Sage Reference Manual: Chow

Release 6.4.1

The Sage Development Team

January 10, 2015

CONTENTS

CHAPTER

ONE

CHOW RINGS AND INTERSECTION THEORY ON SCHEMES

Let X be a non singular algebraic variety over an algebraically closed field k.

An *algebraic cycle* of codimension p is a formal linear combination of algebraic subvarieties of codimension p in X; the group of such cycles is generally denoted by $Z^p(X)$. Two algebraic cycles of codimension p are said to be *rationally equivalent* if their difference is in the subgroup $R^p(X) \subset Z^p(X)$ generated by the cycles $div(f)_W$ where $f \in k(W)^*$ and W runs over the codimension $p+1$ subvarieties of X.

The group $A^p(X) = Z^p(X)/R^p(X)$ of algebraic cycles modulo rational equivalence is called the *Chow group of codimension p cycles*. Note that $A^0(X) = \mathbf{Z}$ and $A^p(X) = 0$ for $p > \dim(X)$. Classical Schubert calculus on Grassmannians generalizes to the intersection product

$$A^p(X) \otimes_{\mathbf{Z}} A^q(X) \longrightarrow A^{p+q}(X)$$

which makes $A^*(X) = \oplus_p A^p(X)$ into an associative, commutative and graded ring.

If X is moreover projective, then there is a well defined *degree* morphism $\int : A^{\dim(X)}(X) \to \mathbf{Z}$ given by $\int(\sum m_i P_i) = \sum m_i$, allowing to define intersection numbers of cycles in complementary dimension.

The basic reference on Intersection Theory is William Fulton's book [F].

The Chow package provides methods for working with *rational* Chow rings of non singular varieties: $A^*_{\mathbf{Q}}(X) = A^*(X) \otimes_{\mathbf{Z}} \mathbf{Q}$. The basic idea is to represent a non singular algebraic variety X by its rational Chow ring $A^*_{\mathbf{Q}}(X)$ and a morphism f between such varieties X and Y by the induced morphism on the level of Chow ring: $f^* : A^*_{\mathbf{Q}}(Y) \to A^*_{\mathbf{Q}}(X)$.

Concretely, a *ChowScheme* for a $\mathbf{Q}$-algebra A will correspond to any non singular algebraic variety X such that $A^*_{\mathbf{Q}}(X) \simeq A$ and a *ChowSchemeMorphism* for a morphism g of $\mathbf{Q}$-algebras B and A to any morphism f of ChowSchemes for A and B such that $f^* = g$. This makes only sense for $\mathbf{Q}$-algebras A that are indeed Chow rings of non singular algebraic varieties, but this is not required as for many formal calculations this would be by far too restrictive.

Disclaimer:

In view of the above loose correspondence, this package should be understood merely as a tool for computations in intersection theory, when you know *precisely what you are doing*.

1.1 Tutorial

For example, the projective plane is a ChowScheme for $\mathbf{Q}[h]/(h^3)$ and is defined in Chow as follows.

```
sage: P2 = ChowScheme(2, 'h', 1, 'h^3')
```

The first argument sets the dimension 2, the second the generator h, the third its degree 1 and finally the last the relation h^3.

If $f : \mathbb{P}^2 \longrightarrow \mathbb{P}^5$ is the Veronese embedding, then f is a ChowSchemeMorphism for $\mathbf{Q}[k]/(k^6) \to \mathbf{Q}[h]/(h^3),\ k \mapsto 2h$ and defined in Chow as follows.

```
sage: P5 = ChowScheme(5, 'k', 1, 'k^6')
sage: f = P2.hom(['2*h'], P5)
```

Note that once f is defined we can make basic computations as follows

```
sage: k = P5.gen()
sage: f.upperstar(5*k^4 - 3*k^2 + 1)
-12*h^2 + 1
```

If X is projetive, the degree morphism $\int$ is defined using a point class in $A^{\dim(X)}(X)$.

```
sage: P2.set_point_class('h^2')
sage: P5.set_point_class('k^5')
```

Once point classes are defined, we have Gysin maps:

```
sage: h = P2.chowring().gen()
sage: f.lowerstar(h)
2*k^4
```

Chow supports the idea of libraries to already known ChowSchemes as projective spaces or Grassmannians, that are defined already. By the way, please send us a library file for your favourite variety!:

```
sage: P2 = Proj(2, 'h')
```

Note that `chow.library.proj.Proj` knows already about canonically defined bundles over $\mathbb{P}^n$

```
sage: P3 = Proj(3, 'h')
sage: P3.tangent_bundle()
Bundle(Proj(3, 'h'), 3, [1, 4*h, 6*h^2, 4*h^3])
sage: P3.sheaves["universal_sub"]
Bundle(Proj(3, 'h'), 3, [1, -h, h^2, -h^3])
sage: P3.sheaves["universal_quotient"]
Bundle(Proj(3, 'h'), 1, [1, h])
```

Remark from the above universal bundles that `chow.library.proj.Proj` uses Grothendieck's convention of linear rank 1 *quotients* of a vector space of dimension $n + 1$. For the classical convention of *subspaces* as in Fulton's book [F], simply use

```
sage: P3 = Grass(1, 4, 'h', name='P3')
sage: P3.tangent_bundle()
Bundle(P3, 3, [1, 4*h, 6*h^2, 4*h^3])
sage: P3.sheaves["universal_sub"]
Bundle(P3, 1, [1, -h])
sage: P3.sheaves["universal_quotient"]
Bundle(P3, 3, [1, h, h^2, h^3])
```

Once we have two ChowSchemes with point classes and tangent bundles, we can compute the blowup of a morphism between the two:

```
sage: P2 = Proj(2, 'h')
sage: P5 = Proj(5, 'k')
sage: f = P2.hom(['2*h'], P5)
sage: g = Blowup(f)
```

Note that `chow.blowup` returns a morphism as follows:

$$\begin{array}{ccc} \widetilde{X} & \stackrel{g}{\to} & \widetilde{Y} \\ \downarrow & & \downarrow\sigma \\ X & \stackrel{f}{\to} & Y \end{array}$$

Compute generators, relations, tangent bundle or Betti numbers for $B = \widetilde{Y}$,

```
sage: B = g.codomain()
sage: B.chowring().gens()
(e, k)
sage: B.chowring().rels()
[k^6, e*k^3, e^3 - 9/2*e^2*k + 15/2*e*k^2 - 4*k^3]
sage: B.tangent_bundle().chern_classes()
[1, -2*e + 6*k, -15/2*e*k + 15*k^2, 9/2*e^2*k - 93/4*e*k^2 + 28*k^3, 27/4*e^2*k^2 + 27*k^4, 12*k^5]
sage: B.betti_numbers()
[1, 2, 3, 3, 2, 1]
sage: (e, k) = B.chowring().gens()
sage: ((6*k - 2*e)^5).integral()
3264
```

The last number answers the classical problem of finding the *smooth* plane conics tangent to five given general conics. Each tangency is a degree 6 condition on the $\mathbb{P}^5$ of *all* conics which contain the 2 dimensional family of double lines.

Unlike the Schubert package, `Chow` computes the full ChowScheme of the Blowup B, e.g. its Chow ring given by generators, degrees and relations, the point class as well as its tangent bundle.

Unlike Schubert, there is also no restriction on the number of generators of the Chow ring of the exceptional divisor, e.g. the following example is correctly computed (Schubert reports erroneously [1, 2, 3, 4, 5, 4, 3, 2, 1] as Betti numbers for B):

```
sage: P2xP2 = Proj(2, 'h') * Proj(2, 'k')
sage: P8 = Proj(8, 'l')
sage: f = P2xP2.hom(['h+k'], P8)  # Segre map P2xP2 -> P8
sage: g = Blowup(f)
sage: B = g.codomain()
sage: B.betti_numbers()
[1, 2, 4, 7, 8, 7, 4, 2, 1]
```

As another example, consider the twisted cubic in $\mathbb{P}^3$, as worked out in great detail for example in the expository note [K] (from which we use the notations for this example):

```
sage: X = Proj(3)
sage: Z = Proj(1, 'w')
sage: i = Z.hom(['3 * w'], X)
sage: i.lowerstar('w')
h^3
sage: g = Blowup(i)
sage: ZZ, XX = g.domain(), g.codomain()
sage: ZZ.chowring().gens()  # Expect the class z of the proj bundle and w
(z, w)
sage: XX.chowring().gens()  # Expect the class e of the exceptional divisor and h
(e, h)
sage: XX.chowring().basis()
[h^3, h^2, e*h, h, e, 1]
sage: XX.chowring().intersection_matrix()
[ 0  0  0  0  0  1]
[ 0  0  0  1  0  0]
[ 0  0  0  0 -3  0]
```

```
[ 0  1  0  0  0  0]
[ 0  0 -3  0  0  0]
[ 1  0  0  0  0  0]
sage: (e, h) = XX.chowring().gens()
sage: (e * e * h).integral()
-3
sage: gw = g.lowerstar('w'); gw
1/3*e*h
sage: gz = g.lowerstar('z'); gz
-10/3*e*h + 3*h^2
sage: (e * gw).integral()  # Example of computation
-1
sage: (e * gz).integral()  # Example of computation
10
```

1.2 Examples

1.2.1 Twisted cubics

Recall that a twisted cubic is a rational smooth curve of degree 3 in $\mathbb{P}^3$. All such curves are projectively equivalent to the twisted cubic $\mathbb{P}^1 \to \mathbb{P}^3$ given by $[x : y] \mapsto [x^3 : x^2y : xy^2 : y^3]$ hence the space of twisted cubics can be identified with the homogeneous space $SL(4)/SL(2)$. In particular, it is a smooth and irreducible 12-dimensional variety. It admits a natural compactification as the component H of the Hilbert scheme $Hilb^{3m+1}(\mathbb{P}^3)$ containing the points corresponding to twisted cubics. Ellingsrud, Piene and Strømme have shown that H can be obtained as the blowup of the variety of nets of quadrics X defined by twisted cubics along the incidence variety I ([EPS], [ES]) and they have also shown how the Chow ring of H can be computed. This has been used in [ES] in the relative case over $\mathbb{P}^4$ and for the computation of the Euler number of the holomorphically symplectic manifolds constructed in [LLSvS] in the relative case to $Grass(6, 4)$.

The library `chow.library.twisted_cubics` provides all the necessary methods to compute H, I and the morphism $f : I \to H$:

```
sage: P = Point()
sage: W = Bundle(P, 4, [1])  # Vector space of dimension 4
sage: f = map_incidence_to_nets_of_quadrics(W)
sage: I = f.domain()
sage: X = f.codomain()
```

Now we check Fulton [1], Prop. 9.1.2 (NIX is the normal bundle):

```
sage: (e1, e2, e3, f2) = X.chowring().gens()
sage: NIX = f.upperstar(X.tangent_bundle()) - I.tangent_bundle()
sage: c = (1 - f.upperstar(e1)) ** 12 * NIX.total_chern_class() ** (-1)
sage: c.integral()
38860
```

The next number is Schubert's number of twisted cubics intersecting 12 general lines in $\mathbb{P}^3$ ([S], pp. 178–179).

```
sage: ((-e1)^12).integral() - c.integral()
80160
```

Use `chow.Blowup` to compute H:

```
sage: H = Blowup(f).codomain()
sage: H.betti_numbers()
[1, 2, 6, 10, 16, 19, 22, 19, 16, 10, 6, 2, 1]
sage: H.euler_number()
130
sage: TH = H.tangent_bundle()
sage: top = TH.chern_classes()[H.dimension()]
sage: top.integral() == H.euler_number()
True
```

Finally, compute some values in table 1 of [ES]:

```
sage: (e1^12).integral()
119020
sage: (e1^10 * e2).integral()
45748
sage: (e1^5 * f2^2 * e3).integral()
490
```

1.2.2 Twisted cubics on quintic threefolds

A more challenging example is the computation of the number of twisted cubics on a general quintic threefold following [ES].

Let $\mathcal{H}$ be the component of the Hilbert scheme compactifiying twisted cubics in $\mathbb{P}^4$. If $\mathcal{C} \subset \mathcal{H} \times \mathbb{P}^4$ is the universal curve and p and q are the projections, then $0 \to \mathcal{I}_{\mathcal{C}} \to \mathcal{O}_{\mathcal{H}\times\mathbb{P}^4} \to \mathcal{O}_{\mathcal{C}} \to 0$ twisted by $q^*(\mathcal{O}_{\mathbb{P}^4}(5))$ induces the short exact sequence

$$0 \to p_*\mathcal{I}_{\mathcal{C}}(5) \to H^0(\mathbb{P}^4, \mathcal{O}_{\mathbb{P}^4}(5)) \otimes_{\mathbb{C}} \mathcal{O}_{\mathcal{H}} \to p_*\mathcal{O}_{\mathcal{C}}(5) \to 0$$

($R^1p_*\mathcal{I}_{\mathcal{C}}(5) = 0$ is easy to see). If $s \in H^0(\mathbb{P}^4, \mathcal{O}_{\mathbb{P}^4}(5))$, then the induced section in $p_*\mathcal{O}_{\mathcal{C}}(5)$ vanishes precisely in those curves [C] of $\mathcal{H}$ contained in the quintic defined by s. It turns out that $p_*\mathcal{O}_{\mathcal{C}}(5)$ is a vector bundle of rank 16 ($= \dim \mathcal{H}$) hence number of twisted cubics on a general quintic threefold is given by the degree of the top Chern class of this bundle.

Every such curve spans a hyperplane in $\mathbb{P}^4$ hence there is a map $\mathcal{H} \to \mathbb{P}^{4\vee}$ with fibres the component H of the Hilbert scheme parameterising twisted cubics in the corresponding $\mathbb{P}^3$. Using `chow.library.twisted_cubics` we get f relative to this $\mathbb{P}^{4\vee}$:

```
sage: P = Grass(1, 5, 'w')
sage: W = P.sheaves["universal_quotient"]
sage: f = map_incidence_to_nets_of_quadrics(W)
```

In order to obtain $\mathcal{H}$ it is then sufficient to compute the blowup of f:

```
sage: g = Blowup(f)
sage: Exc, H = g.domain(), g.codomain()
sage: I, X = Exc.base_chowscheme(), H.base_chowscheme()
```

Finally, as Chow determines the blowup by generators and relations, it is enough to use the exact sequence (6-1)

$$0 \to g_*\mathcal{A}_{Exc} \to \sigma^*\mathcal{A}_X \to p_*\mathcal{O}_{\mathcal{C}}(5) \to 0$$

of [ES] and the explicit description of $\mathcal{A}_{Exc}$ and $\mathcal{A}_X$ given there to compute as follows:

```
sage: K1, K2 = I.sheaves["K1"], I.sheaves["K2"]
sage: L1, L2 = I.sheaves["L1"], I.sheaves["L2"]
sage: Q = L1 + K2.dual()
sage: Exc_Q = Exc.base_morphism().upperstar(Q)
sage: Exc_K1 = Exc.base_morphism().upperstar(K1)
sage: AExc = Exc_Q.symm(2) * Exc_K1.determinant() * Exc.o(-1)
sage: AExcH = g.lowerstar(AExc, normal_bundle=Exc.o(-1))

sage: E, F = X.sheaves["E"], X.sheaves["F"]
sage: XW = X.base_morphism().upperstar(W)
sage: AX = (F * XW.symm(2)) - (E * XW.symm(3)) + XW.symm(5)
sage: AXH = H.base_morphism().upperstar(AX)

sage: (AXH - AExcH).chern_classes()[H.dimension()].integral()
317206375
```

1.2.3 Twisted cubics on cubic fourfolds

Here is the computation of the Euler number of the holomorphically symplectic manifolds constructed in [LLSvS]. This computation takes some time (45 minutes on a MacBook Air 2012), so that we add a '# long time' after each line from now on in order to avoid that automated doctesting (without the –long flag) does the computations. We start by defining $f : I_G \to X_G$ for $G = Grass(6, 4)$ using the library `chow.library.twisted_cubics`:

```
sage: G = Grass(6, 4, 'w')  # long time
sage: W = G.sheaves["universal_quotient"]  # long time
sage: f = map_incidence_to_nets_of_quadrics(W)  # long time
sage: I = f.domain()  # long time
sage: X = f.codomain()  # long time
```

Then we compute the blowup.

```
sage: g = Blowup(f, var_name='t')  # long time
```

If you prefer to follow the computations, add verbose=True, e.g. use g = Blowup(f, var_name='t', verbose=True). Once the Blowup is computed, one may check against Euler and Betti numbers of the Blowup.

```
sage: Exc, H = g.domain(), g.codomain()  # long time
sage: I, X = Exc.base_chowscheme(), H.base_chowscheme()  # long time
sage: Exc.euler_number()  # long time
1260
sage: Exc.betti_numbers()  # long time
[1, 4, 11, 23, 41, 64, 90, 115, 135, 146, 146, 135, 115, 90, 64, 41, 23, 11, 4, 1]
sage: H.euler_number()  # long time
1950
sage: H.betti_numbers()  # long time
[1, 3, 10, 22, 45, 75, 117, 158, 200, 225, 238, 225, 200, 158, 117, 75, 45, 22, 10, 3, 1]
```

Now we can compute the Euler number.

```
sage: IW = I.base_morphism().upperstar(W)  # long time
sage: L1 = I.sheaves["L1"]  # long time
sage: VB = Exc.base_morphism().upperstar(IW - L1)  # long time
sage: LL = VB.wedge(3) * Exc.o(-1)  # long time
sage: CC = g.lowerstar(LL, normal_bundle=Exc.o(-1))  # long time
```

Pull back E, F from X and W from $Grass(6, 4)$ to H:

```
sage: HE = H.base_morphism().upperstar(X.sheaves["E"])  # long time
sage: HF = H.base_morphism().upperstar(X.sheaves["F"])  # long time
sage: h = X.base_morphism() * H.base_morphism()  # long time
sage: HW = h.upperstar(W)  # long time
```

Compute the fundamental class, then the virtual tangent bundle of M_3 (see [LLSvS]):

```
sage: EW = HE * HW  # long time
sage: U1 = EW - HF  # long time
sage: U = CC + U1  # long time
sage: S3HW = HW.symm(3)  # long time
sage: V = S3HW - U  # long time

sage: fundamental_class = V.chern_classes()[10]  # long time

sage: TM = H.tangent_bundle() - V  # long time
sage: euler_class = TM.chern_classes()[10]  # long time
sage: euler_number_M = (fundamental_class * euler_class).integral()  # long time
```

Dividing by 3 be get the Euler number of the P2-contraction:

```
sage: eM = euler_number_M / 3; eM  # long time
25731
```

Still we have to subtract 81 for the final contraction:

```
sage: eM - 81  # long time
25650
```

Remark: Only the blowup takes a long time and can be saved in an intermediate step:

```
sage: g.save('/tmp/blowup_h_g64')  # long time
```

Then it can be resumed later by:

```
sage: g = load('/tmp/blowup_h_g64')  # long time
```

1.3 Notes

Chow is inspired and based on the beautiful Maple package *Schubert* [KS] by Sheldon Katz and Stein Arild Strømme, even though, under the hood, the computations are done completely differently: Chow, except from written from scratch in Python, makes essential use of Singular through Sage's libsingular interface.

Note that there is also a package for *Macaulay2* [GSE], called *Schubert2* [GSSEC], written by Daniel R. Grayson, Michael E. Stillman, Stein A. Strømme, David Eisenbud and Charley Crissman, equally based on the above Maple package.

CHAPTER

TWO

CHOW: THE CHOW RING OF A SCHEME

In general, the Chow ring of an algebraic variety is a graded ring, an algebraic analogue to the cohomology ring of its underlying topological space. Its elements correspond to formal linear combinations of algebraic subvarieties modulo rational equivalence. The product structure comes from the intersection of subvarieties.

See http://en.wikipedia.org/wiki/Chow_ring for an introduction to Chow Rings.

Calculating the Chow ring of an algebraic (projective, smooth) variety is a non trivial task. Here we suppose that the Chow ring is already given by generators and relations and will first provide some standard methods as computing a linear basis, the intersection matrix or the dual basis. Note also that we always work over $\mathbf{Q}$.

Later we will provide methods to compute the Chow ring for related algebraic varieties as a Grassmann bundle or a blowup (grass, blowup).

For example the Chow ring of the projective line is given by its generator h in degree 1 subject to the relation h^2:

```
sage: A = ChowRing('h', 1, 'h^2')
```

Another example is the Chow ring of the projective plane blown up in a point given by two classes E and H in degree one subject to the relations $E.H = 0$ and $E^2 + H^2 = 0$. Note that the latter relations are not in a standard basis. However when retrieving the relations, a standard basis is returned:

```
sage: A = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: A.rels()
[E^3, H^2 + E^2, H*E]
```

An important method for a ChowRing is the computation of a linear basis of algebraic cycles `ChowRing_generic.basis()`:

```
sage: A = ChowRing('h', 1, 'h^3')  # P2
sage: A.basis()
[h^2, h, 1]
```

Note however we do not ask for a ChowRing to be Artin, even though this is the case, at least for projective smooth algebraic varieties. So the following is perfectly valid for us:

```
sage: A = ChowRing(['c1', 'c2'], [1, 2])
```

The reason for this is that it is sometimes very useful to compute in these more general rings, especially while computing the relations for a particular variety. Note that if A is not Artin, it does not make sens to compute a linear basis and an error message is thrown:

```
sage: A.basis()
Traceback (most recent call last):
...
ValueError: Ring is not of dimension 0.
```

If A is the Chow ring of a projective smooth algebraic variety, a *point_class* is an element in top degree that integrates to one. For example, for the projective space of dimension n with generator h, the point_class is h^n. Once we have a point class, we can compute the intersection matrix and the basis dual to those given by `ChowRing_generic.basis()` with respect to the intersection matrix:

```
sage: A.<h> = ChowRing('h', 1, 'h^4')  # P3
sage: A.basis()
[h^3, h^2, h, 1]
sage: A.set_point_class(h^3)
sage: A.intersection_matrix()
[0 0 0 1]
[0 0 1 0]
[0 1 0 0]
[1 0 0 0]
sage: A.dual_basis_slow()
[1, h, h^2, h^3]
```

Note that `dual_basis_slow()` is, as its name indicates, slow at least for large examples (e.g. with a large linear basis of A). This is due to the fact that `ChowRing_generic.dual_basis_slow()` simply computes the intersection matrix, then takes its inverse. A much faster method is `ChowRing_generic.dual_basis()` which computes only the relevant (i.e. in complementary degrees) part of the intersection matrix. Setting the optional parameter verbose=True allows to follow the computation:

```
sage: A.<H, E> = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: A.basis()
[E^2, E, H, 1]
sage: A.set_point_class(H^2)
sage: A.dual_basis(verbose=True)
Computing block 0 of size 1 ...
Computing block 1 of size 2 ...
[-1, -E, H, -E^2]
```

Finally, there is a particular PointChowRing which is defined as the ChowRing with no generators and relations:

```
sage: A = ChowRing()
sage: A.set_dimension(0)
sage: A.set_point_class(1)
sage: B = PointChowRing
sage: A == B
True
```

In this case the underlying ring is $\mathbf{Q}[]/(O)$ and not simply $\mathbf{Q}$ as one might suspect, in order to get uniformly quotient rings:

```
sage: A = ChowRing(); A
Quotient of Multivariate Polynomial Ring in no variables over Rational Field by the ideal (0)
sage: A.gens()
()
sage: A.ngens()
0
sage: A.gen()
Traceback (most recent call last):
...
ValueError: Generator not defined.
sage: A.variable_names()
()
sage: A.variable_name()
Traceback (most recent call last):
```

```
...
IndexError: tuple index out of range
```

In the general case, when get as expected:

```
sage: A = ChowRing('h', 1, 'h^2')  # ChowRing of P1
sage: A.gens()
(h,)
sage: A.variable_names()
('h',)
sage: A.variable_name()
'h'
sage: A = ChowRing(['c1', 'c2'], [1, 2])
sage: A.variable_names()
('c1', 'c2')
sage: A.variable_name()
'c1'
```

TESTS:

```
sage: A = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: TestSuite(A).run()
```

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

sage.schemes.chow.ring.**ChowRing**(*generators=None*, *degrees=None*, *relations=None*, *names=None*, *name=None*, *latex_name=None*)

Returns a ChowRing, given its dimension, generators, degrees, relations and point_class.

INPUT:

- `generators` – An optional (list of) strings, the generators
- `degrees` – An optional (list of) integers, the degrees of the generators
- `relations` – An optional (list of) strings, the relations between the generators
- `names` – An optional (list of) strings, names of the generators in the ChowRing
- `name` – An optional string, the name of the ChowRing
- `latex_name` – An optional string, a latex representation of the ChowRing

OUTPUT:

- A ChowRing

EXAMPLES:

The ChowRing of the projective line is given by a generator h in degree 1, subject to the relation h^2:

```
sage: A = ChowRing('h', 1, 'h^2')
```

It can equally be defined more verbosely by:

```
sage: B = ChowRing(generators='h', degrees=1, relations='h^2')
sage: A == B
True
```

If there is more than one generator (or more than one relation) they have to be specified as lists of strings:

```
sage: A = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
```

Note that by default the generators in the ChowRing have the same names as in the cover_ring as opposed to a quotient by a PolynomialRing:

```
sage: A = ChowRing('h', 1, 'h^2')
sage: A.gens()
(h,)
sage: P = PolynomialRing(QQ, 'h').quotient('h^2')
sage: P.gens()
(hbar,)
```

As for QuotientRings, the generators can be named explicitly:

```
sage: A.<x> = ChowRing('h', 1, 'h^2')
sage: A.gens()
(x,)
```

Only the dimension is mandatory. For example, one may define a point Chow Ring as follows:

```
sage: A = ChowRing(0); A
Quotient of Multivariate Polynomial Ring in no variables over Rational Field by the ideal (0)
```

Please note that the internal structure of A is not the field $\mathbf{Q}$ but rather the Quotient of $\mathbf{Q}$ by the ideal (0). This is due to the fact that ChowRings are uniformly represented as quotient rings for us.

TESTS:

```
sage: A = ChowRing('c1', 'c2')
Traceback (most recent call last):
...
TypeError: Expect list of integers for degrees.

sage: A = ChowRing('h', 0)
Traceback (most recent call last):
...
ValueError: Expect positive integers for degrees.

sage: A = ChowRing('h', [1,2])
Traceback (most recent call last):
...
ValueError: Number of generators and degrees differ.

sage: A = ChowRing(degrees=1, relations='h^2')
No generators, degrees ignored.
No generators, relations ignored.

sage: A = ChowRing([], [], [])
sage: A.set_point_class(1)
sage: A(3).integral()
3
```

sage.schemes.chow.ring.**ChowRingFromRing**(*S*, *names=None*, *name=None*, *latex_name=None*)

Returns a ChowRing, given a ring S.

INPUT:

- S – A (quotient of) a multivariate Polynomial Ring

OUTPUT:

- The ChowRing defined by S

EXAMPLES:

```
sage: from sage.schemes.chow.ring import ChowRingFromRing
sage: variables, degrees, relations = ['x', 'y'], [2, 3], ['x*y']
sage: AX = ChowRing(variables, degrees, relations)
sage: OT = TermOrder('wdegrevlex', (2, 3))
sage: S = PolynomialRing(QQ, ['x', 'y'], order=OT).quotient('x*y'); S
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the ideal (x*y)
sage: AS = ChowRingFromRing(S)
sage: AS == AX
True
```

Note that we expect the ring S to be (quotient of) a multivariate Polynomial Ring:

```
sage: S = PolynomialRing(QQ, 'h').quotient('h^2'); S
Univariate Quotient Polynomial Ring in hbar over Rational Field with modulus h^2
sage: AS = ChowRingFromRing(S)
Traceback (most recent call last):
...
TypeError: Expected (Quotient of) Multivariate Polynomial Ring.
```

class sage.schemes.chow.ring.**ChowRing_generic**(*R*, *I*, *names=None*, *name=None*, *latex_name=None*)

Bases: sage.rings.quotient_ring.QuotientRing_generic

Construct a ChowRing_generic.

> **Warning:** This class does not perform any checks of correctness of input. Use ChowRing() or ChowRingFromRing() to construct a ChowRing.

INPUT:

- R – **Q** or a multivariate polynomial ring;
- I – an ideal of R;
- names – list of variable names;
- name– An optional string, the name of the ChowRing
- latex_name– An optional string, a latex representation of the ChowRing

OUTPUT:

- ChowRing.

TESTS

```
sage: A = ChowRing()
```

ChowRingElement
alias of ChowRingElement

Element
alias of ChowRingElement

basis()
Returns a basis of the underlying ring of this ChowRing if the underlying ring is Artin.

OUTPUT:

- A list of elements of the underlying ring.

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^4')  # P3
sage: A.basis()
[h^3, h^2, h, 1]

sage: A = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: A.basis()
[E^2, E, H, 1]

sage: A = ChowRing(['c1', 'c2'], [1, 2])  # No relations!
sage: A.basis()
Traceback (most recent call last):
...
ValueError: Ring is not of dimension 0.
```

TESTS:

```
sage: A = PointChowRing
sage: A.basis()
[1]
```

basis_by_degree()

Return a basis of the underlying ring of this ChowRing ordered by the degrees of the elements.

OUTPUT:

- A list of list of elements of the underlying ring.

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^4')  # P3
sage: A.basis_by_degree()
[[1], [h], [h^2], [h^3]]

sage: A = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: A.basis_by_degree()
[[1], [E, H], [E^2]]
```

TESTS:

```
sage: A = PointChowRing
sage: A.basis_by_degree()
[[1]]
```

deg(*i=None*)

Return the degree of the i-th generator. If i is unspecified the degree of the first generator (eg i=0) is returned if exists.

INPUT:

- `i` – an (optional) integer.

OUTPUT:

The degree of the i-th generator.

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^2')  # ChowRing of P1
sage: A.deg()
1
sage: A = ChowRing(['c1', 'c2'], [1, 2])
```

```
sage: A.deg()
1
sage: A.deg(1)
2
```

CORNER CASE:

```
sage: A = PointChowRing
sage: A.deg()
Traceback (most recent call last):
...
IndexError: tuple index out of range
```

degs()

Return the degrees of the generators of this ChowRing.

OUTPUT:

A tuple of integers representing the degrees of the generators.

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^2')  # ChowRing of P1
sage: A.degs()
(1,)
sage: A = ChowRing(['c1', 'c2'], [1, 2])
sage: A.degs()
(1, 2)
```

CORNER CASE:

```
sage: A = PointChowRing
sage: A.degs()
()
```

dimension()

Return the variety dimension (see : $meth$: $set_dimension$).

OUTPUT:

- an integer

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^2')
sage: A.dimension()  # None as not set yet.
sage: A.set_dimension(1)
sage: A.dimension()
1
```

dual_basis(*verbose=False*)

Returns the basis dual to `basis()` with respect to the intersection matrix. We carefully compute only the relevant part of the intersection matrix in order to invert several smaller matrices instead of one large matrix as in `dual_basis_slow()`. This method is mainly needed to compute f_* for morphisms $f : X \to Y$ hence especially while computing a blowup and its tangentbundle.

Remark: This requires quite some computational time for large examples, so we print out which block we compute if verbose is True.

INPUT:

- `verbose` – an (optional) flag for printing intermediate steps

OUTPUT:

•A list of ring elements representing the basis dual to `basis()`.

EXAMPLES:

```
sage: A.<H, E> = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: A.set_point_class(H^2)
sage: A.dual_basis(verbose=True)
Computing block 0 of size 1 ...
Computing block 1 of size 2 ...
[-1, -E, H, -E^2]
```

TESTS:

```
sage: A = ChowRing()
sage: A.set_point_class(2)
sage: A.dual_basis()
[1/2]
```

dual_basis_dict(*verbose=False*)

Returns the dictionary $e_i \to e_i^*$ where e_i is the basis given by `basis()` and e_i^* is the dual element. If verbose is set to True, intermediate steps are printing during the calculation of the dual basis.

INPUT:

•`verbose` – an (optional) flag for printing intermediate steps

OUTPUT:

•A dictionary associating $e_i \to e_i^*$.

EXAMPLES:

```
sage: A = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: A.set_point_class('H^2')
sage: DualBasis = A.dual_basis_dict()
sage: [(key, DualBasis[key]) for key in sorted(DualBasis)]
[(1, -E^2), (E, -E), (H, H), (E^2, -1)]
```

dual_basis_slow()

Returns the basis dual to self.basis() with respect to the intersection matrix. It is computed by simply inverting the intersection matrix. For a large ring basis, this method is somehow slow and has been optimized in self.dual_basis().

OUTPUT:

•A list of ring elements representing the basis dual to self.basis()

EXAMPLES:

```
sage: A = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: A.set_point_class('H^2')
sage: A.dual_basis_slow()
[-1, -E, H, -E^2]
```

TESTS:

```
sage: A = ChowRing()
sage: A.set_point_class(2)
sage: A.intersection_matrix()
[1/2]
```

identity_morphism()
Returns the identity morphism of this ChowRing.

OUTPUT:

•The identity morphism

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^2')
sage: f = A.identity_morphism()
sage: h = A.gen()
sage: f(h)
h
sage: f(1)
1

sage: A = PointChowRing
sage: f = A.identity_morphism()
sage: f(1)
1
```

intersection_matrix()
Returns the intersection matrix of this ChowRing.

OUTPUT:

•A matrix, the intersection matrix of this ChowRing

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^3')  # P2
sage: A.set_point_class('h^2')
sage: A.intersection_matrix()
[0 0 1]
[0 1 0]
[1 0 0]

sage: A.<H, E> = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: A.basis()
[E^2, E, H, 1]
sage: A.set_point_class(H^2)
sage: A.intersection_matrix()
[ 0  0  0 -1]
[ 0 -1  0  0]
[ 0  0  1  0]
[-1  0  0  0]
```

TESTS:

```
sage: A = ChowRing()
sage: A.set_point_class(2)
sage: A.intersection_matrix()
[1/2]
```

is_point()
Returns True if this ChowRing corresponds to a point, e.g. is of dimension 0 and has no generators.

OUTPUT:

•True or False

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^4')  # ChowRing of P3
sage: A.is_point()
False

sage: A = ChowRing()
sage: A.is_point()
True
```

is_point_chowring()

Returns True if this ChowRing is the instance PointChowRing.

OUTPUT:

- True or False

EXAMPLES:

```
sage: A = ChowRing()
sage: A.is_point_chowring()
False

sage: A = PointChowRing
sage: A.is_point_chowring()
True
```

krull_dimension()

Returns the Krull dimension of the underlying ring of this ChowRing. Remark: this is *not* the dimension of this ChowRing specified while defining this ChowRing.

OUTPUT:

- An integer, the Krull dimension of this ring.

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^4')  # P3
sage: A.krull_dimension()
0
sage: A = ChowRing(['c1', 'c2'], [1, 2])  # No relations!
sage: A.krull_dimension()
2
```

max_degree()

Return the maximal degree of this ChowRing if Artin else 0.

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^4')  # P3
sage: A.max_degree()
3

sage: A = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: A.max_degree()
2

sage: A = ChowRing('h', 1)  # not Artin
sage: A.max_degree()
0
```

TESTS:

```
sage: A = PointChowRing
sage: A.max_degree()
0
```

nrels()
Return the number of relations of this ChowRing

OUTPUT:

An integer, the number of relations.

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^2')  # ChowRing of P1
sage: A.nrels()
1
sage: A = ChowRing(['c1', 'c2'], [1, 2])
sage: A.nrels()
0
```

point_class()
Return the point class.

OUTPUT:

- a ring element

EXAMPLES:

```
sage: A.<h> = ChowRing('h', 1, 'h^3')    # P2
sage: A.set_point_class(h^2)
sage: A.point_class()
h^2
```

rel(*i=None*)
Return the i-th relation. If i is unspecified the first relation (eg i=0) is returned if exists.

INPUT:

- `i` – an (optional) integer.

OUTPUT:

The i-th relation.

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^2')  # ChowRing of P1
sage: A.rel()
h^2
```

TESTS:

```
sage: A = ChowRing(['c1', 'c2'], [1, 2])
sage: A.rel()
Traceback (most recent call last):
...
IndexError: list index out of range
```

rels()
Return the relations of this ChowRing.

OUTPUT:

A list of ring elements representing the relations.

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^2')  # ChowRing of P1
sage: A.rels()
[h^2]
sage: A = ChowRing(['c1', 'c2'], [1, 2])
sage: A.rels()
[]
```

TESTS:

```
sage: A = PointChowRing
sage: A.rels()
[]
```

set_dimension(*dimension*)

Set the dimension. If this ChowRing is the Chow ring of a variety X, set the dimension to $\dim(X)$

INPUT:

- an integer

EXAMPLES:

```
sage: A = ChowRing('h', 1, 'h^3')   # P2
sage: A.set_dimension(2)    # dim(P2)
```

TESTS:

```
sage: A = ChowRing('h', 1, 'h^3')   # P2
sage: A.set_dimension(1)
Traceback (most recent call last):
...
ValueError: Expect the dimension to be at least max-degree.
```

set_point_class(*point_class*)

Set the point class.

EXAMPLES:

```
sage: A.<h> = ChowRing('h', 1, 'h^2')   # P2
sage: A.set_point_class(h)
```

TESTS:

```
sage: A = ChowRing('h', 1, 'h^3')   # P2
sage: A.set_point_class('h^2')
sage: A.point_class()
h^2
sage: A.set_point_class('k^2')
Traceback (most recent call last):
...
TypeError: Can't coerce pointclass to ring element.
sage: A.set_point_class('h')
Traceback (most recent call last):
...
ValueError: Expect the point class in max-degree.
```

sage.schemes.chow.ring.**Kernel**(*f*)

Given a morphism $f : A \to B$ between (quotients of) multivariate polynomial rings, the kernel of f is returned.

EXAMPLE (http://trac.sagemath.org/sage_trac/ticket/9792):

```
sage: from sage.schemes.chow.ring import Kernel
sage: R = PolynomialRing(QQ, 4, names=['x', 'y', 'z', 'w'])
sage: S = PolynomialRing(QQ, 2, names=['s', 't'])
sage: s, t = S.gens()
sage: f = R.hom([s ** 4, s ** 3 * t, s * t ** 3, t ** 4], S)
sage: Kernel(f).gens()
[y*z - x*w, z^3 - y*w^2, x*z^2 - y^2*w, y^3 - x^2*z]
```

TESTS (taken from the Singular documentation of alg_kernel and kernel):

```
sage: from sage.schemes.chow.ring import Kernel
sage: R = PolynomialRing(QQ, 3, names=['a', 'b', 'c'])
sage: S = PolynomialRing(QQ, 6, names=['x', 'y', 'z', 'u', 'v', 'w'])
sage: x, y, z, u, v, w = S.gens()
sage: f = R.hom([x - w, u^2 * w + 1, y * z - v], S)
sage: Kernel(f).gens()
[0]

sage: from sage.schemes.chow.ring import Kernel
sage: R = PolynomialRing(QQ, 3, names=['a', 'b', 'c'])
sage: S = PolynomialRing(QQ, 3, names=['x', 'y', 'z']).quotient(x - y)
sage: x, y, z = S.gens()
sage: f = R.hom([x, y, x^2 - y^3], S)
sage: Kernel(f).gens()
[a - b, b^3 - b^2 + c]

sage: from sage.schemes.chow.ring import Kernel
sage: R = PolynomialRing(QQ, 4, names=['a', 'b', 'c', 'd'])
sage: S = PolynomialRing(QQ, 3, names=['a', 'b', 'c'])
sage: a, b, c = S.gens()
sage: f = R.hom([a, b, c, 0], S)
sage: Kernel(f).gens()
[d]
```

sage.schemes.chow.ring.**Kernel2**(f)

Given a morphism $f : A \to B$ between (quotients of) multivariate polynomial rings, the kernel of f is returned.

EXAMPLE (http://trac.sagemath.org/sage_trac/ticket/9792):

```
sage: from sage.schemes.chow.ring import Kernel
sage: R = PolynomialRing(QQ, 4, names=['x', 'y', 'z', 'w'])
sage: S = PolynomialRing(QQ, 2, names=['s', 't'])
sage: s, t = S.gens()
sage: f = R.hom([s ** 4, s ** 3 * t, s * t ** 3, t ** 4], S)
sage: Kernel(f).gens()
[y*z - x*w, z^3 - y*w^2, x*z^2 - y^2*w, y^3 - x^2*z]
```

TESTS (taken from the Singular documentation of alg_kernel and kernel):

```
sage: from sage.schemes.chow.ring import Kernel
sage: R = PolynomialRing(QQ, 3, names=['a', 'b', 'c'])
sage: S = PolynomialRing(QQ, 6, names=['x', 'y', 'z', 'u', 'v', 'w'])
sage: x, y, z, u, v, w = S.gens()
sage: f = R.hom([x - w, u^2 * w + 1, y * z - v], S)
sage: Kernel(f).gens()
[0]

sage: from sage.schemes.chow.ring import Kernel
sage: R = PolynomialRing(QQ, 3, names=['a', 'b', 'c'])
```

```
sage: S = PolynomialRing(QQ, 3, names=['x', 'y', 'z']).quotient(x - y)
sage: x, y, z = S.gens()
sage: f = R.hom([x, y, x^2 - y^3], S)
sage: Kernel(f).gens()
[a - b, b^3 - b^2 + c]

sage: from sage.schemes.chow.ring import Kernel
sage: R = PolynomialRing(QQ, 4, names=['a', 'b', 'c', 'd'])
sage: S = PolynomialRing(QQ, 3, names=['a', 'b', 'c'])
sage: a, b, c = S.gens()
sage: f = R.hom([a, b, c, 0], S)
sage: Kernel(f).gens()
[d]
```

sage.schemes.chow.ring.**ideals_are_equal**(*I*, *J*, *A*)

Internal Helper. Fast check that $I \subset J$ and $J \subset I$ in A using Singular.

INPUT:

- I – an ideal in the ring A
- J – an ideal in the ring A
- A – a ring

OUTPUT: True or False depending whether I = J in A or not.

TESTS:

```
sage: from sage.schemes.chow.ring import ideals_are_equal
sage: R = PolynomialRing(QQ, 3, names=['a', 'b', 'c'])
sage: a, b, c = R.gens()
sage: I = R.ideal(a^2 - b, a^3)
sage: J = R.ideal(b^2 , a * b, a^2 - b)
sage: ideals_are_equal(I, J, R)
True

sage: from sage.schemes.chow.ring import ideals_are_equal
sage: K = R.ideal(a^2 - b, a^2)
sage: ideals_are_equal(I, K, R)
False
```

sage.schemes.chow.ring.**is_chowRing**(*R*)

Test whether R is a ChowRing.

INPUT:

- R – anything.

OUTPUT:

Boolean. Whether R derives from ChowRing_generic.

TESTS:

```
sage: A = PointChowRing
sage: is_chowRing(A)
True
sage: S = PolynomialRing(QQ, ['x', 'y']).quotient('x*y')
sage: is_chowRing(S)
False
```

CHAPTER

THREE

CHOW: METHODS FOR CALCULATIONS WITH ELEMENTS OF A CHOWRING

Provide methods for calculations with elements of a ChowRing. For example, suppose we work with the ChowRing of the projective plane blown up in a point $\widehat{\mathbb{P}^2}$ and consider the special element $x = (2 * H - E)^2 + E$:

```
sage: A.<H,E> = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: x = (2 * H - E)^2 + E; x
-3*E^2 + E
```

Compute the decomposition of x in the graded ring A and return its part in degree equal to the dimension:

```
sage: x.by_degrees()
[0, E, -3*E^2]
```

The codimension is the top degree of an element:

```
sage: x.codimension()
2
sage: (E + H).codimension()
1
```

Compute the integral of x with respect to the point class H^2:

```
sage: A.set_point_class(H^2)        # Need to set the point class first.
sage: x.integral()
3
```

The other methods are related to the elements defined as the Chern character of a vector bundle. Recall that if X is a smooth algebraic variety, then the Chern character gives an isomorphism

$$ch : K_0(X) \otimes \mathbf{Q} \longrightarrow A^*(X) \otimes \mathbf{Q}$$

In our example if T is the tangent bundle on $\mathbb{P}^2$ then T is of rank 2 with Chern classes $[1, 3 * H - E, 4 * H^2]$ and its Chern character is $t = 2 - E + 3 * H$:

```
sage: A.set_dimension(2)     # Need to set the dimension first.
sage: t = 2 - E + 3 * H
sage: t._expp()
-4*E^2 + 3*H - E + 1
sage: t.wedge(2)._expp()
3*H - E + 1
sage: t.symm(2)._expp()
-32*E^2 + 9*H - 3*E + 1
```

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

class sage.schemes.chow.ring_element.**ChowRingElement**(*parent*, *rep*)
Bases: sage.rings.quotient_ring_element.QuotientRingElement

Class representing the elements of a ChowRing.

adams(*k*)
Return the result of the k-th Adams operator applied to this Chow ring element.

INPUT :

•k– an integer

EXAMPLE:

```
sage: A.<c1, c2, c3> = ChowRing(['c1', 'c2', 'c3'], [1, 2, 3])
sage: (1 + c1 + c2).adams(2)
4*c2 + 2*c1 + 1
```

by_degrees()
Returns this Chow ring element as a list $[e_0, \ldots, e_d]$ where e_k has degree k.

EXAMPLES:

```
sage: A.<H,E> = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: (-2 - E + 3*H - 4*H^2).by_degrees()
[-2, 3*H - E, 4*E^2]
sage: (-E + 3*H - 4*H^2).by_degrees()
[0, 3*H - E, 4*E^2]
sage: (-2 - E + 3*H).by_degrees()
[-2, 3*H - E]
```

TESTS:

```
sage: A = ChowRing()
sage: A(5).by_degrees()
[5]
```

codimension()
Return the codimension of this Chow ring element.

EXAMPLES:

```
sage: A.<H> = ChowRing('H', 1, 'H^5')
sage: H.codimension()
1
sage: (H + H^2).codimension()
2
```

TESTS:

```
sage: A = ChowRing(0)
sage: A(5).codimension()
0
```

integral()
Return the integral of this Chow ring element with respect to the parents point_class.

EXAMPLES:

```
sage: A.<H,E> = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: A.set_point_class('H^2')
sage: x = (2*H-E)^2; x.integral()
3
```

TESTS:

```
sage: A = ChowRing()
sage: A.set_point_class(2)
sage: A(6).integral()
3
```

symm(*p*)

Return the p-th symmetric power of this Chow ring element. Truncate above degree d.

INPUT:

- p– an integer
- d– an integer

OUTPUT:

- A Chow ring element.

EXAMPLES:

```
sage: A.<h> = ChowRing('h', 1, 'h^3')  # P2
sage: A.set_dimension(2)
sage: t = 1/2*h^2 + h + 1  # ch(O(1))
sage: t._expp()
h + 1
sage: t.symm(0)._expp()
1
sage: t.symm(1)._expp()
h + 1
sage: t.symm(2)._expp()
2*h + 1
sage: t.symm(5) == t^5  # t corresponds to a line bundle.
True
```

todd()

Return the todd class of this Chow ring element.

OUTPUT:

- An element of the parent Chow ring.

EXAMPLE:

```
sage: A.<c1, c2, c3> = ChowRing(['c1', 'c2', 'c3'], [1, 2, 3])
sage: A.set_dimension(3)
sage: x = 1 + c1 + 1/2*c1^2 + 1/6*c1^3

sage: x._expp()  # Check that x=ch(L) with L l.b. c1(L)=c1
c1 + 1
sage: x.todd()
1/12*c1^2 + 1/2*c1 + 1
```

TEST:

```
sage: A = ChowRing()
sage: A.set_dimension(0)
```

```
sage: A(7).todd()
1
```

truncate(*a=0*, *b=0*)

Return this Chow ring element truncated in degrees below a and above b (strictly).

INPUT:

- a – an optional integer (defaults to 0)
- b – an optional integer (defaults to 0)

OUTPUT:

- a Chow ring element.

EXAMPLES:

```
sage: A.<H,E> = ChowRing(['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'])
sage: x = -2 - E + 3*H - 4*H^2; x.by_degrees()
[-2, 3*H - E, 4*E^2]
sage: x.truncate()
-2
sage: x.truncate(0, 1)
3*H - E - 2
sage: x.truncate(0, 2)
4*E^2 + 3*H - E - 2
sage: x.truncate(1, 2)
4*E^2 + 3*H - E
```

wedge(*p*)

Return the p-th exterior power of this Chow ring element. Truncate above degree d.

INPUT:

- p– an integer

By convention, 0 is returned for negative values of p.

EXAMPLES:

```
sage: A.<h> = ChowRing('h', 1, 'h^3')  # P2
sage: A.set_dimension(2)
sage: t = 3/2*h^2 + 3*h + 2  # ch(TP^2)
sage: t._expp()
3*h^2 + 3*h + 1
sage: t.wedge(0)._expp()
1
sage: t.wedge(1)._expp()
3*h^2 + 3*h + 1
sage: t.wedge(2)._expp()
3*h + 1
sage: t.wedge(3)._expp()
Traceback (most recent call last):
...
ValueError: Wedge for powers outside 0..rank not allowed.
```

TESTS:

```
sage A = ChowRing(0)
sage A(5).wedge(0)
1
```

CHAPTER

FOUR

CHOW: THE SET OF MORPHISM BETWEEN CHOWRINGS

Space of ChowRing homomorphisms.

Derives from the space of quotient ring homomorphisms implement in $sage.ring.homset.RingHomset_quo_ring$ in order to explicitly allow "no generators", e.g. im_gens = [].

EXAMPLES:

```
sage: A.<h> = ChowRing('h', 1, 'h^3')
sage: B.<k> = ChowRing('k', 1, 'k^6')
sage: phi = B.hom([2*h], A); phi
Ring morphism:
  From: Quotient of Multivariate Polynomial Ring in k over Rational Field by the ideal (k^6)
  To:   Quotient of Multivariate Polynomial Ring in h over Rational Field by the ideal (h^3)
  Defn: k |--> 2*h
sage: phi(2)
2
sage: phi(k)
2*h

sage: A = ChowRing()
sage: f = A.hom([], A)
sage: f(1)
1
sage: B.<h> = ChowRing('h', 1, 'h^3')
sage: f = B.hom([0], A)
sage: f(h)
0
sage: f(3)
3
```

TESTS:

```
sage: A = ChowRing()
sage: B.<h> = ChowRing('h', 1, 'h^3')
sage: H = B.Hom(A)
sage: H == loads(dumps(H))
True

sage: TestSuite(phi).run()
```

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

class sage.schemes.chow.ring_homset.**ChowRingHomSet**(*R*, *S*, *category=None*)

Bases: sage.rings.homset.RingHomset_generic

Initialize self.

EXAMPLES:

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

CHAPTER

FIVE

CHOW: FINITE RING EXTENSIONS

Let $f : A \to B$ be a finite ring extension, given as a ring map between (quotients of) multivariate polynomial rings.

Let $m_1, \ldots, m_s$ be elements of B that generate B as an A-module. Consider the surjective ring morphism $\psi : A[z_1, ..., z_s] \to B$ defined by f and evaluation on $m_1, \ldots, m_s$.

Then the class FiniteRingExtension returns the quotient ring C/I with $C = A[z_1, ..., z_s]$ and $I = Ker(\psi)$ together with a ring morphism $\phi : B \to C/I$, inverse to the morphism $\psi : C/I \to B$ induced and still denoted by ψ.

Moreover, the annihilator $Ann_A(B)$ of B as an A-module is calculated as well as a presentation matrix P of B as an A-module:

$$A^p \xrightarrow{P} A^s \xrightarrow{m} B \longrightarrow 0$$

EXAMPLE:

Consider the twisted cubic in $\mathbb{P}^3$. On the level of ChowRings the inclusion i induces the ring map

$$A^*\mathbb{P}^3 \xrightarrow{j} A^*\mathbb{P}^1.$$

```
sage: A.<h> = ChowRing('h', 1, 'h^4')  # P3
sage: B.<w> = ChowRing('w', 1, 'w^2')  # P1
sage: j = A.hom([3*w], B)  # j=i^{*} sends h to 3*w.
sage: F = FiniteRingExtension(j); F
Quotient of Multivariate Polynomial Ring in z, h over Rational Field by the ideal (h^2, z - 1)
sage: F.psi()
Ring morphism:
  From: Quotient of Multivariate Polynomial Ring in z, h over Rational Field by the ideal (h^2, z - 1
  To:   Quotient of Multivariate Polynomial Ring in w over Rational Field by the ideal (w^2)
  Defn: 1 |--> 1
        h |--> 3*w
sage: F.phi()
Ring morphism:
  From: Quotient of Multivariate Polynomial Ring in w over Rational Field by the ideal (w^2)
  To:   Quotient of Multivariate Polynomial Ring in z, h over Rational Field by the ideal (h^2, z - 1
  Defn: w |--> 1/3*h
sage: F.ann()
Ideal (h^2) of Quotient of Multivariate Polynomial Ring in h over Rational Field by the ideal (h^4)
sage: F.prm()  # presentation matrix
[0]
sage: F.mgs()  # module generators
(1,)
sage: F.mds()  # module generator degrees
(0,)
sage: F.nvs()  # new variables that have been introduced
('z',)
```

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

class sage.schemes.chow.finite_ring_extension.**FiniteRingExtension**(*f*, *var_name='z'*)

Bases: sage.rings.quotient_ring.QuotientRing_generic

Construct a FiniteRingExtension.

INPUT:

- f – a finite ring morphism between (quotients of) multivariate polynomial rings.

OUTPUT:

- FiniteRingExtension.

EXAMPLES:

Consider the morphism on the level of Chow rings induced by the Veronese embedding (http://en.wikipedia.org/wiki/Veronese_surface) $i : \mathbb{P}^2 \longrightarrow \mathbb{P}^5$:

```
sage: A.<y> = ChowRing('y', 1, 'y^6')  # P5
sage: B.<x> = ChowRing('x', 1, 'x^3')  # P2
sage: f = A.hom([2*x], B)
sage: F = FiniteRingExtension(f)
sage: F.cover_ring().gens()
(z, y)
sage: F.defining_ideal().gens()
[y^3, z - 1]
```

Another examples is the morphism on the level of Chow rings induced by the Segre embedding $\mathbb{P}^2 \times \mathbb{P}^2 \longrightarrow \mathbb{P}^8$:

```
sage: A.<l> = ChowRing('l', 1, 'l^9')
sage: B.<h,k> = ChowRing(['h', 'k'], [1, 1], ['h^3', 'k^3'])
sage: f = A.hom([h+k], B)  # Segre embedding P2 x P2 to P8
sage: F = FiniteRingExtension(f);
sage: F.cover_ring().gens()
(z1, z2, z3, l)
sage: F.defining_ideal().gens()
[l^5, z3 - 1, 2*z2*l^3 - l^4, 3*z1*l - 3*z2*l^2 + l^3, z2^2 - z1, z1*z2, z1^2]
```

TESTS:

We test a corner case with codomain a quotient of a multivariate polynomial ring in no variables (used while blowing up $\mathbb{P}^2$ in a point):

```
sage: A.<h> = ChowRing('h', 1, 'h^3')
sage: B = ChowRing()
sage: f = A.hom([B(0)], B)
sage: F = FiniteRingExtension(f)
sage: F.cover_ring().gens()
(z, h)
sage: F.defining_ideal().gens()
[h, z - 1]
```

Finally run the test suite on F:

```
sage: TestSuite(F).run()
```

ann()

Return the annihilator $Ann_A(B)$ of B seen as an A-module via f.

EXAMPLES:

```
sage: A.<y> = ChowRing('y', 1, 'y^6')  # P5
sage: B.<x> = ChowRing('x', 1, 'x^3')  # P2
sage: f = A.hom([2*x], B)
sage: F = FiniteRingExtension(f)
sage: F.ann().gens()
[y^3]

sage: A.<l> = ChowRing('l', 1, 'l^9')
sage: B.<h,k> = ChowRing(['h', 'k'], [1, 1], ['h^3', 'k^3'])
sage: f = A.hom([h+k], B)  # Segre embedding P2 x P2 to P8
sage: F = FiniteRingExtension(f)
sage: F.ann().gens()
[l^5]
```

TESTS:

```
sage: A.<h> = ChowRing('h', 1, 'h^3')
sage: B = ChowRing()
sage: f = A.hom([B(0)], B)
sage: F = FiniteRingExtension(f)
sage: F.ann().gens()
[h]
```

mds()

Return the tuple of the degrees of the module generators of B seen as an A-module via f.

EXAMPLES:

```
sage: A.<y> = ChowRing('y', 1, 'y^6')  # P5
sage: B.<x> = ChowRing('x', 1, 'x^3')  # P2
sage: f = A.hom([2*x], B)
sage: F = FiniteRingExtension(f)
sage: F.mds()
(0,)

sage: A.<l> = ChowRing('l', 1, 'l^9')
sage: B.<h,k> = ChowRing(['h', 'k'], [1, 1], ['h^3', 'k^3'])
sage: f = A.hom([h+k], B)  # Segre embedding P2 x P2 to P8
sage: F = FiniteRingExtension(f)
sage: F.mds()
(2, 1, 0)
```

TESTS:

```
sage: B = ChowRing()
sage: f = A.hom([B(0)], B)
sage: F = FiniteRingExtension(f)
sage: F.mds()
(0,)
```

mgs()

Return the tuple of module generators of B seen as an A-module via f.

EXAMPLES:

```
sage: A.<y> = ChowRing('y', 1, 'y^6')  # P5
sage: B.<x> = ChowRing('x', 1, 'x^3')  # P2
sage: f = A.hom([2*x], B)
sage: F = FiniteRingExtension(f)
sage: F.mgs()
(1,)

sage: A.<l> = ChowRing('l', 1, 'l^9')
sage: B.<h,k> = ChowRing(['h', 'k'], [1, 1], ['h^3', 'k^3'])
sage: f = A.hom([h+k], B)  # Segre embedding P2 x P2 to P8
sage: F = FiniteRingExtension(f)
sage: F.mgs()
(k^2, k, 1)
```

TESTS:

```
sage: A.<h> = ChowRing('h', 1, 'h^3')
sage: B = ChowRing()
sage: f = A.hom([B(0)], B)
sage: F = FiniteRingExtension(f)
sage: F.mgs()
(1,)
```

nvs()

Return the tuple of 'new' variables $(z_1, \ldots, z_s)$ corresponding to the module generators of the A-module B (via f).

EXAMPLES:

```
sage: A.<y> = ChowRing('y', 1, 'y^6')  # P5
sage: B.<x> = ChowRing('x', 1, 'x^3')  # P2
sage: f = A.hom([2*x], B)
sage: F = FiniteRingExtension(f, var_name='t')
sage: F.nvs()
('t',)

sage: A.<l> = ChowRing('l', 1, 'l^9')
sage: B.<h,k> = ChowRing(['h', 'k'], [1, 1], ['h^3', 'k^3'])
sage: f = A.hom([h+k], B)  # Segre embedding P2 x P2 to P8
sage: F = FiniteRingExtension(f)
sage: F.nvs()
('z1', 'z2', 'z3')
```

TESTS:

```
sage: A.<h> = ChowRing('h', 1, 'h^3')
sage: B = ChowRing()
sage: f = A.hom([B(0)], B)
sage: F = FiniteRingExtension(f)
sage: F.nvs()
('z',)
```

phi()

Return the isomorphism $\phi : B \longrightarrow C/I$ inverse to ψ.

EXAMPLES:

```
sage: A.<y> = ChowRing('y', 1, 'y^6')  # P5
sage: B.<x> = ChowRing('x', 1, 'x^3')  # P2
sage: f = A.hom([2*x], B)
```

```
sage: F = FiniteRingExtension(f)
sage: F.phi()
Ring morphism:
  From: Quotient of Multivariate Polynomial Ring in x over Rational Field by the ideal (x^3)
  To:   Quotient of Multivariate Polynomial Ring in z, y over Rational Field by the ideal (y
  Defn: x |--> 1/2*y

sage: A.<l> = ChowRing('l', 1, 'l^9')
sage: B.<h,k> = ChowRing(['h', 'k'], [1, 1], ['h^3', 'k^3'])
sage: f = A.hom([h+k], B)  # Segre embedding P2 x P2 to P8
sage: F = FiniteRingExtension(f)
sage: F.phi()
Ring morphism:
  From: Quotient of Multivariate Polynomial Ring in h, k over Rational Field by the ideal (h
  To:   Quotient of Multivariate Polynomial Ring in z1, z2, z3, l over Rational Field by the
  Defn: h |--> -z2 + l
        k |--> z2
```

TESTS:

```
sage: A.<h> = ChowRing('h', 1, 'h^3')
sage: B = ChowRing()
sage: f = A.hom([B(0)], B)
sage: F = FiniteRingExtension(f)
sage: F.phi()
Ring morphism:
  From: Quotient of Multivariate Polynomial Ring in no variables over Rational Field by the
  To:   Quotient of Multivariate Polynomial Ring in z, h over Rational Field by the ideal (h
```

prm()

Return the presentation matrix.

$$A^*\mathbb{P}^3 \xrightarrow{j} A^*\mathbb{P}^1.$$

EXAMPLES:

```
sage: A.<y> = ChowRing('y', 1, 'y^6')  # P5
sage: B.<x> = ChowRing('x', 1, 'x^3')  # P2
sage: f = A.hom([2*x], B)
sage: F = FiniteRingExtension(f)
sage: F.prm()
[0]

sage: A.<l> = ChowRing('l', 1, 'l^9')
sage: B.<h,k> = ChowRing(['h', 'k'], [1, 1], ['h^3', 'k^3'])
sage: f = A.hom([h+k], B)  # Segre embedding P2 x P2 to P8
sage: F = FiniteRingExtension(f)
sage: F.prm()
[     0    3*l]
[ 2*l^3 -3*l^2]
[  -l^4    l^3]
```

TESTS:

```
sage: A.<h> = ChowRing('h', 1, 'h^3')
sage: B = ChowRing()
sage: f = A.hom([B(0)], B)
sage: F = FiniteRingExtension(f)
sage: F.prm()
[0]
```

psi()

Return he isomorphism $\psi : C/I \longrightarrow B$ given by f and the module generators.

EXAMPLES:

```
sage: A.<y> = ChowRing('y', 1, 'y^6')  # P5
sage: B.<x> = ChowRing('x', 1, 'x^3')  # P2
sage: f = A.hom([2*x], B)
sage: F = FiniteRingExtension(f)
sage: F.psi()
Ring morphism:
  From: Quotient of Multivariate Polynomial Ring in z, y over Rational Field by the ideal (y
  To:   Quotient of Multivariate Polynomial Ring in x over Rational Field by the ideal (x^3)
  Defn: 1 |--> 1
        y |--> 2*x

sage: A.<l> = ChowRing('l', 1, 'l^9')
sage: B.<h,k> = ChowRing(['h', 'k'], [1, 1], ['h^3', 'k^3'])
sage: f = A.hom([h+k], B)  # Segre embedding P2 x P2 to P8
sage: F = FiniteRingExtension(f)
sage: F.psi()
Ring morphism:
  From: Quotient of Multivariate Polynomial Ring in z1, z2, z3, l over Rational Field by the
  To:   Quotient of Multivariate Polynomial Ring in h, k over Rational Field by the ideal (h
  Defn: z1 |--> k^2
        z2 |--> k
        1 |--> 1
        l |--> h + k
```

TESTS:

```
sage: A.<h> = ChowRing('h', 1, 'h^3')
sage: B = ChowRing(0)
sage: f = A.hom([B(0)], B)
sage: F = FiniteRingExtension(f)
sage: F.psi()
Ring morphism:
  From: Quotient of Multivariate Polynomial Ring in z, h over Rational Field by the ideal (h
  To:   Quotient of Multivariate Polynomial Ring in no variables over Rational Field by the
  Defn: 1 |--> 1
        0 |--> 0
```

push_down(v)

Return the push_down of a sequence of elements in B.

INPUT:

- v– a list or tuple of elements in B

EXAMPLES:

```
sage: A.<y> = ChowRing('y', 1, 'y^6')  # P5
sage: B.<x> = ChowRing('x', 1, 'x^3')  # P2
sage: f = A.hom([2*x], B)
sage: F = FiniteRingExtension(f)
sage: F.push_down([x^2])
[1/4*y^2]

sage: A.<l> = ChowRing('l', 1, 'l^9')
```

```
sage: B.<h,k> = ChowRing(['h', 'k'], [1, 1], ['h^3', 'k^3'])
sage: f = A.hom([h+k], B)  # Segre embedding P2 x P2 to P8
sage: F = FiniteRingExtension(f)
sage: F.push_down((h, k))
[ 0  0]
[-1  1]
[ 1  0]
```

TESTS:

```
sage: A.<h> = ChowRing('h', 1, 'h^3')
sage: B = ChowRing()
sage: f = A.hom([B(0)], B)
sage: F = FiniteRingExtension(f)
sage: F.push_down([1])
[1]
```

CHAPTER

SIX

CHOW: SCHEMES

A ChowScheme is thought about as a smooth algebraic variety having a given Chow ring. A morphism between ChowSchemes is thought about as a morphism between these varieties represented by the induced morphism on the level of Chow rings.

As an example, consider $\mathbb{P}^2$ and $\mathbb{P}^5$ and the morphism $f : \mathbb{P}^2 \longrightarrow \mathbb{P}^5$ given by the Veronese embedding $[x : y : z] \mapsto [x^2 : y^2 : z^2 : yz : xz : xy]$ which we encode as follows:

```
sage: P2 = ChowScheme(2, 'h', 1, 'h^3')
sage: P5 = ChowScheme(5, 'k', 1, 'k^6')
sage: f = P2.hom(['2*h'], P5)
```

In the first two lines, we define $\mathbb{P}^2$ and $\mathbb{P}^5$ by their (rational) Chow rings respectively $A^*(\mathbb{P}^2) = \mathbf{Q}[h]/(h^3)$ and $A^*(\mathbb{P}^5) = \mathbf{Q}[k]/(k^6)$. The Veronese embedding f induced on the level of Chow rings

$$f^* : A^*(\mathbb{P}^5) \longrightarrow A^*(\mathbb{P}^2)$$

is given by sending the generator k to $2 * h$ which explains the third line.

Actually the projective space is already defined in the libraries and one might code as follows:

```
sage: P2 = Proj(2)
sage: P5 = Proj(5, hyperplane_class='k')
sage: f = P2.hom(['2*h'], P5)
```

TESTS:

```
sage: X = ChowScheme(2, ['H', 'E'], [1, 1], ['E*H', 'E^2+H^2'], 'H^2')
sage: TestSuite(X).run(skip=["_test_an_element", "_test_some_elements"])
```

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

sage.schemes.chow.scheme.**ChowScheme**(*dimension*, *generators=None*, *degrees=None*, *relations=None*, *point_class=None*, *names=None*, *name=None*, *latex_name=None*)

Returns a ChowScheme, given its dimension and (optional) generators, degrees, relations, point_class, names, name, latex_name.

INPUT:

- `dimension` – The dimension of the ChowScheme
- `generators` – An optional (list of) strings, the generators
- `degrees` – An optional (list of) integers, the degrees of the generators

- relations – An optional (list of) strings, the relations between the generators
- point_class – An optional point_class of the ChowScheme.
- names – An optional (list of) strings, names of the generators in the ChowScheme
- name – An optional string, the name of the ChowScheme
- latex_name– An optional string, the latex representation of the ChowScheme

OUTPUT:

- A ChowScheme

EXAMPLES:

```
sage: P2.<h> = ChowScheme(2, 'h', 1, 'h^3')
sage: h^3
0
```

Note that the optional name is not the unique string representation of the ChowScheme (as given by python's : $meth$: $repr()$) but is rather used in the string representations of other objects as morphisms or bundles on the ChowScheme in order to simplify the output.

EXAMPLES:

```
sage: P2.<h> = ChowScheme(2, 'h', 1, 'h^3', 'h^2', name='P2'); P2
ChowScheme(2, 'h', 1, 'h^3', 'h^2')
sage: str(P2)
'P2'
sage: P2.<h> = Proj(2, name='P2'); P2
ChowScheme(2, 'h', 1, 'h^3', 'h^2')
sage: P2.o(-1)  # The Hopf bundle on P2
Bundle(P2, 1, [1, -h])
```

TESTS:: sage: P2.objgen() (ChowScheme(2, 'h', 1, 'h^3', 'h^2'), h) sage: P2.objgens() (ChowScheme(2, 'h', 1, 'h^3', 'h^2'), (h,)) sage: P2.inject_variables() Defining h sage: P2.latex_variable_names() ['h'] sage: P2.variable_names() ('h',)

class sage.schemes.chow.scheme.**ChowScheme_generic**(*R=None*, *name=None*, *latex_name=None*)

Bases: sage.structure.parent.Parent

The base class for all ChowSchemes.

base()

Return the base ChowScheme of this ChowScheme. Synonym for $meth$: $base_c howscheme()$

OUTPUT :

A ChowScheme.

Examples:

```
sage: G = Grass(4, 2)
sage: G.base()
PointChowScheme
sage: Q = G.sheaves["universal_quotient"]
sage: ProjQ = ProjBundle(Q.symm(2).dual())  # Proj(Q)->G
sage: ProjQ.base()
ChowScheme(4, ['c1', 'c2'], [1, 2], ['c2^3', 'c1*c2^2', 'c1^2*c2 - c2^2', 'c1^3 - 2*c1*c2'],
```

base_change(*f*)
Return this ChowScheme with base changes to f.codomain().

INPUT:

- `f` – a morphism from this ChowScheme to another ChowScheme

OUTPUT:

This ChowScheme based changed to f.codomain()

EXAMPLES:

```
sage: P4 = ChowScheme(4, 'w', 1, 'w^5')  # P4
sage: X = ChowScheme(6, ['e1', 'e2', 'w'], [1, 2, 1], ['w^5'])
sage: X.base()
PointChowScheme
sage: X = X.base_change(X.hom('w', P4))
sage: X.base()
ChowScheme(4, 'w', 1, 'w^5')
```

base_chowring()
Return the Chow Ring of the Base Chow Scheme of this ChowScheme.

EXAMPLES:

```
sage: P1 = ChowScheme(1, 'h', 1, 'h^2', 'h')
sage: P1.base_chowring()
Quotient of Multivariate Polynomial Ring in no variables over Rational Field by the ideal (0
```

base_chowring_morphism()
Return the Chow Ring morphism from the Base Chow Ring to the Chow Ring

EXAMPLES:

```
sage: P1 = ChowScheme(1, 'h', 1, 'h^2', 'h')
sage: f = P1.base_chowring_morphism()
```

base_chowscheme()
Return the base ChowScheme of this ChowScheme.

OUTPUT:

A chowscheme.

EXAMPLES:

```
sage: P2 = Proj(2)
sage: P2.base_chowscheme()
PointChowScheme
```

TESTS:

```
sage: P = PointChowScheme
sage: P.base_chowscheme()
PointChowScheme
```

base_morphism()
Return the structure morphism from `self` to its base chow scheme.

OUTPUT:

A ChowScheme morphism.

EXAMPLES:

```
sage: G = Grass(4, 3)
sage: Q = G.sheaves["universal_quotient"]
sage: PG = ProjBundle(Q.symm(2).dual())
sage: PG.base_morphism()
ChowScheme morphism:
  From: Proj(Bundle(Grass(4, 3), 6, [1, -4*c, 10*c^2, -20*c^3]))
  To:   Grass(4, 3)
  Defn: Structure map
```

base_ring()

Return the Chow ring of the base of this ChowScheme. Synonym of : $meth$: $base_chowring()$.

EXAMPLES:

```
sage: P1 = ChowScheme(1, 'h', 1, 'h^2', 'h')
sage: P1.base_ring()
Quotient of Multivariate Polynomial Ring in no variables over Rational Field by the ideal (0
```

betti_numbers()

Return the Betti number of this ChowScheme.

EXAMPLES:

sage: P2 = Proj(2) sage: P = PointChowScheme sage: f = P.hom([0], P2) sage: B = Blowup(f).codomain() # P2 blown up in a point sage: B.betti_numbers() [1, 2, 1]

TESTS:

```
sage: X = PointChowScheme
sage: X.betti_numbers()
[1]
```

canonical_class()

Return the canonical class of this ChowScheme.

EXAMPLES:

```
sage: P2 = Proj(2)
sage: P2.canonical_class()
-3*h
```

chowring()

Return the Chow Ring of this ChowScheme.

EXAMPLES:

```
sage: P1 = ChowScheme(1, 'h', 1, 'h^2', 'h')
sage: P1.chowring()
Quotient of Multivariate Polynomial Ring in h over Rational Field by the ideal (h^2)
```

deg(*i=0*)

Return the degree of the i-th generator of the Chow ring of this ChowScheme. If i is unspecified the degree of the first generator (eg i=0) is returned if exists.

INPUT:

- `i` – an (optional) integer.

OUTPUT:

The degree of the i-th generator.

EXAMPLES:

```
sage: X = ChowScheme(1, 'h', 1, 'h^2', 'h')  # ChowScheme of P1
sage: X.deg()
1
sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2])
sage: X.deg()
1
sage: X.deg(1)
2
```

CORNER CASE:

```
sage: X = PointChowScheme
sage: X.deg()
Traceback (most recent call last):
...
IndexError: tuple index out of range
```

degs()

Return the degrees of the generators of the Chow ring of this ChowScheme.

OUTPUT:

A tuple of integers representing the degrees of the generators.

EXAMPLES:

```
sage: X = ChowScheme(1, 'h', 1, 'h^2', 'h')  # ChowScheme of P1
sage: X.degs()
(1,)
sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2])
sage: X.degs()
(1, 2)
```

CORNER CASE:

```
sage: X = PointChowScheme
sage: X.degs()
()
```

dimension()

Return the dimension of this ChowScheme.

OUTPUT:

An integer, the dimension of this ChowScheme

EXAMPLES:

```
sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2])
sage: X.dimension()
2
```

TESTS:

```
sage: X = PointChowScheme
sage: X.dimension()
0
```

euler_number()

Return the Euler number of this ChowScheme.

EXAMPLES:

sage: G = Proj(2) sage: G.euler_number() 3

TESTS:

```
sage: X = PointChowScheme
sage: X.euler_number()
1
```

gen(*i=0*)

Return the i-th generator for the Chow ring of this ChowScheme.

INPUT:

- `i` – an (optional) integer

OUTPUT:

The element of the Chow ring of this ChowScheme corresponding to the i-th generator.

EXAMPLES:

sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2]) sage: X.gen(1) c2

gens()

Return the generators for the Chow ring of this ChowScheme.

EXAMPLES:

sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2]) sage: X.gens() (c1, c2)

gens_dict()

Return the dictionary variable_name – generator of the Chow ring of this ChowScheme.

EXAMPLES:

```
sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2])
sage: gens = X.gens_dict()
sage: [(key, gens[key]) for key in sorted(gens)]
[('c1', c1), ('c2', c2)]
```

gens_dict_recursive()

Return recursively the dictionary variable_name – generator of the Chow ring of this ChowScheme.

EXAMPLES:

```
sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2])
sage: gens = X.gens_dict_recursive()
sage: [(key, gens[key]) for key in sorted(gens)]
[('c1', c1), ('c2', c2)]
```

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

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

INPUT:

- `x` – anything hat determines a ChowScheme morphism, typically the images of the generators of the ChowRing of Y in the ChowRing of X. 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 ChowScheme morphism from `self` to `Y` defined by `x`.

EXAMPLES:

```
sage: P1.<h> = ChowScheme(1, 'h', 1, 'h^2', 'h')
sage: P3.<k> = ChowScheme(3, 'k', 1, 'k^4', 'k^3')
sage: f = P1.hom([3*h], P3)  # Twisted cubic
sage: f(k)
3*h
```

TESTS:

```
sage: P = PointChowScheme
sage: P2.<h> = Proj(2)
sage: f = P.hom([0], P2)  # PointChowScheme in P2
sage: f(h)
0
```

identity_morphism()

Return the identity morphism.

OUTPUT:

The identity morphism of the ChowScheme `self`.

EXAMPLES:

```
sage: X = ChowScheme(1, 'h', 1, 'h^2', name='P1')
sage: X.identity_morphism()
ChowScheme endomorphism of P1
  Defn: Identity map
```

is_point()

Return True if this ChowScheme corresponds to a point.

EXAMPLES:

```
sage: P2 = Proj(2)
sage: P2.base().is_point()
True
```

TESTS:

```
sage: X = ChowScheme(0, point_class='1')
sage: X.is_point()
True
```

is_point_chowscheme()

Return True if this ChowScheme is the instance PointChowScheme

EXAMPLES:

```
sage: P2 = Proj(2)
sage: P2.is_point_chowscheme()
False
sage: P2.base().is_point_chowscheme()
True
```

TESTS:

```
sage: X = ChowScheme(0, point_class='1')
sage: X.is_point_chowscheme()
False
```

ngens()

Return the number of generators for the Chow ring of this ChowScheme.

EXAMPLES:

sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2]) sage: X.ngens() 2

nrels()

Return the number of relations of the Chow ring of this ChowScheme.

OUTPUT:

An integer, the number of relations.

EXAMPLES:

```
sage: X = ChowScheme(1, 'h', 1, 'h^2', 'h')
sage: X.nrels()
1
sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2])
sage: X.nrels()
0
```

o(*n=0*)

Return $\mathcal{O}(n)$ for this ChowScheme.

INPUT:

- n – an integer

EXAMPLES:

```
sage: G = Grass(4,3)
sage: G.o(1)
Bundle(Grass(4, 3), 1, [1, c])
sage: G.sheaves["universal_quotient"].determinant()
Bundle(Grass(4, 3), 1, [1, c])
```

TESTS:

```
sage: X = ChowScheme(1, 'h', 1, 'h^2', 'h')  # P1
sage: X.o(-2)
Traceback (most recent call last):
...
RuntimeError: o(1) is undefined yet.
```

point_class()

Return the point_class of this ChowScheme

OUTPUT:

The point_class of this ChowScheme.

EXAMPLES:

```
sage: X = ChowScheme(3, 'h', 1, 'h^4', 'h^3')  # ChowScheme of P3
sage: X.point_class()
h^3

sage: X = ChowScheme(1, 'h', 1, 'h^2')  # P1, no point_class
sage: X.point_class()
```

TESTS:

```
sage: X = PointChowScheme
sage: X.point_class()
1
```

rel(*i=0*)

Return the i-th relation of the Chow ring of this ChowScheme. If i is unspecified the first relation (eg i=0) is returned if exists.

INPUT:

- i – an (optional) integer.

OUTPUT:

The i-th relation.

EXAMPLES:

```
sage: X = ChowScheme(1, 'h', 1, 'h^2', 'h')  # P1
sage: X.rel()
h^2
```

TESTS:

```
sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2])
sage: X.rel()
Traceback (most recent call last):
...
IndexError: list index out of range
```

relative_dimension()

Return the relative dimension of this ChowScheme.

EXAMPLES:

```
sage: G = Grass(4, 3)
sage: Q = G.sheaves["universal_quotient"]
sage: Q.symm(2).dual().rank()
6
sage: ProjG = ProjBundle(Q.symm(2).dual())
sage: ProjG.relative_dimension()
5
```

rels()

Return the relations of Chow ring of this ChowScheme.

OUTPUT:

A list of ring elements representing the relations.

EXAMPLES:

```
sage: X = ChowScheme(1, 'h', 1, 'h^2', 'h')
sage: X.rels()
[h^2]
sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2])
sage: X.rels()
[]
```

TESTS:

```
sage: X = PointChowScheme
sage: X.rels()
[]
```

set_point_class(*value*)

Set the point_class of this ChowScheme.

INPUT:

- v – an element (or its string representation) of the point_class

EXAMPLES:

```
sage: X = ChowScheme(3, 'h', 1, 'h^4')  # ChowScheme of P3
sage: h = X.gen()
sage: X.set_point_class(h^3)

sage: X = ChowScheme(3, 'h', 1, 'h^4')  # ChowScheme of P3
sage: X.set_point_class('h^3')
```

TESTS:

```
sage: X = ChowScheme(3, 'h', 1, 'h^4')  # ChowRing of P3
sage: X.set_point_class('k^3')
Traceback (most recent call last):
...
TypeError: Can't coerce pointclass to ring element.
```

sheaves

Return a dictionary with sheaves on this ChowScheme. The naming convention is to suppress the word "bundle" in the keys. For instance, the *tangent bundle* has the key 'tangent', the *universal sub bundle* the key 'universal_sub' and the *universal quotient bundle* the key 'universal_quotient'.

EXAMPLES:

```
sage: P4 = Proj(4, name='P4')  # P4 in the sens of Grothendieck
sage: P4.sheaves["tangent"]
Bundle(P4, 4, [1, 5*h, 10*h^2, 10*h^3, 5*h^4])
sage: P4.sheaves["universal_sub"]
Bundle(P4, 4, [1, -h, h^2, -h^3, h^4])
sage: P4.sheaves["universal_quotient"]
Bundle(P4, 1, [1, h])
```

tangent_bundle()

Return the tangent bundle of this ChowScheme if defined.

EXAMPLES:

```
sage: G = Grass(4,2)
sage: G.tangent_bundle()
Bundle(Grass(4, 2), 4, [1, 4*c1, 7*c1^2, 12*c1*c2, 6*c2^2])
```

TESTS:

```
sage: P = PointChowScheme
sage: P.tangent_bundle()
Bundle(PointChowScheme, 0, [1])
```

todd_class()

Return the todd class of this ChowScheme.

EXAMPLES:

```
sage: Proj(3).todd_class()
h^3 + 11/6*h^2 + 2*h + 1
```

TESTS:

```
sage: P2 = ChowScheme(2, 'h', 1, 'h^3', 'h^2')
sage: P2.todd_class()
Traceback (most recent call last):
...
RuntimeError: Tangent bundle is undefined yet.
```

variable_name()

Return the (first) variable name of the Chow ring of this ChowScheme.

EXAMPLES:

```
sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2])
sage: X.variable_name()
'c1'
```

variable_names()

Return the variable names of the Chow ring of this ChowScheme.

EXAMPLES:

```
sage: X = ChowScheme(2, ['c1', 'c2'], [1, 2])
sage: X.variable_names()
('c1', 'c2')
```

sage.schemes.chow.scheme.**is_chowScheme**(*x*)

Test whether x is a chow scheme.

INPUT:

- x – anything.

OUTPUT:

Boolean. Whether x derives from ChowScheme_generic.

TESTS:

```
sage: X = ChowScheme(1)
sage: is_chowScheme(X)
True
sage: is_chowScheme(1)
False
sage: is_chowScheme(PointChowScheme)
True
```

CHAPTER

SEVEN

CHOW: THE CATEGORY OF CHOWSCHEMES

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

class sage.schemes.chow.schemes.**ChowSchemes**(*s=None*)
Bases: sage.categories.category.Category

Construct a category of ChowSchemes.

class HomCategory(*category*, **args*)
Bases: sage.categories.homsets.HomsetsCategory

TESTS:

```
sage: from sage.categories.covariant_functorial_construction import CovariantConstructionCat
sage: class FooBars(CovariantConstructionCategory):
...       _functor_category = "FooBars"
sage: Category.FooBars = lambda self: FooBars.category_of(self)
sage: C = FooBars(ModulesWithBasis(ZZ))
sage: C
Category of foo bars of modules with basis over Integer Ring
sage: C.base_category()
Category of modules with basis over Integer Ring
sage: latex(C)
\mathbf{FooBars}(\mathbf{ModulesWithBasis}_{\Bold{Z}})
sage: import __main__; __main__.FooBars = FooBars # Fake FooBars being defined in a python m
sage: TestSuite(C).run()
```

extra_super_categories()
x.__init__(...) initializes x; see help(type(x)) for signature

ChowSchemes.**super_categories**()
x.__init__(...) initializes x; see help(type(x)) for signature

class sage.schemes.chow.schemes.**ChowSchemes_over_base**(*base*, *name=None*)
Bases: sage.categories.category_types.Category_over_base

The category of schemes over a given base scheme.

base_chowscheme()
x.__init__(...) initializes x; see help(type(x)) for signature

super_categories()
x.__init__(...) initializes x; see help(type(x)) for signature

CHAPTER

EIGHT

CHOW: THE SET OF MORPHISMES BETWEEN CHOWSCHEMES

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

class sage.schemes.chow.scheme_homset.**ChowSchemeHomsetFactory**
Bases: sage.structure.factory.UniqueFactory

Factory for SHom-sets of schemes.

create_key_and_extra_args(*X*, *Y*, *category=None*, *base=None*, *check=True*)
Create a key that uniquely determines the SHom-set.

INPUT:

- X – a chowscheme. The domain of the morphisms.
- Y – a chowscheme. The codomain of the morphisms.
- **category – a category for the SHom-sets (default: chowschemes** over given base).
- base – a chowscheme. The base chowscheme of domain and codomain.
- check – boolean (default: True).

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.

class sage.schemes.chow.scheme_homset.**ChowSchemeHomset_generic**(*X*, *Y*, *category=None*, *check=True*, *base=None*)
Bases: sage.categories.homset.HomsetWithBase

The base class for SHom-sets of ChowSchemes.

INPUT:

- X – a chowscheme. The domain of the SHom-set.
- Y – a chowscheme. The codomain of the SHom-set.
- **category – a category (optional). The category of the** SHom-set.

- **check – boolean (optional, default=‘‘True‘‘). Whether to** check the defining data for consistency.

Element
alias of ChowSchemeMorphism

natural_map()
Return a natural map in the SHom space.

OUTPUT:

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

zero_element()
Backward compatibility alias for self.zero()

sage.schemes.chow.scheme_homset.**is_ChowSchemeHomset**(*H*)
Test whether H is a chowscheme SHom-set.

CHAPTER

NINE

CHOW: THE BLOWUP OF A CHOWSCHEME Y ALONG A CHOWSCHEME X

Suppose given two projective non singular varieties X and Y and a morphism of ChowSchemes $f : X \to Y$. Then the `Blowup` computes the Blowup of Y along X.

EXAMPLE (the Veronese embedding):

```
sage: P2.<h> = Proj(2, 'h')
sage: P5.<k> = Proj(5, 'k')
sage: f = P2.hom([2*h], P5)
sage: g = Blowup(f)
```

Note that `Blowup` returns a morphism as follows:

$$\begin{array}{ccc} \widetilde{X} & \stackrel{g}{\to} & \widetilde{Y} \\ \downarrow & & \downarrow\sigma \\ X & \stackrel{f}{\to} & Y \end{array}$$

Hence in order to get $B = \widetilde{Y}$ and to check for example for generators, relations, tangent bundle or betti numbers of the Blowup use the codomain of g:

```
sage: B = Blowup(f).codomain()
```

In this particular example we expect an additional divisor e:

```
sage: B.chowring().gens()
(e, k)
sage: B.chowring().rels()
[k^6, e*k^3, e^3 - 9/2*e^2*k + 15/2*e*k^2 - 4*k^3]
```

The tangent bundle on B is equally computed:

```
sage: TB = B.tangent_bundle()
sage: TB.chern_classes()
[1, -2*e + 6*k, -15/2*e*k + 15*k^2, 9/2*e^2*k - 93/4*e*k^2 + 28*k^3, 27/4*e^2*k^2 + 27*k^4, 12*k^5]
```

as well as the usual topological invariants:

```
sage: B.betti_numbers()
[1, 2, 3, 3, 2, 1]
sage: B.euler_number()
12
sage: B.euler_number() == TB.chern_classes()[5].integral()
True
```

Finally one can answer the classical problem of finding the smooth plane conics tangent to five given general conics. As each tangency is a degree 6 condition on the $\mathbb{P}^5$ of all (not necessarily smooth) conics, containing the double lines, one may compute as follows:

```
sage: (e, k) = B.chowring().gens()
sage: ((6*k - 2*e)^5).integral()
3264
```

There is no restriction on the number of generators of the Chow ring of the exceptional divisor:

```
sage: P2xP2 = Proj(2, 'h') * Proj(2, 'k')
sage: P8 = Proj(8, 'l')
sage: f = P2xP2.hom(['h+k'], P8)  # Segre map P2xP2 -> P8
sage: g = Blowup(f)
sage: B = g.codomain()
sage: B.gens()
(e1, e2, e3, l)
sage: B.betti_numbers()
[1, 2, 4, 7, 8, 7, 4, 2, 1]
```

TESTS:

```
sage: P2.<h> = Proj(2, 'h')
sage: P5.<k> = Proj(5, 'k')
sage: f = P2.hom([2*h], P5)
sage: g = Blowup(f)
sage: TestSuite(g).run(skip=["_test_category"])
```

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

class sage.schemes.chow.blowup.**Blowup**(*f*, *var_name='e'*, *proj_var_name='z'*, *verbose=False*, *domain_name=None*, *codomain_name=None*, *latex_domain_name=None*, *latex_codomain_name=None*)

Bases: sage.schemes.chow.morphism.ChowSchemeMorphism

Construct the blowup of a ChowSchemeMorphism representing an embedding of smooth projective varieties:

EXAMPLE:

```
sage: X.<w> = Proj(1, 'w', name='X')
sage: Y.<h> = Proj(3, 'h', name='Y')
sage: i = X.hom([3*w], Y)
sage: g = Blowup(i)
sage: XX, YY = g.domain(), g.codomain()
sage: XX.chowring().gens()  # XX is a ProjBundle hence the generator z
(z, w)
sage: YY.chowring().gens()  # YY is the Blowup hence the class e
(e, h)
sage: YY.chowring().basis()
[h^3, h^2, e*h, h, e, 1]
sage: YY.chowring().intersection_matrix()
[ 0  0  0  0  0  1]
[ 0  0  0  1  0  0]
[ 0  0  0  0 -3  0]
[ 0  1  0  0  0  0]
[ 0  0 -3  0  0  0]
[ 1  0  0  0  0  0]
```

CHAPTER

TEN

CHOW: MORPHISMS BETWEEN CHOWSCHEMES

A morphism between ChowSchemes is given by a ring map between its Chow rings. For example, the Veronese embedding is given as follows:

```
sage: X.<x> = Proj(2, 'x', name='X')
sage: Y.<y> = Proj(5, 'y', name='Y')
sage: f = X.hom([2*x], Y); f
ChowScheme morphism:
  From: X
  To:   Y
  Defn: y |--> 2*x
```

The domain and codomain are retrieved as usual:

```
sage: f.domain() == X
True
sage: f.codomain() == Y
True
```

Compute the images of elements y in the Chow ring of the codomain, either by `f.upperstar(y)` or simply `f(y)`:

```
sage: f.upperstar(y)
2*x
sage: f(y)
2*x
sage: f(y^2)
4*x^2
sage: f(2*y^3)
0
```

The composition of two morphisms is given by applying the * operator:

```
sage: Z.<z> = Proj(56, 'z', name='Z')
sage: g = Y.hom([3*y], Z)
sage: h = g * f
sage: k = X.hom([6*x], Z)  # Direct definition of the composition
sage: h == k
True
```

If the point_classes of the domain and codomain are defined, e.g. domain and codomain are representing projective non singular varieties, as they are for `library.proj.Proj`, one can compute `f.lowerstar(x)` for elements x in the Chow ring of the domain:

```
sage: f.lowerstar(x)
2*y^4
```

```
sage: f.lowerstar(x^2)
y^5
sage: f.lowerstar(1 + x^2)
y^5 + 4*y^3
```

If E and F are sheaves on X and Y respectively, compute the pullback and pushforward as follows:

```
sage: TY = Y.tangent_bundle()
sage: f.upperstar(TY)
Bundle(X, 5, [1, 12*x, 60*x^2])
sage: TX = X.tangent_bundle()
sage: f.lowerstar(TX)
Sheaf(Y, 0, [1, 0, 0, 16*y^3, 72*y^4, 240*y^5])
```

TESTS:

```
sage: TestSuite(f).run(skip=["_test_category"])
```

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

class sage.schemes.chow.morphism.**ChowSchemeMorphism**(*parent*, *phi*)
Bases: sage.structure.element.Element

Base class for morphisms between ChowSchemes.

category()
Return the category of the SHom-set.

OUTPUT:

A category.

EXAMPLES:

```
sage: X.<w> = Proj(1, 'w')
sage: Y.<h> = Proj(3, 'h')
sage: i = X.hom(['3 * w'], Y)
sage: i.category()
Category of homsets of chowschemes over PointChowScheme
```

chowring_morphism()
Return the morphism on the level of Chow rings for this ChowSchemeMorphism.

EXAMPLES:

```
sage: P1.<h> = ChowScheme(1, 'h', 1, 'h^2', 'h', name='P1')
sage: P3.<k> = ChowScheme(3, 'k', 1, 'k^4', 'k^3', name='P3')
sage: f = P1.hom([3*h], P3)
sage: f.chowring_morphism()
Ring morphism:
  From: A(P3)
  To:   A(P1)
  Defn: k |--> 3*h
```

codomain()
Return the codomain of this ChowSchemeMorphism.

OUTPUT:

A ChowsScheme. The codomain of this ChowSchemeMorphism.

EXAMPLES:

```
sage: X.<w> = Proj(1, 'w')
sage: Y.<h> = Proj(3, 'h')
sage: i = X.hom(['3 * w'], Y)
sage: i.codomain()
ChowScheme(3, 'h', 1, 'h^4', 'h^3')
```

domain()

Return the domain of this ChowSchemeMorphism.

OUTPUT:

A Chowscheme. The domain of this ChowSchemeMorphism

EXAMPLES:

```
sage: X.<w> = Proj(1, 'w')
sage: Y.<h> = Proj(3, 'h')
sage: i = X.hom(['3 * w'], Y)
sage: i.domain()
ChowScheme(1, 'w', 1, 'w^2', 'w')
```

is_endomorphism()

Return whether the morphism is an endomorphism.

OUTPUT:

Boolean. Return true if the domain and codomain are identical.

EXAMPLES:

```
sage: X.<w> = Proj(1, 'w')
sage: Y.<h> = Proj(3, 'h')
sage: i = X.hom(['3 * w'], Y)
sage: i.is_endomorphism()
False
sage: j = X.hom(['2 * w'], X)
sage: j.is_endomorphism()
True
```

lowerstar(*v*, *normal_bundle=None*, *verbose=False*)

Return $f_*(v)$ where v can be a class in $A^*(X)$ or a sheaf on the domain X. Both, the domain X and the codomain Y are supposed to be projective, e.g. their point_classes respectively have to be known.

In case, v is a sheaf, the tangent bundles on X and Y have to be known as Grothendieck-Hirzebruch-Riemann-Roch is used for the computation. Note, as indicated already above, the result should be understood as a virtual sheaf (e.g. the alternating sum of $R^i f_* v$).

If the normal bundle is specified, f is supposed to be an embedding and Riemann-Roch without denominateurs is used for the computation. In this case no need for the tangent bundles on X and Y and the result is not virtual.

INPUT:

- `v` – a class in the Chow ring of the domain or a sheaf on the domain.
- `normal_bundle` – the (optional) normal bundle of this ChowSchemeMorphism (supposed to be an embedding in this case).

OUTPUT:

A class in the Chow ring of the codomain or a sheaf on the codomain depending whether v is a class or a sheaf.

EXAMPLES:

```
sage: X.<w> = Proj(1, 'w', name='X')
sage: Y.<h> = Proj(3, 'h', name='Y')
sage: i = X.hom([3*w], Y)
sage: i.lowerstar(w)
h^3
sage: i.lowerstar(X.o(1))
Sheaf(Y, 0, [1, 0, -3*h^2, -8*h^3])
sage: i.lowerstar(X.tangent_bundle())
Sheaf(Y, 0, [1, 0, -3*h^2, -6*h^3])

sage: P2.<h> = ChowScheme(2, 'h', 1, 'h^3', 'h^2', name='P2')
sage: P5.<k> = ChowScheme(5, 'k', 1, 'k^6', 'k^5', name='P5')
sage: f = P2.hom([2*h], P5)
sage: E = Bundle(P2, 2, [1, 3*h, 3*h^2])
sage: N = Bundle(P2, 3, [1, 9*h, 30*h^2])  # Normal bundle
sage: f.lowerstar(E, normal_bundle=N)
Sheaf(P5, 0, [1, 0, 0, 16*k^3, 72*k^4, 240*k^5])

sage: P2 = Proj(2)
sage: f = P2.base_morphism()  # P2 -> Pt
sage: f.lowerstar(P2.o(1)).rank()  # = dim H^0(P2, O(1)) par GHRR
3
```

upperstar(*v*)

Return $f^*(v)$ where v can be a class in $A^*(Y)$ or a sheaf on Y.

INPUT:

- v – a class in the Chow ring of the codomain or a sheaf on the codomain.

OUTPUT:

A class in the Chow ring of the domain or a sheaf on the domain depending whether v is a class or a sheaf.

EXAMPLES:

```
sage: X.<w> = Proj(1, 'w', name='X')
sage: Y.<h> = Proj(3, 'h', name='Y')
sage: i = X.hom(['3 * w'], Y)
sage: i.upperstar(h)
3*w
sage: i.upperstar(Y.o(1))
Bundle(X, 1, [1, 3*w])
sage: i.upperstar(Y.tangent_bundle())
Bundle(X, 3, [1, 12*w])
```

class sage.schemes.chow.morphism.**ChowSchemeMorphism_id**(*X*)

Bases: sage.schemes.chow.morphism.ChowSchemeMorphism

Return the identity morphism from X to itself.

INPUT:

- X – the ChowScheme.

EXAMPLES:

```
sage: from sage.schemes.chow.morphism import ChowSchemeMorphism_id
sage: X = ChowScheme(1, 'h', 1, 'h^2', name='P1')
sage: ChowSchemeMorphism_id(X)
ChowScheme endomorphism of P1
  Defn: Identity map
```

class sage.schemes.chow.morphism.**ChowSchemeMorphism_structure_map**(*parent*, *f*)

Bases: sage.schemes.chow.morphism.ChowSchemeMorphism

The structure morphism

INPUT:

- parent – SHom-set with codomain equal to the base ChowScheme of the domain.

sage.schemes.chow.morphism.**is_chowSchemeMorphism**(*f*)

Test whether f is a chowscheme morphism.

INPUT:

- f – anything.

OUTPUT:

Boolean. Return True if f is a chowscheme morphism.

EXAMPLES:

```
sage: P1.<h> = ChowScheme(1, 'h', 1, 'h^2', 'h', name='P1')
sage: P3.<k> = ChowScheme(3, 'k', 1, 'k^4', 'k^3', name='P3')
sage: H = P1.Hom(P3)
sage: f = H([3*h])
sage: is_chowSchemeMorphism(f)
True
```

CHAPTER

ELEVEN

CHOW: SHEAVES ON CHOWSCHEMES

A sheaf on a ChowScheme X is an element of the Grothendieck ring $G(X)$ of coherent sheaves on X with the rational coefficients. In view of the ring isomorphism given by the Chern character:

$$ch : G(X) \otimes_{\mathbf{Z}} \mathbf{Q} \xrightarrow{\simeq} A(X)^* \otimes_{\mathbf{Z}} \mathbf{Q}$$

a sheaf will be represented by its Chern character.

The class `Sheaf` implements most of the common operations on sheaves. Note that in view of the above all these operations should be understood as "virtual". For example, the tensor product of two sheaves E and F returns the alternating sum of the $\mathrm{Tor}_i(E,F)$ and $\mathrm{SHom}(E,F)$ the alternating sum of the $\mathrm{Ext}^i(E,F)$. Similarly, if f is a proper morphism of ChowSchemes, f_* of a sheaf E returns the alternating sum of the higher direct images $R^i f_* E$.

A sheaf on a ChowScheme X may be specified either by its Chern character or its rank r and Chern classes $c_1, \ldots, c_r$. Typically, for example in order to define a rank 2 sheaf with Chern classes $c_1 = 0$ and $c_2 = 4$ on $\mathbb{P}^2$ one writes

```
sage: P2.<h> = Proj(2, 'h', name='P2')
sage: E = Sheaf(P2, 2, [1, 0, 4*h^2]); E
Sheaf(P2, 2, [1, 0, 4*h^2])
```

Note that if one wants to emphasize that E is actually a bundle, one may use the module `bundle`:

```
sage: E = Bundle(P2, 2, [1, 0, 4*h^2]); E
Bundle(P2, 2, [1, 0, 4*h^2])
```

The class :class‘bundle.Bundle‘ derives from `Sheaf` and may have some additional methods in future versions which apply only for vector bundles. Also tensor products, etc. of bundles return bundles.

TESTS:

```
sage: P2.<h> = ChowScheme(2, 'h', 1, 'h^3', name='P2')
sage: O1 = Sheaf(P2, 1, [1, h])
sage: TestSuite(O1).run()
```

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

`sage.schemes.chow.sheaf.`**`SEnd`**`(E)`

Return the sheaf of endomorphisms of E.

INPUT:

- `E` - a sheaf

OUTPUT:

A sheaf.

EXAMPLES:

```
sage: P2 = Proj(2, 'h')
sage: SEnd(P2.tangent_bundle())
Bundle(Proj(2, 'h'), 4, [1, 0, 3*h^2])
```

sage.schemes.chow.sheaf.**SHom**(*E*, *F*)

Return the sheaf of homomorphisms from E to F.

INPUT:

- E - a sheaf
- F - a sheaf

OUTPUT:

A sheaf.

EXAMPLES:

```
sage: P2 = Proj(2, 'h')
sage: SHom(P2.sheaves['universal_sub'], P2.sheaves['universal_quotient'])
Bundle(Proj(2, 'h'), 2, [1, 3*h, 3*h^2])
sage: P2.tangent_bundle()
Bundle(Proj(2, 'h'), 2, [1, 3*h, 3*h^2])
```

class sage.schemes.chow.sheaf.**Sheaf**(*X*, *r=None*, *cc=None*, *ch=None*, *name=None*, *latex_name=None*)

Bases: sage.structure.sage_object.SageObject

A *sheaf* is an element of the Grothendieck ring of coherent sheaves on a ChowScheme (tensored with $\mathbf{Q}$) and represented by its Chern character. This class provides operations on sheaves to be understood as virtual, e.g. in the Grothendieck ring.

adams(*k*)

Return the k-th Adams operator applied to this sheaf.

EXAMPLES:

```
sage: X.<c1, c2> = ChowScheme(2, ['c1', 'c2'], [1, 2], name='X')
sage: E = Bundle(X, 1, [1, c1])
sage: all([(E.adams(n) == E^n) for n in range(-2,3)])
True
sage: F = Bundle(X, 2, [1, c1, c2])
sage: all([E.adams(k) + F.adams(k) == (E + F).adams(k) for k in range(-2, 3)])
True
sage: F.adams(2).adams(3) == F.adams(2*3)
True
```

chern_character()

Return the Chern character of this Sheaf.

OUTPUT:

- an element of the underlying Chow ring of this Sheaf.

EXAMPLE:

```
sage: X.<c1, c2, c3> = ChowScheme(3, ['c1', 'c2', 'c3'], [1, 2, 3])
sage: S = Sheaf(X, 3, [1, c1, c2, c3])
```

```
sage: ch = S.chern_character(); ch.by_degrees()
[3, c1, 1/2*c1^2 - c2, 1/6*c1^3 - 1/2*c1*c2 + 1/2*c3]
```

chern_classes()

Return the Chern classes of this Sheaf

OUTPUT:

- a list of elements of the underlying Chow ring of this Sheaf.

EXAMPLE:

```
sage: X.<c1, c2, c3> = ChowScheme(3, ['c1', 'c2', 'c3'], [1, 2, 3])
sage: ch = 3 + c1 + (1/2*c1^2 - c2) + (1/6*c1^3 - 1/2*c1*c2 + 1/2*c3)
sage: S = Sheaf(X, ch=ch)
sage: S.chern_classes()
[1, c1, c2, c3]
```

chowscheme()

Return the underlying ChowScheme.

OUTPUT:

- a ChowScheme

EXAMPLE:

```
sage: X = ChowScheme(4, 'h', 1, 'h^5')
sage: S = Sheaf(X, 1, [1])
sage: S.chowscheme()
ChowScheme(4, 'h', 1, 'h^5')
```

det()

Return the determinant of this sheaf. Synonym for determinant().

TESTS:

```
sage: P1 = Proj(1, name='P1')
sage: (P1.o() - P1.o(-1)).det()
Sheaf(P1, 1, [1, h])
```

determinant()

Return the determinant of this sheaf. Synonym for determinant().

EXAMPLES:

```
sage: X.<c1, c2> = ChowScheme(2, ['c1', 'c2'], [1, 2], name='X')
sage: E = Sheaf(X, 2, [1, c1, c2], name='E')
sage: E.det()
Sheaf(X, 1, [1, c1])
```

TESTS:

```
sage: X.<c1, c2> = ChowScheme(2, ['c1', 'c2'], [1, 2], name='X')
sage: E = Sheaf(X, 2, [1, c1, c2], name='E')
sage: str(E.det())
'det(E)'
sage: latex(E.det())
\det(\mathcal{E})
```

dual()

Return the dual of this Sheaf. If the sheaf is a bundle, this is the dual bundle. In general, the result is

virtual.

OUTPUT:

•a sheaf (or a bundle this sheaf is a bundle)

TESTS:

```
sage: variables = ['c1', 'c2', 'c3', 'c4']
sage: degrees = [1, 2, 3, 4]
sage: X = ChowScheme(4, variables, degrees, name='X')
sage: (c1, c2, c3, c4) = X.gens()
sage: E = Sheaf(X, 4, [1, c1, c2, c3, c4], name='E')
sage: ED = E.dual(); ED
Sheaf(X, 4, [1, -c1, c2, -c3, c4])
sage: str(ED)
'E^*'
sage: latex(ED)
\mathcal{E}^{*}
sage: E = Sheaf(X, 1, [1, c1], name='E', latex_name='\\bold{E}')
sage: latex(E.dual())
\bold{E}^{*}
```

euler_characteristic()

Return the euler-characteristic of this sheaf using Hirzebruch-Riemann-Roch. Requires the underlying ChowScheme to be a point.

EXAMPLES:

```
sage: P3 = Proj(3)
sage: P3.o(-4).euler_characteristic()
-1
```

rank()

Return the rank of this Sheaf.

OUTPUT:

•an integer.

EXAMPLE:

```
sage: X.<c1, c2, c3> = ChowScheme(3, ['c1', 'c2', 'c3'], [1, 2, 3])
sage: S = Sheaf(X, 3, [1, c1, c2, c3])
sage: S.rank()
3
```

sans_denominateurs(E)

Returns the expression P(self,E) (see Fulton 296-297, Lemma 15.3).

EXAMPLES:

```
sage: P2 = Proj(2)
sage: P5 = Proj(5, 'k')
sage: f = P2.hom(['2*h'], P5)
sage: N = f.upperstar(P5.tangent_bundle()) - P2.tangent_bundle(); N
Sheaf(Proj(2, 'h'), 3, [1, 9*h, 30*h^2])
sage: N.sans_denominateurs(P2.tangent_bundle())
[4, 36*h, 240*h^2]
```

segre_classes()

Returns $[s_0, ..., s_r]$ where the s_i are the Segre classes.

Nota Bene: We use the convention that $s_t(E) * c_t(E^*) = 1$ (in Fulton's book, $s_t(E) * c_t(E) = 1$ is used).

OUTPUT:

•a list of elements of the underlying Chow ring of this Sheaf.

EXAMPLE:

```
sage: X.<c1, c2, c3> = ChowScheme(3, ['c1', 'c2', 'c3'], [1, 2, 3])
sage: S = Sheaf(X, 3, [1, c1, c2, c3])
sage: S.segre_classes()
[1, c1, c1^2 - c2, c1^3 - 2*c1*c2 + c3]
```

symbolic_chern_polynomial(*var_name='t'*)

Return the symbolic Chern polynomial of this Sheaf.

INPUT:

•`var_name` – a (optional) string representing the formal variable.

OUTPUT:

•a symbolic polynomial in `var_name`.

EXAMPLE:

```
sage: X.<c1, c2> = ChowScheme(3, ['c1', 'c2'], [1, 2], name='X')
sage: S = Sheaf(X, 2, [1, -c1, c2])
sage: S.symbolic_chern_polynomial()
1 - c1*t + c2*t^2
```

symbolic_segre_polynomial(*var_name='t'*)

Return the symbolic Segre polynomial s_t.

Nota Bene: We use the convention that $s_t(E) * c_t(E^*) = 1$ (in Fulton's book, $s_t(E) * c_t(E) = 1$ is used).

INPUT:

•`var_name` – a (optional) string representing the formal variable.

OUTPUT:

•a symbolic polynomial in `var_name`.

EXAMPLES:

```
sage: X.<c1, c2, c3> = ChowScheme(3, ['c1', 'c2', 'c3'], [1, 2, 3])
sage: S = Sheaf(X, 3, [1, c1, c2, c3])
sage: S.symbolic_segre_polynomial()
1 + c1*t + (c1^2 - c2)*t^2 + (c1^3 - 2*c1*c2 + c3)*t^3
```

symm(*p*)

Return the p-th symmetric power of this sheaf.

EXAMPLES:

```
sage: X.<c1, c2> = ChowScheme(2, ['c1', 'c2'], [1, 2], name='X')
sage: F = Bundle(X, 2, [1, c1, c2])
sage: F.adams(2) == (F.symm(2) - F.wedge(2))
True
```

TESTS:

```
sage: X.<c1, c2> = ChowScheme(2, ['c1', 'c2'], [1, 2], name='X')
sage: F = Bundle(X, 2, [1, c1, c2])
sage: F.symm(0) == TrivialBundle(X, 1)
```

```
True
sage: F.symm(1) == F
True
```

todd_class()

Returns the todd class of this sheaf.

EXAMPLES:

```
sage: P3 = Proj(3)
sage: P3.todd_class()
h^3 + 11/6*h^2 + 2*h + 1
sage: P3.todd_class() == P3.tangent_bundle().todd_class()
True
```

total_chern_class()

Return the total Chern class of this Sheaf

OUTPUT:

- an element of the underlying Chow ring of this Sheaf.

EXAMPLE:

```
sage: X.<c1, c2, c3> = ChowScheme(3, ['c1', 'c2', 'c3'], [1, 2, 3])
sage: S = Sheaf(X, 3, [1, c1, c2, c3])
sage: S.total_chern_class()
c3 + c2 + c1 + 1
```

total_segre_class()

Return the total Segre class of this Sheaf.

Nota Bene: We use the convention that $s_t(E) * c_t(E^*) = 1$ (in Fulton's book, $s_t(E) * c_t(E) = 1$ is used).

OUTPUT:

- an element of the underlying Chow ring of this Sheaf.

EXAMPLES:

```
sage: X.<c1, c2, c3> = ChowScheme(3, ['c1', 'c2', 'c3'], [1, 2, 3])
sage: S = Sheaf(X, 3, [1, c1, c2, c3])
sage: S.total_segre_class()
c1^3 - 2*c1*c2 + c3 + c1^2 - c2 + c1 + 1
```

wedge(p)

Return the p-th exterior power of this sheaf.

EXAMPLES:

```
sage: X.<c1, c2> = ChowScheme(2, ['c1', 'c2'], [1, 2], name='X')
sage: F = Bundle(X, 2, [1, c1, c2])
sage: F.wedge(2) == F.det()
True
```

TESTS:

```
sage: X.<c1, c2> = ChowScheme(2, ['c1', 'c2'], [1, 2], name='X')
sage: F = Bundle(X, 2, [1, c1, c2])
sage: F.wedge(0) == TrivialBundle(X, 1)
True
sage: F.wedge(1) == F
True
```

sage.schemes.chow.sheaf.**is_sheaf**(*x*)

Test whether x is a sheaf.

INPUT:

- x – anything.

OUTPUT:

Boolean. Return True if x is a Sheaf, False otherwise.

TESTS:

```
sage: P5 = Proj(5)
sage: is_sheaf(P5.o(1))
True
```

sage.schemes.chow.sheaf.**sans_denominateurs_p**(*n*, *d*, *e*)

Return the expression $P(D, E)$ (see Fulton 296-297, Lemma 15.3) where D and E are bundles of ranks d and e respectively on a variety V of dimension n.

Note that we compute *up to dimension* $n+d$ as in applications D is generally the normal bundle of an embedding $V \to W$.

INPUT:

- n – the dimension of V
- d – the rank of D
- e – the rank of E

OUTPUT:

A tuple $(A, [p_O, \ldots, p_{n+d}])$ of a ring of coefficients A and a list of coefficients p_i such that $P(D, E) = \sum p_i$.

EXAMPLES:

```
sage: from sage.schemes.chow.sheaf import sans_denominateurs_p
sage: A, P = sans_denominateurs_p(3, 1, 1); P
[1, d1 - e1, d1^2 - 2*d1*e1 + e1^2, d1^3 - 3*d1^2*e1 + 3*d1*e1^2 - e1^3]
sage: A, P = sans_denominateurs_p(3, 1, 4); P
[4, 10*d1 - e1, 20*d1^2 - 5*d1*e1 + e1^2 - 2*e2, 35*d1^3 - 15*d1^2*e1 + 6*d1*e1^2 - e1^3 - 11*d1
sage: A, P = sans_denominateurs_p(2, 2, 4); P
[-4, -4*d1 + 2*e1, -4*d1^2 + 10*d2 + 3*d1*e1 - 3*e1^2 + 6*e2]
sage: A, P = sans_denominateurs_p(1, 3, 4); P
[8, 12*d1 - 6*e1]
```

CHAPTER

TWELVE

CHOW: BUNDLES ON A CHOWSCHEME

A bundle on a ChowScheme X is represented by its rank r and Chern classes $c_0, \ldots, c_r$. Typically, for example in order to define a rank 2 bundles with Chern classes 0 and 4 on the projective plane:

```
sage: P2.<h> = ChowScheme(2, 'h', 1, 'h^3', name='P2')
sage: E = Bundle(P2, 2, [1, 0, 4*h^2]); E
Bundle(P2, 2, [1, 0, 4*h^2])
sage: E.chern_character()
-4*h^2 + 2
sage: Sheaf(P2, ch=E.chern_character())
Sheaf(P2, 2, [1, 0, 4*h^2])
```

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

class sage.schemes.chow.bundle.**Bundle**(*X*, *r=None*, *cc=None*, *ch=None*, *name=None*, *latex_name=None*)

Bases: sage.schemes.chow.sheaf.Sheaf

Class for Bundles on ChowSchemes.

TESTS:

```
sage: P2.<h> = ChowScheme(2, 'h', 1, 'h^3', name='P2')
sage: E = Bundle(P2, 2, [1, 0, 4*h^2])
sage: TestSuite(E).run()
```

sage.schemes.chow.bundle.**BundleDiffRelations**(*B*, *A*)

Return the relations given by the difference of two bundles B and A on a ChowScheme X.

INPUT:

- B – a bundle on a ChowScheme X
- A – a bundle on a ChowScheme X

OUTPUT:

A list of elements of the Chow ring of X.

EXAMPLE (get the relations for Grass(6, 4)):

```
sage: from sage.schemes.chow.bundle import BundleDiffRelations
sage: G64 = ChowScheme(8, ['w1', 'w2'], [1, 2])
sage: O = TrivialBundle(G64, 6)
sage: S = Bundle(G64, 2, [1, '-w1', 'w1^2-w2'])  # Universal Sub
sage: rels = BundleDiffRelations(O, S); rels
```

```
[-2*w1^3*w2 + 3*w1*w2^2, w1^6 - 2*w1^4*w2 + w2^3]
sage: A = ChowRing(['w1', 'w2'], [1, 2], [str(x) for x in rels])
sage: A.rels()  # Returns the relations in a standard basis.
[w2^5, w1*w2^4, w1^2*w2^3 - 4/3*w2^4, w1^6 - 3*w1^2*w2^2 + w2^3, w1^3*w2 - 3/2*w1*w2^2]

sage: Grass(6, 4, 'w').rels()
[w2^5, w1*w2^4, w1^2*w2^3 - 4/3*w2^4, w1^6 - 3*w1^2*w2^2 + w2^3, w1^3*w2 - 3/2*w1*w2^2]
```

sage.schemes.chow.bundle.**TrivialBundle**(*X*, *r*)

Return the trivial bundle of rank r on the ChowScheme X.

INPUT:

- X – a ChowScheme, the base
- r – an integer, the rank.

OUTPUT:

The trivial bundle of rank r on X.

TESTS:

sage: P2.<h> = ChowScheme(2, 'h', 1, 'h^3', name='P2') sage: TrivialBundle(P2, 3) Bundle(P2, 3, [1])

sage.schemes.chow.bundle.**is_bundle**(*x*)

Test whether x is a bundle.

INPUT:

- x – anything.

OUTPUT:

Boolean. Return True if x is a bundle.

EXAMPLES:

```
sage: P2.<h> = ChowScheme(2, 'h', 1, 'h^3', name='P2')
sage: B = Bundle(P2, 2, [1, 0, 3*h^2])
sage: is_bundle(B)
True
sage: is_bundle(P2)
False
```

CHAPTER

THIRTEEN

CHOW: THE PROJ CHOWSCHEME

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

sage.schemes.chow.library.proj.**Proj**(*n*, *hyperplane_class='h'*, *names=None*, *name=None*, *latex_name=None*)

Return the projective space $\mathbb{P}^n$ of dimension n.

INPUT:

- n – An integer, the dimension of the projective space.
- hyperplane_class - An (optional) name for the hyperplane class
- name – An optional string, the name of the ChowScheme
- latex_name– An optional string, the latex representation of the ChowScheme

OUTPUT:

- The ChowScheme corresponding to the projective space in the sense of Grothendieck, i.e. the rank 1 quotients of $mathbbC^{n+1}$.

EXAMPLES:

```
sage: X = Proj(3)  # P3 of rank 1 quotients of a 4 dim. vector space
sage: X.sheaves["universal_sub"]
Bundle(Proj(3, 'h'), 3, [1, -h, h^2, -h^3])
```

sage.schemes.chow.library.proj.**ProjBundle**(*E*, *hyperplane_class='h'*, *names=None*, *name=None*, *latex_name=None*)

Return the *Proj* of a bundle E.

INPUT:

- E – A sheaf on a ChowScheme
- hyperplane_class - An (optional) name for the hyperplane class
- name – An optional string, the name of the ProjBundle
- latex_name– An optional string, the latex representation of the ProjBundle

OUTPUT:

- The ChowScheme corresponding to $\mathbb{P}(E)$ in the sense of Grothendieck, i.e. the rank 1 quotient modules of E.

EXAMPLES:

```
sage: P3 = Proj(3, name='P3')
sage: S = P3.sheaves['universal_sub']
sage: PS = ProjBundle(S); str(PS)
'Proj(Bundle(P3, 3, [1, -h, h^2, -h^3]))'
```

CHAPTER

FOURTEEN

CHOW: THE GRASSMANNIAN CHOWSCHEME

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

sage.schemes.chow.library.grass.**Grass**(*n*, *r*, *chern_class='c'*, *names=None*, *name=None*, *latex_name=None*)

Return either depending respectively whether $n > r$ or $n < r$:

•The Grassmannian of quotients of rank r of an n dimensional vector space;

•The Grassmannian of subspaces of rank n of an r dimensional vector space.

EXAMPLES:

```
sage: G = Grass(6, 4, chern_class='w')
sage: G.dimension()
8
sage: G.rels()
[w2^5, w1*w2^4, w1^2*w2^3 - 4/3*w2^4, w1^6 - 3*w1^2*w2^2 + w2^3, w1^3*w2 - 3/2*w1*w2^2]
sage: G.sheaves["universal_sub"]
Bundle(Grass(6, 4), 2, [1, -w1, w1^2 - w2])
sage: G.sheaves["universal_quotient"]
Bundle(Grass(6, 4), 4, [1, w1, w2, -w1^3 + 2*w1*w2, -w1^4 + w1^2*w2 + w2^2])

sage: H = Grass(6, 2, chern_class='v')
sage: H.dimension()
8
sage: H.rels()
[v2^5, v1*v2^4, v1^2*v2^3 - v2^4, v1^3*v2^2 - 2*v1*v2^3, v1^4*v2 - 3*v1^2*v2^2 + v2^3, v1^5 - 4*
sage: H.sheaves["universal_sub"]
Bundle(Grass(6, 2), 4, [1, -v1, v1^2 - v2, -v1^3 + 2*v1*v2, v1^4 - 3*v1^2*v2 + v2^2])
sage: H.sheaves["universal_quotient"]
Bundle(Grass(6, 2), 2, [1, v1, v2])
```

sage.schemes.chow.library.grass.**GrassBundle**(*A*, *B*, *chern_class='c'*, *names=None*, *name=None*, *latex_name=None*)

Return either

•the Grassmannian of quotients of rank B of A if A is a sheaf or

•the Grassmannian of subbundles of B of rank A if B is a bundle.

CHAPTER

FIFTEEN

CHOW: TWISTED CUBICS

The variety of nets of quadrics defining twisted cubics relative to a base.

We briefly explain the algorithm of Ellingsrud and Strømme in order to compute the variety of nets of quadrics defining twisted cubics as well as the incidence variety relative to a base.

Let E_0 and F_0 denote vector spaces of dimension 3 and 2, respectively, considered as representations of the group $G = \mathrm{Gl}_3 \times \mathrm{Gl}_2$ via the standard action of the left and right factor, respectively. If $L = \det(E) \otimes \det(F)^*$, then the restriction of the representations $E \otimes L^*$ and $F \otimes L^*$ to the diagonal subgroup $\Delta = \{(tI_3, tI_2) \mid t \in \mathbb{C}^*\}$ are trivial.

Let W denote a locally free sheaf on a scheme S and let $U := \mathrm{SHom}(E_0, F_0 \otimes W)$ denote the S-scheme of linear maps $A : E_0 \to F_0 \otimes_{\mathbb{C}} W_s$, $s \in S$. The open subset $U^{ss} \subset U$ of semistable points for the action of G on U consists of matrices A such that there are no non-zero subspaces $F' \subset F_0$ and $E' \subset E_0$ with $A(F_0) \subset E_0 \otimes W$ and $\dim(E')/\dim(F') < \dim(E_0)/\dim(F_0)$. Let $b : X_W \to S$ denote the GIT-quotient of U^{ss} by the action of G.

The cohomology ring of X_W can be described as follows: The stabiliser subgroup of any point in U^{ss} is the diagonal group Δ. Since this group acts trivially on $\mathcal{O}_U \otimes (E_0 \otimes L^*)$ and $\mathcal{O}_U \otimes (F_0 \otimes L^*)$, these bundles descend to vector bundles E and F (of rank 3 and 2, resp.) on X_W, and there is a tautological homomorphism $a : F \to E \otimes b^*W$. The Chern classes e_1, e_2, e_3 of E and f_1, f_2 of F generate $H^*(X_W, \mathbb{Q})$ as an $H^*(S, \mathbb{Q})$-algebra. One checks directly that $\det(E) = \det(F)$ so that $e_1 = f_1$.

Let $t : T \to X_W$ denote the variety of pairs of full flags of F and E, and let

$$0 \subset B_1 \subset t^*F, \quad 0 \subset A_1 \subset A_2 \subset t^*E$$

denote the universal flag of subsheaves. As $H^*(S, \mathbb{Q})$-algebra, the cohomology ring of T is generated by the Chern roots

$$b_1 = c_1(B_1),\ b_2 = c_1(t^*F/B_1), \qquad a_1 = c_1(A_1), a_2 = c_1(A_2/A_1), a_3 = c_1(t^*E/A_2).$$

In particular, the classes f_1, f_2 and e_1, e_2, e_3 are the elementary symmetric polynomials of the b_1, b_2 and a_1, a_2, a_3, respectively. The composite homomorphism

$$B_1 \to t^*F \to t^*E \otimes W \to (t^*E/A_1) \otimes W \text{ and } t^*F \to t^*E \otimes W \to (t^*E/A_2) \otimes W$$

cannot vanish anywhere on T due to the semistability of the points on in X_W. This implies that the top Chern class of the corresponding SHom-bundles must vanish, which yields the relations

$$c_8\mathcal{H}om(B_1, (t^*E/A_1) \otimes W) = 0, \quad c_8\mathcal{H}om(t^*F, (t^*E/A_2) \otimes W) = 0.$$

The relations of the Chow-ring of X_W are now the coefficients of these Chern classes with respect to a basis of $H^*(S, \mathbb{Q})[a_1, a_2, a_3, b_1, b_2]$ considered as a finite free module over $H^*(S, \mathbb{Q})[f_1, f_2, e_1, e_1, e_3]$ plus the relation $e_1 = f_2$.

The tangent bundle is computed using the exact sequence,

$$0 \to \mathcal{O}_X \to End(E) \oplus End(F) \to SHom(F, E \otimes W) \to TX \to 0$$

This is implemented in variety_of_nets_of_quadrics().

The incidence variety can be obtained in a two-step process $I_W = \mathbb{P}(K_1^*) \xrightarrow{v} \mathbb{P}(W) \xrightarrow{u} S$, where $0 \to K_1 \to u^*W \to L_1 \to 0$ is the tautological sequence on $\mathbb{P}(W)$.

This is implemented in incidence_variety().

Also, let $0 \to K_2 \to v^*K_1^* \to L_2 \to 0$ be the tautological sequence on I_W. Since the Chow ring of $H^*(X_W)$ is generated by the Chern classes of E and F, the embedding $f : I_W \to X_W$ is determined as soon as one can identify f^*E and f^*F. According to Ellingsrud and Strømme , these bundles are $f^*E = K_1 \otimes L_2^*$ and $f^*F = K_2 \otimes L_2^* \otimes \det(K_1)$.

Finally the morphism f is implemented in map_incidence_to_nets_of_quadrics().

REFERENCE:

> Ellingsrud, Geir and Strømme, Stein Arild: The number of twisted cubics on the general quintic threefold, Math. Scand. 76 (1995) 5-34

AUTHORS:

- Manfred Lehn (2013)
- Christoph Sorger (2013)

sage.schemes.chow.library.twisted_cubics.**incidence_variety**(*W*, *name=None*, *latex_name=None*)

The Incidence variety I.

EXAMPLES:

```
sage: P = PointChowScheme
sage: W = Bundle(P, 4, [1])
sage: I = incidence_variety(W)
sage: I.dimension()
5
sage: I.euler_number()
12
sage: I.betti_numbers()
[1, 2, 3, 3, 2, 1]
sage: c5 = I.tangent_bundle().chern_classes()[5]
sage: c5.integral() == I.euler_number()
True

sage: P = Grass(1, 5, 'w')
sage: W = P.sheaves["universal_quotient"]
sage: I = incidence_variety(W)
sage: I.betti_numbers()
[1, 3, 6, 9, 11, 11, 9, 6, 3, 1]
sage: I.euler_number()
60
sage: top = I.tangent_bundle().chern_classes()[I.dimension()]
sage: top.integral() == I.euler_number()
True

sage: G = Grass(6, 4, 'w')
sage: W = G.sheaves["universal_quotient"]
sage: I = incidence_variety(W)
sage: I.betti_numbers()
[1, 3, 7, 12, 18, 23, 26, 26, 23, 18, 12, 7, 3, 1]
sage: I.euler_number()
180
sage: top = I.tangent_bundle().chern_classes()[I.dimension()]
```

```
sage: top.integral() == I.euler_number()
True
```

sage.schemes.chow.library.twisted_cubics.**map_incidence_to_nets_of_quadrics**(*W*, *domain_name=None*, *codomain_name=None*, *latex_domain_name=No*, *latex_codomain_name=*

Returns the map $f : I_S \to X_S$ from the incidence variety to the variety of nets of quadrics.

EXAMPLES:

```
sage: P = PointChowScheme
sage: W = Bundle(P, 4, [1])
sage: f = map_incidence_to_nets_of_quadrics(W)
```

sage.schemes.chow.library.twisted_cubics.**variety_of_nets_of_quadrics**(*W*, *name=None*, *latex_name=None*)

Returns the variety of nets of quadrics defining twisted cubics relative to S and a rank 4 vector bundle W of S.

If S is given as parameter the variety will be relative to this instance of S.

EXAMPLES:

```
sage: P = PointChowScheme
sage: W = Bundle(P, 4, [1])
sage: X = variety_of_nets_of_quadrics(W)
sage: X.euler_number()
58
sage: top = X.tangent_bundle().chern_classes()[X.dimension()]
sage: top.integral() == X.euler_number()
True

sage: P = Grass(1, 5, 'w')
sage: W = P.sheaves["universal_quotient"]
sage: X = variety_of_nets_of_quadrics(W)
sage: X.euler_number()
290
sage: top = X.tangent_bundle().chern_classes()[X.dimension()]
sage: top.integral() == X.euler_number()
True
```

CHAPTER

SIXTEEN

INDICES AND TABLES

- Index
- Module Index
- Search Page

[EPS] Ellingsrud, Geir and Piene, Ragni and Strømme, Stein Arild: On the variety of nets of quadrics defining twisted cubics, Space curves (Rocca di Papa, 1985), Springer (1987), Lecture Notes in Math. 1266, pp. 84–96

[ES] Ellingsrud, Geir and Strømme, Stein Arild: The number of twisted cubics on the general quintic threefold, Math. Scand. 76 (1995) 5-34

[F] Fulton, William: Intersection theory, 2nd edition (1998) Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer Verlag

[GSE] Macaulay2 — a software system for research in algebraic geometry. by Daniel R. Grayson, Michael E. Stillman, David Eisenbud

[GSSEC] Schubert2 — computation in intersection theory. by Daniel R. Grayson, Michael E. Stillman, Stein A. Strømme, David Eisenbud and Charley Crissman

[K] Kass, Jesse Leo: The Chow ring of the blow-up of P3 along the twisted cubic.

[KS] Schubert — a Maple package for Intersection Theory by Sheldon Katz and Stein Arild Stromme

[LLSvS] Twisted cubics on cubic fourfolds, by Christian Lehn, Manfred Lehn, Christoph Sorger and Duco van Straten

[S] Schubert, Hermann: Kalkül der abzählenden Geometrie Teubner Verlag (1879)

S

A

B

C

L

M

N

O

P

R

S

T

U

V

W

Z