COMPUTING CUP-PRODUCTS IN INTEGRAL COHOMOLOGY OF HILBERT SCHEMES OF POINTS ON K3 SURFACES

SIMON KAPFER

Abstract. We study cup products in integral cohomology of the Hilbert scheme of n points on a K3 surface and present a computer program for this purpose. In particular, we deal with the question, which classes can be represented by products of lower degrees.

The Hilbert schemes of n points on a complex surface parametrize all zero-dimensional subschemes of length n. Studying their rational cohomology, Nakajima [8] was able to give an explicit description of the vector space structure in terms of the action of a Heisenberg algebra. The Hilbert schemes of points on a K3 surface are one of the few known classes of Irreducible Holomorphic Symplectic Manifolds. Lehn and Sorger [4] developed an algebraic model to describe the cohomological ring structure. On the other hand, Qin and Wang [9] found a base for integral cohomology in the projective case. By combining these results, we are able to compute everything explicitely in the cohomology rings of Hilbert schemes of n points on a projective K3 with integral coefficients. For $n = 2$, this was done by Boissière, Nieper-Wißkirchen and Sarti [1], who applied their results to automorphism groups of prime order. When n is increasing, the cohomology rings become very large, so we need the help of a computer. The source code is available under `https://github.com/s--kapfer/HilbK3`

Our goal here is to give some characteristics for low degrees. Denote by $S^{[3]}$ the Hilbert scheme of 3 points on a projective K3 surface (or a deformation equivalent space). We identify $\operatorname{Sym}^2 H^2(S^{[3]}, \mathbb{Z})$ with its image in $H^4(S^{[3]}, \mathbb{Z})$ under the cup product mapping.

Theorem 0.1. *The cup product mappings for the Hilbert scheme of 3 points on a projective K3 surface have the following cokernels:*

$$\frac{H^4(S^{[3]}, \mathbb{Z})}{\operatorname{Sym}^2 H^2(S^{[3]}, \mathbb{Z})} \cong \frac{\mathbb{Z}}{3\mathbb{Z}} \oplus \mathbb{Z}^{\oplus 23} \tag{1}$$

$$\frac{H^6(S^{[3]}, \mathbb{Z})}{H^2(S^{[3]}, \mathbb{Z}) \smile H^4(S^{[3]}, \mathbb{Z})} \cong \left(\frac{\mathbb{Z}}{3\mathbb{Z}}\right)^{\oplus 23} \tag{2}$$

Although the case $n = 3$ is most interesting for us, our computer program allows computations for arbitrary n. We give some numerical results results in Section 2.

Acknowledgements. The author thanks Samuel Boissière and Marc Nieper-Wißkirchen for their supervision and many helpful comments. He also thanks Grégoire Menet for stimulating proposals and the Laboratoire de Mathématiques et Applications of the university of Poitiers for its hospitality. This work was partially supported by a DAAD grant.

Date: October 31, 2014.

1. Preliminaries

Definition 1.1. Let n be a natural number. A partition of n is a sequence $\lambda = (\lambda_1 \geq \ldots \geq \lambda_k > 0)$ of natural numbers such that $\sum_i \lambda_i = n$. It is convenient to write $\lambda = (1^{m_1}, 2^{m_2}, \ldots)$ as a sequence of multiplicities. We define the weight $\|\lambda\| := \sum m_i i = n$ and the length $|\lambda| := \sum_i m_i = k$. We also define $z_\lambda := \prod_i i^{m_i} m_i!$.

Definition 1.2. Let Λ be the ring of symmetric functions. Let m_λ and p_λ denote the monomial and the power sum symmetric functions. They are both indexed by partitions and form a basis of Λ. They are linearly related by $p_\lambda = \sum_\mu \psi_{\lambda\mu} m_\mu$, the sum being over partitions with the same weight as λ. It turns out that the base change matrix $(\psi_{\lambda\mu})$ has integral entries, but its inverse $(\psi^{-1}_{\mu\lambda})$ has not. For example, $p_{(2,1,1)} = 2m_{(2,1,1)} + 2m_{(2,2)} + 2m_{(3,1)} + m_{(4)}$ but $m_{(2,1,1)} = \frac{1}{2}p_{(2,1,1)} - \frac{1}{2}p_{(2,2)} - p_{(3,1)} + p_{(4)}$. A method to determine the $(\psi_{\lambda\mu})$ is given by Lascoux in [3, Sect. 3.7].

Definition 1.3. A lattice L is a free $\mathbb{Z}$-module of finite rank, equipped with a non-degenerate symmetric integral bilinear form B. The lattice L is called odd, if there exists a $v \in L$, such that $B(v,v)$ is odd, otherwise it is called even. If the map $v \mapsto B(v,v)$ takes both negative and positive values on L, the lattice is called indefinite. Choosing a base $\{e_i\}_i$ of our lattice, we can write B as a symmetric matrix. L is called unimodular, if the matrix B has determinant ± 1. The difference between the number of positive eigenvalues and the number of negative eigenvalues of B (regarded as a matrix over $\mathbb{R}$) is called the signature.

There is the following classification theorem. See [7, Chap. II] for reference.

Theorem 1.4. *Any two indefinite unimodular lattices L, L' are isometric iff they have the same rank, signature and parity. Evenness implies that the signature is divisible by 8. In particular, if L is odd, then L possesses an orthogonal basis and is hence isometric to $\langle 1 \rangle^{\oplus k} \oplus \langle -1 \rangle^{\oplus l}$ for some $k, l \geq 0$. If L is even, then L is isometric to $U^{\oplus k} \oplus (\pm E_8)^{\oplus l}$ for some $k, l \geq 0$.*

Definition 1.5. Let S be a projective K3 surface. We fix integral bases 1 of $H^0(S,\mathbb{Z})$, x of $H^4(S,\mathbb{Z})$ and $\alpha_1, \ldots, \alpha_{22}$ of $H^2(S,\mathbb{Z})$. The cup product induces a symmetric bilinear form B_{H^2} on $H^2(S,\mathbb{Z})$ and thus the structure of a unimodular lattice. We may extend B_{H^2} to a symmetric non-degenerate bilinear form B on $H^*(S,\mathbb{Z})$ by setting $B(1,1) = 0,\ B(1,\alpha_i) = 0,\ B(1,x) = 1,\ B(x,x) = 0$.

By the Hirzebruch index theorem, we know that $H^2(S,\mathbb{Z})$ has signature -16 and, by the classification theorem for indefinite unimodular lattices, is isomorphic to $U^{\oplus 3} \oplus (-E_8)^{\oplus 2}$.

Definition 1.6. B induces a form $B \otimes B$ on $\operatorname{Sym}^2 H^*(S,\mathbb{Z})$. So the cup-product

$$\mu : \operatorname{Sym}^2 H^*(S,\mathbb{Z}) \longrightarrow H^*(S,\mathbb{Z})$$

induces an adjoint comultiplication Δ that is coassociative, given by:

$$\Delta : H^*(S,\mathbb{Z}) \longrightarrow \operatorname{Sym}^2 H^*(S,\mathbb{Z}), \quad \Delta = -(B \otimes B)^{-1} \mu^T B$$

with the property $B(\Delta(a), b \otimes c) = -B(a, b \smile c)$. Note that this does not define a bialgebra structure. The image of 1 under the composite map $\mu(\Delta(1)) = B(\Delta(1), \Delta(1)) = 24x$, denoted by e is called the Euler Class.

We denote by $S^{[n]}$ the Hilbert scheme of n points on S, *i.e.* the classifying space of all zero-dimensional closed subschemes of length n. $S^{[0]}$ consists of a single point and $S^{[1]} = S$. Fogarty proved that the Hilbert scheme is a smooth variety. A theorem by Nakajima gives an explicit description of the vector space structure of $H^*(S^{[n]},\mathbb{Q})$ in terms of creation operators

$$\mathfrak{q}_l(\beta) : H^*(S^{[n]},\mathbb{Q}) \longrightarrow H^{*+k+l-1}(S^{[n+l]},\mathbb{Q}),$$

where $\beta \in H^k(S,\mathbb{Q})$, acting on the direct sum $\mathbb{H} := \bigoplus_n H^*(S^{[n]},\mathbb{Q})$. The operators $\mathfrak{q}_l(\beta)$ are linear and commute with each other. The vacuum vector $|0\rangle$ is defined as the generator of $H^0(S^{[0]},\mathbb{Q}) \cong \mathbb{Q}$. The images of $|0\rangle$ under the polynomial algebra generated by the creation operators span $\mathbb{H}$ as a vector space. Following [9], we abbreviate $\mathfrak{q}_{l_1}(\beta)\dots\mathfrak{q}_{l_k}(\beta) =: \mathfrak{q}_\lambda(\beta)$, where the partition λ is composed by the l_i.

An integral basis for $H^*(S^{[n]},\mathbb{Z})$ in terms of Nakajima's operators was given by Qin–Wang:

Theorem 1.7. [9, Thm. 5.4.] *Let $\mathfrak{m}_{\nu,\alpha} := \sum_\rho \psi^{-1}_{\nu\rho}\,\mathfrak{q}_\rho(\alpha)$, with coefficients $\psi^{-1}_{\nu\rho}$ as in Definition 1.2. The classes*

$$\frac{1}{z_\lambda}\mathfrak{q}_\lambda(1)\mathfrak{q}_\mu(x)\mathfrak{m}_{\nu^1,\alpha_1}\dots\mathfrak{m}_{\nu^{22},\alpha_{22}}|0\rangle, \qquad \|\lambda\| + \|\mu\| + \sum_{i=1}^{22}\|\nu^i\| = n$$

form an integral basis for $H^*(S^{[n]},\mathbb{Z})$. Here, λ, μ, ν^i are partitions.

Notation 1.8. To enumerate the basis of $H^*(S^{[n]},\mathbb{Z})$, we introduce the following abbreviation:

$$\boldsymbol{\alpha}^{\boldsymbol{\lambda}} := 1^{\lambda^0}\alpha_1^{\lambda^1}\dots\alpha_{22}^{\lambda^{22}}x^{\lambda^{23}} := \frac{1}{z_{\widetilde{\lambda^0}}}\mathfrak{q}_{\widetilde{\lambda^0}}(1)\mathfrak{q}_{\lambda^{23}}(x)\mathfrak{m}_{\lambda^1,\alpha_1}\dots\mathfrak{m}_{\lambda^{22},\alpha_{22}}|0\rangle$$

where the partition $\widetilde{\lambda^0}$ is built from λ^0 by appending sufficiently many Ones, such that $\left\|\widetilde{\lambda^0}\right\| + \sum_{i\geq 1}\left\|\lambda^i\right\| = n$. If $\sum_{n\geq 0}\left\|\lambda^i\right\| > n$, we put $\boldsymbol{\alpha}^{\boldsymbol{\lambda}} = 0$. Thus we can interpret $\boldsymbol{\alpha}^{\boldsymbol{\lambda}}$ as an element of $H^*(S^{[n]},\mathbb{Z})$ for arbitrary n. We say that the symbol $\boldsymbol{\alpha}^{\boldsymbol{\lambda}}$ is reduced, if λ^0 contains no Ones. We define also $\|\boldsymbol{\lambda}\| := \sum_{n\geq 0}\left\|\lambda^i\right\|$.

Lemma 1.9. *Let $\boldsymbol{\alpha}^{\boldsymbol{\lambda}}$ represent a class of cohomological degree $2k$. If $\boldsymbol{\alpha}^{\boldsymbol{\lambda}}$ is reduced, then $\frac{k}{2} \leq \|\boldsymbol{\lambda}\| \leq 2k$.*

Proof. This is a simple combinatorial observation. The lower bound is witnessed by $x^{(\frac{k}{2})}$ (if k is even) and the upper bound is witnessed by $1^{(2^k)}$. □

The ring structure of $H^*(S^{[n]},\mathbb{Q})$ has been studied by Lehn and Sorger in [4], where an explicit algebraic model is constructed, which we recall briefly:

Definition 1.10. [4, Sect. 2] Let π be a permutation of n letters, written as a sum of disjoint cycles. To each cycle we may associate an element of $A := H^*(S,\mathbb{Q})$. This defines an element in $A^{\otimes m}$, m being the number of cycles. For example, a term like $(1\,2\,3)_{\alpha_1}(4\,5)_{\alpha_2}$ may describe a permutation consisting of two cycles with associated classes $\alpha_1,\alpha_2 \in A$. Thus we construct a vector space $A\{S_n\} := \bigoplus_{\pi\in S_n} A^{\otimes(\langle\pi\rangle\backslash[n])}$.

To define a ring structure, take two permutations π,τ with associated elements of A. The result of the multiplication will be the permutation $\pi\tau$, together with a mapping of cycles. To construct the mappings to A, look first at the orbit space of the group of permutations $\langle\pi,\tau\rangle$, generated by π and τ. For each cycle of π,τ contained in one orbit B of $\langle\pi,\tau\rangle$, multiply with the associated element of A.

Also multiply with a certain power of the Euler class e^g. Afterwards, apply the comultiplication Δ repeatedly on the product to get a mapping from the cycles of $\pi\tau$ contained in B to A. Here the "graph defect" g is defined as follows: Let u, v, w be the number of cycles contained in B of π, τ, $\pi\tau$, respectively. Then $g := \frac{1}{2}(|B| + 2 - u - v - w)$. Now follow this procedure for each orbit B.

The symmetric group S_n acts on $A\{S_n\}$ by conjugation. This action preserves the ring structure. Therefore the space of invariants $A^{[n]} := (A\{S_n\})^{S_n}$ becomes a subring. The main theorem of [4] can now be stated:

Theorem 1.11. [4, Thm. 3.2.] *The following map is an isomorphism of rings:*

$$H^*(S^{[n]}, \mathbb{Q}) \longrightarrow A^{[n]}$$
$$\mathfrak{q}_{n_1}(\beta_1) \dots \mathfrak{q}_{n_k}(\beta_k)|0\rangle \longmapsto \sum_{\sigma \in S_n} \sigma a \sigma^{-1}$$

with $n_1 + \ldots + n_k = n$ and $a \in A\{S_n\}$ corresponds to an arbitrary permutation with k cycles of lengths $n_1, \ldots, n_k$ that are associated to the classes $\beta_1, \ldots, \beta_k \in H^*(S, \mathbb{Q})$, respectively.

Since $H^{\text{odd}}(S^{[n]}, \mathbb{Z}) = 0$ and $H^{\text{even}}(S^{[n]}, \mathbb{Z})$ is torsion-free by [5], we can apply these results to $H^*(S^{[n]}, \mathbb{Z})$ to determine the multiplicative structure of cohomology with integer coefficients. It turns out, that it is somehow independent of n. More precisely, we have the following stability theorem, by Li, Qin and Wang:

Theorem 1.12. (Derived from [9, Thm. 2.1]). *Let $Q_1, \ldots, Q_s$ be products of creation operators, i.e. $Q_i = \prod_j \mathfrak{q}_{\lambda_{i,j}}(\beta_{i,j})$ for some partitions $\lambda_{i,j}$ and classes $\beta_{i,j} \in H^*(S, \mathbb{Z})$. Set $n_i := \sum_j \|\lambda_{i,j}\|$. Then the cup product $\prod_{i=1}^s \left(\frac{1}{(n-n_i)!}\mathfrak{q}_{n-n_i}(1)\, Q_i\, |0\rangle\right)$ equals a finite linear combination of classes of the form $\frac{1}{(n-m)!}\mathfrak{q}_{n-m}(1) \prod_j \mathfrak{q}_{\mu_j}(\gamma_j)|0\rangle$, with $\gamma \in H^*(S, \mathbb{Z})$, $m = \sum_j \|\mu_j\|$, whose coefficients are independent of n. We have the upper bound $m \leq \sum_i n_i$.*

Corollary 1.13. *Let $\boldsymbol{\alpha}^{\boldsymbol{\lambda}}, \boldsymbol{\alpha}^{\boldsymbol{\mu}}, \boldsymbol{\alpha}^{\boldsymbol{\nu}}$ be reduced. Assume $n \geq \|\boldsymbol{\lambda}\|, \|\boldsymbol{\mu}\|$. Then the coefficients $c_{\boldsymbol{\nu}}^{\boldsymbol{\lambda}\boldsymbol{\mu}}$ of the cup product in $H^*(S^{[n]}, \mathbb{Z})$*

$$\boldsymbol{\alpha}^{\boldsymbol{\lambda}} \smile \boldsymbol{\alpha}^{\boldsymbol{\mu}} = \sum_{\boldsymbol{\nu}} c_{\boldsymbol{\nu}}^{\boldsymbol{\lambda}\boldsymbol{\mu}} \boldsymbol{\alpha}^{\boldsymbol{\nu}}$$

are polynomials in n of degree at most $\|\boldsymbol{\lambda}\| + \|\boldsymbol{\mu}\| - \|\boldsymbol{\nu}\|$.

Proof. Set $Q_{\boldsymbol{\lambda}} := \mathfrak{q}_{\lambda^0}(1)\mathfrak{q}_{\lambda^{23}}(x)\prod_{1 \leq j \leq 22} \mathfrak{q}_{\lambda^j}(\alpha_j)$ and $n_{\boldsymbol{\lambda}} := \|\boldsymbol{\lambda}\|$. Then we have: $\boldsymbol{\alpha}^{\boldsymbol{\lambda}} = \frac{1}{(n-n_{\boldsymbol{\lambda}})!\, z_{\lambda^0}}\mathfrak{q}_{n-n_{\boldsymbol{\lambda}}}(1) Q_{\boldsymbol{\lambda}}|0\rangle$ and $\boldsymbol{\alpha}^{\boldsymbol{\mu}} = \frac{1}{(n-n_{\boldsymbol{\mu}})!\, z_{\mu^0}}\mathfrak{q}_{n-n_{\boldsymbol{\mu}}}(1) Q_{\boldsymbol{\mu}}|0\rangle$. Thus the coefficient $c_{\boldsymbol{\nu}}^{\boldsymbol{\lambda}\boldsymbol{\mu}}$ in the product expansion is a constant, which depends on $\|\boldsymbol{\lambda}\|, \|\boldsymbol{\mu}\|, \|\boldsymbol{\nu}\|$, but not on n, multiplied with $\frac{(n-n_{\boldsymbol{\nu}})!}{(n-m)!}$ for a certain $m \leq n_{\boldsymbol{\lambda}} + n_{\boldsymbol{\mu}}$. This is a polynomial of degree $m - n_{\boldsymbol{\nu}} \leq n_{\boldsymbol{\lambda}} + n_{\boldsymbol{\mu}} - n_{\boldsymbol{\nu}} = \|\boldsymbol{\lambda}\| + \|\boldsymbol{\mu}\| - \|\boldsymbol{\nu}\|$. □

Remark 1.14. The above condition, $n \geq \|\boldsymbol{\lambda}\|, \|\boldsymbol{\mu}\|$, seems to be unnecessary. In particular, if $\|\boldsymbol{\nu}\| \leq n < \max\{\|\boldsymbol{\lambda}\|, \|\boldsymbol{\nu}\|\}$, the polynomial $c_{\boldsymbol{\nu}}^{\boldsymbol{\lambda}\boldsymbol{\mu}}$ has a root at n.

Example 1.15. Here are some explicit examples for illustration.

(1) $1^{(2,2)} \smile \alpha_i^{(2)} = -2 \cdot 1^{(2)}\alpha_i^{(1)}x^{(1)} + 1^{(2,2)}\alpha_i^{(2)} + 2 \cdot 1^{(2)}\alpha_i^{(3)} + \alpha_i^{(4)}$ for $i \in \{1..22\}$.
(2) Let $i, j \in \{1 \ldots 22\}$. If $i \neq j$, then $\alpha_i^{(2)} \smile \alpha_j^{(1)} = \alpha_i^{(2)}\alpha_j^{(1)} + 2B(\alpha_i, \alpha_j) \cdot x^{(1)}$. Otherwise, $\alpha_i^{(2)} \smile \alpha_i^{(1)} = \alpha_i^{(3)} + \alpha_i^{(2,1)} + 2B(\alpha_i, \alpha_i) \cdot x^{(1)}$.

(3) Set $\boldsymbol{\alpha^\lambda} = 1^{(2)}$ and $\boldsymbol{\alpha^\nu} = x^{(1)}$. Then $c_{\boldsymbol{\nu}}^{\boldsymbol{\lambda\lambda}} = -(n-1)$.
(4) Set $\boldsymbol{\alpha^\lambda} = 1^{(2,2)}$ and $\boldsymbol{\alpha^\nu} = x^{(1,1)}$. Then $c_{\boldsymbol{\nu}}^{\boldsymbol{\lambda\lambda}} = \frac{(n-3)(n-2)}{2}$.
(5) Let i,j be indices, such that $B(\alpha_i,\alpha_j)=1,\ B(\alpha_i,\alpha_i)=0=B(\alpha_j,\alpha_j)$ and let $k\geq 0$. Set $\boldsymbol{\alpha^\lambda} = \alpha_i^{(1)}\alpha_j^{(1)}x^{(1^k)}$ and $\boldsymbol{\alpha^\nu} = x^{(1^{2k})}$. Then $c_{\boldsymbol{\nu}}^{\boldsymbol{\lambda\lambda}} = 1$.

2. Computational results

We now give some results in low degrees, obtained by computing multiplication matrices with respect to the integral basis. To get their cokernels, one has to reduce them to Smith normal form. Both was done using a computer.

Remark 2.1. Denote $h^k(S^{[n]})$ the rank of $H^k(S^{[n]},\mathbb{Z})$. We have:

- $h^2(S^{[n]}) = 23$ for $n\geq 2$.
- $h^4(S^{[n]}) = 276,\ 299,\ 300$ for $n = 2,3,\geq 4$ resp.
- $h^6(S^{[n]}) = 23,\ 2554,\ 2852,\ 2875,\ 2876$ for $n = 2,3,4,5,\geq 6$ resp.

The algebra generated by classes of degree 2 is an interesting object to study. For cohomology with complex coefficients, Verbitsky has proven in [10] that the cup product mapping from $\mathrm{Sym}^k H^2(S^{[n]},\mathbb{C})$ to $H^{2k}(S^{[n]},\mathbb{C})$ is injective for $k\leq n$. Since there is no torsion, one concludes that this also holds for integral coefficients.

Proposition 2.2. *We identify* $\mathrm{Sym}^2 H^2(S^{[n]},\mathbb{Z})$ *with its image in* $H^4(S^{[n]},\mathbb{Z})$ *under the cup product mapping. Then:*

$$\frac{H^4(S^{[2]},\mathbb{Z})}{\mathrm{Sym}^2 H^2(S^{[2]},\mathbb{Z})} \cong \left(\frac{\mathbb{Z}}{2\mathbb{Z}}\right)^{\oplus 23} \oplus \frac{\mathbb{Z}}{5\mathbb{Z}}, \tag{1}$$

$$\frac{H^4(S^{[3]},\mathbb{Z})}{\mathrm{Sym}^2 H^2(S^{[3]},\mathbb{Z})} \cong \frac{\mathbb{Z}}{3\mathbb{Z}} \oplus \mathbb{Z}^{\oplus 23}, \tag{2}$$

$$\frac{H^4(S^{[n]},\mathbb{Z})}{\mathrm{Sym}^2 H^2(S^{[n]},\mathbb{Z})} \cong \mathbb{Z}^{\oplus 24}, \quad \textit{for } n\geq 4. \tag{3}$$

The 3-torsion part in (2) is generated by the integral class $1^{(3)}$.

Remark 2.3. The torsion in the case $n=2$ was also computed by Boissière, Nieper-Wißkirchen and Sarti, [1, Prop. 3] using similar techniques. For all the author knows, the result for $n=3$ is new. The freeness result for $n\geq 4$ was already proven by Markman, [6, Thm. 1.10], using a completely different method.

Proposition 2.4. *For triple products of* $H^2(S^{[n]},\mathbb{Z})$*, we have:*

$$\frac{H^6(S^{[2]},\mathbb{Z})}{\mathrm{Sym}^3 H^2(S^{[2]},\mathbb{Z})} \cong \frac{\mathbb{Z}}{2\mathbb{Z}}$$

The quotient is generated by the integral class $1^{(2)}$.

$$\frac{H^6(S^{[3]},\mathbb{Z})}{\mathrm{Sym}^3 H^2(S^{[3]},\mathbb{Z})} \cong \left(\frac{\mathbb{Z}}{2\mathbb{Z}}\right)^{\oplus 230} \oplus \left(\frac{\mathbb{Z}}{36\mathbb{Z}}\right)^{\oplus 22} \oplus \frac{\mathbb{Z}}{72\mathbb{Z}} \oplus \mathbb{Z}^{\oplus 507}$$

$$\frac{H^6(S^{[4]},\mathbb{Z})}{\mathrm{Sym}^3 H^2(S^{[4]},\mathbb{Z})} \cong \frac{\mathbb{Z}}{2\mathbb{Z}} \oplus \mathbb{Z}^{\oplus 552}$$

For $n\geq 5$*, the quotient is free.*

Proof. For the freeness result, it is enough to check the case $n=5$, since we have the canonical split inclusions $\mathfrak{q}_1(1): H^k(S^{[n]},\mathbb{Z}) \hookrightarrow H^k(S^{[n+1]},\mathbb{Z})$ for all n,k. ☐

We study now cup products between classes of degree 2 and 4. The case of $S^{[3]}$ is of particular interest.

Proposition 2.5. *The cup product mapping* $: H^2(S^{[n]},\mathbb{Z})\otimes H^4(S^{[n]},\mathbb{Z}) \to H^6(S^{[n]},\mathbb{Z})$ *is neither injective (unless $n=0$) nor surjective (unless $n\leq 2$). We have:*

$$\frac{H^6(S^{[3]},\mathbb{Z})}{H^2(S^{[3]},\mathbb{Z})\smile H^4(S^{[3]},\mathbb{Z})} \cong \left(\frac{\mathbb{Z}}{3\mathbb{Z}}\right)^{\oplus 22} \oplus \frac{\mathbb{Z}}{3\mathbb{Z}} \tag{1}$$

$$\frac{H^6(S^{[4]},\mathbb{Z})}{H^2(S^{[4]},\mathbb{Z})\smile H^4(S^{[4]},\mathbb{Z})} \cong \left(\frac{\mathbb{Z}}{6\mathbb{Z}}\right)^{\oplus 22} \oplus \frac{\mathbb{Z}}{108\mathbb{Z}} \oplus \frac{\mathbb{Z}}{2\mathbb{Z}} \tag{2}$$

$$\frac{H^6(S^{[5]},\mathbb{Z})}{H^2(S^{[5]},\mathbb{Z})\smile H^4(S^{[5]},\mathbb{Z})} \cong \mathbb{Z}^{\oplus 22} \oplus \mathbb{Z} \tag{3}$$

$$\frac{H^6(S^{[n]},\mathbb{Z})}{H^2(S^{[n]},\mathbb{Z})\smile H^4(S^{[n]},\mathbb{Z})} \cong \mathbb{Z}^{\oplus 22} \oplus \mathbb{Z} \oplus \mathbb{Z},\ n\geq 6. \tag{4}$$

In each case, the first 22 parts of the quotient are generated by the integral classes

$$\alpha_i^{(1,1,1)} - 3\cdot\alpha_i^{(2,1)} + 3\cdot\alpha_i^{(3)} + 3\cdot 1^{(2)}\alpha_i^{(1,1)} - 6\cdot 1^{(2)}\alpha_i^{(2)} + 6\cdot 1^{(2,2)}\alpha_i^{(1)} - 3\cdot 1^{(3)}\alpha_i^{(1)},$$

for $i = 1\ldots 22$. Now define an integral class

$$K := \sum_{i\neq j} B(\alpha_i,\alpha_j)\left[\alpha_i^{(1,1)}\alpha_j^{(1)} - 2\cdot\alpha_i^{(2)}\alpha_j^{(1)} + \frac{3}{2}\cdot 1^{(2)}\alpha_i^{(1)}\alpha_j^{(1)}\right] + \\ + \sum_i B(\alpha_i,\alpha_i)\left[\alpha_i^{(1,1,1)} - 2\cdot\alpha_i^{(2,1)} + \frac{3}{2}\cdot 1^{(2)}\alpha_i^{(1,1)}\right] + x^{(2)} - 1^{(2)}x^{(1)}.$$

In the case $n=3$, the last part of the quotient is generated by K.
In the case $n=4$, the class $1^{(4)}$ generates the 2-torsion part and $K-38\cdot 1^{(4)}$ generates the 108-torsion part.
In the case $n=5$, the last part of the quotient is generated by $K-16\cdot 1^{(4)}+21\cdot 1^{(3,2)}$.
If $n\geq 6$, the two last parts of the quotient are generated over the rationals by $K+\frac{4}{3}(45-n)1^{(2,2,2)}-(48-n)1^{(3,2)}$ and $K+\frac{1}{2}(40-n)1^{(2,2,2)}-\frac{1}{4}(48-n)1^{(4)}$. Over $\mathbb{Z}$, one has to take appropriate multiples depending on n, such that the coefficients become integral numbers.

Proof. The last assertion for arbitrary n follows from Corollary 1.13. First observe that for $\boldsymbol{\alpha^\lambda}\in H^2$, $\boldsymbol{\alpha^\mu}\in H^4$, $\boldsymbol{\alpha^\nu}\in H^6$, we have $\|\boldsymbol{\lambda}\|\leq 2$, $\|\boldsymbol{\mu}\|\leq 4$ and $\|\boldsymbol{\nu}\|\geq 2$, according to Lemma 1.9. The coefficient of the cup product martix are thus polynomials of degree at most $2+4-2=4$ and it suffices to compute only a finite number of instances for n. It turns out that the maximal degree is 1 and the cokernel of the multiplication map is given as stated. $\square$

In what follows, we compare some well-known facts about Hilbert schemes of points on K3 surfaces with our numerical calculations. This means, we have some tests that may justify the correctness of our computer program. We state now computational results for the middle cohomology group. Since $S^{[n]}$ is a projective variety of complex dimension $2n$, Poincaré duality gives $H^{2n}(S^{[n]},\mathbb{Z})$ the structure of an unimodular lattice.

Proposition 2.6. *Let L denote the unimodular lattice $H^{2n}(S^{[n]},\mathbb{Z})$. We have:*

(1) *For $n=2$, L is an odd lattice of rank 276 and signature 156.*
(2) *For $n=3$, L is an even lattice of rank 2554 and signature -1152.*

(3) *For $n = 4$, L is an odd lattice of rank* 19298 *and signature* 7082.

For n even, L is always odd.

Proof. The numerical results come from an explicit calculation. For n even, we always have the norm-1-vector given by Example 1.15 (5), so L is odd. To obtain the signature, we could equivalently use Hirzebruch's signature theorem and compute the L-genus of $S^{[n]}$. For the signature, we need nothing but the Pontryagin numbers, which can be derived from the Chern numbers of $S^{[n]}$. These in turn are known by Ellingsrud, Göttsche and Lehn, [2, Rem. 5.5]. $\square$

Another test is to compute the lattice structure of $H^2(S^{[2]}, \mathbb{Z})$, with bilinear form given by $(a, b) \longmapsto \int \left(a \smile b \smile 1^{(2)} \smile 1^{(2)}\right)$. The signature of this lattice is 17, as shown by Boissière, Nieper-Wißkirchen and Sarti [1, Lemma 6.9].

Appendix A. Source Code

We give the source code for our computer program. It is available online under `https://github.com/s--kapfer/HilbK3`. We used the language Haskell. The project is divided into 4 modules.

A.1. Module for cup product structure of K3 surfaces.

Here the hyperbolic and the E_8 lattice and the bilinear form on the cohomology of a K3 surface are defined. Furthermore, cup products and their adjoints are implemented.

```haskell
-- a module for the integer cohomology structure of a K3 surface
module K3 (
  K3Domain,
  degK3,
  rangeK3,
  oneK3, xK3,
  cupLSparse,
  cupAdLSparse
  ) where

import Data.Array
import Data.List
import Data.MemoTrie

-- type for indexing the cohomology base
type K3Domain = Int

rangeK3 = [0..23] :: [K3Domain]

oneK3 = 0 :: K3Domain
xK3 = 23 :: K3Domain

rangeK3Deg :: Int -> [K3Domain]
rangeK3Deg 0 = [0]
rangeK3Deg 2 = [1..22]
rangeK3Deg 4 = [23]
rangeK3Deg _ = []

delta i j = if i==j then 1 else 0

-- degree of the element of H^*(S), indexed by i
degK3 :: (Num d) => K3Domain -> d
degK3 0 = 0
degK3 23 = 4
degK3 i = if i>0 && i < 23 then 2 else error "Not a K3 index"

-- the negative e8 intersection matrix
e8 = array ((1,1),(8,8)) $
  zip [(i,j) | i <- [1..8], j <-[1..8]] [
  -2, 1, 0, 0, 0, 0, 0, 0,
  1, -2, 1, 0, 0, 0, 0, 0,
```

```
  0, 1, −2, 1, 0, 0, 0, 0,
  0, 0, 1, −2, 1, 0, 0, 0,
  0, 0, 0, 1, −2, 1, 1, 0,
  0, 0, 0, 0, 1, −2, 0, 1,
  0, 0, 0, 0, 1, 0, −2, 0,
  0, 0, 0, 0, 0, 1, 0, −2 :: Int]

−− the inverse matrix of e8
inve8 = array ((1,1),(8,8)) $
  zip [(i,j) | i <− [1..8],j <−[1..8]] [
  −2, −3, −4, −5, −6, −4, −3, −2,
  −3, −6, −8,−10,−12, −8, −6, −4,
  −4, −8,−12,−15,−18,−12, −9, −6,
  −5,−10,−15,−20,−24,−16,−12, −8,
  −6,−12,−18,−24,−30,−20,−15,−10,
  −4, −8,−12,−16,−20,−14,−10, −7,
  −3, −6, −9,−12,−15,−10, −8, −5,
  −2, −4, −6, −8,−10, −7, −5, −4 :: Int]

−− hyperbolic lattice
u 1 2 = 1
u 2 1 = 1
u 1 1 = 0
u 2 2 = 0
u i j = undefined

−− cup product pairing for K3 cohomology
bilK3 :: K3Domain −> K3Domain −> Int
bilK3 ii jj = let
  (i,j) = (min ii jj, max ii jj)
  in
  if (i < 0) || (j > 23) then undefined else
  if (i == 0) then delta j 23 else
  if (i >= 1) && (j <= 2) then u i j else
  if (i >= 3) && (j <= 4) then u (i−2) (j−2) else
  if (i >= 5) && (j <= 6) then u (i−4) (j−4) else
  if (i >= 7) && (j <= 14) then e8 ! ((i−6), (j−6)) else
  if (i >= 15) && (j<= 22) then e8 ! ((i−14), (j−14))  else
  0

−− inverse matrix to cup product pairing
bilK3inv :: K3Domain −> K3Domain −> Int
bilK3inv ii jj = let
  (i,j) = (min ii jj, max ii jj)
  in
  if (i < 0) || (j > 23) then undefined else
  if (i == 0) then delta j 23 else
  if (i >= 1) && (j <= 2) then u i j else
  if (i >= 3) && (j <= 4) then u (i−2) (j−2) else
  if (i >= 5) && (j <= 6) then u (i−4) (j−4) else
  if (i >= 7) && (j <= 14) then inve8 ! ((i−6), (j−6)) else
  if (i >= 15) && (j<= 22) then inve8 ! ((i−14), (j−14))  else
  0

−− cup product with two factors
−− a_i * a_j = sum [cup k (i,j) * a_k | k<− rangeK3]
cup :: K3Domain −> (K3Domain,K3Domain) −> Int
cup = memo2 r where
  r k (0,i) = delta k i
  r k (i,0) = delta k i
  r _ (i,23) = 0
  r _ (23,i) = 0
  r 23 (i,j) =  bilK3 i j
  r _ _ = 0

−− indices where the cup product does not vanish
cupNonZeros :: [ (K3Domain,(K3Domain,K3Domain)) ]
cupNonZeros = [ (k,(i,j)) | i<−rangeK3, j<−rangeK3, k<−rangeK3, cup k (i,j) /= 0]

−− cup product of a list of factors
cupLSparse :: [K3Domain] −> [(K3Domain,Int)]
cupLSparse = cu . filter (/=oneK3) where
```

```
  cu [] = [(oneK3,1)]; cu [i] = [(i,1)]
  cu [i,j] = [(k,z) | k<-rangeK3, let z = cup k (i,j), z/=0]
  cu _ = []

-- comultiplication, adjoint to the cup product
-- Del a_k = sum [cupAd (i,j) k * a_i `tensor` a_k | i<-rangeK3, j<-rangeK3]
cupAd :: (K3Domain,K3Domain) -> K3Domain -> Int
cupAd = memo2 ad where
  ad (i,j) k = negate $ sum [bilK3inv i ii * bilK3inv j jj
    * cup kk (ii,jj) * bilK3 kk k |(kk,(ii,jj)) <- cupNonZeros ]

-- n-fold comultiplication
cupAdLSparse :: Int -> K3Domain -> [([K3Domain],Int)]
cupAdLSparse = memo2 cals where
  cals 0 k = if k == xK3 then [([],1)] else []
  cals 1 k = [([k], 1)]
  cals 2 k = [([i,j],ca) | i<-rangeK3, j<-rangeK3, let ca = cupAd (i,j) k, ca /=0]
  cals n k = clean [(i:r,v*w) |([i,j],w)<-cupAdLSparse 2 k, (r,v)<-cupAdLSparse(n-1) j]
  clean = map (\g -> (fst$head g, sum$(map snd g))). groupBy cg.sortBy cs
  cs = (.fst).compare.fst; cg = (.fst).(==).fst
```

A.2. **Module for handling partitions.** This module defines the data structures and elementary methods to handle partitions. We define both partitions written as descending sequences of integers (λ-notation) and as sequences of multiplicities (α-notation).

```
{-# LANGUAGE TypeOperators, TypeFamilies #-}

-- implements data structure and basic functions for partitions
module Partitions where

import Data.Permute
import Data.Maybe
import qualified Data.List
import Data.MemoTrie

class (Eq a, HasTrie a) => Partition a where
  -- length of a partition
  partLength :: Integral i => a -> i

  -- weight of a partition
  partWeight :: Integral i => a -> i

  -- degree of a partition = weight - length
  partDegree :: Integral i => a -> i
  partDegree p = partWeight p - partLength p

  -- the z, occuring in all papers
  partZ :: Integral i => a -> i
  partZ = partZ.partAsAlpha

  -- conjugated partition
  partConj :: a -> a
  partConj = res. partAsAlpha where
    make l (m:r) = l : make (l-m) r
    make _ [] = []
    res (PartAlpha r) = partFromLambda $ PartLambda $ make (sum r) r

  -- empty partition
  partEmpty :: a

  -- transformation to alpha-notation
  partAsAlpha :: a -> PartitionAlpha
  -- transformation from alpha-notation
  partFromAlpha :: PartitionAlpha -> a
  -- transformation to lambda-notation
  partAsLambda :: a -> PartitionLambda Int
  -- transformation from lambda-notation
  partFromLambda :: (Integral i, HasTrie i) => PartitionLambda i -> a
```

```
  -- all permutationens of a certain cycle type
  partAllPerms :: a -> [Permute]
```

```
-- data type for partitiones in alpha-notation
-- (list of multiplicities)
newtype PartitionAlpha = PartAlpha { alphList::[Int] }

-- reimplementation of the zipWith function
zipAlpha op (PartAlpha a) (PartAlpha b) = PartAlpha $ z a b where
  z (x:a) (y:b) = op x y : z a b
  z [] (y:b) = op 0 y : z [] b
  z (x:a) [] = op x 0 : z a []
  z [] [] = []

-- reimplementation of the (:) operator
alphaPrepend 0 (PartAlpha []) = partEmpty
alphaPrepend i (PartAlpha  r) = PartAlpha (i:r)

-- all partitions of a given weight
partOfWeight :: Int -> [PartitionAlpha]
partOfWeight = let
  build n 1 acc = [alphaPrepend n acc]
  build n c acc = concat [ build (n-i*c) (c-1) (alphaPrepend i acc) | i<-[0..div n c]]
  a 0 = [PartAlpha []]
  a w =  if w<0 then [] else  build w w partEmpty
  in memo a

-- all partitions of given weight and length
partOfWeightLength = let
  build 0 0 _ = [partEmpty]
  build w 0 _ = []
  build w l c = if l > w || c>w then [] else
    concat [ map (alphaPrepend i) $ build (w-i*c) (l-i) (c+1)
      | i <- [0..min l $ div w c]]
  a w l = if w<0 || l<0 then [] else build w l 1
  in memo2 a

-- determines the cycle type of a permutation
cycleType :: Permute -> PartitionAlpha
cycleType p = let
  lengths = Data.List.sort $ map Data.List.length $ cycles p
  count i 0 [] = partEmpty
  count i m [] = PartAlpha [m]
  count i m (x:r) = if x==i then count i (m+1) r
    else alphaPrepend m (count (i+1) 0 (x:r))
  in count 1 0 lengths

-- constructs a permutation from a partition
partPermute :: Partition a => a -> Permute
partPermute = let
  make l n acc (PartAlpha x) = f x where
    f [] = cyclesPermute n acc
    f (0:r) = make (l+1) n acc $ PartAlpha r
    f (i:r) = make l (n+l) ([n..n+l-1]:acc) $ PartAlpha ((i-1):r)
  in make 1 0 [] . partAsAlpha

instance Partition PartitionAlpha where
  partWeight (PartAlpha r) = fromIntegral $ sum $ zipWith (*) r [1..]
  partLength (PartAlpha r) = fromIntegral $ sum r
  partEmpty = PartAlpha []
  partZ (PartAlpha l) = foldr (*) 1 $
    zipWith (\a i-> factorial a*i^a) (map fromIntegral l) [1..] where
      factorial n = if n==0 then 1 else n*factorial(n-1)
  partAsAlpha = id
  partFromAlpha = id
  partAsLambda (PartAlpha l) = PartLambda $ reverse $ f 1 l where
    f i [] = []
    f i (0:r) = f (i+1) r
    f i (m:r) = i : f i ((m-1):r)
  partFromLambda = lambdaToAlpha
```

```
  partAllPerms = partAllPerms . partAsLambda

instance Eq PartitionAlpha where
  PartAlpha p == PartAlpha q = findEq p q where
    findEq [] [] = True
    findEq (a:p) (b:q) = (a==b) && findEq p q
    findEq [] q = isZero q
    findEq p [] = isZero p
    isZero = all (==0)

instance Ord PartitionAlpha where
  compare a1 a2 = compare (partAsLambda a1) (partAsLambda a2)

instance Show PartitionAlpha where
  show p = let
    leftBracket = "(|"
    rightBracket = "|)"
    rest [] = rightBracket
    rest [i] = show i ++ rightBracket
    rest (i:q) = show i ++ "," ++ rest q
    in leftBracket ++ rest (alphList p)

instance HasTrie PartitionAlpha where
  newtype PartitionAlpha :->: a =  TrieType { unTrieType :: [Int] :->: a }
  trie f = TrieType $ trie $ f . PartAlpha
  untrie f =  untrie (unTrieType f) . alphList
  enumerate f  = map (\(a,b) -> (PartAlpha a,b)) $ enumerate (unTrieType f)
```

```
-- data type for partitions in lambda-notation
-- (descending list of positive numbers)
newtype PartitionLambda i = PartLambda { lamList :: [i] }

lambdaToAlpha :: Integral i => PartitionLambda i -> PartitionAlpha
lambdaToAlpha (PartLambda []) = PartAlpha[]
lambdaToAlpha (PartLambda (s:p)) = lta 1 s p [] where
  lta _ 0 _ a = PartAlpha a
  lta m c [] a = lta 0 (c-1) [] (m:a)
  lta m c (s:p) a = if c==s then lta (m+1) c p a else
    lta 0 (c-1) (s:p) (m:a)

instance (Integral i, HasTrie i) => Partition (PartitionLambda i) where
  partWeight (PartLambda r) = fromIntegral $ sum r
  partLength (PartLambda r) = fromIntegral $ length r
  partEmpty = PartLambda []
  partAsAlpha = lambdaToAlpha
  partAsLambda (PartLambda r) = PartLambda $ map fromIntegral r
  partFromAlpha (PartAlpha l) = PartLambda $ reverse $ f 1 l where
    f i [] = []
    f i (0:r) = f (i+1) r
    f i (m:r) = i : f i ((m-1):r)
  partFromLambda (PartLambda r) = PartLambda $ map fromIntegral r
  partAllPerms (PartLambda l) = it $ Just $ permute $ partWeight $ PartLambda l where
    it (Just p) = if Data.List.sort (map length $ cycles p) == r
      then p : it (next p) else it (next p)
    it Nothing = []
    r = map fromIntegral $ reverse l

instance (Eq i, Num i) => Eq (PartitionLambda i) where
  PartLambda p == PartLambda q = findEq p q where
    findEq [] [] = True
    findEq (a:p) (b:q) = (a==b) && findEq p q
    findEq [] q = isZero q
    findEq p [] = isZero p
    isZero = all (==0)

instance (Ord i, Num i) => Ord (PartitionLambda i) where
  compare p1 p2 = if weighteq == EQ then compare l1 l2 else weighteq where
    (PartLambda l1, PartLambda l2) = (p1, p2)
    weighteq = compare (sum l1) (sum l2)
```

```haskell
instance (Show i) => Show (PartitionLambda i) where
  show (PartLambda p) = "[" ++ s ++ "]" where
    s = concat $ Data.List.intersperse "-" $ map show p

instance HasTrie i => HasTrie (PartitionLambda i) where
  newtype (PartitionLambda i) :->: a =  TrieTypeL { unTrieTypeL :: [i] :->: a }
  trie f = TrieTypeL $ trie $ f . PartLambda
  untrie f =  untrie (unTrieTypeL f) . lamList
  enumerate f  = map (\(a,b) -> (PartLambda a,b)) $ enumerate (unTrieTypeL f)
```

A.3. Module for coefficients on Symmetric Functions.

This module provides nothing but the base change matrices $\psi_{\lambda\mu}$ and $\psi^{-1}_{\mu\lambda}$ from Definition 1.2.

```haskell
-- A module implementing base change matrices for symmetric functions
module SymmetricFunctions(
  monomialPower,
  powerMonomial,
  factorial
  ) where

import Data.List
import Data.MemoTrie
import Data.Ratio
import Partitions

-- binomial coefficients
choose n k = ch1 n k where
  ch1 = memo2 ch
  ch 0 0 = 1
  ch n k = if n<0 || k<0 then 0 else if k> div n 2 + 1 then ch1 n (n-k) else
    ch1(n-1) k + ch1 (n-1) (k-1)

-- multinomial coefficients
multinomial 0 [] = 1
multinomial n [] = 0
multinomial n (k:r) = choose n k * multinomial (n-k) r

-- factorial function
factorial 0 = 1
factorial n = n*factorial(n-1)

-- http://www.mat.univie.ac.at/~slc/wpapers/s68vortrag/ALCoursSf2.pdf , p. 48
-- scalar product between monomial symmetric functions and power sums
monomialScalarPower moI poI = (s * partZ poI) `div` quo where
  mI = partAsAlpha moI
  s = sum[a* moebius b | (a,b)<-finerPart mI (partAsLambda poI)]
  quo = product[factorial i| let PartAlpha l =mI, i<-l]
  nUnder 0 [] = [[]]
  nUnder n [] = []
  nUnder n (r:profile) = concat[map (i:) $ nUnder (n-i) profile | i<-[0..min n r]]
  finerPart (PartAlpha a) (PartLambda l) = nub [(a`div` sym sb,sb)
    | (a,b)<-fp 1 a l, let sb = sort b] where
    sym = s 0 []
    s n acc [] = factorial n
    s n acc (a:o) = if a==acc then s (n+1) acc o else factorial n * s 1 a o
    fp i [] l = if all (==0) l then [(1,[[]|x<-l])] else []
    fp i (0:ar) l = fp (i+1) ar l
    fp i (m:ar) l = [(v*multinomial m p,addprof p op)
      | p <- nUnder m (map (flip div i) l),
      (v,op) <- fp (i+1) ar (zipWith (\j mm-> j-mm*i) l p)] where
        addprof = zipWith (\mm l -> replicate mm i ++ l)
  moebius l = product [(-1)^c * factorial c | m<-l, let c = length m - 1]

-- base change matrix from monomials to power sums
-- no integer coefficients
-- m_j = sum [ p_i * powerMonomial i j | i<-partitions]
powerMonomial :: (Partition a, Partition b) => a->b->Ratio Int
powerMonomial poI moI = monomialScalarPower moI poI % partZ poI

-- base change matrix from power sums to monomials
-- p_j = sum [m_i * monomialPower i j | i<-partitions]
```

```haskell
monomialPower :: (Partition a, Partition b, Num i) => a->b->i
monomialPower lambda mu = fromIntegral $ numerator $
  memoizedMonomialPower (partAsLambda lambda) (partAsLambda mu)
memoizedMonomialPower = memo2 mmp1 where
  mmp1 l m  = if partWeight l == partWeight m then mmp2 (partWeight m) l m else 0
  mmp2 w l m = invertLowerDiag (map partAsLambda $ partOfWeight w) powerMonomial l m

-- inversion of lower triangular matrix
invertLowerDiag vs a = ild where
  ild = memo2 inv
  delta i j = if i==j then 1 else 0
  inv i j | i<j = 0
    | otherwise = (delta i j - sum [a i k * ild k j | k<-vs, i>k , k>= j]) / a i i
```

A.4. **Module implementing cup products for Hilbert schemes.** This is our main module. We implement the algebraic model developped by Lehn and Sorger and the change of base due to Qin and Wang. The cup product on the Hilbert scheme is computed by the function `cupInt`.

```haskell
-- implements the cup product according to Lehn-Sorger and Qin-Wang
module HilbK3 where

import Data.Array
import Data.MemoTrie
import Data.Permute hiding (sort,sortBy)
import Data.List
import qualified Data.IntMap as IntMap
import qualified Data.Set as Set
import Data.Ratio
import K3
import Partitions
import SymmetricFunctions

-- elements in A^[n] are indexed by partitions, with attached elements of the base K3
-- is also used for indexing H^*(Hilb, Z)
type AnBase = (PartitionLambda Int, [K3Domain])

-- elements in A{S_n} are indexed by permutations, in cycle notation,
-- where to each cycle an element of the base K3 is attached, see L-S (2.5)
type SnBase = [([Int],K3Domain)]

-- an equivalent to partZ with painted partitions
-- counts multiplicites that occur, when the symmetrization operator is applied
anZ :: AnBase -> Int
anZ (PartLambda l, k) = comp 1 (0,undefined) 0 $ zip l k where
  comp acc old m (e@(x,_):r) | e==old = comp (acc*x) old (m+1) r
    | otherwise = comp (acc*x*factorial m) e 1 r
  comp acc _ m [] = factorial m * acc

-- injection of A^[n] in A{S_n}, see L-S 2.8
-- returns a symmetrized vector of A{S_n}
toSn :: AnBase -> ([SnBase],Int)
toSn = makeSn where
  allPerms = memo p where
    p n = map (array (0,n-1). zip [0..]) (permutations [0..n-1])
  shape l = (map (forth IntMap.!) l, IntMap.fromList $ zip [1..] sl) where
    sl = map head$ group $ sort l;
    forth = IntMap.fromList$ zip sl [1..]
  symmetrize :: AnBase -> ([[([Int],K3Domain)]],Int)
  symmetrize (part,l) = (perms, toInt $ factorial n % length perms)  where
    perms = nub [sortSn$ zipWith (\c cb ->(ordCycle $ map(p!)c, cb) ) cyc l
      | p <- allPerms n]
    cyc = sortBy ((.length).flip compare.length) $ cycles $ partPermute part
    n = partWeight part
  ordCycle cyc = take l $ drop p $ cycle cyc where
    (m,p,l) = foldl findMax (-1,-1,0) cyc
    findMax (m,p,l) ce = if m<ce then (ce,l,l+1) else (m,p,l+1)
  sortSn = sortBy compareSn  where
    compareSn (cyc1,class1) (cyc2,class2) = let
      cL = compare l2 $ length cyc1 ; l2 = length cyc2
```

```
      cC = compare class2 class1
      in if cL /= EQ then cL else
        if cC /= EQ then cC else compare cyc2 cyc1
  mSym = memo symmetrize
  makeSn (part,l) = ([ [(z,im IntMap.! k) | (z,k) <- op ]|op <- res],m)  where
    (repl,im) = shape l
    (res,m) = mSym (part,repl)

-- multiplication in A{S_n}k, see L-S, Prop 2.13
multSn :: SnBase -> SnBase -> [(SnBase,Int)]
multSn l1 l2 = tensor $ map m cmno where
  -- determines the orbits of the group generated by pi, tau
  commonOrbits :: Permute -> Permute -> [[Int]]
  commonOrbits pi tau = Data.List.sortBy ((.length).compare.length) orl where
    orl = foldr (uni [][]) (cycles pi) (cycles tau)
    uni i ni c []  = i:ni
    uni i ni c (k:o) = if Data.List.intersect c k == []
      then uni i (k:ni) c o else uni (i++k) ni c o
  pi1 = cyclesPermute n $ cy1 ; cy1 = map fst l1; n = sum $ map length cy1
  pi2 = cyclesPermute n $ map fst l2
  set1 = map (\(a,b)->(Set.fromList a,b)) l1;
  set2 = map (\(a,b)->(Set.fromList a,b)) l2
  compose s t = swapsPermute (max (size s) (size t)) (swaps s ++ swaps t)
  tau = compose pi1 pi2
  cyt = cycles tau ;
  cmno = map Set.fromList $ commonOrbits pi1 pi2;
  m or = fdown where
    sset12 = [xv | xv <-set1++set2, Set.isSubsetOf (fst xv) or]
    -- fup and fdown correspond to the images of the maps described in L-S (2.8)
    fup = cupLSparse $ map snd sset12 ++ replicate def xK3
    t = [c | c<-cyt, Set.isSubsetOf (Set.fromList c) or]
    fdown = [(zip t l,v*w*24^def)| (r,v) <- fup, (l,w)<-cupAdLSparse(length t) r]
    def = toInt ((Set.size or + 2 - length sset12 - length t)%2)

-- tensor product for a list of arguments
tensor :: Num a =>  [[([b],a)]] -> [([b],a)]
tensor [] = [([],1)]
tensor (t:r) = [(y++x,w*v) |(x,v)<-tensor r, (y,w) <- t ]

-- multiplication in A^[n]
multAn :: AnBase -> AnBase -> [(AnBase,Int)]
multAn a = multb where
  (asl,m) = toSn a
  toAn sn =(PartLambda l, k) where
    (l,k)= unzip$ sortBy (flip compare)$ map (\(c,k)->(length c,k)) sn
  multb (pb,lb) = map ungroup$ groupBy ((.fst).(==).fst) $sort elems where
    ungroup g@((an,_):_) = (an, m*(sum $ map snd g) )
    bs = zip (sortBy ((.length).flip compare.length) $cycles $ partPermute pb) lb
    elems = [(toAn cs,v) | as <- asl, (cs,v) <- multSn as bs]

-- integer base to ordinary base, see Q-W, Thm 1.1
intCrea :: AnBase -> [(AnBase,Ratio Int)]
intCrea = map makeAn. tensor. construct where
  memopM = memo pM
  pM pa = [(pl,v)| p@(PartLambda pl)<-map partAsLambda$ partOfWeight (partWeight pa),
    let v = powerMonomial p pa, v/=0]
  construct pl = onePart pl : xPart pl :
    [ [(zip l $ repeat a,v)| (l,v)<- memopM (subpart pl a)] |a<-[1..22]]
  onePart pl = [(zip l$ repeat oneK3, 1%partZ p)] where
    p@(PartLambda l) = subpart pl oneK3
  xPart pl = [(zip l$ repeat xK3, 1)] where
    (PartLambda l) = subpart pl xK3
  makeAn (list,v) = ((PartLambda x,y),v) where
    (x,y) = unzip$ sortBy (flip compare) list

-- ordinary base to integer base, see Q-W, Thm 1.1
creaInt :: AnBase -> [(AnBase, Int)]
creaInt = map makeAn. tensor. construct where
  memomP = memo mP
  mP pa = [(pl,v)| p@(PartLambda pl)<-map partAsLambda$ partOfWeight (partWeight pa),
    let v = monomialPower p pa, v/=0]
  construct pl = onePart pl : xPart pl :
```

```
    [ [(zip l $ repeat a,v)| (l,v)<- memomP (subpart pl a)] |a<-[1..22]]
  onePart pl = [(zip l$ repeat oneK3, partZ p)] where
    p@(PartLambda l) = subpart pl oneK3
  xPart pl = [(zip l$ repeat xK3, 1)] where
    (PartLambda l) = subpart pl xK3
  makeAn (list,v) = ((PartLambda x,y),v) where
    (x,y) = unzip$ sortBy (flip compare) list

-- cup product for integral classes
cupInt :: AnBase -> AnBase -> [(AnBase,Int)]
cupInt a b = [(s,toInt z)| (s,z) <- y] where
  ia = intCrea a; ib = intCrea b
  x = sparseNub [(e,v*w*fromIntegral z) | (p,v) <- ia,
    let m = multAn p, (q,w) <- ib, (e,z)<- m q]
  y = sparseNub [(s,v*fromIntegral w) | (e,v) <- x, (s,w) <- creaInt e]

-- helper function, adds duplicates in a sparse vector
sparseNub :: (Num a) => [(AnBase, a)] -> [(AnBase,a)]
sparseNub = map (\g->(fst$head g, sum $map snd g)).groupBy ((.fst).(==).fst).
  sortBy ((.fst).compare.fst)

-- cup product for integral classes from a list of factors
cupIntList :: [AnBase] -> [(AnBase,Int)]
cupIntList = makeInt. ci . cL where
  cL [b] = intCrea b
  cL (b:r) = x where
    ib = intCrea b
    x = sparseNub [(e,v*w*fromIntegral z) |
      (p,v) <- cL r, let m = multAn p, (q,w) <- ib, (e,z)<-m q]
  makeInt l = [(e,toInt z) | (e,z) <- l]
  ci l = sparseNub [(s,v*fromIntegral w) | (e,v) <- l, (s,w) <- creaInt e]

-- degree of a base element of cohomology
degHilbK3 :: AnBase -> Int
degHilbK3 (lam,a) = 2*partDegree lam + sum [degK3 i | i<- a]

-- base elements in Hilb^n(K3) of degree d
hilbBase :: Int -> Int -> [AnBase]
hilbBase = memo2 hb where
  hb n d = sort $map ((\(a,b)->(PartLambda a,b)).unzip) $ hilbOperators n d

-- all possible combinations of creation operators of weight n and degree d
hilbOperators :: Int -> Int -> [[ (Int,K3Domain) ]]
hilbOperators = memo2 hb where
  hb 0 0 = [[]] -- empty product of operators
  hb n d = if n<0 || odd d || d<0 then [] else
    nub $ map (Data.List.sortBy (flip compare)) $ f n d
  f n d = [(nn,oneK3):x | nn <-[1..n], x<-hilbOperators(n-nn)(d-2*nn+2)] ++
    [(nn,a):x | nn<-[1..n], a <-[1..22], x<-hilbOperators(n-nn)(d-2*nn)] ++
    [(nn,xK3):x | nn <-[1..n], x<-hilbOperators(n-nn)(d-2*nn-2)]

-- helper function
subpart :: AnBase -> K3Domain -> PartitionLambda Int
subpart (PartLambda pl,l) a = PartLambda $ sb pl l where
  sb [] _ = []
  sb pl [] = sb pl [0,0..]
  sb (e:pl) (la:l) = if la == a then e: sb pl l else sb pl l

-- converts from Rational to Int
toInt :: Ratio Int -> Int
toInt q = if n ==1 then z else error "not integral" where
  (z,n) =(numerator q, denominator q)
```

References

1. S. Boissière, M. Nieper-Wißkirchen, and A. Sarti, *Smith theory and Irreducible Holomorphic Symplectic Manifolds*, eprint arXiv:1204.4118 (2012).
2. G. Ellingsrud, L. Göttsche and M. Lehn, *On the Cobordism Class of the Hilbert Scheme of a Surface*, J. Algebraic Geometry (1999).

3. A. Lascoux, *Symmetric functions*, Notes of the course given at Nankai University, 2001, http://www.mat.univie.ac.at/ slc/wpapers/s68vortrag/ALCoursSf2.pdf .
4. M. Lehn and C. Sorger, *The cup product of Hilbert schemes for $K3$ surfaces*, Invent. Math. **152** (2003), no. 2, 305–329.
5. E. Markman, *Integral generators for the cohomology ring of moduli spaces of sheaves over Poisson surfaces*, Adv. Math. **208** (2007), no. 2, 622–646.
6. E. Markman, *Integral constraints on the monodromy group of the hyperKähler resolution of a symmetric product of a $K3$ surface*, Internat. J. Math. **21** (2010), no. 2, 169–223.
7. J.W. Milnor and D. Husemöller, *Symmetric bilinear forms*, Ergebnisse der Mathematik und ihrer Grenzgebiete, Springer (1973).
8. H. Nakajima, *Heisenberg algebra and Hilbert schemes of points on projective surfaces*, Ann. of Math. (2) **145** (1997), no. 2, 379–388.
9. Z. Qin and W. Wang, *Integral operators and integral cohomology classes of Hilbert schemes*, Math. Ann. **331** (2005), no. 3, 669–692.
10. M. Verbitsky, *Cohomology of compact hyper-Kähler manifolds and its applications*, Geom. Funct. Anal. **6** (1996), no. 4, 601–611.

Simon Kapfer, Laboratoire de Mathématiques et Applications, UMR CNRS 6086, Université de Poitiers, Téléport 2, Boulevard Marie et Pierre Curie, F-86962 Futuroscope Chasseneuil

E-mail address: `simon.kapfer@math.univ-poitiers.fr`