f.as_polynomial_map()
Scheme morphism:
  From: 2-d toric variety covered by 4 affine patches
  To:   4-d toric variety covered by 23 affine patches
  Defn: Defined on coordinates by sending [z0 : z1 : z2 : z3] to
        [1 : 1 : 1 : 1 : z1 : 0 : 1 : z0 : 1 : 1 : 1 : z2 : z3 : 1 : 1]

sage: primitive_cone = Cone([(-1, 2, -1, 0)])
sage: f = fibration.fiber_component(primitive_cone).embedding_morphism()
sage: f.as_polynomial_map()
Traceback (most recent call last):
...
ValueError: The morphism cannot be written using homogeneous polynomials.
```

base_cone()

Return the base cone σ.

The fiber is constant over the base orbit closure $V(\sigma)$.

OUTPUT:

A cone of the base of the toric fibration.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: fc = P1xP1.hom(matrix([[1],[0]]), P1).fiber_component(Cone([(1,0)]))
sage: f = fc.embedding_morphism()
sage: f.defining_cone().rays()
N(1, 0)
```

```
in 2-d lattice N
sage: f.base_cone().rays()
N(1)
in 1-d lattice N
```

defining_cone()

Return the cone corresponding to the fiber torus orbit.

OUTPUT:

A cone of the fan of the total space of the toric fibration.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: fc = P1xP1.hom(matrix([[1],[0]]), P1).fiber_component(Cone([(1,0)]))
sage: f = fc.embedding_morphism()
sage: f.defining_cone().rays()
N(1, 0)
in 2-d lattice N
sage: f.base_cone().rays()
N(1)
in 1-d lattice N
```

pullback_divisor(*divisor*)

Pull back a toric divisor.

INPUT:

- `divisor` – a torus-invariant QQ-Cartier divisor on the codomain of the embedding map.

OUTPUT:

A divisor on the domain of the embedding map (irreducible component of a fiber of a toric morphism) that is isomorphic to the pull-back divisor $f^*(D)$ but with possibly different linearization.

EXAMPLES:

```
sage: A1 = toric_varieties.A1()
sage: fan = Fan([(0,1,2)], [(1,1,0),(1,0,1),(1,-1,-1)]).subdivide(new_rays=[(1,0,0)])
sage: f = ToricVariety(fan).hom(matrix([[1],[0],[0]]), A1)
sage: D = f.domain().divisor([1,1,3,4]); D
V(z0) + V(z1) + 3*V(z2) + 4*V(z3)
sage: fc = f.fiber_component(Cone([(1,1,0)]))
sage: fc.embedding_morphism().pullback_divisor(D)
3*V(z0) + 2*V(z2)
sage: fc = f.fiber_component(Cone([(1,0,0)]))
sage: fc.embedding_morphism().pullback_divisor(D)
-3*V(z0) - 3*V(z1) - V(z2)
```

class `sage.schemes.toric.morphism.`**SchemeMorphism_fan_toric_variety**(*parent*, *fan_morphism*, *check=True*)

Bases: `sage.schemes.generic.morphism.SchemeMorphism`, `sage.categories.morphism.Morphism`

Construct a morphism determined by a fan morphism

> **Warning:** You should not create objects of this class directly. Use the `hom()` method of `toric varieties` instead.

INPUT:

- `parent` – Hom-set whose domain and codomain are toric varieties.
- `fan_morphism` – A morphism of fans whose domain and codomain fans equal the fans of the domain and codomain in the `parent` Hom-set.
- `check` – boolean (optional, default:`True`). Whether to check the input for consistency.

> **Warning:** A fibration is a dominant morphism; if you are interested in these then you have to make sure that your fan morphism is dominant. For example, this can be achieved by `factoring the morphism`. See `SchemeMorphism_fan_toric_variety_dominant` for additional functionality for fibrations.

OUPUT:

A `SchemeMorphism_fan_toric_variety`.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: f = P1.hom(matrix([[1,0]]), P1xP1);  f
Scheme morphism:
  From: 1-d CPR-Fano toric variety covered by 2 affine patches
  To:   2-d CPR-Fano toric variety covered by 4 affine patches
  Defn: Defined by sending Rational polyhedral fan in 1-d lattice N
        to Rational polyhedral fan in 2-d lattice N.
sage: type(f)
<class 'sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety'>
```

Slightly more explicit construction:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: hom_set = P1xP1.Hom(P1)
sage: fm = FanMorphism( matrix(ZZ,[[1],[0]]), P1xP1.fan(), P1.fan() )
sage: hom_set(fm)
Scheme morphism:
  From: 2-d CPR-Fano toric variety covered by 4 affine patches
  To:   1-d CPR-Fano toric variety covered by 2 affine patches
  Defn: Defined by sending Rational polyhedral fan in 2-d lattice N
       to Rational polyhedral fan in 1-d lattice N.

sage: P1xP1.hom(fm, P1)
Scheme morphism:
  From: 2-d CPR-Fano toric variety covered by 4 affine patches
  To:   1-d CPR-Fano toric variety covered by 2 affine patches
  Defn: Defined by sending Rational polyhedral fan in 2-d lattice N
        to Rational polyhedral fan in 1-d lattice N.
```

as_polynomial_map()

Express the morphism via homogeneous polynomials.

OUTPUT:

A `SchemeMorphism_polynomial_toric_variety`. Raises a `TypeError` if the morphism cannot be written in terms of homogeneous polynomials.

EXAMPLES:

```
sage: A1 = toric_varieties.A1()
sage: square = A1.hom(matrix([[2]]), A1)
```

```
sage: square.as_polynomial_map()
Scheme endomorphism of 1-d affine toric variety
  Defn: Defined on coordinates by sending [z] to
        [z^2]

sage: P1 = toric_varieties.P1()
sage: patch = A1.hom(matrix([[1]]), P1)
sage: patch.as_polynomial_map()
Scheme morphism:
  From: 1-d affine toric variety
  To:   1-d CPR-Fano toric variety covered by 2 affine patches
  Defn: Defined on coordinates by sending [z] to
        [z : 1]
```

factor()

Factor `self` into injective * birational * surjective morphisms.

OUTPUT:

- a triple of toric morphisms (ϕ_i, ϕ_b, ϕ_s), such that ϕ_s is surjective, ϕ_b is birational, ϕ_i is injective, and `self` is equal to $\phi_i \circ \phi_b \circ \phi_s$.

The intermediate varieties are universal in the following sense. Let `self` map X to X' and let X_s, X_i sit in between, that is,

$$X \twoheadrightarrow X_s \to X_i \hookrightarrow X'.$$

Then any toric morphism from X coinciding with `self` on the maximal torus factors through X_s and any toric morphism into X' coinciding with `self` on the maximal torus factors through X_i. In particular, X_i is the closure of the image of `self` in X'.

See `factor()` for a description of the toric algorithm.

EXAMPLES:

We map an affine plane into a projective 3-space in such a way, that it becomes "a double cover of a chart of the blow up of one of the coordinate planes":

```
sage: A2 = toric_varieties.A2()
sage: P3 = toric_varieties.P(3)
sage: m = matrix([(2,0,0), (1,1,0)])
sage: phi = A2.hom(m, P3)
sage: phi.as_polynomial_map()
Scheme morphism:
  From: 2-d affine toric variety
  To:   3-d CPR-Fano toric variety covered by 4 affine patches
  Defn: Defined on coordinates by sending [x : y] to
        [x^2*y : y : 1 : 1]

sage: phi.is_surjective(), phi.is_birational(), phi.is_injective()
(False, False, False)
sage: phi_i, phi_b, phi_s = phi.factor()
sage: phi_s.is_surjective(), phi_b.is_birational(), phi_i.is_injective()
(True, True, True)
sage: prod(phi.factor()) == phi
True
```

Double cover (surjective):

```
sage: phi_s.as_polynomial_map()
Scheme morphism:
  From: 2-d affine toric variety
  To:   2-d affine toric variety
  Defn: Defined on coordinates by sending [x : y] to
        [x^2 : y]
```

Blowup chart (birational):

```
sage: phi_b.as_polynomial_map()
Scheme morphism:
  From: 2-d affine toric variety
  To:   2-d toric variety covered by 3 affine patches
  Defn: Defined on coordinates by sending [z0 : z1] to
        [z0*z1 : z1 : 1]
```

Coordinate plane inclusion (injective):

```
sage: phi_i.as_polynomial_map()
Scheme morphism:
  From: 2-d toric variety covered by 3 affine patches
  To:   3-d CPR-Fano toric variety covered by 4 affine patches
  Defn: Defined on coordinates by sending [z0 : z1 : z2] to
        [z0 : z1 : z2 : z2]
```

fan_morphism()

Return the defining fan morphism.

OUTPUT:

A `FanMorphism`.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: f = P1xP1.hom(matrix([[1],[0]]), P1)
sage: f.fan_morphism()
Fan morphism defined by the matrix
[1]
[0]
Domain fan: Rational polyhedral fan in 2-d lattice N
Codomain fan: Rational polyhedral fan in 1-d lattice N
```

is_birational()

Check if `self` is birational.

See `is_birational()` for fan morphisms for a description of the toric algorithm.

OUTPUT:

Boolean. Whether `self` is birational.

EXAMPLES:

```
sage: dP8 = toric_varieties.dP8()
sage: P2 = toric_varieties.P2()
sage: dP8.hom(identity_matrix(2), P2).is_birational()
True

sage: X = toric_varieties.A(2)
sage: Y = ToricVariety(Fan([Cone([(1,0), (1,1)])]))
```

```
sage: m = identity_matrix(2)
sage: f = Y.hom(m, X)
sage: f.is_birational()
True
```

is_bundle()

Check if `self` is a bundle.

See `is_bundle()` for fan morphisms for details.

OUTPUT:

- `True` if `self` is a bundle, `False` otherwise.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: P1xP1.hom(matrix([[1],[0]]), P1).is_bundle()
True
```

is_dominant()

Return whether `self` is dominant.

See `is_dominant()` for fan morphisms for a description of the toric algorithm.

OUTPUT:

Boolean. Whether `self` is a dominant scheme morphism.

EXAMPLES:

```
sage: P1 = toric_varieties.P1()
sage: A1 = toric_varieties.A1()
sage: phi = A1.hom(identity_matrix(1), P1);  phi
Scheme morphism:
  From: 1-d affine toric variety
  To:   1-d CPR-Fano toric variety covered by 2 affine patches
  Defn: Defined by sending Rational polyhedral fan in 1-d lattice N
        to Rational polyhedral fan in 1-d lattice N.
sage: phi.is_dominant()
True
sage: phi.is_surjective()
False
```

is_fibration()

Check if `self` is a fibration.

See `is_fibration()` for fan morphisms for details.

OUTPUT:

- `True` if `self` is a fibration, `False` otherwise.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: P1xP1.hom(matrix([[1],[0]]), P1).is_fibration()
True
```

is_injective()

Check if `self` is injective.

See `is_injective()` for fan morphisms for a description of the toric algorithm.

OUTPUT:

Boolean. Whether `self` is injective.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: P1xP1.hom(matrix([[1],[0]]), P1).is_injective()
False

sage: X = toric_varieties.A(2)
sage: m = identity_matrix(2)
sage: f = X.hom(m, X)
sage: f.is_injective()
True

sage: Y = ToricVariety(Fan([Cone([(1,0), (1,1)])]))
sage: f = Y.hom(m, X)
sage: f.is_injective()
False
```

is_surjective()

Check if `self` is surjective.

See `is_surjective()` for fan morphisms for a description of the toric algorithm.

OUTPUT:

Boolean. Whether `self` is surjective.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: P1xP1.hom(matrix([[1],[0]]), P1).is_surjective()
True

sage: X = toric_varieties.A(2)
sage: m = identity_matrix(2)
sage: f = X.hom(m, X)
sage: f.is_surjective()
True

sage: Y = ToricVariety(Fan([Cone([(1,0), (1,1)])]))
sage: f = Y.hom(m, X)
sage: f.is_surjective()
False
```

pullback_divisor(*divisor*)

Pull back a toric divisor.

INPUT:

- `divisor` – a torus-invariant QQ-Cartier divisor on the codomain of `self`.

OUTPUT:

The pull-back divisor $f^*(D)$.

EXAMPLES:

```
sage: A2_Z2 = toric_varieties.A2_Z2()
sage: A2 = toric_varieties.A2()
sage: f = A2.hom( matrix([[1,0],[1,2]]), A2_Z2)
sage: f.pullback_divisor(A2_Z2.divisor(0))
V(x)

sage: A1 = toric_varieties.A1()
sage: square = A1.hom(matrix([[2]]), A1)
sage: D = A1.divisor(0);  D
V(z)
sage: square.pullback_divisor(D)
2*V(z)
```

class sage.schemes.toric.morphism.**SchemeMorphism_fan_toric_variety_dominant**(*parent*, *fan_morphism*, *check=True*)

Bases: sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety

Construct a morphism determined by a dominant fan morphism.

A dominant morphism is one that is surjective onto a dense subset. In the context of toric morphisms, this means that it is onto the big torus orbit.

> **Warning:** You should not create objects of this class directly. Use the `hom()` method of toric varieties instead.

INPUT:

See SchemeMorphism_fan_toric_variety. The given fan morphism `must be dominant`.

OUPUT:

A SchemeMorphism_fan_toric_variety_dominant.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: dP8 = toric_varieties.dP8()
sage: f = dP8.hom(identity_matrix(2), P2);  f
Scheme morphism:
  From: 2-d CPR-Fano toric variety covered by 4 affine patches
  To:   2-d CPR-Fano toric variety covered by 3 affine patches
  Defn: Defined by sending Rational polyhedral fan in 2-d lattice N
        to Rational polyhedral fan in 2-d lattice N.
sage: type(f)
<class 'sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety_dominant'>
```

fiber_component(*domain_cone*, *multiplicity=False*)

Return a fiber component corresponding to `domain_cone`.

INPUT:

- `domain_cone` – a cone of the domain fan of `self`.
- `multiplicity` (default: `False`) – whether to return the number of fiber components corresponding to `domain_cone` as well.

OUTPUT:

- either X or a tuple (X, n), where X is a toric variety with the embedding morphism into domain of `self` and n is an integer.

Let $\phi : \Sigma \to \Sigma'$ be the fan morphism corresponding to self. Let $\sigma \in \Sigma$ and $\sigma' \in \Sigma'$ be the image_cone() of σ. The fiber over any point of the torus orbit corresponding to σ' consists of n isomorphic connected components with each component being a union of toric varieties intersecting along their torus invariant subvarieties. The latter correspond to preimage_cones() of σ' and X is one of the n components corresponding to σ. The irreducible components correspond to primitive_preimage_cones().

EXAMPLES:

```
sage: polytope = LatticePolytope(
...       [(-3,0,-1,-1),(-1,2,-1,-1),(0,-1,0,0),(0,0,0,1),(0,0,1,0),
...        (0,1,0,0),(0,2,-1,-1),(1,0,0,0),(2,0,-1,-1)])
sage: coarse_fan = FaceFan(polytope)
sage: P2 = toric_varieties.P2()
sage: proj24 = matrix([[0,0],[1,0],[0,0],[0,1]])
sage: fm = FanMorphism(proj24, coarse_fan, P2.fan(), subdivide=True)
sage: fibration = ToricVariety(fm.domain_fan()).hom(fm, P2)
sage: primitive_cones = fibration.fan_morphism().primitive_preimage_cones(P2.fan(1)[0])
sage: primitive_cone = primitive_cones[0]
sage: fibration.fiber_component(primitive_cone)
2-d toric variety covered by 4 affine patches
sage: fibration.fiber_component(primitive_cone, True)
(2-d toric variety covered by 4 affine patches, 1)

sage: for primitive_cone in primitive_cones:
...       print fibration.fiber_component(primitive_cone)
2-d toric variety covered by 4 affine patches
2-d toric variety covered by 3 affine patches
2-d toric variety covered by 3 affine patches
```

fiber_dimension(*codomain_cone*)

Return the dimension of the fiber over a particular torus orbit in the base.

INPUT:

- codomain_cone – a cone σ of the codomain, specifying a torus orbit $O(\sigma)$.

OUTPUT:

An integer. The dimension of the fiber over the torus orbit corresponding to codomain_cone. If the fiber is the empty set, -1 is returned. Note that all fibers over this torus orbit are isomorphic, and therefore have the same dimension.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: f = P1xP1.hom(matrix([[1],[0]]), P1)
sage: f.fiber_dimension(P1.fan(0)[0])
1
sage: f.fiber_dimension(P1.fan(1)[0])
1
sage: f.fiber_dimension(P1.fan(1)[1])
1
```

Here is a more complicated example that is not a flat fibration:

```
sage: A2_Z2 = toric_varieties.A2_Z2()
sage: O2_P1 = A2_Z2.resolve(new_rays=[(1,1)])
sage: blowup = O2_P1.hom(identity_matrix(2), A2_Z2)
sage: blowup.fiber_dimension(A2_Z2.fan(0)[0])
0
```

```
sage: blowup.fiber_dimension(A2_Z2.fan(1)[0])
0
sage: blowup.fiber_dimension(A2_Z2.fan(2)[0])
1
```

This corresponds to the three different fibers:

```
sage: blowup.fiber_generic()
(0-d affine toric variety, 1)
sage: blowup.fiber_component(Cone([(1,0)]))
0-d affine toric variety
sage: blowup.fiber_component(Cone([(1,1)]))
1-d toric variety covered by 2 affine patches
```

fiber_generic()

Return the generic fiber.

OUTPUT:

- a tuple (X, n), where X is a `toric variety` with the embedding morphism into domain of `self` and n is an integer.

The fiber over the base point with homogeneous coordinates $[1 : 1 : \cdots : 1]$ consists of n disjoint toric varieties isomorphic to X. Note that fibers of a dominant toric morphism are isomorphic over all points of a fixed torus orbit of its codomain, in particular over all points of the maximal torus, so it makes sense to talk about "the generic" fiber.

The embedding of X is a toric morphism with the `domain_fan()` being the `kernel_fan()` of the defining fan morphism. By contrast, embeddings of fiber components over lower-dimensional torus orbits of the image are not toric morphisms. Use `fiber_component()` for the latter (non-generic) fibers.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: fiber = P1xP1.hom(matrix([[1],[0]]), P1).fiber_generic()
sage: fiber
(1-d toric variety covered by 2 affine patches, 1)
sage: f = fiber[0].embedding_morphism();  f
Scheme morphism:
  From: 1-d toric variety covered by 2 affine patches
  To:   2-d CPR-Fano toric variety covered by 4 affine patches
  Defn: Defined by sending Rational polyhedral fan in Sublattice <N(0, 1)> to
        Rational polyhedral fan in 2-d lattice N.
sage: f.as_polynomial_map()
Scheme morphism:
  From: 1-d toric variety covered by 2 affine patches
  To:   2-d CPR-Fano toric variety covered by 4 affine patches
  Defn: Defined on coordinates by sending [z0 : z1] to
        [1 : 1 : z0 : z1]

sage: A1 = toric_varieties.A1()
sage: fan = Fan([(0,1,2)], [(1,1,0),(1,0,1),(1,-1,-1)])
sage: fan = fan.subdivide(new_rays=[(1,0,0)])
sage: f = ToricVariety(fan).hom(matrix([[1],[0],[0]]), A1)
sage: f.fiber_generic()
(2-d affine toric variety, 1)
sage: _[0].fan().generating_cones()
(0-d cone of Rational polyhedral fan in Sublattice <N(0, 1, 0), N(0, 0, 1)>,)
```

fiber_graph (*codomain_cone*)

Return the fiber over a given torus orbit in the codomain.

INPUT:

- `codomain_cone` – a cone σ of the codomain, specifying a torus orbit $O(\sigma)$.

OUTPUT:

A graph whose nodes are the irreducible components of a connected component of the fiber over a point of $O(\sigma)$. If two irreducible components intersect, the corresponding nodes of the graph are joined by an edge. Note that irreducible components do not have to be of the same dimension.

See also:

`fiber_component()`.

EXAMPLES:

```
sage: polytope = Polyhedron(
...       [(-3,0,-1,-1),(-1,2,-1,-1),(0,-1,0,0),(0,0,0,1),(0,0,1,0),
...        (0,1,0,0),(0,2,-1,-1),(1,0,0,0),(2,0,-1,-1)])
sage: coarse_fan = FaceFan(polytope, lattice=ToricLattice(4))

sage: P2 = toric_varieties.P2()
sage: proj34 = block_matrix(2,1,[zero_matrix(2,2), identity_matrix(2)])
sage: fm = FanMorphism(proj34, coarse_fan, P2.fan(), subdivide=True)
sage: fibration = ToricVariety(fm.domain_fan()).hom(fm, P2)

sage: fibration.fiber_graph( P2.fan(0)[0] )
Graph on 1 vertex
sage: for c1 in P2.fan(1):
...       fibration.fiber_graph(c1)
Graph on 1 vertex
Graph on 1 vertex
Graph on 4 vertices

sage: fibration.fiber_graph(P2.fan(1)[2]).get_vertices()
{0: 2-d toric variety covered by 4 affine patches,
 1: 2-d toric variety covered by 3 affine patches,
 2: 2-d toric variety covered by 3 affine patches,
 3: 2-d toric variety covered by 4 affine patches}

sage: fibration
Scheme morphism:
  From: 4-d toric variety covered by 18 affine patches
  To:   2-d CPR-Fano toric variety covered by 3 affine patches
  Defn: Defined by sending Rational polyhedral fan in 4-d lattice N
        to Rational polyhedral fan in 2-d lattice N.
```

class `sage.schemes.toric.morphism.`**SchemeMorphism_orbit_closure_toric_variety**(*parent*, *defining_cone*, *ray_map*)

Bases: `sage.schemes.generic.morphism.SchemeMorphism`, `sage.categories.morphism.Morphism`

The embedding of an orbit closure.

INPUT:

- `parent` – the parent homset.
- `defining_cone` – the defining cone.

- ray_map – a dictionary {ambient ray generator: orbit ray generator}. Note that the image of the ambient ray generator is not necessarily primitive.

Warning: You should not create objects of this class directly. Use the orbit_closure() method of toric varieties instead.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: H = P1xP1.fan(1)[0]
sage: V = P1xP1.orbit_closure(H)
sage: V.embedding_morphism()
Scheme morphism:
  From: 1-d toric variety covered by 2 affine patches
  To:   2-d CPR-Fano toric variety covered by 4 affine patches
  Defn: Defined by embedding the torus closure associated to the 1-d
        cone of Rational polyhedral fan in 2-d lattice N.
```

TESTS:

```
sage: V.embedding_morphism()._reverse_ray_map()
{N(-1): 3, N(1): 2}
sage: V.embedding_morphism()._defining_cone
1-d cone of Rational polyhedral fan in 2-d lattice N
```

as_polynomial_map()

Express the morphism via homogeneous polynomials.

OUTPUT:

A SchemeMorphism_polynomial_toric_variety. Raises a TypeError if the morphism cannot be written in terms of homogeneous polynomials.

The defining polynomials are not necessarily unique. There are choices if multiple ambient space ray generators project to the same orbit ray generator, and one such choice is made implicitly. The orbit embedding can be written as a polynomial map if and only if each primitive orbit ray generator is the image of at least one primitive ray generator of the ambient toric variety.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: V = P2.orbit_closure(P2.fan(1)[0]);  V
1-d toric variety covered by 2 affine patches
sage: V.embedding_morphism().as_polynomial_map()
Scheme morphism:
  From: 1-d toric variety covered by 2 affine patches
  To:   2-d CPR-Fano toric variety covered by 3 affine patches
  Defn: Defined on coordinates by sending [z0 : z1] to
        [0 : z1 : z0]
```

If the toric variety is singular, then some orbit closure embeddings cannot be written with homogeneous polynomials:

```
sage: P2_112 = toric_varieties.P2_112()
sage: P1 = P2_112.orbit_closure(Cone([(1,0)]))
sage: P1.embedding_morphism().as_polynomial_map()
Traceback (most recent call last):
...
TypeError: The embedding cannot be written with homogeneous polynomials.
```

defining_cone()

Return the cone corresponding to the torus orbit.

OUTPUT:

A cone of the fan of the ambient toric variety.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: cone = P2.fan(1)[0]
sage: P1 = P2.orbit_closure(cone)
sage: P1.embedding_morphism().defining_cone()
1-d cone of Rational polyhedral fan in 2-d lattice N
sage: _ is cone
True
```

pullback_divisor(*divisor*)

Pull back a toric divisor.

INPUT:

- `divisor` – a torus-invariant QQ-Cartier divisor on the codomain of the embedding map.

OUTPUT:

A divisor on the domain of the embedding map (the orbit closure) that is isomorphic to the pull-back divisor $f^*(D)$ but with possibly different linearization.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P1 = P2.orbit_closure(P2.fan(1)[0])
sage: f = P1.embedding_morphism()
sage: D = P2.divisor([1,2,3]); D
V(x) + 2*V(y) + 3*V(z)
sage: f.pullback_divisor(D)
4*V(z0) + 2*V(z1)
```

class `sage.schemes.toric.morphism.`**SchemeMorphism_point_toric_field**(*X*, *coordinates*, *check=True*)

Bases: `sage.schemes.generic.morphism.SchemeMorphism_point`, `sage.categories.morphism.Morphism`

A point of a toric variety determined by homogeneous coordinates in a field.

> **Warning:** You should not create objects of this class directly. Use the `hom()` method of `toric varieties` instead.

INPUT:

- `X` – toric variety or subscheme of a toric variety.
- `coordinates` – list of coordinates in the base field of `X`.
- `check` – if `True` (default), the input will be checked for correctness.

OUTPUT:

A `SchemeMorphism_point_toric_field`.

TESTS:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1(1,2,3,4)
[1 : 2 : 3 : 4]
```

class sage.schemes.toric.morphism.**SchemeMorphism_polynomial_toric_variety**(*parent*, *polynomials*, *check=True*)

Bases: sage.schemes.generic.morphism.SchemeMorphism_polynomial, sage.categories.morphism.Morphism

A morphism determined by homogeneous polynomials.

> **Warning:** You should not create objects of this class directly. Use the hom() method of toric varieties instead.

INPUT:

Same as for SchemeMorphism_polynomial.

OUPUT:

A SchemeMorphism_polynomial_toric_variety.

TESTS:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1.inject_variables()
Defining s, t, x, y
sage: P1 = P1xP1.subscheme(s-t)
sage: H = P1xP1.Hom(P1)
sage: import sage.schemes.toric.morphism as MOR
sage: MOR.SchemeMorphism_polynomial_toric_variety(H, [s, s, x, y])
Scheme morphism:
  From: 2-d CPR-Fano toric variety covered by 4 affine patches
  To:   Closed subscheme of 2-d CPR-Fano toric variety
        covered by 4 affine patches defined by:
  s - t
  Defn: Defined on coordinates by sending [s : t : x : y] to
        [s : s : x : y]
```

as_fan_morphism()

Express the morphism as a map defined by a fan morphism.

OUTPUT:

A SchemeMorphism_polynomial_toric_variety. Raises a TypeError if the morphism cannot be written in such a way.

EXAMPLES:

```
sage: A1.<z> = toric_varieties.A1()
sage: P1 = toric_varieties.P1()
sage: patch = A1.hom([1,z], P1)
sage: patch.as_fan_morphism()
Traceback (most recent call last):
...
NotImplementedError: expressing toric morphisms as fan morphisms is
not implemented yet!
```

16.9 Weierstrass form of a toric elliptic curve.

There are 16 reflexive polygons in the plane, see `ReflexivePolytopes()`. Each of them defines a toric Fano variety. And each of them has a unique crepant resolution to a smooth toric surface [CLSsurfaces] by subdividing the face fan. An anticanonical hypersurface defines an elliptic curve in this ambient space, which we call a toric elliptic curve. The purpose of this module is to write an anticanonical hypersurface equation in the short Weierstrass form $y^2 = x^3 + fx + g$. This works over any base ring as long as its characteristic $\neq 2, 3$.

For an analogous treatment of elliptic curves defined as complete intersection in higher dimensional toric varieties, see the module `weierstrass_higher`.

Technically, this module computes the Weierstrass form of the Jacobian of the elliptic curve. This is why you will never have to specify the origin (or zero section) in the following.

It turns out [VolkerBraun] that the anticanonical hypersurface equation of any one of the above 16 toric surfaces is a specialization (that is, set one or more of the coefficients to zero) of the following three cases. In inhomogeneous coordinates, they are

- Cubic in $\mathbb{P}^2$:

$$\begin{aligned} p(x, y) = a_{30}x^3 + a_{21}x^2y + a_{12}xy^2 + a_{03}y^3 + a_{20}x^2 + \\ a_{11}xy + a_{02}y^2 + a_{10}x + a_{01}y + a_{00} \end{aligned}$$

- Biquadric in $\mathbb{P}^1 \times \mathbb{P}^1$:

$$\begin{aligned} p(x, y) = a_{22}x^2y^2 + a_{21}x^2y + a_{20}x^2 + a_{12}xy^2 + \\ a_{11}xy + xa_{10} + y^2a_{02} + ya_{01} + a_{00} \end{aligned}$$

- Anticanonical hypersurface in weighted projective space $\mathbb{P}^2[1, 1, 2]$:

$$\begin{aligned} p(x, y) = a_{40}x^4 + a_{30}x^3 + a_{21}x^2y + a_{20}x^2 + \\ a_{11}xy + a_{02}y^2 + a_{10}x + a_{01}y + a_{00} \end{aligned}$$

EXAMPLES:

The main functionality is provided by `WeierstrassForm()`, which brings each of the above hypersurface equations into Weierstrass form:

```
sage: R.<x,y> = QQ[]
sage: cubic = x^3 + y^3 + 1
sage: WeierstrassForm(cubic)
(0, -27/4)
sage: WeierstrassForm(x^4 + y^2 + 1)
(-4, 0)
sage: WeierstrassForm(x^2*y^2 + x^2 + y^2 + 1)
(-16/3, 128/27)
```

Only the affine span of the Newton polytope of the polynomial matters. For example:

```
sage: R.<x,y,z> = QQ[]
sage: WeierstrassForm(x^3 + y^3 + z^3)
(0, -27/4)
sage: WeierstrassForm(x * cubic)
(0, -27/4)
```

This allows you to work with either homogeneous or inhomogeneous variables. For exmple, here is the del Pezzo surface of degree 8:

```
sage: dP8 = toric_varieties.dP8()
sage: dP8.inject_variables()
Defining t, x, y, z
sage: WeierstrassForm(x*y^2 + y^2*z + t^2*x^3 + t^2*z^3)
(-3, -2)
sage: WeierstrassForm(x*y^2 + y^2 + x^3 + 1)
(-3, -2)
```

By specifying only certain variables we can compute the Weierstrass form over the polynomial ring generated by the remaining variables. For example, here is a cubic over $\mathbf{Q}[a]$

```
sage: R.<a, x, y, z> = QQ[]
sage: cubic = x^3 + a*y^3 + a^2*z^3
sage: WeierstrassForm(cubic, variables=[x,y,z])
(0, -27/4*a^6)
```

TESTS:

```
sage: R.<f, g, x, y> = QQ[]
sage: cubic = -y^2 + x^3 + f*x + g
sage: WeierstrassForm(cubic, variables=[x,y])
(f, g)
```

REFERENCES:

`sage.schemes.toric.weierstrass.`**`Discriminant`**(*polynomial*, *variables=None*)

The discriminant of the elliptic curve.

INPUT:

See `WeierstrassForm()` for how to specify the input polynomial(s) and variables.

OUTPUT:

The discriminant of the elliptic curve.

EXAMPLES:

```
sage: from sage.schemes.toric.weierstrass import Discriminant
sage: R.<x, y, z> = QQ[]
sage: Discriminant(x^3+y^3+z^3)
19683/16
sage: Discriminant(x*y*z)
0
sage: R.<w,x,y,z> = QQ[]
sage: quadratic1 = w^2+x^2+y^2
sage: quadratic2 = z^2 + w*x
sage: Discriminant([quadratic1, quadratic2])
-1/16
```

`sage.schemes.toric.weierstrass.`**`Newton_polygon_embedded`**(*polynomial*, *variables*)

Embed the Newton polytope of the polynomial in one of the three maximal reflexive polygons.

This function is a helper for `WeierstrassForm()`

INPUT:

Same as `WeierstrassForm()` with only a single polynomial passed.

OUTPUT:

A tuple $(\Delta, P, (x, y))$ where

•Δ is the Newton polytope of `polynomial`.

•$P(x, y)$ equals the input `polynomial` but with redefined variables such that its Newton polytope is Δ.

EXAMPLES:

```
sage: from sage.schemes.toric.weierstrass import Newton_polygon_embedded
sage: R.<x,y,z> = QQ[]
sage: cubic = x^3 + y^3 + z^3
sage: Newton_polygon_embedded(cubic, [x,y,z])
(A 2-dimensional lattice polytope in ZZ^3 with 3 vertices,
 x^3 + y^3 + 1,
 (x, y))

sage: R.<a, x,y,z> = QQ[]
sage: cubic = x^3 + a*y^3 + a^2*z^3
sage: Newton_polygon_embedded(cubic, variables=[x,y,z])
(A 2-dimensional lattice polytope in ZZ^3 with 3 vertices,
 a^2*x^3 + y^3 + a,
 (x, y))

sage: R.<s,t,x,y> = QQ[]
sage: biquadric = (s+t)^2 * (x+y)^2
sage: Newton_polygon_embedded(biquadric, [s,t,x,y])
(A 2-dimensional lattice polytope in ZZ^4 with 4 vertices,
 s^2*t^2 + 2*s^2*t + 2*s*t^2 + s^2 + 4*s*t + t^2 + 2*s + 2*t + 1,
 (s, t))
```

`sage.schemes.toric.weierstrass.`**`Newton_polytope_vars_coeffs`**(*polynomial*, *variables*)

Return the Newton polytope in the given variables.

INPUT:

See `WeierstrassForm()` for how to specify the input polynomial and variables.

OUTPUT:

A tuple containing of the affine span of the Netwton polytope and a dictionary with keys the integral values of the Newton polytope and values the corresponding coefficient of `polynomial`.

EXAMPLES:

```
sage: from sage.schemes.toric.weierstrass import Newton_polytope_vars_coeffs
sage: R.<x,y,z,a30,a21,a12,a03,a20,a11,a02,a10,a01,a00> = QQ[]
sage: p = (a30*x^3 + a21*x^2*y + a12*x*y^2 + a03*y^3 + a20*x^2*z +
....:      a11*x*y*z + a02*y^2*z + a10*x*z^2 + a01*y*z^2 + a00*z^3)
sage: p_data = Newton_polytope_vars_coeffs(p, [x,y,z]);  p_data
{(0, 0, 3): a00,
 (0, 1, 2): a01,
 (0, 2, 1): a02,
 (0, 3, 0): a03,
 (1, 0, 2): a10,
 (1, 1, 1): a11,
 (1, 2, 0): a12,
 (2, 0, 1): a20,
 (2, 1, 0): a21,
 (3, 0, 0): a30}
```

```
sage: from sage.geometry.polyhedron.ppl_lattice_polytope import LatticePolytope_PPL
sage: polytope = LatticePolytope_PPL(p_data.keys());  polytope
A 2-dimensional lattice polytope in ZZ^3 with 3 vertices
sage: polytope.vertices()
((0, 0, 3), (3, 0, 0), (0, 3, 0))
sage: polytope.embed_in_reflexive_polytope()
The map A*x+b with A=
[-1 -1]
[ 0  1]
[ 1  0]
b =
(3, 0, 0)
```

sage.schemes.toric.weierstrass.**WeierstrassForm**(*polynomial*, *variables=None*, *transformation=False*)

Return the Weierstrass form of an elliptic curve inside either inside a toric surface or $\mathbb{P}^3$.

INPUT:

- `polynomial` – either a polynomial or a list of polynomials defining the elliptic curve. A single polynomial can be either a cubic, a biquadric, or the hypersurface in $\mathbb{P}^2[1,1,2]$. In this case the equation need not be in any standard form, only its Newton polyhedron is used. If two polynomials are passed, they must both be quadratics in $\mathbb{P}^3$.
- `variables` – a list of variables of the parent polynomial ring or `None` (default). In the latter case, all variables are taken to be polynomial ring variables. If a subset of polynomial ring variables are given, the Weierstrass form is determined over the function field generated by the remaining variables.
- `transformation` – boolean (default: `False`). Whether to return the new variables that bring `polynomial` into Weierstrass form.

OUTPUT:

The pair of coefficients (f,g) of the Weierstrass form $y^2 = x^3 + fx + g$ of the hypersurface equation.

If `transformation=True`, a triple (X,Y,Z) of polynomials defining a rational map of the toric hypersurface or complete intersection in $\mathbb{P}^3$ to its Weierstrass form in $\mathbb{P}^2[2,3,1]$ is returned. That is, the triple satisfies

$$Y^2 = X^3 + fXZ^4 + gZ^6$$

when restricted to the toric hypersurface or complete intersection.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: cubic = x^3 + y^3 + z^3
sage: f, g = WeierstrassForm(cubic);  (f, g)
(0, -27/4)
```

Same in inhomogeneous coordinates:

```
sage: R.<x,y> = QQ[]
sage: cubic = x^3 + y^3 + 1
sage: f, g = WeierstrassForm(cubic);  (f, g)
(0, -27/4)

sage: X,Y,Z = WeierstrassForm(cubic, transformation=True);  (X,Y,Z)
(-x^3*y^3 - x^3 - y^3,
 1/2*x^6*y^3 - 1/2*x^3*y^6 - 1/2*x^6 + 1/2*y^6 + 1/2*x^3 - 1/2*y^3,
 x*y)
```

Note that plugging in $[X : Y : Z]$ to the Weierstrass equation is a complicated polynomial, but contains the hypersurface equation as a factor:

```
sage: -Y^2 + X^3 + f*X*Z^4 + g*Z^6
-1/4*x^12*y^6 - 1/2*x^9*y^9 - 1/4*x^6*y^12 + 1/2*x^12*y^3
- 7/2*x^9*y^6 - 7/2*x^6*y^9 + 1/2*x^3*y^12 - 1/4*x^12 - 7/2*x^9*y^3
- 45/4*x^6*y^6 - 7/2*x^3*y^9 - 1/4*y^12 - 1/2*x^9 - 7/2*x^6*y^3
- 7/2*x^3*y^6 - 1/2*y^9 - 1/4*x^6 + 1/2*x^3*y^3 - 1/4*y^6
sage: cubic.divides(-Y^2 + X^3 + f*X*Z^4 + g*Z^6)
True
```

Only the affine span of the Newton polytope of the polynomial matters. For example:

```
sage: R.<x,y,z> = QQ[]
sage: cubic = x^3 + y^3 + z^3
sage: WeierstrassForm(cubic.subs(z=1))
(0, -27/4)
sage: WeierstrassForm(x * cubic)
(0, -27/4)
```

This allows you to work with either homogeneous or inhomogeneous variables. For example, here is the del Pezzo surface of degree 8:

```
sage: dP8 = toric_varieties.dP8()
sage: dP8.inject_variables()
Defining t, x, y, z
sage: WeierstrassForm(x*y^2 + y^2*z + t^2*x^3 + t^2*z^3)
(-3, -2)
sage: WeierstrassForm(x*y^2 + y^2 + x^3 + 1)
(-3, -2)
```

By specifying only certain variables we can compute the Weierstrass form over the function field generated by the remaining variables. For example, here is a cubic over $\mathbf{Q}[a]$

```
sage: R.<a, x,y,z> = QQ[]
sage: cubic = x^3 + a*y^3 + a^2*z^3
sage: WeierstrassForm(cubic, variables=[x,y,z])
(0, -27/4*a^6)
```

TESTS:

```
sage: for P in ReflexivePolytopes(2):
....:     S = ToricVariety(FaceFan(P))
....:     p = sum((-S.K()).sections_monomials())
....:     print WeierstrassForm(p)
(-25/48, -1475/864)
(-97/48, 17/864)
(-25/48, -611/864)
(-27/16, 27/32)
(47/48, -199/864)
(47/48, -71/864)
(5/16, -21/32)
(23/48, -235/864)
(-1/48, 161/864)
(-25/48, 253/864)
(5/16, 11/32)
(-25/48, 125/864)
(-67/16, 63/32)
(-11/16, 3/32)
(-241/48, 3689/864)
(215/48, -5291/864)
```

sage.schemes.toric.weierstrass.**WeierstrassForm_P1xP1**(*biquadric*, *variables=None*)

Bring a biquadric into Weierstrass form

Input/output is the same as WeierstrassForm(), except that the input polynomial must be a standard biquadric in $\mathbb{P}^2$,

$$p(x,y) = a_{40}x^4 + a_{30}x^3 + a_{21}x^2y + a_{20}x^2 + a_{11}xy + a_{02}y^2 + a_{10}x + a_{01}y + a_{00}$$

EXAMPLES:

```
sage: from sage.schemes.toric.weierstrass import WeierstrassForm_P1xP1
sage: R.<x0,x1,y0,y1>= QQ[]
sage: biquadric = ( x0^2*y0^2 + x0*x1*y0^2*2 + x1^2*y0^2*3
....:     + x0^2*y0*y1*4 + x0*x1*y0*y1*5 + x1^2*y0*y1*6
....:     + x0^2*y1^2*7 + x0*x1*y1^2*8 )
sage: WeierstrassForm_P1xP1(biquadric, [x0, x1, y0, y1])
(1581/16, -3529/32)
```

Since there is no $x_1^2 y_1^2$ term in biquadric, we can dehomogenize it and get a cubic:

```
sage: from sage.schemes.toric.weierstrass import WeierstrassForm_P2
sage: WeierstrassForm_P2(biquadric(x0=1,y0=1))
(1581/16, -3529/32)
```

TESTS:

```
sage: R.<x0,x1,y0,y1,a00,a10,a20,a01,a11,a21,a02,a12,a22> = QQ[]
sage: biquadric = ( x0^2*y0^2*a00 + x0*x1*y0^2*a10 + x1^2*y0^2*a20
....:     + x0^2*y0*y1*a01 + x0*x1*y0*y1*a11 + x1^2*y0*y1*a21
....:     + x0^2*y1^2*a02 + x0*x1*y1^2*a12 )
sage: WeierstrassForm_P1xP1(biquadric, [x0, x1, y0, y1])
(-1/48*a11^4 + 1/6*a01*a11^2*a21 - 1/3*a01^2*a21^2
 + 1/6*a20*a11^2*a02 + 1/3*a20*a01*a21*a02 - 1/2*a10*a11*a21*a02
 + a00*a21^2*a02 - 1/3*a20^2*a02^2 - 1/2*a20*a01*a11*a12
 + 1/6*a10*a11^2*a12 + 1/3*a10*a01*a21*a12 - 1/2*a00*a11*a21*a12
 + 1/3*a10*a20*a02*a12 - 1/3*a10^2*a12^2 + a00*a20*a12^2, 1/864*a11^6
 - 1/72*a01*a11^4*a21 + 1/18*a01^2*a11^2*a21^2 - 2/27*a01^3*a21^3
 - 1/72*a20*a11^4*a02 + 1/36*a20*a01*a11^2*a21*a02
 + 1/24*a10*a11^3*a21*a02 + 1/9*a20*a01^2*a21^2*a02
 - 1/6*a10*a01*a11*a21^2*a02 - 1/12*a00*a11^2*a21^2*a02
 + 1/3*a00*a01*a21^3*a02 + 1/18*a20^2*a11^2*a02^2
 + 1/9*a20^2*a01*a21*a02^2 - 1/6*a10*a20*a11*a21*a02^2
 + 1/4*a10^2*a21^2*a02^2 - 2/3*a00*a20*a21^2*a02^2 - 2/27*a20^3*a02^3
 + 1/24*a20*a01*a11^3*a12 - 1/72*a10*a11^4*a12
 - 1/6*a20*a01^2*a11*a21*a12 + 1/36*a10*a01*a11^2*a21*a12
 + 1/24*a00*a11^3*a21*a12 + 1/9*a10*a01^2*a21^2*a12
 - 1/6*a00*a01*a11*a21^2*a12 - 1/6*a20^2*a01*a11*a02*a12
 + 1/36*a10*a20*a11^2*a02*a12 + 1/18*a10*a20*a01*a21*a02*a12
 - 1/6*a10^2*a11*a21*a02*a12 + 5/6*a00*a20*a11*a21*a02*a12
 - 1/6*a00*a10*a21^2*a02*a12 + 1/9*a10*a20^2*a02^2*a12
 + 1/4*a20^2*a01^2*a12^2 - 1/6*a10*a20*a01*a11*a12^2
 + 1/18*a10^2*a11^2*a12^2 - 1/12*a00*a20*a11^2*a12^2
 + 1/9*a10^2*a01*a21*a12^2 - 1/6*a00*a20*a01*a21*a12^2
 - 1/6*a00*a10*a11*a21*a12^2 + 1/4*a00^2*a21^2*a12^2
 + 1/9*a10^2*a20*a02*a12^2 - 2/3*a00*a20^2*a02*a12^2
 - 2/27*a10^3*a12^3 + 1/3*a00*a10*a20*a12^3)
```

```
sage: _ == WeierstrassForm_P1xP1(biquadric.subs(x1=1,y1=1), [x0, y0])
True
```

sage.schemes.toric.weierstrass.**WeierstrassForm_P2**(*polynomial*, *variables=None*)

Bring a cubic into Weierstrass form.

Input/output is the same as `WeierstrassForm()`, except that the input polynomial must be a standard cubic in $\mathbb{P}^2$,

$$p(x,y) = a_{30}x^3 + a_{21}x^2y + a_{12}xy^2 + a_{03}y^3 + a_{20}x^2 + a_{11}xy + a_{02}y^2 + a_{10}x + a_{01}y + a_{00}$$

EXAMPLES:

```
sage: from sage.schemes.toric.weierstrass import WeierstrassForm_P2
sage: R.<x,y,z> = QQ[]
sage: WeierstrassForm_P2( x^3+y^3+z^3 )
(0, -27/4)

sage: R.<x,y,z, a,b> = QQ[]
sage: WeierstrassForm_P2( -y^2*z+x^3+a*x*z^2+b*z^3, [x,y,z] )
(a, b)
```

TESTS:

```
sage: R.<x,y,z,a30,a21,a12,a03,a20,a11,a02,a10,a01,a00> = QQ[]
sage: p = ( a30*x^3 + a21*x^2*y + a12*x*y^2 + a03*y^3 + a20*x^2*z +
....:       a11*x*y*z + a02*y^2*z + a10*x*z^2 + a01*y*z^2 + a00*z^3 )
sage: WeierstrassForm_P2(p, [x,y,z])
(-1/48*a11^4 + 1/6*a20*a11^2*a02 - 1/3*a20^2*a02^2 - 1/2*a03*a20*a11*a10
 + 1/6*a12*a11^2*a10 + 1/3*a12*a20*a02*a10 - 1/2*a21*a11*a02*a10
 + a30*a02^2*a10 - 1/3*a12^2*a10^2 + a21*a03*a10^2 + a03*a20^2*a01
 - 1/2*a12*a20*a11*a01 + 1/6*a21*a11^2*a01 + 1/3*a21*a20*a02*a01
 - 1/2*a30*a11*a02*a01 + 1/3*a21*a12*a10*a01 - 3*a30*a03*a10*a01
 - 1/3*a21^2*a01^2 + a30*a12*a01^2 + a12^2*a20*a00 - 3*a21*a03*a20*a00
 - 1/2*a21*a12*a11*a00 + 9/2*a30*a03*a11*a00 + a21^2*a02*a00
 - 3*a30*a12*a02*a00,
 1/864*a11^6 - 1/72*a20*a11^4*a02 + 1/18*a20^2*a11^2*a02^2
 - 2/27*a20^3*a02^3 + 1/24*a03*a20*a11^3*a10 - 1/72*a12*a11^4*a10
 - 1/6*a03*a20^2*a11*a02*a10 + 1/36*a12*a20*a11^2*a02*a10
 + 1/24*a21*a11^3*a02*a10 + 1/9*a12*a20^2*a02^2*a10
 - 1/6*a21*a20*a11*a02^2*a10 - 1/12*a30*a11^2*a02^2*a10
 + 1/3*a30*a20*a02^3*a10 + 1/4*a03^2*a20^2*a10^2
 - 1/6*a12*a03*a20*a11*a10^2 + 1/18*a12^2*a11^2*a10^2
 - 1/12*a21*a03*a11^2*a10^2 + 1/9*a12^2*a20*a02*a10^2
 - 1/6*a21*a03*a20*a02*a10^2 - 1/6*a21*a12*a11*a02*a10^2
 + a30*a03*a11*a02*a10^2 + 1/4*a21^2*a02^2*a10^2
 - 2/3*a30*a12*a02^2*a10^2 - 2/27*a12^3*a10^3 + 1/3*a21*a12*a03*a10^3
 - a30*a03^2*a10^3 - 1/12*a03*a20^2*a11^2*a01 + 1/24*a12*a20*a11^3*a01
 - 1/72*a21*a11^4*a01 + 1/3*a03*a20^3*a02*a01 - 1/6*a12*a20^2*a11*a02*a01
 + 1/36*a21*a20*a11^2*a02*a01 + 1/24*a30*a11^3*a02*a01
 + 1/9*a21*a20^2*a02^2*a01 - 1/6*a30*a20*a11*a02^2*a01
 - 1/6*a12*a03*a20^2*a10*a01 - 1/6*a12^2*a20*a11*a10*a01
 + 5/6*a21*a03*a20*a11*a10*a01 + 1/36*a21*a12*a11^2*a10*a01
 - 3/4*a30*a03*a11^2*a10*a01 + 1/18*a21*a12*a20*a02*a10*a01
 - 3/2*a30*a03*a20*a02*a10*a01 - 1/6*a21^2*a11*a02*a10*a01
 + 5/6*a30*a12*a11*a02*a10*a01 - 1/6*a30*a21*a02^2*a10*a01
 + 1/9*a21*a12^2*a10^2*a01 - 2/3*a21^2*a03*a10^2*a01
 + a30*a12*a03*a10^2*a01 + 1/4*a12^2*a20^2*a01^2
```

```
 - 2/3*a21*a03*a20^2*a01^2 - 1/6*a21*a12*a20*a11*a01^2
 + a30*a03*a20*a11*a01^2 + 1/18*a21^2*a11^2*a01^2
 - 1/12*a30*a12*a11^2*a01^2 + 1/9*a21^2*a20*a02*a01^2
 - 1/6*a30*a12*a20*a02*a01^2 - 1/6*a30*a21*a11*a02*a01^2
 + 1/4*a30^2*a02^2*a01^2 + 1/9*a21^2*a12*a10*a01^2
 - 2/3*a30*a12^2*a10*a01^2 + a30*a21*a03*a10*a01^2
 - 2/27*a21^3*a01^3 + 1/3*a30*a21*a12*a01^3 - a30^2*a03*a01^3
 - a03^2*a20^3*a00 + a12*a03*a20^2*a11*a00 - 1/12*a12^2*a20*a11^2*a00
 - 3/4*a21*a03*a20*a11^2*a00 + 1/24*a21*a12*a11^3*a00
 + 5/8*a30*a03*a11^3*a00 - 2/3*a12^2*a20^2*a02*a00
 + a21*a03*a20^2*a02*a00 + 5/6*a21*a12*a20*a11*a02*a00
 - 3/2*a30*a03*a20*a11*a02*a00 - 1/12*a21^2*a11^2*a02*a00
 - 3/4*a30*a12*a11^2*a02*a00 - 2/3*a21^2*a20*a02^2*a00
 + a30*a12*a20*a02^2*a00 + a30*a21*a11*a02^2*a00
 - a30^2*a02^3*a00 + 1/3*a12^3*a20*a10*a00
 - 3/2*a21*a12*a03*a20*a10*a00 + 9/2*a30*a03^2*a20*a10*a00
 - 1/6*a21*a12^2*a11*a10*a00 + a21^2*a03*a11*a10*a00
 - 3/2*a30*a12*a03*a11*a10*a00 - 1/6*a21^2*a12*a02*a10*a00
 + a30*a12^2*a02*a10*a00 - 3/2*a30*a21*a03*a02*a10*a00
 - 1/6*a21*a12^2*a20*a01*a00 + a21^2*a03*a20*a01*a00
 - 3/2*a30*a12*a03*a20*a01*a00 - 1/6*a21^2*a12*a11*a01*a00
 + a30*a12^2*a11*a01*a00 - 3/2*a30*a21*a03*a11*a01*a00
 + 1/3*a21^3*a02*a01*a00 - 3/2*a30*a21*a12*a02*a01*a00
 + 9/2*a30^2*a03*a02*a01*a00 + 1/4*a21^2*a12^2*a00^2
 - a30*a12^3*a00^2 - a21^3*a03*a00^2
 + 9/2*a30*a21*a12*a03*a00^2 - 27/4*a30^2*a03^2*a00^2)
```

sage.schemes.toric.weierstrass.**WeierstrassForm_P2_112**(*polynomial*, *variables=None*)

Bring an anticanonical hypersurface in $\mathbb{P}^2[1,1,2]$ into Weierstrass form.

Input/output is the same as `WeierstrassForm()`, except that the input polynomial must be a standard anticanonical hypersurface in weighted projective space $\mathbb{P}^2[1,1,2]$:

$$p(x,y) = a_{40}x^4 + a_{30}x^3 + a_{21}x^2y + a_{20}x^2 + \\ a_{11}xy + a_{02}y^2 + a_{10}x + a_{01}y + a_{00}$$

EXAMPLES:

```
sage: from sage.schemes.toric.weierstrass import WeierstrassForm_P2_112
sage: fan = Fan(rays=[(1,0),(0,1),(-1,-2),(0,-1)],cones=[[0,1],[1,2],[2,3],[3,0]])
sage: P112.<x,y,z,t> = ToricVariety(fan)
sage: (-P112.K()).sections_monomials()
(z^4*t^2, x*z^3*t^2, x^2*z^2*t^2, x^3*z*t^2,
 x^4*t^2, y*z^2*t, x*y*z*t, x^2*y*t, y^2)
sage: WeierstrassForm_P2_112(sum(_), [x,y,z,t])
(-97/48, 17/864)
```

TESTS:

```
sage: R.<x,y,z,t,a40,a30,a20,a10,a00,a21,a11,a01,a02> = QQ[]
sage: p = ( a40*x^4*t^2 + a30*x^3*z*t^2 + a20*x^2*z^2*t^2 + a10*x*z^3*t^2 +
....:       a00*z^4*t^2 + a21*x^2*y*t + a11*x*y*z*t + a01*y*z^2*t + a02*y^2 )
sage: WeierstrassForm_P2_112(p, [x,y,z,t])
(-1/48*a11^4 + 1/6*a21*a11^2*a01 - 1/3*a21^2*a01^2 + a00*a21^2*a02
 - 1/2*a10*a21*a11*a02 + 1/6*a20*a11^2*a02 + 1/3*a20*a21*a01*a02
 - 1/2*a30*a11*a01*a02 + a40*a01^2*a02 - 1/3*a20^2*a02^2 + a30*a10*a02^2
 - 4*a40*a00*a02^2, 1/864*a11^6 - 1/72*a21*a11^4*a01
 + 1/18*a21^2*a11^2*a01^2 - 2/27*a21^3*a01^3 - 1/12*a00*a21^2*a11^2*a02
 + 1/24*a10*a21*a11^3*a02 - 1/72*a20*a11^4*a02 + 1/3*a00*a21^3*a01*a02
```

```
- 1/6*a10*a21^2*a11*a01*a02 + 1/36*a20*a21*a11^2*a01*a02
+ 1/24*a30*a11^3*a01*a02 + 1/9*a20*a21^2*a01^2*a02
- 1/6*a30*a21*a11*a01^2*a02 - 1/12*a40*a11^2*a01^2*a02
+ 1/3*a40*a21*a01^3*a02 + 1/4*a10^2*a21^2*a02^2
- 2/3*a20*a00*a21^2*a02^2 - 1/6*a20*a10*a21*a11*a02^2
+ a30*a00*a21*a11*a02^2 + 1/18*a20^2*a11^2*a02^2
- 1/12*a30*a10*a11^2*a02^2 - 2/3*a40*a00*a11^2*a02^2
+ 1/9*a20^2*a21*a01*a02^2 - 1/6*a30*a10*a21*a01*a02^2
- 4/3*a40*a00*a21*a01*a02^2 - 1/6*a30*a20*a11*a01*a02^2
+ a40*a10*a11*a01*a02^2 + 1/4*a30^2*a01^2*a02^2
- 2/3*a40*a20*a01^2*a02^2 - 2/27*a20^3*a02^3
+ 1/3*a30*a20*a10*a02^3 - a40*a10^2*a02^3 - a30^2*a00*a02^3
+ 8/3*a40*a20*a00*a02^3)

sage: _ == WeierstrassForm_P2_112(p.subs(z=1,t=1), [x,y])
True

sage: cubic = p.subs(a40=0)
sage: a,b = WeierstrassForm_P2_112(cubic, [x,y,z,t])
sage: a = a.subs(t=1,z=1)
sage: b = b.subs(t=1,z=1)
sage: from sage.schemes.toric.weierstrass import WeierstrassForm_P2
sage: (a,b) == WeierstrassForm_P2(cubic.subs(t=1,z=1), [x,y])
True
```

sage.schemes.toric.weierstrass.**j_invariant**(*polynomial*, *variables=None*)

Return the j-invariant of the elliptic curve.

INPUT:

See `WeierstrassForm()` for how to specify the input polynomial(s) and variables.

OUTPUT:

The j-invariant of the (irreducible) cubic. Notable special values:

- The Fermat cubic: $j(x^3 + y^3 + z^3) = 0$
- A nodal cubic: $j(-y^2 + x^2 + x^3) = \infty$
- A cuspidal cubic $y^2 = x^3$ has undefined j-invariant. In this case, a `ValueError` is returned.

EXAMPLES:

```
sage: from sage.schemes.toric.weierstrass import j_invariant
sage: R.<x,y,z> = QQ[]
sage: j_invariant(x^3+y^3+z^3)
0
sage: j_invariant(-y^2 + x^2 + x^3)
+Infinity
sage: R.<x,y,z, a,b> = QQ[]
sage: j_invariant( -y^2*z + x^3 + a*x*z^2, [x,y,z])
1728
```

TESTS:

```
sage: j_invariant(x*y*z)
Traceback (most recent call last):
...
ValueError: curve is singular and has no well-defined j-invariant
```

16.10 Map to the Weierstrass form of a toric elliptic curve.

There are 16 reflexive polygons in 2-d. Each defines a toric Fano variety, which (since it is 2-d) has a unique crepant resolution to a smooth toric surface. An anticanonical hypersurface defines a genus one curve C in this ambient space, with Jacobian elliptic curve $J(C)$ which can be defined by the Weierstrass model $y^2 = x^3 + fx + g$. The coefficients f and g can be computed with the `weierstrass` module. The purpose of this model is to give an explicit rational map $C \to J(C)$. This is an n^2-cover, where n is the minimal multi-section of C.

Since it is technically often easier to deal with polynomials than with fractions, we return the rational map in terms of homogeneous coordinates. That is, the ambient space for the Weierstrass model is the weighted projective space $\mathbb{P}^2[2,3,1]$ with homogeneous coordinates $[X : Y : Z] = [\lambda^2 X, \lambda^3 Y, \lambda Z]$. The homogenized Weierstrass equation is

$$Y^2 = X^3 + fXZ^4 + gZ^6$$

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: cubic = x^3 + y^3 + 1
sage: f, g = WeierstrassForm(cubic);  (f,g)
(0, -27/4)
```

That is, this hypersurface $C \in \mathbb{P}^2$ has a Weierstrass equation $Y^2 = X^3 + 0 \cdot XZ^4 - \frac{27}{4}Z^6$ where $[X : Y : Z]$ are projective coordinates on $\mathbb{P}^2[2,3,1]$. The form of the map $C \to J(C)$ is:

```
sage: X,Y,Z = WeierstrassForm(cubic, transformation=True);  (X,Y,Z)
(-x^3*y^3 - x^3 - y^3,
 1/2*x^6*y^3 - 1/2*x^3*y^6 - 1/2*x^6 + 1/2*y^6 + 1/2*x^3 - 1/2*y^3,
 x*y)
```

Note that plugging in $[X : Y : Z]$ to the Weierstrass equation is a complicated polynomial, but contains the hypersurface equation as a factor:

```
sage: -Y^2 + X^3 + f*X*Z^4 + g*Z^6
-1/4*x^12*y^6 - 1/2*x^9*y^9 - 1/4*x^6*y^12 + 1/2*x^12*y^3
- 7/2*x^9*y^6 - 7/2*x^6*y^9 + 1/2*x^3*y^12 - 1/4*x^12 - 7/2*x^9*y^3
- 45/4*x^6*y^6 - 7/2*x^3*y^9 - 1/4*y^12 - 1/2*x^9 - 7/2*x^6*y^3
- 7/2*x^3*y^6 - 1/2*y^9 - 1/4*x^6 + 1/2*x^3*y^3 - 1/4*y^6
sage: cubic.divides(-Y^2 + X^3 + f*X*Z^4 + g*Z^6)
True
```

If you prefer you can also use homogeneous coordinates for $C \in \mathbb{P}^2$

```
sage: R.<x,y,z> = QQ[]
sage: cubic = x^3 + y^3 + z^3
sage: f, g = WeierstrassForm(cubic);  (f,g)
(0, -27/4)
sage: X,Y,Z = WeierstrassForm(cubic, transformation=True)
sage: cubic.divides(-Y^2 + X^3 + f*X*Z^4 + g*Z^6)
True
```

The 16 toric surfaces corresponding to the 16 reflexive polygons can all be blown down to $\mathbb{P}^2$, $\mathbb{P}^1 \times \mathbb{P}^1$, or $\mathbb{P}^2[1,1,2]$. Their (and hence in all 16 cases) anticanonical hypersurface can equally be brought into Weierstrass form. For example, here is an anticanonical hypersurface in $\mathbb{P}^2[1,1,2]$

```
sage: P2_112 = toric_varieties.P2_112()
sage: C = P2_112.anticanonical_hypersurface(coefficients=[1]*4);  C
Closed subscheme of 2-d CPR-Fano toric variety
covered by 3 affine patches defined by:
```

```
  z0^4 + z2^4 + z0*z1*z2 + z1^2
sage: eq = C.defining_polynomials()[0]
sage: f, g = WeierstrassForm(eq)
sage: X,Y,Z = WeierstrassForm(eq, transformation=True)
sage: (-Y^2 + X^3 + f*X*Z^4 + g*Z^6).reduce(C.defining_ideal())
0
```

Finally, you sometimes have to manually specify the variables to use. This is either because the equation is degenerate or because it contains additional variables that you want to treat as coefficients:

```
sage: R.<a, x,y,z> = QQ[]
sage: cubic = x^3 + y^3 + z^3 + a*x*y*z
sage: f, g = WeierstrassForm(cubic, variables=[x,y,z])
sage: X,Y,Z = WeierstrassForm(cubic, variables=[x,y,z], transformation=True)
sage: cubic.divides(-Y^2 + X^3 + f*X*Z^4 + g*Z^6)
True
```

REFERENCES:

`sage.schemes.toric.weierstrass_covering.`**`WeierstrassMap`**(*polynomial*, *variables=None*)

Return the Weierstrass form of an anticanonical hypersurface.

You should use `sage.schemes.toric.weierstrass.WeierstrassForm()` with `transformation=True` to get the transformation. This function is only for internal use.

INPUT:

- `polynomial` – a polynomial. The toric hypersurface equation. Can be either a cubic, a biquadric, or the hypersurface in $\mathbb{P}^2[1,1,2]$. The equation need not be in any standard form, only its Newton polyhedron is used.
- `variables` – a list of variables of the parent polynomial ring or `None` (default). In the latter case, all variables are taken to be polynomial ring variables. If a subset of polynomial ring variables are given, the Weierstrass form is determined over the function field generated by the remaining variables.

OUTPUT:

A triple (X, Y, Z) of polynomials defining a rational map of the toric hypersurface to its Weierstrass form in $\mathbb{P}^2[2,3,1]$. That is, the triple satisfies

$$Y^2 = X^3 + fXZ^4 + gZ^6$$

when restricted to the toric hypersurface.

EXAMPLES:

```
sage: R.<x,y,z> = QQ[]
sage: cubic = x^3 + y^3 + z^3
sage: X,Y,Z = WeierstrassForm(cubic, transformation=True);  (X,Y,Z)
(-x^3*y^3 - x^3*z^3 - y^3*z^3,
 1/2*x^6*y^3 - 1/2*x^3*y^6 - 1/2*x^6*z^3 + 1/2*y^6*z^3
     + 1/2*x^3*z^6 - 1/2*y^3*z^6,
 x*y*z)
 sage: f, g = WeierstrassForm(cubic);  (f,g)
 (0, -27/4)
 sage: cubic.divides(-Y^2 + X^3 + f*X*Z^4 + g*Z^6)
 True
```

Only the affine span of the Newton polytope of the polynomial matters. For example:

```
sage: WeierstrassForm(cubic.subs(z=1), transformation=True)
(-x^3*y^3 - x^3 - y^3,
 1/2*x^6*y^3 - 1/2*x^3*y^6 - 1/2*x^6
     + 1/2*y^6 + 1/2*x^3 - 1/2*y^3,
 x*y)
sage: WeierstrassForm(x * cubic, transformation=True)
(-x^3*y^3 - x^3*z^3 - y^3*z^3,
 1/2*x^6*y^3 - 1/2*x^3*y^6 - 1/2*x^6*z^3 + 1/2*y^6*z^3
     + 1/2*x^3*z^6 - 1/2*y^3*z^6,
 x*y*z)
```

This allows you to work with either homogeneous or inhomogeneous variables. For example, here is the del Pezzo surface of degree 8:

```
sage: dP8 = toric_varieties.dP8()
sage: dP8.inject_variables()
Defining t, x, y, z
sage: WeierstrassForm(x*y^2 + y^2*z + t^2*x^3 + t^2*z^3, transformation=True)
(-1/27*t^4*x^6 - 2/27*t^4*x^5*z - 5/27*t^4*x^4*z^2
     - 8/27*t^4*x^3*z^3 - 5/27*t^4*x^2*z^4 - 2/27*t^4*x*z^5
     - 1/27*t^4*z^6 - 4/81*t^2*x^4*y^2 - 4/81*t^2*x^3*y^2*z
     - 4/81*t^2*x*y^2*z^3 - 4/81*t^2*y^2*z^4 - 2/81*x^2*y^4
     - 4/81*x*y^4*z - 2/81*y^4*z^2,
0,
1/3*t^2*x^2*z + 1/3*t^2*x*z^2 - 1/9*x*y^2 - 1/9*y^2*z)
sage: WeierstrassForm(x*y^2 + y^2 + x^3 + 1, transformation=True)
(-1/27*x^6 - 4/81*x^4*y^2 - 2/81*x^2*y^4 - 2/27*x^5
     - 4/81*x^3*y^2 - 4/81*x*y^4 - 5/27*x^4 - 2/81*y^4 - 8/27*x^3
     - 4/81*x*y^2 - 5/27*x^2 - 4/81*y^2 - 2/27*x - 1/27,
 0,
 -1/9*x*y^2 + 1/3*x^2 - 1/9*y^2 + 1/3*x)
```

By specifying only certain variables we can compute the Weierstrass form over the function field generated by the remaining variables. For example, here is a cubic over $\mathbf{Q}[a]$

```
sage: R.<a, x,y,z> = QQ[]
sage: cubic = x^3 + a*y^3 + a^2*z^3
sage: WeierstrassForm(cubic, variables=[x,y,z], transformation=True)
(-a^9*y^3*z^3 - a^8*x^3*z^3 - a^7*x^3*y^3,
 -1/2*a^14*y^3*z^6 + 1/2*a^13*y^6*z^3 + 1/2*a^13*x^3*z^6
     - 1/2*a^11*x^3*y^6 - 1/2*a^11*x^6*z^3 + 1/2*a^10*x^6*y^3,
 a^3*x*y*z)
```

TESTS:

```
sage: for P in ReflexivePolytopes(2):
....:     S = ToricVariety(FaceFan(P))
....:     p = sum( (-S.K()).sections_monomials() )
....:     f, g = WeierstrassForm(p)
....:     X,Y,Z = WeierstrassForm(p, transformation=True)
....:     assert p.divides(-Y^2 + X^3 + f*X*Z^4 + g*Z^6)
```

`sage.schemes.toric.weierstrass_covering.`**`WeierstrassMap_P1xP1`**`(`*polynomial*, *variables=None*`)`

Map an anticanonical hypersurface in $\mathbb{P}^1 \times \mathbb{P}^1$ into Weierstrass form.

Input/output is the same as `WeierstrassMap()`, except that the input polynomial must be a standard anticanonical hypersurface in the toric surface $\mathbb{P}^1 \times \mathbb{P}^1$:

EXAMPLES:

```
sage: from sage.schemes.toric.weierstrass_covering import WeierstrassMap_P1xP1
sage: from sage.schemes.toric.weierstrass import WeierstrassForm_P1xP1
sage: R.<x0,x1,y0,y1,a>= QQ[]
sage: biquadric = ( x0^2*y0^2 + x1^2*y0^2 + x0^2*y1^2 + x1^2*y1^2 +
....:     a * x0*x1*y0*y1*5 )
sage: f, g = WeierstrassForm_P1xP1(biquadric, [x0, x1, y0, y1]);  (f,g)
(-625/48*a^4 + 25/3*a^2 - 16/3, 15625/864*a^6 - 625/36*a^4 - 100/9*a^2 + 128/27)
sage: X, Y, Z = WeierstrassMap_P1xP1(biquadric, [x0, x1, y0, y1])
sage: (-Y^2 + X^3 + f*X*Z^4 + g*Z^6).reduce(R.ideal(biquadric))
0

sage: R = PolynomialRing(QQ, 'x,y,s,t', order='lex')
sage: R.inject_variables()
Defining x, y, s, t
sage: equation = ( s^2*(x^2+2*x*y+3*y^2) + s*t*(4*x^2+5*x*y+6*y^2)
....:              + t^2*(7*x^2+8*x*y+9*y^2) )
sage: X, Y, Z = WeierstrassMap_P1xP1(equation, [x,y,s,t])
sage: f, g = WeierstrassForm_P1xP1(equation, variables=[x,y,s,t])
sage: (-Y^2 + X^3 + f*X*Z^4 + g*Z^6).reduce(R.ideal(equation))
0

sage: R = PolynomialRing(QQ, 'x,s', order='lex')
sage: R.inject_variables()
Defining x, s
sage: equation = s^2*(x^2+2*x+3) + s*(4*x^2+5*x+6) + (7*x^2+8*x+9)
sage: X, Y, Z = WeierstrassMap_P1xP1(equation)
sage: f, g = WeierstrassForm_P1xP1(equation)
sage: (-Y^2 + X^3 + f*X*Z^4 + g*Z^6).reduce(R.ideal(equation))
0
```

`sage.schemes.toric.weierstrass_covering.`**`WeierstrassMap_P2`**(*polynomial*, *variables=None*)

Map a cubic to its Weierstrass form

Input/output is the same as `WeierstrassMap()`, except that the input polynomial must be a cubic in $\mathbb{P}^2$,

$$p(x, y) = a_{30}x^3 + a_{21}x^2y + a_{12}xy^2 + a_{03}y^3 + a_{20}x^2 + a_{11}xy + a_{02}y^2 + a_{10}x + a_{01}y + a_{00}$$

EXAMPLES:

```
sage: from sage.schemes.toric.weierstrass import WeierstrassForm_P2
sage: from sage.schemes.toric.weierstrass_covering import WeierstrassMap_P2
sage: R.<x,y,z> = QQ[]
sage: equation =  x^3+y^3+z^3+x*y*z
sage: f, g = WeierstrassForm_P2(equation)
sage: X,Y,Z = WeierstrassMap_P2(equation)
sage: equation.divides(-Y^2 + X^3 + f*X*Z^4 + g*Z^6)
True

sage: from sage.schemes.toric.weierstrass import WeierstrassForm_P2
sage: from sage.schemes.toric.weierstrass_covering import WeierstrassMap_P2
sage: R.<x,y> = QQ[]
sage: equation =  x^3+y^3+1
sage: f, g = WeierstrassForm_P2(equation)
sage: X,Y,Z = WeierstrassMap_P2(equation)
sage: equation.divides(-Y^2 + X^3 + f*X*Z^4 + g*Z^6)
True
```

sage.schemes.toric.weierstrass_covering.**WeierstrassMap_P2_112**(*polynomial*, *variables=None*)

Map an anticanonical hypersurface in $\mathbb{P}^2[1,1,2]$ into Weierstrass form.

Input/output is the same as WeierstrassMap(), except that the input polynomial must be a standard anticanonical hypersurface in weighted projective space $\mathbb{P}^2[1,1,2]$:

$$p(x,y) = a_{40}x^4 + a_{30}x^3 + a_{21}x^2y + a_{20}x^2 + a_{11}xy + a_{02}y^2 + a_{10}x + a_{01}y + a_{00}$$

EXAMPLES:

```
sage: from sage.schemes.toric.weierstrass_covering import WeierstrassMap_P2_112
sage: from sage.schemes.toric.weierstrass import WeierstrassForm_P2_112
sage: R = PolynomialRing(QQ, 'x,y,a0,a1,a2,a3,a4', order='lex')
sage: R.inject_variables()
Defining x, y, a0, a1, a2, a3, a4
sage: equation = y^2 + a0*x^4 + 4*a1*x^3 + 6*a2*x^2 + 4*a3*x + a4
sage: X, Y, Z = WeierstrassMap_P2_112(equation, [x,y])
sage: f, g = WeierstrassForm_P2_112(equation, variables=[x,y])
sage: (-Y^2 + X^3 + f*X*Z^4 + g*Z^6).reduce(R.ideal(equation))
0
```

Another example, this time in homogeneous coordinates:

```
sage: fan = Fan(rays=[(1,0),(0,1),(-1,-2),(0,-1)],cones=[[0,1],[1,2],[2,3],[3,0]])
sage: P112.<x,y,z,t> = ToricVariety(fan)
sage: (-P112.K()).sections_monomials()
(z^4*t^2, x*z^3*t^2, x^2*z^2*t^2, x^3*z*t^2,
 x^4*t^2, y*z^2*t, x*y*z*t, x^2*y*t, y^2)
sage: C_eqn = sum(_)
sage: C = P112.subscheme(C_eqn)
sage: WeierstrassForm_P2_112(C_eqn, [x,y,z,t])
(-97/48, 17/864)
sage: X, Y, Z = WeierstrassMap_P2_112(C_eqn, [x,y,z,t])
sage: (-Y^2 + X^3 - 97/48*X*Z^4 + 17/864*Z^6).reduce(C.defining_ideal())
0
```

16.11 Weierstrass for Elliptic Curves in Higher Codimension

The weierstrass module lets you transform a genus-one curve, given as a hypersurface in a toric surface, into Weierstrass form. The purpose of this module is to extend this to higher codimension subschemes of toric varieties. In general, this is an unsolved problem. However, for certain special cases this is known.

The simplest codimension-two case is the complete intersection of two quadratic equations in $\mathbb{P}^3$

```
sage: R.<w,x,y,z> = QQ[]
sage: quadratic1 = w^2+x^2+y^2
sage: quadratic2 = z^2 + w*x
sage: WeierstrassForm([quadratic1, quadratic2])
(-1/4, 0)
```

Hence, the Weierstrass form of this complete intersection is $Y^2 = X^3 - \frac{1}{4}XZ^4$.

sage.schemes.toric.weierstrass_higher.**WeierstrassForm2**(*polynomial*, *variables=None*, *transformation=False*)

Helper function for WeierstrassForm()

Currently, only the case of the complete intersection of two quadratic equations in $\mathbb{P}^3$ is supported.

INPUT / OUTPUT:

See WeierstrassForm()

TESTS:

```
sage: from sage.schemes.toric.weierstrass_higher import WeierstrassForm2
sage: R.<w,x,y,z> = QQ[]
sage: quadratic1 = w^2+x^2+y^2
sage: quadratic2 = z^2 + w*x
sage: WeierstrassForm2([quadratic1, quadratic2])
(-1/4, 0)
```

sage.schemes.toric.weierstrass_higher.**WeierstrassForm_P3**(*quadratic1*, *quadratic2*, *variables=None*)

Bring a complete intersection of two quadratics into Weierstrass form.

Input/output is the same as sage.schemes.toric.weierstrass.WeierstrassForm(), except that the two input polynomials must be quadratic polynomials in $\mathbb{P}^3$.

EXAMPLES:

```
sage: from sage.schemes.toric.weierstrass_higher import WeierstrassForm_P3
sage: R.<w,x,y,z> = QQ[]
sage: quadratic1 = w^2+x^2+y^2
sage: quadratic2 = z^2 + w*x
sage: WeierstrassForm_P3(quadratic1, quadratic2)
(-1/4, 0)
```

TESTS:

```
sage: R.<w,x,y,z,a0,a1,a2,a3,b0,b1,b2,b3,b4,b5> = QQ[]
sage: p1 = w^2 + x^2 + y^2 + z^2
sage: p2 = a0*w^2 + a1*x^2 + a2*y^2 + a3*z^2
sage: p2 += b0*x*y + b1*x*z + b2*x*w + b3*y*z + b4*y*w + b5*z*w
sage: a, b = WeierstrassForm_P3(p1, p2, [w,x,y,z])
sage: a.total_degree(), len(a.coefficients())
(4, 107)
sage: b.total_degree(), len(b.coefficients())
(6, 648)
```

sage.schemes.toric.weierstrass_higher.**WeierstrassMap_P3**(*quadratic1*, *quadratic2*, *variables=None*)

Bring a complete intersection of two quadratics into Weierstrass form.

Input/output is the same as sage.schemes.toric.weierstrass.WeierstrassForm(), except that the two input polynomials must be quadratic polynomials in $\mathbb{P}^3$.

EXAMPLES:

```
sage: from sage.schemes.toric.weierstrass_higher import \
....:     WeierstrassMap_P3, WeierstrassForm_P3
sage: R.<w,x,y,z> = QQ[]
sage: quadratic1 = w^2+x^2+y^2
sage: quadratic2 = z^2 + w*x
sage: X, Y, Z = WeierstrassMap_P3(quadratic1, quadratic2)
sage: X
1/1024*w^8 + 3/256*w^6*x^2 + 19/512*w^4*x^4 + 3/256*w^2*x^6 + 1/1024*x^8
sage: Y
1/32768*w^12 - 7/16384*w^10*x^2 - 145/32768*w^8*x^4 - 49/8192*w^6*x^6
- 145/32768*w^4*x^8 - 7/16384*w^2*x^10 + 1/32768*x^12
```

```
sage: Z
-1/8*w^2*y*z + 1/8*x^2*y*z

sage: a, b = WeierstrassForm_P3(quadratic1, quadratic2);  a, b
(-1/4, 0)

sage: ideal = R.ideal(quadratic1, quadratic2)
sage: (-Y^2 + X^3 + a*X*Z^4 + b*Z^6).reduce(ideal)
0
```

TESTS:

```
sage: R.<w,x,y,z,a0,a1,a2,a3> = GF(101)[]
sage: p1 = w^2 + x^2 + y^2 + z^2
sage: p2 = a0*w^2 + a1*x^2 + a2*y^2 + a3*z^2
sage: X, Y, Z = WeierstrassMap_P3(p1, p2, [w,x,y,z])
sage: X.total_degree(), len(X.coefficients())
(22, 4164)
sage: Y.total_degree(), len(Y.coefficients())
(33, 26912)
sage: Z.total_degree(), len(Z.coefficients())
(10, 24)
sage: Z
w*x*y*z*a0^3*a1^2*a2 - w*x*y*z*a0^2*a1^3*a2 - w*x*y*z*a0^3*a1*a2^2
+ w*x*y*z*a0*a1^3*a2^2 + w*x*y*z*a0^2*a1*a2^3 - w*x*y*z*a0*a1^2*a2^3
- w*x*y*z*a0^3*a1^2*a3 + w*x*y*z*a0^2*a1^3*a3 + w*x*y*z*a0^3*a2^2*a3
- w*x*y*z*a1^3*a2^2*a3 - w*x*y*z*a0^2*a2^3*a3 + w*x*y*z*a1^2*a2^3*a3
+ w*x*y*z*a0^3*a1*a3^2 - w*x*y*z*a0*a1^3*a3^2 - w*x*y*z*a0^3*a2*a3^2
+ w*x*y*z*a1^3*a2*a3^2 + w*x*y*z*a0*a2^3*a3^2 - w*x*y*z*a1*a2^3*a3^2
- w*x*y*z*a0^2*a1*a3^3 + w*x*y*z*a0*a1^2*a3^3 + w*x*y*z*a0^2*a2*a3^3
- w*x*y*z*a1^2*a2*a3^3 - w*x*y*z*a0*a2^2*a3^3 + w*x*y*z*a1*a2^2*a3^3
```

16.12 Set of homomorphisms between two toric varieties.

For schemes X and Y, this module implements the set of morphisms $Hom(X, Y)$. This is done by `SchemeHomset_generic`.

As a special case, the Hom-sets can also represent the points of a scheme. Recall that the K-rational points of a scheme X over k can be identified with the set of morphisms $Spec(K) \to X$. In Sage, the rational points are implemented by such scheme morphisms. This is done by `SchemeHomset_points` and its subclasses.

Note: You should not create the Hom-sets manually. Instead, use the `Hom()` method that is inherited by all schemes.

AUTHORS:

- Volker Braun (2012-02-18): Initial version

EXAMPLES:

Here is a simple example, the projection of $\mathbb{P}^1 \times \mathbb{P}^1 \to \mathbb{P}^1$

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: hom_set = P1xP1.Hom(P1);  hom_set
Set of morphisms
  From: 2-d CPR-Fano toric variety covered by 4 affine patches
  To:   1-d CPR-Fano toric variety covered by 2 affine patches
```

In terms of the fan, we can define this morphism by the projection onto the first coordinate. The Hom-set can construct the morphism from the projection matrix alone:

```
sage: hom_set(matrix([[1],[0]]))
Scheme morphism:
  From: 2-d CPR-Fano toric variety covered by 4 affine patches
  To:   1-d CPR-Fano toric variety covered by 2 affine patches
  Defn: Defined by sending Rational polyhedral fan in 2-d lattice N
        to Rational polyhedral fan in 1-d lattice N.
sage: _.as_polynomial_map()
Scheme morphism:
  From: 2-d CPR-Fano toric variety covered by 4 affine patches
  To:   1-d CPR-Fano toric variety covered by 2 affine patches
  Defn: Defined on coordinates by sending [s : t : x : y] to
        [s : t]
```

In the case of toric algebraic schemes (defined by polynomials in toric varieties), this module defines the underlying morphism of the ambient toric varieties:

```
sage: P1xP1.inject_variables()
Defining s, t, x, y
sage: S = P1xP1.subscheme([s*x-t*y])
sage: type(S.Hom(S))
<class 'sage.schemes.toric.homset.SchemeHomset_toric_variety_with_category'>
```

Finally, you can have morphisms defined through homogeneous coordinates where the codomain is not implemented as a toric variety:

```
sage: P2_toric.<x,y,z> = toric_varieties.P2()
sage: P2_native.<u,v,w> = ProjectiveSpace(QQ, 2)
sage: toric_to_native = P2_toric.Hom(P2_native);  toric_to_native
Set of morphisms
  From: 2-d CPR-Fano toric variety covered by 3 affine patches
  To:   Projective Space of dimension 2 over Rational Field
sage: type(toric_to_native)
<class 'sage.schemes.toric.homset.SchemeHomset_toric_variety_with_category'>
sage: toric_to_native([x^2, y^2, z^2])
Scheme morphism:
  From: 2-d CPR-Fano toric variety covered by 3 affine patches
  To:   Projective Space of dimension 2 over Rational Field
  Defn: Defined on coordinates by sending [x : y : z] to
        (x^2 : y^2 : z^2)

sage: native_to_toric = P2_native.Hom(P2_toric);  native_to_toric
Set of morphisms
  From: Projective Space of dimension 2 over Rational Field
  To:   2-d CPR-Fano toric variety covered by 3 affine patches
sage: type(native_to_toric)
<class 'sage.schemes.generic.homset.SchemeHomset_generic_with_category'>
sage: native_to_toric([u^2, v^2, w^2])
Scheme morphism:
  From: Projective Space of dimension 2 over Rational Field
  To:   2-d CPR-Fano toric variety covered by 3 affine patches
  Defn: Defined on coordinates by sending (u : v : w) to
        [u^2 : v^2 : w^2]
```

class sage.schemes.toric.homset.**SchemeHomset_points_subscheme_toric_field**(*X*, *Y*, *category=None*, *check=True*, *base=Integer Ring*)

Bases: sage.schemes.toric.homset.SchemeHomset_points_toric_base

Python constructor.

INPUT:

See SchemeHomset_generic.

EXAMPLES:

```
sage: from sage.schemes.generic.homset import SchemeHomset_points
sage: SchemeHomset_points(Spec(QQ), AffineSpace(ZZ,2))
Set of rational points of Affine Space of dimension 2 over Rational Field
```

cardinality()

Return the number of points of the toric variety.

OUTPUT:

An integer or infinity. The cardinality of the set of points.

EXAMPLES:

```
sage: P2.<x,y,z> = toric_varieties.P2(base_ring=GF(5))
sage: cubic = P2.subscheme([x^3 + y^3 + z^3])
sage: list(cubic.point_set())
[[0 : 1 : 4], [1 : 0 : 4], [1 : 4 : 0], [1 : 2 : 1], [1 : 1 : 2], [1 : 3 : 3]]
sage: cubic.point_set().cardinality()
6
```

class sage.schemes.toric.homset.**SchemeHomset_points_toric_base**(*X*, *Y*, *category=None*, *check=True*, *base=Integer Ring*)

Bases: sage.schemes.generic.homset.SchemeHomset_points

Base class for homsets with toric ambient spaces

INPUT:

- same as for SchemeHomset_points.

OUPUT:

A scheme morphism of type SchemeHomset_points_toric_base.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1(QQ)
Set of rational points of 2-d CPR-Fano toric variety
covered by 4 affine patches
```

TESTS:

```
sage: import sage.schemes.toric.homset as HOM
sage: HOM.SchemeHomset_points_toric_base(Spec(QQ), P1xP1)
Set of rational points of 2-d CPR-Fano toric variety covered by 4 affine patches
```

is_finite()

Return whether there are finitely many points.

OUTPUT:

Boolean.

EXAMPLES:

```
sage: P2 = toric_varieties.P2()
sage: P2.point_set().is_finite()
False
sage: P2.change_ring(GF(7)).point_set().is_finite()
True
```

class sage.schemes.toric.homset.**SchemeHomset_points_toric_field**(*X*, *Y*, *category=None*, *check=True*, *base=Integer Ring*)

Bases: sage.schemes.toric.homset.SchemeHomset_points_toric_base

Set of rational points of a toric variety.

You should not use this class directly. Instead, use the point_set() method to construct the point set of a toric variety.

INPUT:

•same as for SchemeHomset_points.

OUPUT:

A scheme morphism of type SchemeHomset_points_toric_field.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1xP1.point_set()
Set of rational points of 2-d CPR-Fano toric variety
covered by 4 affine patches
sage: P1xP1(QQ)
Set of rational points of 2-d CPR-Fano toric variety
covered by 4 affine patches
```

The quotient $\mathbb{P}^2/\mathbf{Z}_3$ over $GF(7)$ by the diagonal action. This is tricky because the base field has a 3-rd root of unity:

```
sage: fan = NormalFan(ReflexivePolytope(2, 0))
sage: X = ToricVariety(fan, base_field=GF(7))
sage: point_set = X.point_set()
sage: point_set.cardinality()
21
sage: sorted(X.point_set().list())
[[0 : 0 : 1], [0 : 1 : 0], [0 : 1 : 1], [0 : 1 : 3],
 [1 : 0 : 0], [1 : 0 : 1], [1 : 0 : 3], [1 : 1 : 0],
 [1 : 1 : 1], [1 : 1 : 2], [1 : 1 : 3], [1 : 1 : 4],
 [1 : 1 : 5], [1 : 1 : 6], [1 : 3 : 0], [1 : 3 : 1],
```

```
 [1 : 3 : 2], [1 : 3 : 3], [1 : 3 : 4], [1 : 3 : 5],
 [1 : 3 : 6]]
```

As for a non-compact example, the blow-up of the plane is the line bundle $O_{\mathbf{P}^1}(-1)$. Its point set is the cartesian product of the points on the base $\mathbf{P}^1$ with the points on the fiber:

```
sage: fan = Fan([Cone([(1,0), (1,1)]), Cone([(1,1), (0,1)])])
sage: blowup_plane = ToricVariety(fan, base_ring=GF(3))
sage: point_set = blowup_plane.point_set()
sage: sorted(point_set.list())
[[0 : 1 : 0], [0 : 1 : 1], [0 : 1 : 2],
 [1 : 0 : 0], [1 : 0 : 1], [1 : 0 : 2],
 [1 : 1 : 0], [1 : 1 : 1], [1 : 1 : 2],
 [1 : 2 : 0], [1 : 2 : 1], [1 : 2 : 2]]
```

Toric varieties with torus factors (that is, where the fan is not full-dimensional) also work:

```
sage: F_times_Fstar = ToricVariety(Fan([Cone([(1,0)])]), base_field=GF(3))
sage: sorted(F_times_Fstar.point_set().list())
[[0 : 1], [0 : 2], [1 : 1], [1 : 2], [2 : 1], [2 : 2]]
```

TESTS:

```
sage: import sage.schemes.toric.homset as HOM
sage: HOM.SchemeHomset_points_toric_field(Spec(QQ), P1xP1)
Set of rational points of 2-d CPR-Fano toric variety covered by 4 affine patches
```

cardinality()

Return the number of points of the toric variety.

OUTPUT:

An integer or infinity. The cardinality of the set of points.

EXAMPLES:

```
sage: o = lattice_polytope.cross_polytope(3)
sage: V = ToricVariety(FaceFan(o))
sage: V.change_ring(GF(2)).point_set().cardinality()
27
sage: V.change_ring(GF(8, "a")).point_set().cardinality()
729
sage: V.change_ring(GF(101)).point_set().cardinality()
1061208
```

For non-smooth varieties over finite fields, the homogeneous rescalings are solved. This is somewhat slower:

```
sage: fan = NormalFan(ReflexivePolytope(2, 0))
sage: X = ToricVariety(fan, base_field=GF(7))
sage: X.point_set().cardinality()
21
```

Fulton's formula does not apply since the variety is not smooth. And, indeed, naive application gives a different result:

```
sage: q = X.base_ring().order()
sage: n = X.dimension()
sage: d = map(len, fan().cones())
sage: sum(dk * (q-1)**(n-k) for k, dk in enumerate(d))
57
```

Over infinite fields the number of points is not very tricky:

```
sage: V.count_points()
+Infinity
```

ALGORITHM:

Uses the formula in Fulton [F], section 4.5.

REFERENCES:

AUTHORS:

- Beth Malmskog (2013-07-14)
- Adriana Salerno (2013-07-14)
- Yiwei She (2013-07-14)
- Christelle Vincent (2013-07-14)
- Ursula Whitcher (2013-07-14)

class sage.schemes.toric.homset.**SchemeHomset_toric_variety**(*X*, *Y*, *category=None*, *check=True*, *base=Integer Ring*)

Bases: sage.schemes.generic.homset.SchemeHomset_generic

Set of homomorphisms between two toric varieties.

EXAMPLES:

```
sage: P1xP1 = toric_varieties.P1xP1()
sage: P1 = toric_varieties.P1()
sage: hom_set = P1xP1.Hom(P1);  hom_set
Set of morphisms
  From: 2-d CPR-Fano toric variety covered by 4 affine patches
  To:   1-d CPR-Fano toric variety covered by 2 affine patches
sage: type(hom_set)
<class 'sage.schemes.toric.homset.SchemeHomset_toric_variety_with_category'>

sage: hom_set(matrix([[1],[0]]))
Scheme morphism:
  From: 2-d CPR-Fano toric variety covered by 4 affine patches
  To:   1-d CPR-Fano toric variety covered by 2 affine patches
  Defn: Defined by sending Rational polyhedral fan in 2-d lattice N
        to Rational polyhedral fan in 1-d lattice N.
```

16.13 Enumerate Points of a Toric Variety

The classes here are not meant to be instatiated manually. Instead, you should always use the methods of the point set of the variety.

In this module, points are always represented by tuples instead of Sage's class for points of the toric variety. All Sage library code must then convert it to proper point objects before returning it to the user.

EXAMPLES:

```
sage: P2 = toric_varieties.P2(base_ring=GF(3))
sage: point_set = P2.point_set()
sage: point_set.cardinality()
```

```
13
sage: next(iter(point_set))
[0 : 0 : 1]
sage: list(point_set)[0:5]
[[0 : 0 : 1], [1 : 0 : 0], [0 : 1 : 0], [0 : 1 : 1], [0 : 1 : 2]]
```

class sage.schemes.toric.points.**FiniteFieldPointEnumerator**(*fan*, *ring*)

Bases: sage.schemes.toric.points.NaiveFinitePointEnumerator

The naive point enumerator.

This is very slow.

INPUT:

- fan – fan of the toric variety.
- ring – finite base ring over which to enumerate points.

EXAMPLES:

```
sage: from sage.schemes.toric.points import NaiveFinitePointEnumerator
sage: fan = toric_varieties.P2().fan()
sage: n = NaiveFinitePointEnumerator(fan, GF(3))
sage: next(iter(n))
(0, 0, 1)
```

cardinality()

Return the cardinality of the point set.

OUTPUT:

Integer. The number of points.

EXAMPLES:

```
sage: fan = NormalFan(ReflexivePolytope(2, 0))
sage: X = ToricVariety(fan, base_ring=GF(7))
sage: point_set = X.point_set()
sage: ffe = point_set._finite_field_enumerator()
sage: ffe.cardinality()
21
```

cone_points_iter()

Iterate over the open torus orbits and yield distinct points.

OUTPUT:

For each open torus orbit (cone): A triple consisting of the cone, the nonzero homogeneous coordinates in that orbit (list of integers), and the nonzero log coordinates of distinct points as a cokernel.

EXAMPLES:

```
sage: fan = NormalFan(ReflexivePolytope(2, 0))
sage: X = ToricVariety(fan, base_ring=GF(7))
sage: point_set = X.point_set()
sage: ffe = point_set._finite_field_enumerator()
sage: cpi = ffe.cone_points_iter()
sage: cone, nonzero_points, cokernel = list(cpi)[5]
sage: cone
1-d cone of Rational polyhedral fan in 2-d lattice N
sage: cone.ambient_ray_indices()
(2,)
```

```
sage: nonzero_points
[0, 1]
sage: cokernel
Finitely generated module V/W over Integer Ring with invariants (2)
sage: list(cokernel)
[(0), (1)]
sage: [p.lift() for p in cokernel]
[(0, 0), (0, 1)]
```

exp(*powers*)

Return the component-wise exp of `z`

INPUT:

- `powers` – a list/tuple/iterable of integers.

OUTPUT:

Tuple of finite field elements. The powers of the `multiplicative_generator()`.

EXAMPLES:

```
sage: F.<a> = GF(5^2)
sage: point_set = toric_varieties.P2_123(base_ring=F).point_set()
sage: ffe = point_set._finite_field_enumerator()
sage: powers = range(24)
sage: ffe.exp(powers)
(1, a, a + 3, 4*a + 3, 2*a + 2, 4*a + 1, 2, 2*a, 2*a + 1, 3*a + 1,
 4*a + 4, 3*a + 2, 4, 4*a, 4*a + 2, a + 2, 3*a + 3, a + 4, 3, 3*a,
 3*a + 4, 2*a + 4, a + 1, 2*a + 3)
sage: ffe.log(ffe.exp(powers)) == tuple(powers)
True
```

log(*z*)

Return the component-wise log of `z`

INPUT:

- `z` – a list/tuple/iterable of non-zero finite field elements.

OUTPUT:

Tuple of integers. The logarithm with base the `multiplicative_generator()`.

EXAMPLES:

```
sage: F.<a> = GF(5^2)
sage: point_set = toric_varieties.P2_123(base_ring=F).point_set()
sage: ffe = point_set._finite_field_enumerator()
sage: z = tuple(a^i for i in range(25)); z
(1, a, a + 3, 4*a + 3, 2*a + 2, 4*a + 1, 2, 2*a, 2*a + 1, 3*a + 1,
 4*a + 4, 3*a + 2, 4, 4*a, 4*a + 2, a + 2, 3*a + 3, a + 4, 3, 3*a,
 3*a + 4, 2*a + 4, a + 1, 2*a + 3, 1)
sage: ffe.log(z)
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
 17, 18, 19, 20, 21, 22, 23, 0)
sage: ffe.exp(ffe.log(z)) == z
True
sage: ffe.log(ffe.exp(range(24))) == tuple(range(24))
True
```

multiplicative_generator()

Return the multiplicative generator of the finite field.

OUTPUT:

A finite field element.

EXAMPLES:

sage: point_set = toric_varieties.P2(base_ring=GF(5^2, 'a')).point_set() sage: ffe = point_set._finite_field_enumerator() sage: ffe.multiplicative_generator() a

multiplicative_group_order()

EXAMPLES:

```
sage: class Foo:
....:     def __init__(self, x):
....:         self._x = x
....:     @cached_method
....:     def f(self):
....:         return self._x^2
sage: a = Foo(2)
sage: print a.f.get_cache()
None
sage: a.f()
4
sage: a.f.get_cache()
4
```

rescaling_log_generators()

Return the log generators of `rescalings()`.

OUTPUT:

A tuple containing the logarithms (see `log()`) of the generators of the multiplicative group of `rescalings()`.

EXAMPLES:

```
sage: point_set = toric_varieties.P2_123(base_ring=GF(5)).point_set()
sage: ffe = point_set._finite_field_enumerator()
sage: ffe.rescalings()
((1, 1, 1), (1, 4, 4), (4, 2, 3), (4, 3, 2))
sage: map(ffe.log, ffe.rescalings())
[(0, 0, 0), (0, 2, 2), (2, 1, 3), (2, 3, 1)]
sage: ffe.rescaling_log_generators()
((2, 3, 1),)
```

root_generator(*n*)

Return a generator for `roots()`.

INPUT:

- `n` integer.

OUTPUT:

A multiplicative generator for `roots()`.

EXAMPLES:

```
sage: point_set = toric_varieties.P2(base_ring=GF(5)).point_set()
sage: ffe = point_set._finite_field_enumerator()
sage: ffe.root_generator(2)
```

```
4
sage: ffe.root_generator(3)
1
sage: ffe.root_generator(4)
2
```

TESTS:

```
sage: for p in primes(10):
....:     for k in range(1,5):
....:         F = GF(p^k, 'a')
....:         N = F.cardinality() - 1
....:         ffe = point_set._finite_field_enumerator(F)
....:         assert N == ffe.multiplicative_group_order()
....:         for n in N.divisors():
....:             x = ffe.root_generator(n)
....:             assert set(x**i for i in range(N)) == set(ffe.roots(n))
```

class `sage.schemes.toric.points.`**`FiniteFieldSubschemePointEnumerator`**(*polynomials*, *ambient*)

Bases: `sage.schemes.toric.points.NaiveSubschemePointEnumerator`

Point enumerator for algebraic subschemes of toric varieties.

INPUT:

- `polynomials` – list/tuple/iterabel of polynomials. The defining polynomials.
- `ambient` – enumerator for ambient space points.

TESTS:

```
sage: P2.<x,y,z> = toric_varieties.P2()
sage: from sage.schemes.toric.points import NaiveSubschemePointEnumerator
sage: ne = NaiveSubschemePointEnumerator(
....:     [x^2+y^2-2*z^2], P2.point_set()._enumerator())
sage: next(iter(ne))
(1, 1, 1)
```

`cardinality`()

Return the cardinality of the point set.

OUTPUT:

Integer. The number of points.

EXAMPLES:

```
sage: fan = NormalFan(ReflexivePolytope(2, 0))
sage: X.<u,v,w> = ToricVariety(fan, base_ring=GF(7))
sage: Y = X.subscheme(u^3 + v^3 + w^3 + u*v*w)
sage: point_set = Y.point_set()
sage: list(point_set)
[[0 : 1 : 3],
 [1 : 0 : 3],
 [1 : 3 : 0],
 [1 : 1 : 6],
 [1 : 1 : 4],
 [1 : 3 : 2],
 [1 : 3 : 5]]
sage: ffe = point_set._enumerator()
```

```
sage: ffe.cardinality()
7
```

homogeneous_coordinates (*log_t*, *nonzero_coordinates*, *cokernel*)

Convert the log of inhomogeneous coordinates back to homogeneous coordinates

INPUT:

- `log_t` – log of inhomogeneous coordinates of a point.
- `nonzero_coordinates` – the nonzero homogeneous coordinates in the patch.
- `cokernel` – the logs of the nonzero coordinates of all distinct points as a cokernel. See `FiniteFieldPointEnumerator.cone_points_iter()`.

OUTPUT:

The same point, but as a tuple of homogeneous coordinates.

EXAMPLES:

```
sage: P2.<x,y,z> = toric_varieties.P2(base_ring=GF(7))
sage: X = P2.subscheme([x^3 + 2*y^3 + 3*z^3, x*y*z + x*y^2])
sage: point_set = X.point_set()
sage: ffe = point_set._enumerator()
sage: cone, nonzero_coordinates, cokernel = list(ffe.ambient.cone_points_iter())[5]
sage: cone.ambient_ray_indices(), nonzero_coordinates
((2,), [0, 1])
sage: ffe.homogeneous_coordinates([0], nonzero_coordinates, cokernel)
(1, 1, 0)
sage: ffe.homogeneous_coordinates([1], nonzero_coordinates, cokernel)
(1, 3, 0)
sage: ffe.homogeneous_coordinates([2], nonzero_coordinates, cokernel)
(1, 2, 0)
```

inhomogeneous_equations (*ring*, *nonzero_coordinates*, *cokernel*)

Inhomogenize the defining polynomials

INPUT:

- `ring` – the polynomial ring for inhomogeneous coordinates.
- `nonzero_coordinates` – list of integers. The indices of the non-zero homogeneous coordinates in the patch.
- `cokernel` – the logs of the nonzero coordinates of all distinct points as a cokernel. See `FiniteFieldPointEnumerator.cone_points_iter()`.

EXAMPLES:

```
sage: R.<s> = QQ[]
sage: P2.<x,y,z> = toric_varieties.P2(base_ring=GF(7))
sage: X = P2.subscheme([x^3 + 2*y^3 + 3*z^3, x*y*z + x*y^2])
sage: point_set = X.point_set()
sage: ffe = point_set._enumerator()
sage: cone, nonzero_coordinates, cokernel = list(ffe.ambient.cone_points_iter())[5]
sage: cone.ambient_ray_indices(), nonzero_coordinates
((2,), [0, 1])
sage: ffe.inhomogeneous_equations(R, nonzero_coordinates, cokernel)
[2*s^3 + 1, s^2]
```

solutions (*inhomogeneous_equations*, *log_range*)

Parallel version of `solutions_serial()`

INPUT/OUTPUT:

Same as solutions_serial(), except that the output points are in random order. Order depends on the number of processors and relative speed of separate processes.

EXAMPLES:

```
sage: R.<s> = GF(7)[]
sage: P2.<x,y,z> = toric_varieties.P2(base_ring=GF(7))
sage: X = P2.subscheme(1)
sage: point_set = X.point_set()
sage: ffe = point_set._enumerator()
sage: ffe.solutions([s^2-1, s^6-s^2], [range(6)])
<generator object solutions at 0x...>
sage: sorted(_)
[[0], [3]]
```

solutions_serial(*inhomogeneous_equations*, *log_range*)

Iterate over solutions in a range.

INPUT:

- inhomogeneous_equations – list/tuple/iterable of inhomogeneous equations (i.e. output from inhomogeneous_equations()).
- log_range – list/tuple/iterable of integer ranges. One for each inhomogeneous coordinate. The logarithms of the homogeneous coordinates.

OUTPUT:

All solutions (as tuple of log inhomogeneous coordinates) in the Cartesian product of the ranges.

EXAMPLES:

```
sage: R.<s> = GF(7)[]
sage: P2.<x,y,z> = toric_varieties.P2(base_ring=GF(7))
sage: X = P2.subscheme(1)
sage: point_set = X.point_set()
sage: ffe = point_set._enumerator()
sage: ffe.solutions_serial([s^2-1, s^6-s^2], [range(6)])
<generator object solutions_serial at 0x...>
sage: list(_)
[[0], [3]]
```

class sage.schemes.toric.points.**InfinitePointEnumerator**(*fan*, *ring*)

Bases: object

Point enumerator for infinite fields.

INPUT:

- fan – fan of the toric variety.
- ring – infinite base ring over which to enumerate points.

TESTS:

```
sage: from sage.schemes.toric.points import InfinitePointEnumerator
sage: fan = toric_varieties.P2().fan()
sage: n = InfinitePointEnumerator(fan, QQ)
sage: ni = iter(n)
sage: [next(ni) for k in range(10)]
[(0, 1, 1), (1, 1, 1), (-1, 1, 1), (1/2, 1, 1), (-1/2, 1, 1),
 (2, 1, 1), (-2, 1, 1), (1/3, 1, 1), (-1/3, 1, 1), (3, 1, 1)]
```

```
sage: X = ToricVariety(Fan([], lattice=ZZ^0))
sage: X.point_set().cardinality()
1
sage: X.base_ring().is_finite()
False
sage: X.point_set().list()
([],)
```

class `sage.schemes.toric.points.`**`NaiveFinitePointEnumerator`**(*fan*, *ring*)

Bases: `object`

The naive point enumerator.

This is very slow.

INPUT:

- `fan` – fan of the toric variety.
- `ring` – finite base ring over which to enumerate points.

EXAMPLES:

```
sage: from sage.schemes.toric.points import NaiveFinitePointEnumerator
sage: fan = toric_varieties.P2().fan()
sage: n = NaiveFinitePointEnumerator(fan, GF(3))
sage: next(iter(n))
(0, 0, 1)
```

`cone_iter`()

Iterate over all cones of the fan

OUTPUT:

Iterator over the cones, starting with the high-dimensional ones.

EXAMPLES:

```
sage: ne = toric_varieties.dP6(base_ring=GF(11)).point_set()._naive_enumerator()
sage: for cone in ne.cone_iter():
....:     print cone.ambient_ray_indices()
(0, 1)
(1, 2)
(2, 3)
(3, 4)
(4, 5)
(0, 5)
(0,)
(1,)
(2,)
(3,)
(4,)
(5,)
()
```

`coordinate_iter`()

Iterate over all distinct homogeneous coordinates.

This method does NOT identify homogeneous coordinates that are equivalent by a homogeneous rescaling.

OUTPUT:

An iterator over the points.

EXAMPLES:

```
sage: F2 = GF(2)
sage: ni = toric_varieties.P2(base_ring=F2).point_set()._naive_enumerator()
sage: list(ni.coordinate_iter())
[(0, 0, 1), (1, 0, 0), (0, 1, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0), (1, 1, 1)]

sage: ni = toric_varieties.P1xP1(base_ring=F2).point_set()._naive_enumerator()
sage: list(ni.coordinate_iter())
[(0, 1, 0, 1), (1, 0, 0, 1), (1, 0, 1, 0),
 (0, 1, 1, 0), (0, 1, 1, 1), (1, 0, 1, 1),
 (1, 1, 0, 1), (1, 1, 1, 0), (1, 1, 1, 1)]
```

TESTS:

```
sage: V = ToricVariety(Fan([Cone([(1,1)])]), base_ring=GF(3))
sage: ni = V.point_set()._naive_enumerator()
sage: list(ni.coordinate_iter())
[(0, 1), (0, 2), (1, 1), (1, 2), (2, 1), (2, 2)]
```

orbit (*point*)

Return the orbit of homogeneous coordinates under rescalings.

OUTPUT:

The set of all homogeneous coordinates that are equivalent to `point`.

EXAMPLES:

```
sage: ne = toric_varieties.P2_123(base_ring=GF(7)).point_set()._naive_enumerator()
sage: sorted(ne.orbit([1, 0, 0]))
[(1, 0, 0), (2, 0, 0), (4, 0, 0)]
sage: sorted(ne.orbit([0, 1, 0]))
[(0, 1, 0), (0, 6, 0)]
sage: sorted(ne.orbit([0, 0, 1]))
[(0, 0, 1), (0, 0, 2), (0, 0, 3), (0, 0, 4), (0, 0, 5), (0, 0, 6)]
sage: sorted(ne.orbit([1, 1, 0]))
[(1, 1, 0), (1, 6, 0), (2, 1, 0), (2, 6, 0), (4, 1, 0), (4, 6, 0)]
```

rays ()

Return all rays (real and virtual).

OUTPUT:

Tuple of rays of the fan.

EXAMPLES:

```
sage: from sage.schemes.toric.points import NaiveFinitePointEnumerator
sage: fan = toric_varieties.torus(2).fan()
sage: fan.rays()
Empty collection
in 2-d lattice N
sage: n = NaiveFinitePointEnumerator(fan, GF(3))
sage: n.rays()
N(1, 0),
N(0, 1)
in 2-d lattice N
```

rescalings ()

Return the rescalings of homogeneous coordinates.

OUTPUT:

A tuple containing all points that are equivalent to $[1 : 1 : \cdots : 1]$, the distinguished point of the big torus orbit.

EXAMPLES:

```
sage: ni = toric_varieties.P2_123(base_ring=GF(5)).point_set()._naive_enumerator()
sage: ni.rescalings()
((1, 1, 1), (1, 4, 4), (4, 2, 3), (4, 3, 2))

sage: ni = toric_varieties.dP8(base_ring=GF(3)).point_set()._naive_enumerator()
sage: ni.rescalings()
((1, 1, 1, 1), (1, 2, 2, 2), (2, 1, 2, 1), (2, 2, 1, 2))

sage: ni = toric_varieties.P1xP1(base_ring=GF(3)).point_set()._naive_enumerator()
sage: ni.rescalings()
((1, 1, 1, 1), (1, 1, 2, 2), (2, 2, 1, 1), (2, 2, 2, 2))
```

roots(*n*)

Return the n-th roots in the base field

INPUT:

- `n` integer.

OUTPUT:

Tuple containing all n-th roots (not only the primitive ones). In particular, 1 is included.

EXAMPLES:

```
sage: ne = toric_varieties.P2(base_ring=GF(5)).point_set()._naive_enumerator()
sage: ne.roots(2)
(1, 4)
sage: ne.roots(3)
(1,)
sage: ne.roots(4)
(1, 2, 3, 4)
```

units()

Return the units in the base field.

EXAMPLES:

```
sage: ne = toric_varieties.P2(base_ring=GF(5)).point_set()._naive_enumerator()
sage: ne.units()
(1, 2, 3, 4)
```

class sage.schemes.toric.points.**NaiveSubschemePointEnumerator**(*polynomials*, *ambient*)

Bases: `object`

Point enumerator for algebraic subschemes of toric varieties.

INPUT:

- `polynomials` – list/tuple/iterabel of polynomials. The defining polynomials.
- `ambient` – enumerator for ambient space points.

TESTS:

```
sage: P2.<x,y,z> = toric_varieties.P2()
sage: from sage.schemes.toric.points import NaiveSubschemePointEnumerator
```

```
sage: ne = NaiveSubschemePointEnumerator(
....:     [x^2+y^2-2*z^2], P2.point_set()._enumerator())
sage: next(iter(ne))
(1, 1, 1)
```

CHAPTER

SEVENTEEN

INDICES AND TABLES

- Index
- Module Index
- Search Page

[LazarsfeldJacobian] Robert Lazarsfeld: Positivity in algebraic geometry II; Positivity for Vector Bundles, and Multiplier Ideals, page 181.

[FMV] Computing Conjugating Sets and Automorphism Groups of Rational Functions by Xander Faber, Michelle Manes, and Bianca Viray

[Hutz] B. Hutz. Efficient determination of rational preperiodic points for endomorphisms of projective space. Arxiv 1210.6246, 2012.

[MoPa] P. Morton and P. Patel. The Galois theory of periodic points of polynomial maps. Proc. London Math. Soc., 68 (1994), 225-263.

[Bruin-Molnar] N. Bruin and A. Molnar, *Minimal models for rational functions in a dynamical setting*, LMS Journal of Computation and Mathematics, Volume 15 (2012), pp 400-417.

[Molnar] A. Molnar, *Fractional Linear Minimal Models of Rational Functions*, M.Sc. Thesis.

[FaHu] J. A. de Faria, B. Hutz. Combinatorics of Cycle Lengths on Wehler K3 Surfaces over finite fields Arxiv 1309.6598, 2013.

[CaSi] G. Call and J. Silverman. Computing the Canonical Height on K3 Surfaces. Mathematics of Comp. , 65 (1996), 259-290.

[Wehl] J. Wehler. Hypersurfaces of the Flag Variety: Deformation Theory and the Theorems of Kodaira-Spencer, Torelli, Lefschetz, M. Noether, and Serre. Math. Z. 198 (1988), 21-38

[Hutzthesis] B. Hutz. Arithmetic Dynamics on Varieties of dimension greater than one. PhD Thesis, Brown University 2007

[CLS] David A. Cox, John B. Little, Hal Schenck, "Toric Varieties", Graduate Studies in Mathematics, Amer. Math. Soc., Providence, RI, 2011

[Demazure] M. Demazure Sous-groupes algébriques de rang maximum du groupe de Cremona. Ann. Sci. Ecole Norm. Sup. 1970, 3, 507–588.

[Bazhov] Ivan Bazhov: On orbits of the automorphism group on a complete toric variety. Arxiv 1110.4275, doi:10.1007/s13366-011-0084-0.

[Schubert] Sheldon Katz and Stein Arild Stromme, A Maple package for intersection theory and enumerative geometry.

[Batyrev1994] Victor V. Batyrev, "Dual polyhedra and mirror symmetry for Calabi-Yau hypersurfaces in toric varieties", J. Algebraic Geom. 3 (1994), no. 3, 493-535. arXiv:alg-geom/9310003v1

[Borisov1993] Lev A. Borisov, "Towards the mirror symmetry for Calabi-Yau complete intersections in Gorenstein Fano toric varieties", 1993. arXiv:alg-geom/9310001v1

[CD2007] Adrian Clingher and Charles F. Doran, “Modular invariants for lattice polarized K3 surfaces”, Michigan Math. J. 55 (2007), no. 2, 355-393. arXiv:math/0602146v1 [math.AG]

[Nill2005] Benjamin Nill, “Gorenstein toric Fano varieties”, Manuscripta Math. 116 (2005), no. 2, 183-210. arXiv:math/0405448v1 [math.AG]

[BCdlOG] Volker Braun, Philip Candelas, Xendia de la Ossa, Antonella Grassi, “Toric Calabi-Yau Fourfolds, Duality Between N=1 Theories and Divisors that Contribute to the Superpotential”, http://arxiv.org/abs/hep-th/0001208

[HLY] Yi Hu, Chien-Hao Liu, Shing-Tung Yau, “Toric morphisms and fibrations of toric Calabi-Yau hypersurfaces”, http://arxiv.org/abs/math/0010082

[FS] William Fulton, Bernd Sturmfels, “Intersection Theory on Toric Varieties”, http://arxiv.org/abs/alg-geom/9403002

[FultonP65] Page 65, 3rd exercise (Section 3.4) of Wiliam Fulton, “Introduction to Toric Varieties”, Princeton University Press

[Perling] Markus Perling: Divisorial Cohomology Vanishing on Toric Varieties, Arxiv 0711.4836v2

[CoxTutorial] David Cox, “What is a Toric Variety”, http://www.cs.amherst.edu/~dac/lectures/tutorial.ps

[wp:ChowRing] Wikipedia article Chow_ring

[FMSS1] Fulton, MacPherson, Sottile, Sturmfels: *Intersection theory on spherical varieties*, J. of Alg. Geometry 4 (1995), 181-193. http://www.math.tamu.edu/~frank.sottile/research/ps/spherical.ps.gz

[FultonChow] Chapter 5.1 “Chow Groups” of Fulton, William: *Introduction to Toric Varieties*, Princeton University Press

[GRIN] Bernd Sturmfels, Serkan Hosten: GRIN: An implementation of Grobner bases for integer programming, in “Integer Programming and Combinatorial Optimization”, [E. Balas and J. Clausen, eds.], Proceedings of the IV. IPCO Conference (Copenhagen, May 1995), Springer Lecture Notes in Computer Science 920 (1995) 267-276.

[ProcSympPureMath62] Bernd Sturmfels: Equations defining toric varieties, Algebraic Geometry - Santa Cruz 1995, Proc. Sympos. Pure Math., 62, Part 2, Amer. Math. Soc., Providence, RI, 1997, pp. 437-449.

[GBCP] Bernd Sturmfels: Grobner Bases and Convex Polytopes AMS University Lecture Series Vol. 8 (01 December 1995)

[BB] Gavin Brown, Jaroslaw Buczynski: Maps of toric varieties in Cox coordinates, http://arxiv.org/abs/1004.4924

[VolkerBraun] Volker Braun: Toric Elliptic Fibrations and F-Theory Compactifications Arxiv 1110.4883

[Duistermaat] J. J. Duistermaat, Discrete integrable systems. QRT maps and elliptic surfaces. Springer Monographs in Mathematics. Berlin: Springer. xxii, 627 p., 2010

[ArtinVillegasTate] Michael Artin, Fernando Rodriguez-Villegas, John Tate, On the Jacobians of plane cubics, Advances in Mathematics 198 (2005) 1, pp. 366–382 doi:10.1016/j.aim.2005.06.004 http://www.math.utexas.edu/users/villegas/publications/jacobian-cubics.pdf

[CLSsurfaces] Section 10.4 in David A. Cox, John B. Little, Hal Schenck, “Toric Varieties”, Graduate Studies in Mathematics, Amer. Math. Soc., Providence, RI, 2011

[AnEtAl] An, Sang Yook et al: Jacobians of Genus One Curves, Journal of Number Theory 90 (2002), pp.304–315, http://www.math.arizona.edu/~wmc/Research/JacobianFinal.pdf

[F] Fulton, W., “Introduction to Toric Varieties”, Princeton University Press, 1993.

S

sage.schemes.affine.affine_homset, 97
sage.schemes.affine.affine_morphism, 84
sage.schemes.affine.affine_point, 80
sage.schemes.affine.affine_rational_point, 95
sage.schemes.affine.affine_space, 75
sage.schemes.generic.algebraic_scheme, 23
sage.schemes.generic.ambient_space, 19
sage.schemes.generic.divisor, 69
sage.schemes.generic.divisor_group, 73
sage.schemes.generic.glue, 15
sage.schemes.generic.homset, 53
sage.schemes.generic.hypersurface, 51
sage.schemes.generic.morphism, 59
sage.schemes.generic.point, 17
sage.schemes.generic.scheme, 3
sage.schemes.generic.spec, 13
sage.schemes.product_projective.homset, 184
sage.schemes.product_projective.morphism, 185
sage.schemes.product_projective.point, 185
sage.schemes.product_projective.space, 179
sage.schemes.product_projective.wehlerK3, 187
sage.schemes.projective.endPN_automorphism_group, 165
sage.schemes.projective.endPN_minimal_model, 175
sage.schemes.projective.projective_homset, 162
sage.schemes.projective.projective_morphism, 120
sage.schemes.projective.projective_morphism_helper, 159
sage.schemes.projective.projective_point, 108
sage.schemes.projective.projective_rational_point, 159
sage.schemes.projective.projective_space, 101
sage.schemes.readme, 1
sage.schemes.toric.chow_group, 294
sage.schemes.toric.divisor, 273
sage.schemes.toric.divisor_class, 292
sage.schemes.toric.fano_variety, 241
sage.schemes.toric.homset, 343
sage.schemes.toric.ideal, 305

sage.schemes.toric.library, 258
sage.schemes.toric.morphism, 308
sage.schemes.toric.points, 348
sage.schemes.toric.variety, 207
sage.schemes.toric.weierstrass, 328
sage.schemes.toric.weierstrass_covering, 337
sage.schemes.toric.weierstrass_higher, 341

INDEX

A

A() (sage.schemes.toric.ideal.ToricIdeal method), 307
A() (sage.schemes.toric.library.ToricVarietyFactory method), 259
A1() (sage.schemes.toric.library.ToricVarietyFactory method), 259
A2() (sage.schemes.toric.library.ToricVarietyFactory method), 260
A2_Z2() (sage.schemes.toric.library.ToricVarietyFactory method), 260
add_variables() (in module sage.schemes.toric.fano_variety), 257
affine_algebraic_patch() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_toric method), 46
affine_algebraic_patch() (sage.schemes.toric.variety.ToricVariety_field method), 221
affine_minimal() (in module sage.schemes.projective.endPN_minimal_model), 176
affine_open() (sage.schemes.generic.point.SchemeTopologicalPoint_affine_open method), 17
affine_patch() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_product_projective method), 40
affine_patch() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_projective method), 42
affine_patch() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_toric method), 46
affine_patch() (sage.schemes.product_projective.space.ProductProjectiveSpaces_ring method), 180
affine_patch() (sage.schemes.projective.projective_space.ProjectiveSpace_ring method), 106
affine_patch() (sage.schemes.toric.variety.ToricVariety_field method), 222
AffineHypersurface (class in sage.schemes.generic.hypersurface), 51
AffineScheme (class in sage.schemes.generic.scheme), 3
AffineSpace() (in module sage.schemes.affine.affine_space), 75
AffineSpace_field (class in sage.schemes.affine.affine_space), 75
AffineSpace_finite_field (class in sage.schemes.affine.affine_space), 76
AffineSpace_generic (class in sage.schemes.affine.affine_space), 76
AffineToricVariety() (in module sage.schemes.toric.variety), 211
AlgebraicScheme (class in sage.schemes.generic.algebraic_scheme), 24
AlgebraicScheme_quasi (class in sage.schemes.generic.algebraic_scheme), 28
AlgebraicScheme_subscheme (class in sage.schemes.generic.algebraic_scheme), 29
AlgebraicScheme_subscheme_affine (class in sage.schemes.generic.algebraic_scheme), 36
AlgebraicScheme_subscheme_affine_toric (class in sage.schemes.generic.algebraic_scheme), 38
AlgebraicScheme_subscheme_product_projective (class in sage.schemes.generic.algebraic_scheme), 40
AlgebraicScheme_subscheme_projective (class in sage.schemes.generic.algebraic_scheme), 42
AlgebraicScheme_subscheme_toric (class in sage.schemes.generic.algebraic_scheme), 45
all_rational_preimages() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space_field method), 149
ambient_space() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme method), 24
ambient_space() (sage.schemes.generic.ambient_space.AmbientSpace method), 19

AmbientSpace (class in sage.schemes.generic.ambient_space), 19
anticanonical_hypersurface() (sage.schemes.toric.fano_variety.CPRFanoToricVariety_field method), 250
AnticanonicalHypersurface (class in sage.schemes.toric.fano_variety), 243
as_fan_morphism() (sage.schemes.toric.morphism.SchemeMorphism_polynomial_toric_variety method), 327
as_polynomial_map() (sage.schemes.toric.morphism.SchemeMorphism_fan_fiber_component_toric_variety method), 314
as_polynomial_map() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety method), 316
as_polynomial_map() (sage.schemes.toric.morphism.SchemeMorphism_orbit_closure_toric_variety method), 325
Aut_dimension() (sage.schemes.toric.variety.ToricVariety_field method), 216
automorphism_group() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 121
automorphism_group() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space_finite_field method), 156
automorphism_group_FF() (in module sage.schemes.projective.endPN_automorphism_group), 167
automorphism_group_FF_alg2() (in module sage.schemes.projective.endPN_automorphism_group), 168
automorphism_group_FF_alg3() (in module sage.schemes.projective.endPN_automorphism_group), 168
automorphism_group_QQ_CRT() (in module sage.schemes.projective.endPN_automorphism_group), 169
automorphism_group_QQ_fixedpoints() (in module sage.schemes.projective.endPN_automorphism_group), 169
automorphisms_fixing_pair() (in module sage.schemes.projective.endPN_automorphism_group), 170

B

base_cone() (sage.schemes.toric.morphism.SchemeMorphism_fan_fiber_component_toric_variety method), 314
base_extend() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 30
base_extend() (sage.schemes.generic.ambient_space.AmbientSpace method), 19
base_extend() (sage.schemes.generic.divisor_group.DivisorGroup_generic method), 73
base_extend() (sage.schemes.generic.scheme.AffineScheme method), 3
base_extend() (sage.schemes.generic.scheme.Scheme method), 6
base_extend() (sage.schemes.projective.projective_homset.SchemeHomset_points_abelian_variety_field method), 163
base_extend() (sage.schemes.toric.divisor.ToricDivisorGroup method), 277
base_morphism() (sage.schemes.generic.scheme.Scheme method), 6
base_ring() (sage.schemes.generic.morphism.SchemeMorphism_polynomial method), 64
base_ring() (sage.schemes.generic.scheme.Scheme method), 6
base_scheme() (sage.schemes.generic.scheme.Scheme method), 7
BCdlOG() (sage.schemes.toric.library.ToricVarietyFactory method), 260
BCdlOG_base() (sage.schemes.toric.library.ToricVarietyFactory method), 261
bCheck() (in module sage.schemes.projective.endPN_minimal_model), 177
blift() (in module sage.schemes.projective.endPN_minimal_model), 177

C

c() (sage.schemes.toric.variety.ToricVariety_field method), 222
canonical_height() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 191
canonical_height() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 123
canonical_height() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring method), 111
canonical_height_minus() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 192
canonical_height_plus() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 193
cardinality() (sage.schemes.generic.homset.SchemeHomset_points method), 56
cardinality() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_finite_field method), 187
cardinality() (sage.schemes.toric.homset.SchemeHomset_points_subscheme_toric_field method), 345

cardinality() (sage.schemes.toric.homset.SchemeHomset_points_toric_field method), 347
cardinality() (sage.schemes.toric.points.FiniteFieldPointEnumerator method), 349
cardinality() (sage.schemes.toric.points.FiniteFieldSubschemePointEnumerator method), 352
cartesian_product() (sage.schemes.projective.projective_space.ProjectiveSpace_ring method), 106
cartesian_product() (sage.schemes.toric.fano_variety.CPRFanoToricVariety_field method), 252
cartesian_product() (sage.schemes.toric.variety.ToricVariety_field method), 223
category() (sage.schemes.generic.morphism.SchemeMorphism method), 60
category_for() (sage.schemes.generic.morphism.SchemeMorphism method), 60
certify_names() (in module sage.schemes.toric.variety), 238
ch() (sage.schemes.toric.divisor.ToricDivisor_generic method), 280
ch() (sage.schemes.toric.variety.ToricVariety_field method), 223
change_ring() (sage.schemes.affine.affine_space.AffineSpace_generic method), 77
change_ring() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 30
change_ring() (sage.schemes.generic.ambient_space.AmbientSpace method), 19
change_ring() (sage.schemes.generic.morphism.SchemeMorphism_point method), 62
change_ring() (sage.schemes.generic.morphism.SchemeMorphism_polynomial method), 64
change_ring() (sage.schemes.product_projective.point.ProductProjectiveSpaces_point_ring method), 186
change_ring() (sage.schemes.product_projective.space.ProductProjectiveSpaces_ring method), 181
change_ring() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 193
change_ring() (sage.schemes.projective.projective_space.ProjectiveSpace_ring method), 106
change_ring() (sage.schemes.toric.fano_variety.CPRFanoToricVariety_field method), 252
change_ring() (sage.schemes.toric.variety.ToricVariety_field method), 224
Chern_character() (sage.schemes.toric.divisor.ToricDivisor_generic method), 278
Chern_character() (sage.schemes.toric.variety.ToricVariety_field method), 216
Chern_class() (sage.schemes.toric.variety.ToricVariety_field method), 217
chi() (sage.schemes.toric.variety.ToricVariety_field method), 224
Chow_cycle() (sage.schemes.toric.divisor.ToricDivisor_generic method), 279
Chow_group() (sage.schemes.toric.variety.ToricVariety_field method), 217
ChowCycle (class in sage.schemes.toric.chow_group), 295
ChowGroup_class (class in sage.schemes.toric.chow_group), 300
ChowGroup_degree_class (class in sage.schemes.toric.chow_group), 303
ChowGroupFactory (class in sage.schemes.toric.chow_group), 299
clear_denominators() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_field method), 109
codimension() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 31
coefficient() (sage.schemes.generic.divisor.Divisor_curve method), 70
coefficient() (sage.schemes.toric.divisor.ToricDivisor_generic method), 280
cohomology() (sage.schemes.toric.divisor.ToricDivisor_generic method), 280
cohomology_basis() (sage.schemes.toric.variety.ToricVariety_field method), 225
cohomology_class() (sage.schemes.toric.chow_group.ChowCycle method), 296
cohomology_class() (sage.schemes.toric.divisor.ToricDivisor_generic method), 282
cohomology_ring() (sage.schemes.toric.variety.ToricVariety_field method), 225
cohomology_support() (sage.schemes.toric.divisor.ToricDivisor_generic method), 282
CohomologyClass (class in sage.schemes.toric.variety), 212
CohomologyRing (class in sage.schemes.toric.variety), 213
complement() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 31
cone_iter() (sage.schemes.toric.points.NaiveFinitePointEnumerator method), 355
cone_points_iter() (sage.schemes.toric.points.FiniteFieldPointEnumerator method), 349
Conifold() (sage.schemes.toric.library.ToricVarietyFactory method), 261
conjugate() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space

method), 123
connected_rational_component() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space_field method), 149
coordinate_iter() (sage.schemes.toric.points.NaiveFinitePointEnumerator method), 355
coordinate_point_to_coordinate() (sage.schemes.toric.fano_variety.CPRFanoToricVariety_field method), 253
coordinate_points() (sage.schemes.toric.fano_variety.CPRFanoToricVariety_field method), 253
coordinate_ring() (sage.schemes.affine.affine_space.AffineSpace_generic method), 77
coordinate_ring() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme method), 25
coordinate_ring() (sage.schemes.generic.morphism.SchemeMorphism_polynomial method), 66
coordinate_ring() (sage.schemes.generic.scheme.AffineScheme method), 3
coordinate_ring() (sage.schemes.generic.scheme.Scheme method), 7
coordinate_ring() (sage.schemes.projective.projective_space.ProjectiveSpace_ring method), 107
coordinate_ring() (sage.schemes.toric.variety.ToricVariety_field method), 226
coordinate_vector() (sage.schemes.toric.chow_group.ChowGroup_class method), 300
count_points() (sage.schemes.generic.scheme.Scheme method), 7
count_points() (sage.schemes.toric.chow_group.ChowCycle method), 297
count_points() (sage.schemes.toric.variety.ToricVariety_field method), 226
CPRFanoToricVariety() (in module sage.schemes.toric.fano_variety), 243
CPRFanoToricVariety_field (class in sage.schemes.toric.fano_variety), 248
create_key_and_extra_args() (sage.schemes.generic.homset.SchemeHomsetFactory method), 54
create_key_and_extra_args() (sage.schemes.toric.chow_group.ChowGroupFactory method), 299
create_object() (sage.schemes.generic.homset.SchemeHomsetFactory method), 54
create_object() (sage.schemes.toric.chow_group.ChowGroupFactory method), 299
critical_height() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 124
critical_point_portrait() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 125
critical_points() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 126
CRT_automorphisms() (in module sage.schemes.projective.endPN_automorphism_group), 165
CRT_helper() (in module sage.schemes.projective.endPN_automorphism_group), 166
Cube_deformation() (sage.schemes.toric.library.ToricVarietyFactory method), 262
Cube_face_fan() (sage.schemes.toric.library.ToricVarietyFactory method), 263
Cube_nonpolyhedral() (sage.schemes.toric.library.ToricVarietyFactory method), 263
Cube_sublattice() (sage.schemes.toric.library.ToricVarietyFactory method), 264
CurvePointToIdeal() (in module sage.schemes.generic.divisor), 69
cyclegraph() (sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space_finite_field method), 94
cyclegraph() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space_finite_field method), 157

D

defining_cone() (sage.schemes.toric.morphism.SchemeMorphism_fan_fiber_component_toric_variety method), 315
defining_cone() (sage.schemes.toric.morphism.SchemeMorphism_orbit_closure_toric_variety method), 325
defining_ideal() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 32
defining_polynomial() (sage.schemes.generic.hypersurface.AffineHypersurface method), 51
defining_polynomial() (sage.schemes.generic.hypersurface.ProjectiveHypersurface method), 51
defining_polynomials() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 32
defining_polynomials() (sage.schemes.generic.ambient_space.AmbientSpace method), 20
defining_polynomials() (sage.schemes.generic.morphism.SchemeMorphism_polynomial method), 66

deg() (sage.schemes.toric.variety.CohomologyClass method), 212
degenerate_fibers() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 194
degenerate_primes() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 194
degree() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 126
degree() (sage.schemes.toric.chow_group.ChowCycle method), 297
degree() (sage.schemes.toric.chow_group.ChowGroup_class method), 300
dehomogenize() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 127
dehomogenize() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring method), 112
Delta() (sage.schemes.toric.fano_variety.CPRFanoToricVariety_field method), 249
Delta_polar() (sage.schemes.toric.fano_variety.CPRFanoToricVariety_field method), 249
Demazure_roots() (sage.schemes.toric.variety.ToricVariety_field method), 218
dimension() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_affine method), 36
dimension() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_affine_toric method), 39
dimension() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_product_projective method), 41
dimension() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_projective method), 43
dimension() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_toric method), 47
dimension() (sage.schemes.generic.ambient_space.AmbientSpace method), 20
dimension() (sage.schemes.generic.scheme.AffineScheme method), 4
dimension() (sage.schemes.generic.scheme.Scheme method), 7
dimension() (sage.schemes.product_projective.space.ProductProjectiveSpaces_ring method), 181
dimension_absolute() (sage.schemes.generic.ambient_space.AmbientSpace method), 20
dimension_absolute() (sage.schemes.generic.scheme.AffineScheme method), 4
dimension_absolute() (sage.schemes.generic.scheme.Scheme method), 8
dimension_absolute() (sage.schemes.product_projective.space.ProductProjectiveSpaces_ring method), 181
dimension_absolute_components() (sage.schemes.product_projective.space.ProductProjectiveSpaces_ring method), 181
dimension_components() (sage.schemes.product_projective.space.ProductProjectiveSpaces_ring method), 182
dimension_relative() (sage.schemes.generic.ambient_space.AmbientSpace method), 21
dimension_relative() (sage.schemes.generic.scheme.AffineScheme method), 4
dimension_relative() (sage.schemes.generic.scheme.Scheme method), 8
dimension_relative() (sage.schemes.product_projective.space.ProductProjectiveSpaces_ring method), 182
dimension_relative_components() (sage.schemes.product_projective.space.ProductProjectiveSpaces_ring method), 182
dimension_singularities() (sage.schemes.toric.variety.ToricVariety_field method), 226
Discriminant() (in module sage.schemes.toric.weierstrass), 329
divisor() (sage.schemes.toric.variety.ToricVariety_field method), 227
divisor_class() (sage.schemes.toric.divisor.ToricDivisor_generic method), 283
Divisor_curve (class in sage.schemes.generic.divisor), 69
Divisor_generic (class in sage.schemes.generic.divisor), 71
divisor_group() (sage.schemes.toric.variety.ToricVariety_field method), 227
DivisorGroup() (in module sage.schemes.generic.divisor_group), 73
DivisorGroup_curve (class in sage.schemes.generic.divisor_group), 73
DivisorGroup_generic (class in sage.schemes.generic.divisor_group), 73
dP6() (sage.schemes.toric.library.ToricVarietyFactory method), 270
dP6xdP6() (sage.schemes.toric.library.ToricVarietyFactory method), 271
dP7() (sage.schemes.toric.library.ToricVarietyFactory method), 271
dP8() (sage.schemes.toric.library.ToricVarietyFactory method), 272
dynatomic_polynomial() (sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space

method), 84
dynatomic_polynomial() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 128

E

Element (sage.schemes.generic.homset.SchemeHomset_generic attribute), 55
Element (sage.schemes.generic.scheme.AffineScheme attribute), 3
Element (sage.schemes.toric.chow_group.ChowGroup_class attribute), 300
Element (sage.schemes.toric.divisor.ToricRationalDivisorClassGroup attribute), 291
Element (sage.schemes.toric.divisor.ToricRationalDivisorClassGroup_basis_lattice attribute), 292
embedding_center() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme method), 25
embedding_morphism() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme method), 26
embedding_morphism() (sage.schemes.toric.variety.ToricVariety_field method), 228
embedding_of_affine_open() (sage.schemes.generic.point.SchemeTopologicalPoint_affine_open method), 17
enum_affine_finite_field() (in module sage.schemes.affine.affine_rational_point), 95
enum_affine_number_field() (in module sage.schemes.affine.affine_rational_point), 96
enum_affine_rational_field() (in module sage.schemes.affine.affine_rational_point), 96
enum_projective_finite_field() (in module sage.schemes.projective.projective_rational_point), 159
enum_projective_number_field() (in module sage.schemes.projective.projective_rational_point), 160
enum_projective_rational_field() (in module sage.schemes.projective.projective_rational_point), 161
Euler_number() (sage.schemes.toric.variety.ToricVariety_field method), 218
exp() (sage.schemes.toric.points.FiniteFieldPointEnumerator method), 350
exp() (sage.schemes.toric.variety.CohomologyClass method), 212
extended_codomain() (sage.schemes.generic.homset.SchemeHomset_points method), 56

F

factor() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety method), 317
fan() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_toric method), 47
fan() (sage.schemes.toric.variety.ToricVariety_field method), 228
fan_morphism() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety method), 318
fiber() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 195
fiber_component() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety_dominant method), 321
fiber_dimension() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety_dominant method), 322
fiber_generic() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety_dominant method), 323
fiber_graph() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety_dominant method), 323
field_descent() (in module sage.schemes.projective.endPN_automorphism_group), 171
FiniteFieldPointEnumerator (class in sage.schemes.toric.points), 349
FiniteFieldSubschemePointEnumerator (class in sage.schemes.toric.points), 352
function_value() (sage.schemes.toric.divisor.ToricDivisor_generic method), 283

G

gen() (sage.schemes.generic.ambient_space.AmbientSpace method), 21
gen() (sage.schemes.toric.chow_group.ChowGroup_degree_class method), 303
gen() (sage.schemes.toric.divisor.ToricDivisorGroup method), 277
gen() (sage.schemes.toric.variety.CohomologyRing method), 214
gens() (sage.schemes.generic.ambient_space.AmbientSpace method), 21
gens() (sage.schemes.toric.chow_group.ChowGroup_class method), 302
gens() (sage.schemes.toric.chow_group.ChowGroup_degree_class method), 303
gens() (sage.schemes.toric.divisor.ToricDivisorGroup method), 277
gens() (sage.schemes.toric.variety.CohomologyRing method), 214

global_height() (sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space method), 85
global_height() (sage.schemes.affine.affine_point.SchemeMorphism_point_affine method), 80
global_height() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 131
global_height() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring method), 113
glue_along_domains() (sage.schemes.generic.morphism.SchemeMorphism method), 60
GluedScheme (class in sage.schemes.generic.glue), 15
gluing_maps() (sage.schemes.generic.glue.GluedScheme method), 15
Gpoly() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 188
green_function() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 132
green_function() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring method), 113

H

height_bound() (in module sage.schemes.projective.endPN_automorphism_group), 171
height_difference_bound() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 132
hom() (sage.schemes.generic.scheme.AffineScheme method), 4
hom() (sage.schemes.generic.scheme.Scheme method), 8
homogeneous_coordinates() (sage.schemes.toric.points.FiniteFieldSubschemePointEnumerator method), 353
homogenize() (sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space method), 86
homogenize() (sage.schemes.affine.affine_point.SchemeMorphism_point_affine method), 81
Hpoly() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 188

I

identity_morphism() (sage.schemes.generic.scheme.Scheme method), 9
InfinitePointEnumerator (class in sage.schemes.toric.points), 354
inhomogeneous_equations() (sage.schemes.toric.points.FiniteFieldSubschemePointEnumerator method), 353
inject_coefficients() (sage.schemes.toric.variety.ToricVariety_field method), 229
integrate() (sage.schemes.toric.variety.ToricVariety_field method), 229
intersection() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 32
intersection_with_divisor() (sage.schemes.toric.chow_group.ChowCycle method), 298
irreducible_components() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 32
is_affine() (sage.schemes.toric.variety.ToricVariety_field method), 230
is_AffineScheme() (in module sage.schemes.generic.scheme), 11
is_AffineSpace() (in module sage.schemes.affine.affine_space), 79
is_AlgebraicScheme() (in module sage.schemes.generic.algebraic_scheme), 49
is_AmbientSpace() (in module sage.schemes.generic.ambient_space), 21
is_ample() (sage.schemes.toric.divisor.ToricDivisor_generic method), 284
is_birational() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety method), 318
is_bundle() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety method), 319
is_Cartier() (sage.schemes.toric.divisor.ToricDivisor_generic method), 283
is_ChowCycle() (in module sage.schemes.toric.chow_group), 304
is_ChowGroup() (in module sage.schemes.toric.chow_group), 304
is_CohomologyClass() (in module sage.schemes.toric.variety), 238
is_complete() (sage.schemes.toric.variety.ToricVariety_field method), 230
is_CPRFanoToricVariety() (in module sage.schemes.toric.fano_variety), 258
is_degenerate() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 196
is_Divisor() (in module sage.schemes.generic.divisor), 71
is_DivisorGroup() (in module sage.schemes.generic.divisor_group), 74

is_dominant() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety method), 319
is_endomorphism() (sage.schemes.generic.morphism.SchemeMorphism method), 61
is_fibration() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety method), 319
is_finite() (sage.schemes.toric.homset.SchemeHomset_points_toric_base method), 346
is_homogeneous() (sage.schemes.toric.variety.ToricVariety_field method), 230
is_Hypersurface() (in module sage.schemes.generic.hypersurface), 52
is_injective() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety method), 319
is_integral() (sage.schemes.toric.divisor.ToricDivisor_generic method), 286
is_isomorphic() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 196
is_isomorphic() (sage.schemes.toric.variety.ToricVariety_field method), 231
is_morphism() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 134
is_nef() (sage.schemes.toric.divisor.ToricDivisor_generic method), 286
is_noetherian() (sage.schemes.generic.scheme.AffineScheme method), 5
is_orbifold() (sage.schemes.toric.variety.ToricVariety_field method), 232
is_PGL_minimal() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 133
is_postcritically_finite() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 134
is_preperiodic() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring method), 114
is_ProductProjectiveSpaces() (in module sage.schemes.product_projective.space), 184
is_projective() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme method), 27
is_projective() (sage.schemes.generic.ambient_space.AmbientSpace method), 21
is_projective() (sage.schemes.projective.projective_space.ProjectiveSpace_ring method), 107
is_ProjectiveSpace() (in module sage.schemes.projective.projective_space), 108
is_QQ_Cartier() (sage.schemes.toric.divisor.ToricDivisor_generic method), 284
is_QQ_Weil() (sage.schemes.toric.divisor.ToricDivisor_generic method), 284
is_Scheme() (in module sage.schemes.generic.scheme), 12
is_SchemeHomset() (in module sage.schemes.generic.homset), 57
is_SchemeMorphism() (in module sage.schemes.generic.morphism), 67
is_SchemeRationalPoint() (in module sage.schemes.generic.point), 18
is_SchemeTopologicalPoint() (in module sage.schemes.generic.point), 18
is_smooth() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_affine method), 37
is_smooth() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_affine_toric method), 39
is_smooth() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_product_projective method), 41
is_smooth() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_projective method), 44
is_smooth() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_toric method), 48
is_smooth() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 197
is_smooth() (sage.schemes.toric.variety.ToricVariety_field method), 232
is_surjective() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety method), 320
is_symmetric_orbit() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 197
is_ToricDivisor() (in module sage.schemes.toric.divisor), 292
is_ToricRationalDivisorClass() (in module sage.schemes.toric.divisor_class), 293
is_ToricVariety() (in module sage.schemes.toric.variety), 239
is_Weil() (sage.schemes.toric.divisor.ToricDivisor_generic method), 284

J

j_invariant() (in module sage.schemes.toric.weierstrass), 336
jacobian() (sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space method), 87
Jacobian() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 29

Jacobian_matrix() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 30

K

K() (sage.schemes.toric.variety.ToricVariety_field method), 218
Kaehler_cone() (sage.schemes.toric.variety.ToricVariety_field method), 219
ker() (sage.schemes.toric.ideal.ToricIdeal method), 307
Kodaira_map() (sage.schemes.toric.divisor.ToricDivisor_generic method), 279

L

lambda_minus() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 198
lambda_plus() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 198
Lattes_map() (sage.schemes.projective.projective_space.ProjectiveSpace_ring method), 105
lift() (sage.schemes.toric.divisor_class.ToricRationalDivisorClass method), 293
lift_to_rational_periodic() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space_field method), 150
linear_equivalence_ideal() (sage.schemes.toric.variety.ToricVariety_field method), 232
list() (sage.schemes.generic.homset.SchemeHomset_points method), 56
local_height() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 135
local_height() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring method), 116
local_height_arch() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 136
local_height_arch() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring method), 116
log() (sage.schemes.toric.points.FiniteFieldPointEnumerator method), 350
Lxa() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 188
Lyb() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 189

M

m() (sage.schemes.toric.divisor.ToricDivisor_generic method), 287
Min() (in module sage.schemes.projective.endPN_minimal_model), 176
minimal_model() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 136
module() (sage.schemes.toric.chow_group.ChowGroup_degree_class method), 303
monomial() (sage.schemes.toric.divisor.ToricDivisor_generic method), 287
Mori_cone() (sage.schemes.toric.variety.ToricVariety_field method), 219
morphism() (sage.schemes.generic.point.SchemeRationalPoint method), 17
move_away_from() (sage.schemes.toric.divisor.ToricDivisor_generic method), 288
multiplicative_generator() (sage.schemes.toric.points.FiniteFieldPointEnumerator method), 350
multiplicative_group_order() (sage.schemes.toric.points.FiniteFieldPointEnumerator method), 351
multiplier() (sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space method), 88
multiplier() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 138
multiplier() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring method), 117
multiplier_spectra() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 139

N

NaiveFinitePointEnumerator (class in sage.schemes.toric.points), 355
NaiveSubschemePointEnumerator (class in sage.schemes.toric.points), 357
natural_map() (sage.schemes.generic.homset.SchemeHomset_generic method), 55

nef_complete_intersection() (sage.schemes.toric.fano_variety.CPRFanoToricVariety_field method), 254
nef_partition() (sage.schemes.toric.fano_variety.NefCompleteIntersection method), 257
NefCompleteIntersection (class in sage.schemes.toric.fano_variety), 256
neighborhood() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_projective method), 44
neighborhood() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_toric method), 48
Newton_polygon_embedded() (in module sage.schemes.toric.weierstrass), 329
Newton_polytope_vars_coeffs() (in module sage.schemes.toric.weierstrass), 330
ngens() (sage.schemes.affine.affine_space.AffineSpace_generic method), 78
ngens() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme method), 28
ngens() (sage.schemes.generic.ambient_space.AmbientSpace method), 21
ngens() (sage.schemes.product_projective.space.ProductProjectiveSpaces_ring method), 182
ngens() (sage.schemes.projective.projective_space.ProjectiveSpace_ring method), 107
ngens() (sage.schemes.toric.chow_group.ChowGroup_degree_class method), 304
ngens() (sage.schemes.toric.divisor.ToricDivisorGroup method), 278
normalize_coordinates() (sage.schemes.product_projective.point.ProductProjectiveSpaces_point_ring method), 186
normalize_coordinates() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 139
normalize_coordinates() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_field method), 109
normalize_coordinates() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring method), 117
normalize_names() (in module sage.schemes.toric.variety), 239
nth_iterate() (sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space method), 89
nth_iterate() (sage.schemes.affine.affine_point.SchemeMorphism_point_affine method), 81
nth_iterate() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 140
nth_iterate() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring method), 118
nth_iterate_map() (sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space method), 89
nth_iterate_map() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 141
nth_iterate_phi() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 199
nth_iterate_psi() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 199
num_components() (sage.schemes.product_projective.space.ProductProjectiveSpaces_ring method), 182
nvariables() (sage.schemes.toric.ideal.ToricIdeal method), 308

O

orbit() (sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space method), 90
orbit() (sage.schemes.affine.affine_point.SchemeMorphism_point_affine method), 82
orbit() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 142
orbit() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring method), 119
orbit() (sage.schemes.toric.points.NaiveFinitePointEnumerator method), 356
orbit_closure() (sage.schemes.toric.variety.ToricVariety_field method), 232
orbit_phi() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 200
orbit_psi() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 201
orbit_structure() (sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space_finite_field method), 94
orbit_structure() (sage.schemes.affine.affine_point.SchemeMorphism_point_affine_finite_field method), 83
orbit_structure() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space_finite_field method), 157
orbit_structure() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_finite_field method),

110
order_p_automorphisms() (in module sage.schemes.projective.endPN_automorphism_group), 171

P

P() (sage.schemes.toric.library.ToricVarietyFactory method), 264
P1() (sage.schemes.toric.library.ToricVarietyFactory method), 265
P1xA1() (sage.schemes.toric.library.ToricVarietyFactory method), 265
P1xP1() (sage.schemes.toric.library.ToricVarietyFactory method), 266
P1xP1_Z2() (sage.schemes.toric.library.ToricVarietyFactory method), 266
P2() (sage.schemes.toric.library.ToricVarietyFactory method), 266
P2_112() (sage.schemes.toric.library.ToricVarietyFactory method), 267
P2_123() (sage.schemes.toric.library.ToricVarietyFactory method), 267
P4_11133() (sage.schemes.toric.library.ToricVarietyFactory method), 268
P4_11133_resolved() (sage.schemes.toric.library.ToricVarietyFactory method), 268
P4_11169() (sage.schemes.toric.library.ToricVarietyFactory method), 269
P4_11169_resolved() (sage.schemes.toric.library.ToricVarietyFactory method), 269
part_of_degree() (sage.schemes.toric.variety.CohomologyClass method), 213
periodic_points() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 143
PGL_order() (in module sage.schemes.projective.endPN_automorphism_group), 166
PGL_repn() (in module sage.schemes.projective.endPN_automorphism_group), 167
phi() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 201
plot() (sage.schemes.toric.variety.ToricVariety_field method), 233
point() (sage.schemes.generic.scheme.Scheme method), 9
point_homset() (sage.schemes.generic.scheme.Scheme method), 9
point_on_affine() (sage.schemes.generic.point.SchemeTopologicalPoint_affine_open method), 17
point_set() (sage.schemes.generic.scheme.Scheme method), 10
points() (sage.schemes.affine.affine_homset.SchemeHomset_points_affine method), 98
points() (sage.schemes.projective.projective_homset.SchemeHomset_points_projective_field method), 163
points() (sage.schemes.projective.projective_homset.SchemeHomset_points_projective_ring method), 165
points_of_bounded_height() (sage.schemes.affine.affine_space.AffineSpace_field method), 75
points_of_bounded_height() (sage.schemes.projective.projective_space.ProjectiveSpace_field method), 103
polyhedron() (sage.schemes.toric.divisor.ToricDivisor_generic method), 288
possible_periods() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 144
possible_periods() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space_finite_field method), 158
prime_ideal() (sage.schemes.generic.point.SchemeTopologicalPoint_prime_ideal method), 18
primes_of_bad_reduction() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 145
ProductProjectiveSpaces() (in module sage.schemes.product_projective.space), 179
ProductProjectiveSpaces_morphism_ring (class in sage.schemes.product_projective.morphism), 185
ProductProjectiveSpaces_point_ring (class in sage.schemes.product_projective.point), 185
ProductProjectiveSpaces_ring (class in sage.schemes.product_projective.space), 180
project_to_degree() (sage.schemes.toric.chow_group.ChowCycle method), 299
projective_embedding() (sage.schemes.affine.affine_space.AffineSpace_generic method), 78
projective_embedding() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_affine method), 37
ProjectiveHypersurface (class in sage.schemes.generic.hypersurface), 51
ProjectiveSpace() (in module sage.schemes.projective.projective_space), 102
ProjectiveSpace_field (class in sage.schemes.projective.projective_space), 102

ProjectiveSpace_finite_field (class in sage.schemes.projective.projective_space), 103
ProjectiveSpace_rational_field (class in sage.schemes.projective.projective_space), 104
ProjectiveSpace_ring (class in sage.schemes.projective.projective_space), 105
psi() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 202
pullback_divisor() (sage.schemes.toric.morphism.SchemeMorphism_fan_fiber_component_toric_variety method), 315
pullback_divisor() (sage.schemes.toric.morphism.SchemeMorphism_fan_toric_variety method), 320
pullback_divisor() (sage.schemes.toric.morphism.SchemeMorphism_orbit_closure_toric_variety method), 326

Q

Qxa() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 189
Qyb() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 190

R

Ramification_poly() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 190
random_WehlerK3Surface() (in module sage.schemes.product_projective.wehlerK3), 204
rational_class_group() (sage.schemes.toric.variety.ToricVariety_field method), 234
rational_function_coefficient_descent() (in module sage.schemes.projective.endPN_automorphism_group), 172
rational_function_coerce() (in module sage.schemes.projective.endPN_automorphism_group), 173
rational_function_reduce() (in module sage.schemes.projective.endPN_automorphism_group), 173
rational_periodic_points() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space_field method), 151
rational_points() (sage.schemes.affine.affine_space.AffineSpace_generic method), 78
rational_points() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_quasi method), 29
rational_points() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 33
rational_points() (sage.schemes.projective.projective_space.ProjectiveSpace_finite_field method), 103
rational_points() (sage.schemes.projective.projective_space.ProjectiveSpace_rational_field method), 104
rational_points_dictionary() (sage.schemes.projective.projective_space.ProjectiveSpace_finite_field method), 104
rational_preimages() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space_field method), 152
rational_preperiodic_graph() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space_field method), 153
rational_preperiodic_points() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space_field method), 154
rays() (sage.schemes.toric.points.NaiveFinitePointEnumerator method), 356
reduce() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 34
relation_gens() (sage.schemes.toric.chow_group.ChowGroup_class method), 302
remove_redundant_automorphisms() (in module sage.schemes.projective.endPN_automorphism_group), 173
rescaling_log_generators() (sage.schemes.toric.points.FiniteFieldPointEnumerator method), 351
rescalings() (sage.schemes.toric.points.NaiveFinitePointEnumerator method), 356
resolve() (sage.schemes.toric.fano_variety.CPRFanoToricVariety_field method), 255
resolve() (sage.schemes.toric.variety.ToricVariety_field method), 234
resolve_to_orbifold() (sage.schemes.toric.variety.ToricVariety_field method), 235
resultant() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 146
ring_homomorphism() (sage.schemes.generic.morphism.SchemeMorphism_spec method), 67
root_generator() (sage.schemes.toric.points.FiniteFieldPointEnumerator method), 351
roots() (sage.schemes.toric.points.NaiveFinitePointEnumerator method), 357

S

sage.schemes.affine.affine_homset (module), 97

sage.schemes.affine.affine_morphism (module), 84
sage.schemes.affine.affine_point (module), 80
sage.schemes.affine.affine_rational_point (module), 95
sage.schemes.affine.affine_space (module), 75
sage.schemes.generic.algebraic_scheme (module), 23
sage.schemes.generic.ambient_space (module), 19
sage.schemes.generic.divisor (module), 69
sage.schemes.generic.divisor_group (module), 73
sage.schemes.generic.glue (module), 15
sage.schemes.generic.homset (module), 53
sage.schemes.generic.hypersurface (module), 51
sage.schemes.generic.morphism (module), 59
sage.schemes.generic.point (module), 17
sage.schemes.generic.scheme (module), 3
sage.schemes.generic.spec (module), 13
sage.schemes.product_projective.homset (module), 184
sage.schemes.product_projective.morphism (module), 185
sage.schemes.product_projective.point (module), 185
sage.schemes.product_projective.space (module), 179
sage.schemes.product_projective.wehlerK3 (module), 187
sage.schemes.projective.endPN_automorphism_group (module), 165
sage.schemes.projective.endPN_minimal_model (module), 175
sage.schemes.projective.projective_homset (module), 162
sage.schemes.projective.projective_morphism (module), 120
sage.schemes.projective.projective_morphism_helper (module), 159
sage.schemes.projective.projective_point (module), 108
sage.schemes.projective.projective_rational_point (module), 159
sage.schemes.projective.projective_space (module), 101
sage.schemes.readme (module), 1
sage.schemes.toric.chow_group (module), 294
sage.schemes.toric.divisor (module), 273
sage.schemes.toric.divisor_class (module), 292
sage.schemes.toric.fano_variety (module), 241
sage.schemes.toric.homset (module), 343
sage.schemes.toric.ideal (module), 305
sage.schemes.toric.library (module), 258
sage.schemes.toric.morphism (module), 308
sage.schemes.toric.points (module), 348
sage.schemes.toric.variety (module), 207
sage.schemes.toric.weierstrass (module), 328
sage.schemes.toric.weierstrass_covering (module), 337
sage.schemes.toric.weierstrass_higher (module), 341
scale() (in module sage.schemes.projective.endPN_minimal_model), 178
scale_by() (sage.schemes.product_projective.point.ProductProjectiveSpaces_point_ring method), 186
scale_by() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 147
scale_by() (sage.schemes.projective.projective_point.SchemeMorphism_point_projective_ring method), 120
Scheme (class in sage.schemes.generic.scheme), 5
scheme() (sage.schemes.generic.divisor.Divisor_generic method), 71
scheme() (sage.schemes.generic.divisor_group.DivisorGroup_generic method), 74

scheme() (sage.schemes.generic.morphism.SchemeMorphism_point method), 63
scheme() (sage.schemes.generic.point.SchemePoint method), 17
scheme() (sage.schemes.toric.chow_group.ChowGroup_class method), 302
SchemeHomset_generic (class in sage.schemes.generic.homset), 55
SchemeHomset_points (class in sage.schemes.generic.homset), 55
SchemeHomset_points_abelian_variety_field (class in sage.schemes.projective.projective_homset), 162
SchemeHomset_points_affine (class in sage.schemes.affine.affine_homset), 97
SchemeHomset_points_product_projective_spaces_ring (class in sage.schemes.product_projective.homset), 184
SchemeHomset_points_projective_field (class in sage.schemes.projective.projective_homset), 163
SchemeHomset_points_projective_ring (class in sage.schemes.projective.projective_homset), 164
SchemeHomset_points_spec (class in sage.schemes.affine.affine_homset), 98
SchemeHomset_points_subscheme_toric_field (class in sage.schemes.toric.homset), 344
SchemeHomset_points_toric_base (class in sage.schemes.toric.homset), 345
SchemeHomset_points_toric_field (class in sage.schemes.toric.homset), 346
SchemeHomset_toric_variety (class in sage.schemes.toric.homset), 348
SchemeHomsetFactory (class in sage.schemes.generic.homset), 53
SchemeMorphism (class in sage.schemes.generic.morphism), 59
SchemeMorphism_fan_fiber_component_toric_variety (class in sage.schemes.toric.morphism), 313
SchemeMorphism_fan_toric_variety (class in sage.schemes.toric.morphism), 315
SchemeMorphism_fan_toric_variety_dominant (class in sage.schemes.toric.morphism), 321
SchemeMorphism_id (class in sage.schemes.generic.morphism), 61
SchemeMorphism_orbit_closure_toric_variety (class in sage.schemes.toric.morphism), 324
SchemeMorphism_point (class in sage.schemes.generic.morphism), 62
SchemeMorphism_point_abelian_variety_field (class in sage.schemes.projective.projective_point), 108
SchemeMorphism_point_affine (class in sage.schemes.affine.affine_point), 80
SchemeMorphism_point_affine_field (class in sage.schemes.affine.affine_point), 82
SchemeMorphism_point_affine_finite_field (class in sage.schemes.affine.affine_point), 83
SchemeMorphism_point_projective_field (class in sage.schemes.projective.projective_point), 109
SchemeMorphism_point_projective_finite_field (class in sage.schemes.projective.projective_point), 110
SchemeMorphism_point_projective_ring (class in sage.schemes.projective.projective_point), 111
SchemeMorphism_point_toric_field (class in sage.schemes.toric.morphism), 326
SchemeMorphism_polynomial (class in sage.schemes.generic.morphism), 63
SchemeMorphism_polynomial_affine_space (class in sage.schemes.affine.affine_morphism), 84
SchemeMorphism_polynomial_affine_space_field (class in sage.schemes.affine.affine_morphism), 91
SchemeMorphism_polynomial_affine_space_finite_field (class in sage.schemes.affine.affine_morphism), 92
SchemeMorphism_polynomial_projective_space (class in sage.schemes.projective.projective_morphism), 120
SchemeMorphism_polynomial_projective_space_field (class in sage.schemes.projective.projective_morphism), 148
SchemeMorphism_polynomial_projective_space_finite_field (class in sage.schemes.projective.projective_morphism), 155
SchemeMorphism_polynomial_toric_variety (class in sage.schemes.toric.morphism), 327
SchemeMorphism_spec (class in sage.schemes.generic.morphism), 66
SchemeMorphism_structure_map (class in sage.schemes.generic.morphism), 67
SchemePoint (class in sage.schemes.generic.point), 17
SchemeRationalPoint (class in sage.schemes.generic.point), 17
SchemeTopologicalPoint (class in sage.schemes.generic.point), 17
SchemeTopologicalPoint_affine_open (class in sage.schemes.generic.point), 17
SchemeTopologicalPoint_prime_ideal (class in sage.schemes.generic.point), 17
sections() (sage.schemes.toric.divisor.ToricDivisor_generic method), 289
sections_monomials() (sage.schemes.toric.divisor.ToricDivisor_generic method), 290
segre_embedding() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme_product_projective

method), 41
segre_embedding() (sage.schemes.product_projective.space.ProductProjectiveSpaces_ring method), 183
sigma_invariants() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 147
sigmaX() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 202
sigmaY() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 203
solutions() (sage.schemes.toric.points.FiniteFieldSubschemePointEnumerator method), 353
solutions_serial() (sage.schemes.toric.points.FiniteFieldSubschemePointEnumerator method), 354
Spec() (in module sage.schemes.generic.spec), 13
Spec() (sage.schemes.toric.variety.ToricVariety_field method), 220
SpecFunctor (class in sage.schemes.generic.spec), 14
Stanley_Reisner_ideal() (sage.schemes.toric.variety.ToricVariety_field method), 220
structure_morphism() (sage.schemes.generic.scheme.Scheme method), 10
subscheme() (sage.schemes.affine.affine_space.AffineSpace_generic method), 79
subscheme() (sage.schemes.product_projective.space.ProductProjectiveSpaces_ring method), 183
subscheme() (sage.schemes.projective.projective_space.ProjectiveSpace_ring method), 107
subscheme() (sage.schemes.toric.variety.ToricVariety_field method), 236
support() (sage.schemes.generic.divisor.Divisor_curve method), 70
Sxa() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 190
Syb() (sage.schemes.product_projective.wehlerK3.WehlerK3Surface_ring method), 191

T

Td() (sage.schemes.toric.variety.ToricVariety_field method), 220
three_stable_points() (in module sage.schemes.projective.endPN_automorphism_group), 174
Todd_class() (sage.schemes.toric.variety.ToricVariety_field method), 221
toric_divisor_group() (sage.schemes.toric.variety.ToricVariety_field method), 236
ToricDivisor() (in module sage.schemes.toric.divisor), 275
ToricDivisor_generic (class in sage.schemes.toric.divisor), 278
ToricDivisorGroup (class in sage.schemes.toric.divisor), 276
ToricDivisorGroup.Element (class in sage.schemes.toric.divisor), 276
ToricIdeal (class in sage.schemes.toric.ideal), 306
ToricRationalDivisorClass (class in sage.schemes.toric.divisor_class), 293
ToricRationalDivisorClassGroup (class in sage.schemes.toric.divisor), 290
ToricRationalDivisorClassGroup_basis_lattice (class in sage.schemes.toric.divisor), 291
ToricVariety() (in module sage.schemes.toric.variety), 214
ToricVariety_field (class in sage.schemes.toric.variety), 215
ToricVarietyFactory (class in sage.schemes.toric.library), 259
torus() (sage.schemes.toric.library.ToricVarietyFactory method), 272

U

union() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 34
union() (sage.schemes.generic.scheme.Scheme method), 11
units() (sage.schemes.toric.points.NaiveFinitePointEnumerator method), 357

V

valid_automorphisms() (in module sage.schemes.projective.endPN_automorphism_group), 174
value_ring() (sage.schemes.generic.homset.SchemeHomset_points method), 56
volume_class() (sage.schemes.toric.variety.ToricVariety_field method), 237

W

WehlerK3Surface() (in module sage.schemes.product_projective.wehlerK3), 187
WehlerK3Surface_field (class in sage.schemes.product_projective.wehlerK3), 187
WehlerK3Surface_finite_field (class in sage.schemes.product_projective.wehlerK3), 187
WehlerK3Surface_ring (class in sage.schemes.product_projective.wehlerK3), 187
WeierstrassForm() (in module sage.schemes.toric.weierstrass), 331
WeierstrassForm2() (in module sage.schemes.toric.weierstrass_higher), 341
WeierstrassForm_P1xP1() (in module sage.schemes.toric.weierstrass), 333
WeierstrassForm_P2() (in module sage.schemes.toric.weierstrass), 334
WeierstrassForm_P2_112() (in module sage.schemes.toric.weierstrass), 335
WeierstrassForm_P3() (in module sage.schemes.toric.weierstrass_higher), 342
WeierstrassMap() (in module sage.schemes.toric.weierstrass_covering), 338
WeierstrassMap_P1xP1() (in module sage.schemes.toric.weierstrass_covering), 339
WeierstrassMap_P2() (in module sage.schemes.toric.weierstrass_covering), 340
WeierstrassMap_P2_112() (in module sage.schemes.toric.weierstrass_covering), 340
WeierstrassMap_P3() (in module sage.schemes.toric.weierstrass_higher), 342
weil_restriction() (sage.schemes.affine.affine_morphism.SchemeMorphism_polynomial_affine_space_field method), 92
weil_restriction() (sage.schemes.affine.affine_point.SchemeMorphism_point_affine_field method), 82
weil_restriction() (sage.schemes.affine.affine_space.AffineSpace_field method), 76
weil_restriction() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_subscheme method), 35
which_group() (in module sage.schemes.projective.endPN_automorphism_group), 175
WP() (sage.schemes.toric.library.ToricVarietyFactory method), 270
wronskian_ideal() (sage.schemes.projective.projective_morphism.SchemeMorphism_polynomial_projective_space method), 148

X

X() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_quasi method), 28

Y

Y() (sage.schemes.generic.algebraic_scheme.AlgebraicScheme_quasi method), 28

Z

zeta_series() (sage.schemes.generic.scheme.Scheme method), 11