A Maple Program for Computing Landau-Ginzburg A- and B-Models and an Exploration of Mirror Symmetry

Evan D. Merrell
Brigham Young University - Provo

A Maple Program for Computing Landau-Ginzburg A- and B-Models and an Exploration of Mirror Symmetry

Evan D. Merrell

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mathematics

Brigham Young University

August 2012

ABSTRACT

A Maple Program for Computing Landau-Ginzburg A- and B-Models and an Exploration of Mirror Symmetry

Evan D. Merrell
Department of Mathematics, BYU
Master of Science

Mirror symmetry has been a significant area of research for geometry and physics for over two decades. Berglund and Hubsch proposed that for a certain family of singularities W, the so called "transposed" singularity W^T should be the mirror partner of W. [1] The techniques for constructing the orbifold LG models to test this conjecture were developed by FJR in [2] with a cohomological field theory generalized from the study of r-spin curves. The duality of LG A- and B-models became more elaborate when Krawitz [3] generalized the Intriligator-Vafa orbifold B-model to include contributions from more than one sector.

This thesis presents a program written in Maple for explicitly computing bases for both LG A- and B-model rings, as well as the correlators for A-models to the extent of current knowledge. Included is a list of observations and conjectures drawn from computations done in the program

Keywords: Mirror symmetry, Landau-Ginzburg theory, Orbifolds, Programming, Code

Acknowledgments

For the support of my parents, my professors, and Ann.

Contents

Chapter 1. Background

Let $W \in \mathbb{C}[x_1, x_2, \ldots, x_N]$. We say W is *quasihomogeneous* if there are positive integers $d_1, d_2, \ldots, d_N, n$ such that

$$W(t^{d_1}X_1, t^{d_2}X_2, \ldots, t^{d_N}X_N) = t^n W(X_1, X_2, \ldots, X_N).$$

In physics, $[d_1, d_2, \ldots, d_N; n]$ is called the *weight system* for W, and the ratios $q_i = d_i/n$ are called the *charges.* In mathematical context, these q_i are called *weights.* We call W *non-degenerate* if these weights are uniquely determined, the Hessian determinant is non-zero, and there is an isolated singularity at the origin. Associated with such a nondegenerate W is its Milnor ring $\mathcal{Q}_W = \mathbb{C}[x_1, x_2, \ldots, x_N]/\partial W$, a graded ring with top degree $\hat{c} = \sum(1-2q_i)$ and dimension $\mu = \prod(\frac{1}{q_i} - 1)$, and its basis is generated by monomials $\prod_i X_i^{b_i}$. The Milnor ring is graded by quasidegree, where the *quasidegree* of $\prod_i X_i^{b_i}$ is $\sum_i q_i b_i$. The ring has a pairing defined by

$$fg = \frac{\langle f, g\rangle}{\mu}\mathrm{Hess}(W) + \text{terms of lower order},$$

and with this pairing the ring is a Frobenius algebra.

In order to make orbifold Landau-Ginzburg models, we use diagonal symmetries of W. The maximal group of diagonal symmetries of W is defined by

$$G^{max} = \{(\alpha_1, \alpha_2, \ldots, \alpha_N) \in \mathbb{C}^N | W(\alpha_1 x_1, \alpha_2 x_2, \ldots, \alpha_N x_N) = W(x_1, x_2, \ldots, x_N)\}.$$

A symmetry group is called *admissible* if it is the maximal symmetry group for $W + Z$, where Z is some term or terms such that $W + Z$ is again a nondegenerate quasihomogeneous polynomial with the same variables and weights as W. Every admissible subgroup contains the exponential grading element $J = \mathrm{diag}(\exp 2\pi i q_1, \exp 2\pi i q_2, \ldots, \exp 2\pi i q_N)$. In some cases $\langle J\rangle = G^{max}$, but in general one may choose from the entire subgroup lattice from $\langle J\rangle$ to G^{max}

We now define the state space $\mathcal{H}_{W,G}$ for our Landau-Ginzburg A-model and the structure to make it a Frobenius algebra. In [2] this is done in terms of Lefschetz thimbles, but through a (non-cannonical) isomorphism, we may work instead with Milnor rings. Choose some admissible symmetry group $G \subset G^{max}$ and for each $\gamma \in G$, we let Fix $\gamma \subset \mathbb{C}^N$ be the fixed locus of γ and N_γ be the dimension of this fixed locus. We now define the *sector* $\mathcal{H}_\gamma$ by

$$\mathcal{H}_\gamma := \Omega^{N_\gamma}\left(\mathbb{C}^{N_\gamma}\right) / \left(dW|_{\mathrm{Fix}h} \wedge \Omega^{N_\gamma - 1}\right) \cong \mathcal{Q}_{W|_{\mathrm{Fix}\gamma}} \cdot \omega_\gamma$$

where $\omega_\gamma \in \Omega^{N_\gamma}$ is $dX_{i_1} \wedge dX_{i_2} \wedge \ldots \wedge dX_{i_{N_\gamma}}$, the natural choice of a volume form. We now define the state space $\mathcal{H}_{W,G}$ as the G-invariant subspace of the sum of the sectors $\mathcal{H}_\gamma$:

$$\mathcal{H}_{W,G} := \left(\bigoplus_{\gamma \in G} \mathcal{H}_\gamma\right)^G .$$

We define W-degree which will make $\mathcal{H}_{W,G}$ a graded ring as follows: given $\gamma \in G$, it can be written in the form

$$\gamma = \left(e^{2\pi i \theta_1^\gamma}, e^{2\pi i \theta_2^\gamma}, \ldots, e^{2\pi i \theta_N^\gamma}\right),$$

where we use Θ_i^γ to denote the principal choice of phase θ_i^γ such that $0 \le \Theta_\gamma^i < 1$. Thus we may express γ uniquely as

$$\gamma = \left(e^{2\pi i \Theta_1^\gamma}, e^{2\pi i \Theta_2^\gamma}, \ldots, e^{2\pi i \Theta_N^\gamma}\right).$$

The W-degree $\deg_W(\alpha_\gamma)$ for $\alpha_\gamma \in \mathcal{H}_\gamma$ is defined as

$$\deg_W(\alpha_\gamma) := N_\gamma + 2\sum_{i=1}^{N}(\Theta_i^\gamma - q_i),$$

and this gives us a graded ring. We also define a pairing

$$\eta_\gamma : (\mathcal{H}_\gamma)^G \otimes (\mathcal{H}_{\gamma^{-1}})^G \to \mathbb{C}, \qquad \eta_\gamma(a,b) = \langle a, I^{-1}(b)\rangle,$$

where $I : \mathcal{H}_\gamma \to \mathcal{H}_{\gamma^{-1}}$ is the cannonical isomorphism that follows from the fact that $\mathrm{Fix}\gamma = \mathrm{Fix}\gamma^{-1}$ and thus $\mathcal{H}_\gamma, \mathcal{H}_{\gamma^{-1}}$ are two copies of the same space. The pairing on the full state space is the direct sum on the pairings of the individual sectors, and we denote the full pairing matrix in a fixed basis as $\eta_{\alpha,\beta} = \langle \alpha, \beta\rangle$ with inverse $\eta^{\alpha,\beta}$.

Next we require one reference to FJRW theory in its fullness: for each pair of non-negative integers g, k with $2g - 2 + k > 0$ the FJRW cohomological field theory produces classes $\Lambda^W_{g,k}(\alpha_1, \alpha_2, \dots, \alpha_k) \in H^*(\overline{\mathcal{M}}_{g,k})$ of complex codimension D for any k-tuple $(\alpha_1, \alpha_2, \dots, \alpha_k) \in \mathcal{H}^k_{W,G}$. Here $\overline{\mathcal{M}}_{g,k}$ is the Deligne-Mumford stack of stable curves of genus g and k marked points, and the codimension D is given by

$$D = \hat{c}_W(g-1) + \frac{1}{2}\sum_{i=1}^{k} \deg_W(\alpha_i).$$

We define k-point correlators by

$$\langle \alpha_1, \alpha_2, \dots, \alpha_k\rangle := \int_{\overline{\mathcal{M}}_{g,k}} \Lambda^W_{g,k}(\alpha_1, \alpha_2, \dots, \alpha_k)$$

and the α_i's are referred to as insertions of the correlator.

We give axioms for computing FJRW rings, in a somewhat simplified form focusing on three-point genus-zero correlators, since that is all that's needed for the ring structure. We define the product

$$r \star s := \sum_{\alpha,\beta} \langle r, s, \alpha\rangle \eta^{\alpha,\beta}\beta$$

for $r, s \in \mathcal{H}_{W,G}$ as α, β range over some fixed basis.

Axiom 1 Dimension: If $2D \notin \mathbb{Z}$, then $\Lambda^W_{g,k}(\alpha_1, \alpha_2, \dots, \alpha_k) = 0$. Otherwise, 2D is the real codimension of the class $\Lambda^W_{g,k}(\alpha_1, \alpha_2, \ldots, \alpha_k)$. In particular, for 3-point genus-0 correlators,

$$\langle \alpha_1, \alpha_2, \alpha_3 \rangle \neq 0 \implies D = 0 \iff \sum_{i=1}^{3} \deg_W \alpha_i = 2\hat{c}.$$

Axiom 2 Symmetry: For $\sigma \in S_n$,

$$\Lambda^W_{g,k}(\alpha_1, \alpha_2, \ldots, \alpha_k) = \Lambda^W_{g,k}(\alpha_{\sigma(1)}, \alpha_{\sigma(2)}, \ldots, \alpha_{\sigma(k)}).$$

The next axioms deal with the line bundles $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_N$ of the W-structure of an orbicurve, but like most everything for the purposes of this program, the hard math reduces to some very simple conditions. For a class $\Lambda^W_{g,k}(\alpha_1, \alpha_2, \ldots, \alpha_k)$ with $\alpha_i \in (\mathcal{H}_{\gamma_i})^G$, let $l_i = |\mathcal{L}_i|$, the degree of the line bundle, for each variable X_i. We then have by [2]

$$l_i = (2g - 2 + k)q_i - \sum_j \theta_i^{\gamma_j}.$$

Axiom 3 Integer Degrees:

$$\Lambda^W_{g,k}(\alpha_1, \alpha_2, \ldots, \alpha_k) \neq 0 \implies l_i \in \mathbb{Z},$$

for $i \in \{1, 2, \ldots, N\}$.

Axiom 4 Concavity: If $l_i < 0$ for $i = 1, 2, \ldots, N$ then $\langle \alpha_1, \alpha_2, \alpha_3 \rangle = 1$.

For the next axiom, we use the Witten map $\mathcal{W}$:

$$\mathcal{W} : \bigoplus_{1=1}^{N} H^0(\mathcal{C}, \mathcal{L}_i), \rightarrow : \bigoplus_{1=1}^{N} H^1(\mathcal{C}, \mathcal{L}_i),$$

$$\mathcal{W} : W \mapsto \left(\overline{\frac{\partial W}{\partial X_1}}, \overline{\frac{\partial W}{\partial X_2}}, \ldots, \overline{\frac{\partial W}{\partial X_N}} \right).$$

The dimensions of these cohomologies $H^0(\mathcal{C}, \mathcal{L}_i)$ and $H^1(\mathcal{C}, \mathcal{L}_i)$ are h_i^0 and h_i^1 respectively, and they are given by

$$h_i^0 = \begin{cases} 0 & \text{if } l_i < 0; \\ l_i + 1 & \text{if } l_i \geq 0; \end{cases}$$

$$h_i^1 = \begin{cases} -l_i - 1 & \text{if } l_i < 0; \\ 0 & \text{if } l_i \geq 0; \end{cases}$$

and so both are nonnegative integers and we have $h_i^0 - h_i^1 = l_i + 1$, which is Riemann-Roch for $\mathcal{L}_i$.

Axiom 5 Index Zero: Consider the class$\Lambda_{g,k}^W(\alpha_1, \alpha_2, \ldots, \alpha_k)$ with $\alpha_i \in (\mathcal{H}_{\gamma_i})^G$. If Fix$\gamma_i = \{0\}$ for each $\alpha_i \in \{1, \ldots, k\}$ and

$$D = \sum_{i=1}^{k} (h_i^0 - h_i^1) = 0,$$

then $\Lambda_{g,k}^W(\alpha_1, \alpha_2, \ldots, \alpha_k)$ is equal to the degree of the Witten map. Strictly speaking, the integral over this class is equal to the degree of the Witten map, but we abuse notation and identify the codimension zero class with the value of its integral over $\overline{\mathcal{M}}_{g,k}$.

Axiom 6 Composition: If the four point class $\Lambda_{0,4}^W(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ is codimension zero, then it decomposes in terms of three-point correlators in the following way:

$$\Lambda_{0,4}^W(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = \sum_{\beta,\delta} \langle \alpha_1, \alpha_2, \beta \rangle \eta^{\beta,\delta} \langle \delta, \alpha_3, \alpha_4 \rangle$$

where β, δ range over a fixed basis for $\mathcal{H}_{W,G}$.

Note that Fix $J = \{0\}$, so $\mathcal{H}_J \cong \mathbb{C}$, and let $\mathbf{1}$ be the element corresponding to $1 \in \mathbb{C}$. This is the identity in $\mathcal{H}_{W,G}$.

Axiom 7 Pairing: For any $\alpha, \beta \in \mathcal{H}_{W,G}$ we have

$$\langle \alpha, \beta, \mathbf{1} \rangle = \eta_{\alpha,\beta}.$$

Not necessary as an axiom, an elementary consequence will be that an insertion of $\mathbf{1}$ in a genus-zero k-point correlator with $k > 3$ forces the correlator to be zero.

Axiom 8 Sums of Singularities: If $W_1 \in \mathbb{C}[X_1, \ldots, X_s], W_2 \in \mathbb{C}[Y_1, \ldots, Y_s]$ are non-degenerate quasihomogeous polynomials with maximal symmetry groups G_1, G_2 respectively, then $W = W_1 + W_2$ is a non-degeneate quasihomogeneous polynomial with maximal symmetry group $G = G_1 \times G_2$ and there is a Frobenius algebra isomorphism

$$\mathcal{H}_{W,G} \cong \mathcal{H}_{W_1,G_1} \otimes \mathcal{H}_{W_2,G_2}.$$

The FJRW theory is a candidate for being a Landau-Ginzburg A-model. Milnor rings are already known to be Landau-Ginzburg B-models, but orbifold B-models are a recent development due to Intriligator and Vafa, with the specifici implementation in our case due to Krawitz, following a recipe of Kaufmann. The state space $\mathcal{B}_{W,G}$ has a definition parallel to the A-model,

$$\mathcal{B}_{W,G} := \left(\bigoplus_{\gamma \in G} \mathcal{Q}_{W|_{\mathrm{Fix}\gamma}} \cdot \omega_\gamma \right)^G,$$

but it is only an algebra after defining a suitable multiplication of orbifold sectors. For $g \in G$, let $F_g := i : gX_i = X_i$. The structure constants $\gamma_{g,h}$ such that $1_g \star 1_h = \gamma_{g,h} 1_{gh}$ are then defined (up to scaling) by

$$\gamma_{g,h} \frac{\mathrm{Hess} W|_{\mathrm{Fix}g \cap \mathrm{Fix}h}}{\mu_{W|_{\mathrm{Fix}g \cap \mathrm{Fix}h}}} := \begin{cases} \frac{\mathrm{Hess} W|_{\mathrm{Fix}gh}}{\mu_{W|_{\mathrm{Fix}gh}}} & \text{if} F_g \cup F_h \cup F_{gh} = 1, \ldots, N; \\ 0 & \text{otherwise.} \end{cases}$$

For these orbifold B-models, our choice of group must come from the lattice ranging from the trivial group (giving a copy of the Milnor ring) to $SL_{\mathbb{C}}(N) \cap G_W^{max}$. The $SL_{\mathbb{C}}(N)$ restriction is dual to all admissible symmetry groups containing the group $\langle J \rangle$, and the lattice of choices for orbifolding the B-model from the trival group to $SL_{\mathbb{C}}(N) \cap G_W^{max}$ is observed to be isomorphic the lattice from G^{max} to $\langle J \rangle$ in all cases, though this has yet be rigorously proven.

A singularity is called *invertible* if the number of monomials matches the number of variables. We may identify a polynomial W with a matrix $B_{i,j}$ where $b_{i,j}$ is the exponent of the jth variable in the ith monomial of W. The invertibility of W is equivalent to the matrix $B_{i,j}$ being square, and by considering the matrix $B^T_{i,j}$, we get the transpose singularity W^T. If we assume that the charges of W are less than $\frac{1}{2}$ then it has been proven [3] that W^T is the mirror partner of W in the sense that the A- and B-models of W are isomorphic to the B- and A-models respectively of W^T using the maximal symmetry group for the A-model and the trivial group for the B-model. It is also known that for charges less than $\frac{1}{2}$ that W must be a sum of polynomials of the following forms: a Fermat polynomial $X_1^{a_1} + X_2^{a_2} + \ldots + X_N^{a_N}$, a chain $X_1^{a_1}X_2 + X_2^{a_2}X_3 + \ldots + X_{N-1}^{a_{N-1}}X_N + X_N^{a_N}$, or a loop $X_1^{a_1}X_2 + X_2^{a_2}X_3 + \ldots + X_N^{a_N}X_1$. Removing the restriction on charges allows rather stranger things to happen, which will be addressed in the sections on observations and conjectures.

Chapter 2. Observations and Conjectures

We consider the two classes of invertible non-degenerate singularities, chains and loops (a Fermat can be considered a chain with $N = 1$), and compute some of the important values for the general polynomial of the class. First we consider a chain, with

$$W_{\text{Chain}} = X_1^{a_1}X_2 + X_2^{a_2}X_3 + \ldots + X_{N-1}^{a_{N-1}}X_N + X_N^{a_N},$$

which will have

$$\dim\mathcal{H}_{W_{\text{Chain}},G^{max}} = \sum_{i=0}^{N}(-1)^i \prod_{j=1}^{N-i} a_j, \qquad \mu = \sum_{i=0}^{N}(-1)^i \prod_{j=i}^{N} a_j,$$

and a basis for the Milnor ring can be produced by removing all mutiples of certain monomials from a simple starting set:

$$\left\{\prod_i X_i^{\alpha}\middle| 0 \le \alpha < a_i\right\} \setminus \text{Exclusions}$$

$$\text{Exclusions} := \text{ multiples of} \begin{cases} \{X_1^{a_1-1}X_2, X_1^{a_1-1}X_3^{a_3-1}X_4, \ldots, X_1^{a_1-1}X_3^{a_3-1}\ldots X_N^{a_N-1}\} & \text{N odd,} \\ \{X_1^{a_1-1}X_2, X_1^{a_1-1}X_3^{a_3-1}X_4, \ldots, X_1^{a_1-1}X_3^{a_3-1}\ldots X_{N-1}^{a_{N-1}-1}X_N\} & \text{N even,} \end{cases}$$

where this denotes removing from $\{\prod_i X_i^{\alpha} | 0 \le \alpha < a_i\}$ all those monomials divisible by any elements of the exclusion set. The order of G^{max} is

$$\left|G^{max}_{W_{\text{Chain}}}\right| = \prod_i a_i$$

.

For convenience, we define the symbol $\widehat{a}_S$ by

$$\widehat{a}_S = \prod_{i \in \{1,\ldots,N\}\setminus S} a_i$$

with the convention that the empty product is 1. The utility of introducing this notation is that by expressing phases (and in particular, charges) in terms of the exponents a_i, they are now in the same terms as the basis for the Milnor ring, which helps in finding which monomials, if any, will be invariants in a sector.

The charges for W_{Chain} are given by

$$q_i = \frac{1}{|G^{max}|} \sum_{j=0}^{N-i} (-1)^j \widehat{a}_{\{i,\ldots,N\}}.$$

Considering now a loop type singularity

$$W_{\text{Loop}} = X_1^{a_1}X_2 + X_2^{a_2}X_3 + \ldots + X_N^{a_N}X_1,$$

we have

$$\left|G^{max}_{W_{\text{Loop}}}\right| = (-1)^{N+1} + \prod_i a_i,$$

and we can compute

$$\dim \mathcal{H}_{W_{\text{Loop}},G^{max}} = \prod_i a_i, \qquad \mu = \prod_i a_i,$$

with Milnor basis

$$\left\{ \prod_i X_i^{\alpha} \middle| 0 \le \alpha < a_i \right\}.$$

The charges for this W are given by

$$q_i = \frac{1}{|G^{max}|} \sum_{j=0}^{N-1} (-1)^j \widehat{a}_{i,\dots,N,1,\dots,i+j}.$$

For both loops and chains, an especially nice generator for the maximal symmetry group is the element γ_1 with phases

$$\theta_i = \frac{1}{|G^{max}|} (-1)^{i+1} \widehat{a}_{\{i,\dots,N\}}$$

so that we may express the state space sectors as $\mathbf{e_k}$ corresponding to γ_1^k, which are eaily identified since the first phase of $\mathbf{e}_k$ is $\frac{k}{|G^{max}|}$.

A *Ramond sector* is one for which $N_\gamma \neq 0$ and there exists at least one invariant monomial in the restricted Milnor ring $\mathcal{Q}_{\text{Fix}\gamma}$. For loops and chains, using G^{max}, there are Ramond sectors exactly when an even number of variables are fixed. Loops with an odd number of variables have no Ramond sectors, and thus the correlators are especially nice to compute. For loops with an even number of variables, the monomials for the $\mathbf{e}_0$ sector are $X_1^{a_1-1}X_3^{a_3-1}\dots X_{N-1}^{a_{N-1}-1}, X_2^{a_2-1}X_4^{a_4-1}\dots X_N^{a_N-1}$. Chain singularities will have only one-dimensional Ramond sectors when using G^{max}, and then only if an even number of variables are fixed; the invariant monomial in such a case is $X_k^{a_k-1}X_{k+2}^{a_{k+2}-1}\dots X_{N-1}^{a_{N-1}-1}$ if variables k through N are those that are fixed.

Consider the lattice of all admissible symmetries of a non-degenerate W, which forms a subgroup lattice of groups G where $\langle J\rangle \leq G \leq G^{max}$, which we will call the *A-lattice of* W. Similarly, the *B-lattice of* W^T is the lattice of groups $\langle e\rangle \leq G_{W^T} \leq G^{max}_{W^T} \cap \mathrm{SL}_{\mathbb{C}}(N)$ which can be used in creating orbifold B-models.

Conjecture 1. *Lattice Conjecture.* Let W be given. The lattice of all possible subgroups from $\langle J\rangle$ to G^{max}_W is exactly the lattice of admissible symmetries G_W of W. Further, this lattice is isomorphic to the full lattice $\langle e\rangle \leq G' \leq G^{max}_{W^T} \cap \mathrm{SL}_{\mathbb{C}}(N)$ and the order of the group G' corresponding to each G_W has order $|G'| = |G^{max}_W : G_W|$

For G in the A-lattice, let G^T denote this corresponding element G' of the B-lattice. As a caution about notation, $G^T_W \leq G^{max}_{W^T}$, but it is not true that G^T_W is an admissible symmetry of W^T.

If one uses G in the construction of an A-model, the correct choice of group for the orbifold B-model has proved to be G^T. Though there is no motivation known for using one group for orbifolding while using another group for taking invariants, the A- and B-model constructions allow it, and the correspondence of A- and B-lattices extends to this more general situation, and computations with the program lead to the follow conjecture:

Conjecture 2. Let W be an invertible non-degenerate singularity. For any admissible symmetries G_1, G_2 of W,

$$\mathcal{H}^{G_2}_{W,G_1} \cong \mathcal{B}^{G_1^T}_{W^T,G_2^T}$$

as Frobenius algebras.

A proof of conjecture requires only the case of $G_1 = G^{max}, G_2 = \langle J\rangle$, since all other choices for the G_i will give Frobenius subalgebras of $\mathcal{H}^{\langle J\rangle}_{W,G^{max}}$.

In [4] there are explicit computations showing that $\mathcal{H}^{\langle J\rangle}_{W_{1,0}}$ and $\mathcal{H}^{\langle J\rangle}_{J_{3,0}}$ can't be isomorphic to the Milnor ring of any quasihomogeneous singularities and so the more general construction is necessary. These are the first examples of singularities where finding the partner is a bit trickier than just specifying a polynomial. Consider the case of $W_{1,0}$: $x^4 + bx^2y^3 + y^6$ for nonsingular choice of b. The FJRW ring of $x^4 + y^6$ using the maximal symmetry group is just a tensor product of two rings, but we need not use the maximal symmetry group and the lattice conjecture doesn't require W to be irreducible: we can use the symmetry group of $x^4+bx^2y^3+y^6$, and its dual partner will be found by transposing x^4+y^6 and identifying where on the A-lattice we have the group for $x^4+bx^2y^3+y^6$. Then we identify the corresponding element of the B-lattice, namely $\langle\exp(2\pi i\frac{1}{2}, 2\pi i\frac{1}{2})\rangle$, since $|G^{max}_{x^4+y^6} : G^{max}_{x^4+bx^2y^3+y^6}| = 2$, and we do indeed get an orbifold B-model isomorphic to the A-model for $\mathcal{H}^{\langle J\rangle}_{W_{1,0}}$.

Given a weight system and the condition that W be quasihomogeneous of quasidegree 1, there may be many choices of polynomial representatives W for a particular singularity. There may be both invertible and non-invertible representatives, both loops and chains, both reducible and irreducible representatives. All of these can be found in the weight system $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. For any W which has the same weight system as an invertible singularity, we can take the transpose of that invertible singularity, and take an appropriate orbifold B-model to get something isomorphic to the A-model of the W we started with.

Suppose W is a non-invertible singularity. It's known that for any admissible group one can add terms such that the admissible group is the maximal symmetry group of the expanded polynomial, in a way we want to reverse the process. If $W = W_{inv}+Z_1+Z_2+\ldots+Z_k$ with W_{inv} invertible, then we consider the A-lattice of W_{inv}. Each Z_i is associated with an element G_j of the A-lattice, where $G_j = G^{max}_{W_{inv}+Z_j}$. Then $\bigcap_j G_j = G^{max}_W \leq G^{max}_{W_{inv}}$ is in the A-lattice of W_{inv} and we know the element $\left(\bigcap_j G_j\right)^T$ in the B-lattice.

Observation 1. For non-degenerate non-invertible $W = W_{inv} + Z_1 + Z_2 + \ldots + Z_k$ with W_{inv} invertible,

$$\mathcal{H}_{W,G^{max}_W} \cong \mathcal{B}_{W^T_{inv},\left(\bigcap_j G_j\right)^T}.$$

The A-model depends on the choice of symmetry group, which in turn depends on the choice of W representing a given singularity. If it proves necessary to change representation, there may not be as direct a way of specifying the appropriate group in the B-lattice, though the order will be $|G^{max}_{W_{inv}} : G^{max}_W|$.

The simplest singularity which does not admit any invertible singularity with the same weight system is $x^{16} + y^5x + z^3x + y^2z^2$ with weights $(\frac{1}{16}, \frac{3}{16}, \frac{5}{16})$. If we follow Kreuzer and Skarke's polytope duality [5] which generalizes the Berglund and Hubsch [1] transposition construction, the dual weight system has weights $\frac{7}{240}, \frac{1}{5}, \frac{1}{3}$, and the only monomials with quasidegree 1 are $x^{16}yz, y^5, z^3$, which do not give an isolated singularity. Thus, it appears that there are singularities for which there do not exist any non-degenerate W which will give the appropriate orbifold LG models, and FJRW theory in its current form may have hit its limit in this area.

For invertible singularities, $\hat{c}$ is the same for both W, W^T. There certainly seems to be geometric significance to this, but at present it is only an interesting observation. If we presume W, W^T have the same number of variables, this reduces to saying that the sum of the weights for W is the sum of the weights for W^T. For any weight system, there we may consider the simplex described by

$$\sum_i q_i X_i = 1, \qquad X_i \geq 0$$

and the lattice points it includes. These lattice points are exactly the monomials of quasidegree 1 in the given weight system, and they have a direct connection with linear relations among the weights by way of toric diagrams [6].The Newton polytope for W is contained insided this simplex, and can presumably be adapted to give the reflexive polytopes for Batyrev's polar duality [7]. Note that the choice of representative W affects the polytope, as the weight system [1,1,1;3] has 5 invertible representatives W with distinct Newton polytopes up to permutation of variables.

Sadly, I have no guidance as to the meaning of the sum of the weights when mapped from the toric diagram to the simplex.

2.1 Mirror Symmetry

Mirror symmetry in our context of Abelian Landau-Ginzburg orbifolds consist of looking for pairs W_1, G_1, and W_2, G_2 such that

$$\mathcal{H}_{W_1,G_1} \cong \mathcal{B}_{W_2,G_2}, \qquad \mathcal{B}_{W_1,G_1} \cong \mathcal{H}_{W_2,G_2}.$$

The original motivation comes from Witten in [8], where he considers twisted models, replacing the stress-energy tensor $T_{\alpha\beta}$ by

$$T_{\alpha\beta} \to T_{\alpha\beta} + \frac{1}{4}(\epsilon_\alpha^\gamma \partial_\gamma (J_{R\beta} \pm J_{L\beta}) + \alpha \leftrightarrow \beta)$$

where the plus sign gives the B-model, and the minus sign the A-model. Given the symmetry between the signs, there should be a symmetry between A- and B-models, if one can find the right choices of singularities. Batyrev developed an approach using polytopes in dual lattices which produced mirror symmetries not then known to physicists [7], which approach Kreuzer and Skarke found to give better results than only using quasihomogeneous functions, as all reflexive polytopes have a dual, while the hoped for mirror symmetry often failed with the functions [9, 5]. Kreuzer and Skarke did not have the theory to directly compute A-models, but having FJRW theory has simply confirmed that there are quasihomogeneous non-degenerate W which do not have a suitable dual W^T of the same form.

It seems that there should be a generalization or extension of FJRW theory which can extend to reflexive polytopes without requiring a non-degenerate W. Just which hypotheses and requirements should be kept of those necessary for being able to use FJRW theory could be a fruitful area of study.

2.2 What Might Come Next

There are singularites which have isomorphic LG models to chains or loops, but which are not of that form. The singularity $xy + zy + zx + xw^3$ has A- and B-models isomorphic to $w^3x + x^2$, where effectively we have a quasihomogeneous singularity isomorphism identifying the variables x,y, and z which have the same charge, $\frac{1}{2}$. While charges of 1/2 are grudgingly dealt with as special cases when chains have $a_N = 2$, the restriction $q_i < 1/2$ seems to come from FJRW theory being a cohomological theory where the relevan moduli spaces are not compact for $q_i \geq 1/2$ and so the integrals over the relevant cycles may not exist. The very simplest singularity $W = x^2$ has charge 1/2, and we shall see admitting situations of $q_i = 1/2$ will lead to charges of more than 1/2. Alternately, since we're working mod 1, charges greater than 1/2 could be considered negative, and there might be an orientation aspect to have signed weights of absolute values at most 1/2. Mirror symmetry still seems to hold if we go outside of situations where the charges have been restricted and apply the then unfounded algorithms.

It was first noticed in [2] that ordinary singularity isomorphisms do not generally preserve the A-model's structure. It is known that $x^2 + xw^3$ is dual with $x^2w + w^3$, but computation shows $xy + zy + zx + xw^3$ is indeed mirror to its transpose, namely $w^3 + xzw + xy + yz$. It may be of value to study the connection between these last two singularities to see what will preserve structure, and thereby learn more about that structure. In this last singularity, y has a charge of $\frac{2}{3}$ and the computer is going outside of proven methods since the weight is greater than $\frac{1}{2}$. We can map $xy + zy + zx + xw^3$ to $x^2 + xw^3$ simply by identifying the three variables x, y, z which all have weight $\frac{1}{2}$ and rescaling to eliminate constants.

We can't pull quite the same trick with $w^3 + xzw + xy + yz$ since now w, x, z all have weight $\frac{1}{3}$ and we want both w and one of x, z, but we still need to do something unmotivated and ad hoc to deal with y. If we end with $w^3 + x^2w + y^2$ we have singularity isomorphic to $x^2w + w^3$, so there is reason to suspect charges greater than a half can do strange things, if they are valid at all. Still, if one presses forward with eyes closed, things seem to work rather well. If we consider loops or chains where some of the a_i are one, we see more evidence that charges greater than a half always occur along with their complementary charge (q_i and $1-q_i$) and that the variables associated with oversized charges have a feeling of gluing things together. If we have a loop or chain with terms

$$\ldots + X_{i-1}^{a_{i-1}} X_i + X_i^{a_i} X_{i+1} + X_{i+1} X_{i+2} + X_{i+2} X_{i+3} + X_{i+3}^{a_{i+3}} X_{i+4} + \ldots$$

it turns out to be isomorphic to that same loop of chain with the $a_i = 1$ section removed and the ends "spliced" together:

$$\ldots + X_{i-1}^{a_{i-1}} X_i + X_i^{a_i a_{i+3}} X_{i+4} + \ldots$$

and this works for splicing over any even number of variables. If we have $a_i = 1$ for an odd number of consecutive variables, splicing does not give you an isomorphic singularity. This suggests that somehow the technique might be useful for constructing new behaviours, if it can be better understood. The singularity

$$x^2 y + ys + s^3 t + t^4 + yu + u^2 v + v^2$$

is isolated, non-degenerate, and quasihomogeneous, and rather than being a loop or chain, it's basically like a Y, in the sense that if we consider two variables connected when they appear together in the same monomial, this singularity's graph has a vertex of degree 3, namely y. Loops and chains have graphs that are basically discreet 1-manifolds. The charge $q_y = \frac{3}{4}$ means we are away from solid ground, but it should be possible to further extend a singularity of this form, or possibly create singularities with multiple branchings. Note that $q_t = 1/4 \neq q_v = 1/2$ and thus the branches are not copies of each other.

Chapter 3. Tutorial

This program has two principal commands: LGModels, for producing bases for the LG A- and B-models of a given isolated quasihomogeneous singularity, and ComputedCorrsOut, which will return those correlators of the given genus and number of points which are computable given methods available at the time of writing.

The procedure LGModels requires for input a polynomial and a list of variables for that polynomial, for example,

```
LGModels(x^3+y^3x,[x,y])}.
```

Letters not appearing in the input list will be treated as constants, such as b in

```
LGModels(x^16+x*y^5+x*z^3+b*y^2*z^2, [x,y,z]).
```

All output data will have tuples ordered to match the ordering given in the input. To avoid any possible confusion, it may be valuable to explicitly include an ordering in the variable names, such as

```
LGModels(x1^3+x2^3*x1+x3^4*x2,[x1,x2,x3])}.
```

The polynomial representative chosen for a singularity is relevant only insofar as it affects the symmetry group, as LGModels uses the maximal symmetry group for the given polynomial. Instructions for using other symmetry groups will appear further on. For the weights (1/3,1/3,1/3), one may choose almost any set of at least three terms from $x^3+y^3+z^3+x^2y+x^2z+y^2x+y^2z+z^2x+z^2y+xyz$, but the Fermat polynomial $x^3+y^3+z^3$ decomposes as the direct sum of three singularities, and LGModels will report this. One could alternately select terms to have Fermat plus chain $x^3+y^3+z^2y$, Fermat plus loop $x^3+y^2z+z^2y$, chain $x^3+y^2x+z^2y$, or loop $x^2y+y^2z+z^2x$. Thus, despite the fact that quasihomogeneous singularties are effectively classified by the weight system, and the weight system does determine the Milnor ring up to isomorphism [10], not all isolated quasihomogeneous representatives have the same symmetry group, and therefore the FJRW rings will differ.

The output [$\mathbf{q}$,$\mathcal{H}_G$, $\mathcal{Q}_G$]is a list of three lists, the first of which is a list of the charges of the singularity, the second is a basis for the LG A-model, and the third is a basis for the LG B-model. The basis elements themselves are given as a three element lists: the "degree" of the ring element (W-degree for A-model, quasidegree for B-model), the group element of the sector, given as a tuple of phases, and finally the polynomial. For Neveu-Schwarz sectors, the polynomial is given as 0.

The ComputedCorrsOut procedure is used for computing the multiplication of the FJRW ring, insofar as it possible at present. The program as included can compute most genus-zero three-point correlators and concave genus-zero four-point correlators. The inputs needed are the singularity, the list of variables, number of points, genus, charges, and basis for the A-model. The output is in the form of a list of sets: a set of equations giving the values of those correlators which can be explicitly computed, a set of equations from using the composition axiom, and the set of those possibly non-zero correlators which cannot be computed by the methods in the program.

A particularly effective approach is to use LGModels in an assignment of the form

```
RingData:= LGModels(Singularity, Variables):
```

so that the output is stored for use in ComputedCorrsOut as

```
RingCorrelators:= ComputedCorrsOut(Singularity, Variables,3,0,RingData):
```

among other possibilities, and this avoids having overly large ring bases being printed to the screen.

The default approach of LGModels is to use the maximal diagonal symmetry group of W for the A-model and the trivial group for the B-model, which yield the Milnor ring for W. One may use other groups for orbifolding or invariance, and the full syntax of LGModels is LGModels(W, Variables List, A-side Orbifolding group, A-side Invariance group, B-side Orbifolding Group, B-side Invariance group). The group data is entered in the form of a list of generating elements as tuples of phases, a polynomial for which the group elements must be a symmetry, and a list of variables for the polynomial.

For example, if we want to find $\mathcal{H}_{D_5,\langle J\rangle}$, one could input

```
LGModels(x^5+y^2x,[x,y],[[[1/5,2/5]],x^5+y^2x,[x,y]]).
```

Note that the generator [1/5, 2/5] must itself be put into a list, hence [[1/5, 2/5]]. The procedure assumes that the invariance group is to be the same as the orbifolding group. To leave some groups at the default setting while feeding parameters later in the list, enter [[]] in place of the input groups which are to be left to default. Additionally, LGModels will reject polynomials which are the direct sums of two or more singularities unless there is some input into at least one of the group parameters. Since the default setting is using G^{max} and the FJRW ring of a direct sum is the tensor product of the FJRW rings of the summands when using the respective maximal symmetry groups, the presumption is that it is more important to understand the summands than have the tensor product.

Chapter 4. Procedures in the Code

Below are guides to all the procedures defined in the code. The first line gives the procedure name, the syntax without optional arguments, and the output. Below that is the explicit Maple syntax for the procedure input, and finally, a description of what the procedure does and how it works.

Procedure Name ProcedureName(Input for Procedure); Output of Procedure

`Inputs and their Maple type requirements, if applicable`

Explaination of Procedure

The procedures cHat, sPlus, sMinus, kaufmannS, kaufmannSbar, Rshift, iota, SectorMu are all take γ, $\mathbf{q} = (q_1, q_2, \dots, q_N)$ as input and return the relevant number for the appropriate sector, i.e. cHat returns $\hat{c}_\gamma$, ngamma returns N_γ, the dimension of the fixed locus of γ, and SectorMu returns the dimension of the Milnor ring of Fix γ.

IsolatedTest IsolatedTest($W, [x_1, \dots, x_N]$); Boolean

`W::polynom, Vars::list`

Finds the zero locus of the Jacobean ideal (∂W) and tests to see if it contains the origin as an isolated point. Since Maple returns solutions to systems of polynomials in the most general form possible, the solution $x_1 = 0, \ldots, x_N = 0$ will appear exactly if the origin is isolated.

DegRead DegRead$(x_1^{a_1} x_2^{a_2} \ldots x_N^{a_N}, [x_1, \ldots, x_N])$; $[a_1, a_2, \ldots, a_N]$

`Wj, Vars::list`

Converts a monomial into a tuple of the degrees of the respective variables.

GetMatrix GetMatrix$(W, [x_1, \ldots, x_N])$; matrix B, where $B_{i,j}$ is the power of x_j in the ith monomial

`W::polynom, Vars::list`

Builds a matrix whose rows correspond to the monomials of W, where the entries are the degrees of the monomial in the respective variables.

GroupInvariantsA GroupInvariantsA$([\gamma, \mathcal{Q}_{W_\gamma}], \{g_1, \ldots, g_s\}, [x_1, \ldots, x_N])$; $[\gamma, \mathcal{Q}_{W_\gamma}^{\langle g_1, \ldots, g_N \rangle}]$

`SectorMilnorData::list,Generators::set,Vars::list`

Finds monomials invariant under the group action. Interestingly, we can use this procedure for both A- and B-models, since the contribution of the N-form in the A-model group action is exactly the same as the contribution from the determinant twist for the B-model group action.

Reducibility Reducibility$(W, [x_1, \ldots, x_N])$;

[true] or [false,[direct summands comprising W]]

`W::polynom,Vars::list`

Determines if the singularity $W = G^{max}$ is a direct sum of other singularities, and if so, returns the direct sum decomposition of the singularity. When using non-maximal symmetry groups, in particular $\langle J \rangle$, this decomposition does not apply. The approach is to build a graph with the variables $x_1, \ldots, x_N$ as the vertex set, with edges connecting variables that occur together in some monomial. We then look for what the connected components of the graph are.

AddGeneration AddGeneration(List of tuples so far, $\mathbf{q}$, $\hat{c}$, Postion in N-tuple); List of tuples increased by including powers of new variable

`InList::list, Charge::list, cHat, tupleposition`

Used exclusively inside of TrialElts for recursively building a list of all monomials with quasidegree at most $\hat{c}$, mod the Jacobean Ideal. The AddGeneration expands the input list to extend to one variable more than previously, by multiplying everything in the input list by all powers of the new variable which yield a product of quasidegree at most $\hat{c}$. Here, we're actually working with tuples of exponents, rather than using any variables. The InList has entries of the form [[tuple of exponents],quasidegree].

TrialElts TrialElts($\mathbf{q}$, $\hat{c}$,basis of ∂W,[$x_1, \ldots, x_N$]); $\mathcal{Q}_W$ in integer tuple form, with quasidegree appended to each element.

`Charge::list, cHat, JacobIdealBasis, Vars::list`

Produces a list of all monomials in the variables $x_1, \ldots, x_N$ of quasidegree at most $\hat{c}$ modulo ∂W. The list of monomials is built up one variable at a time, where each application of AddGeneration adds the next variable in the list, and after each generation, we reduce the list mod ∂W. The final output is a list with entries of the form [[tuple of exponents],quasidegree].

NotDivisibleBy NotDivisibleBy(tuple of exponents,second tuple of exponents,N); Boolean

`TrialElt::list, Criterion::list, dim::integer`

Identifying the tuples of exponents with monomials, this procedure returns whether or not it is true the second monomial does NOT divide the first.

Exponentiate Exponentiate(tuple of exponents [$a_1, \ldots, a_N$], [$x_1, \ldots, x_N$]);

$x_1^{a_1} x_2^{a_2} \dots x_N^{a_N}$

`tuple; Vars::list`

Converts a list into a monomial in the given variables. Used for converting back to algebraic format after times when it is more convenient to forget the variables and simply deal with monomials as integer tuples.

MilnorBasis MilnorBasis($W_\gamma, [x_{i_1}, \dots, x_{i_N}]$ such that $\mathrm{Fix}_\gamma = \mathrm{Span}\{x_{i_k}\}, \mathcal{Q}_W$); $\mathcal{Q}_{W_\gamma}$,

`W::polynom, Vars::list, RawBasis`

Finds a basis for the Milnor ring of $W_\gamma = W|_{Fix(\gamma)}$ which is a subset of the full basis given. There is a subtlety here, as simply computing the restricted basis from scratch introduces the risk of having different representations for the same equivalence class mod ∂W.

TempWBasis TempWBasis($\gamma, W, \mathcal{Q}_W, \mathbf{q}, [x_1, \dots, x_N]$); $[\gamma, \mathcal{Q}_W, \mathbf{q}|_{Fix(\gamma)}$

`g::list, W::polynom, basislist::listlist, q::list, Vars::list`

Finds the variables fixed by γ and the restrictions $W_\gamma, \mathbf{q}|_{Fix(\gamma)}$ and uses MilnorBasis to return a Milnor basis for the fixed locus, along with the charges for the variables in the fixed locus.

TransposeW Transpose$W(W, [x_1, \dots, x_N])$; W^T

`W::polynom; Vars::list`

Gives the transpose singularity for invertible W.

SortOutZeros SortOutZeros(any list); [[Positions of nonzero entries],[Positions of zero entries]]

`L::list`

Locates all the zeros in a list e.g. finds which variables are fixed by a group element.

IndexZeroSort IndexZeroSort(List of line bundle degrees) [Locations of zeros, locations of -2's, Boolean for if there's exactly one of each]

`L::list`

The readily computable index-zero classes are those with line bundle degrees all -1, except for a single 0 and a single -2. This procedure checks for these, and identifies the relevant variables when they exist.

IndexZeroValuesOut IndexZeroValuesOut($W[x_1, \ldots, x_W]$,genus g,$\mathbf{q}$,List of correlators $\langle e_{i_1}, \ldots, e_{i_n}\rangle$); [List of entries $\langle e_{i_1}, \ldots, e_{i_n}\rangle$ = computed value, Correlators from input list which aren't computable by index zero approach]

`W::polynom, Vars :  :list, genus ::integer, Charges::list,`

`Correlators::listlist`

Tests the input list of correlators to find those which are index zero and for which the degree of the Witten map can be computed at this time. The output consists of a list of equations expressing the values of those correlators in the input list which are computable, and the second list is all those left over. These leftovers might not be index zero, or in rare cases, they may have Witten maps where computing the degree isn't within the scope of the program at this time.

RamPairingOut RamPairingOut($W, [x_1, \ldots, x_N]$,$\mathbf{q}$,List of correlators $\langle e_i, e_j, e_k\rangle$); [List of entries $\langle e_i, e_j, e_k\rangle$ = computed value, Correlators from input list which do not reduce to the pairing]

`W::polynom, Vars::list, Charges::list, Correlators::listlist`

Computes the values of all 3-point correlators determined by the pairing. (Note that only 3-point correlators are directly determined by the pairing, so any higher correlators will be returned in the unknown list, since this approach won't be successful.)

ComputedFourPtCorrsOut ComputedFourPtCorrsOut($W, [x_1, \ldots, x_N]$;genus g, $\mathbf{q}$, List of correlators $\langle e_{i_1}, \ldots, e_{i_n}\rangle$); [List of entries $\langle e_{i_1}, \ldots, e_{i_n}\rangle$ = computed value, Correlators from input list which aren't computable by concavity approach]

`W::polynom; Vars::list, genus::integer, Charges::list,`

`Correlators::listlist`

Computes concave four-point correlators.

CompositionAxiom CompositionAxiom($W, [x_1, \ldots, x_N]$,genus g, $\mathbf{q}$,List of correlators $\langle e_{i_1}, \ldots, e_{i_n}\rangle$); [List of equations in correlators values, Correlators from input list for which the composition axiom doesn't give new information]

`W::polynom; Vars::list; genus::integer; Charges::list;`

`Correlators::listlist`

Finds equations in 3-point correlators with one Ramond insertion by using the composition axiom to relate these to computable index zero 4-point classes. After identifying such a 3-point correlator in the input list, the program next identifies all sectors that have a non-zero pairing with the Ramond insertion sector, and the 3-point correlators with one Ramond insertion which will pair with the original Ramond insertion (called Complements in the code). We then take the NS sectors from the original correlator along with the complements, and look for any index zero four point classes. We then have all the pieces to return exactly the composition axiom's equation

$$\langle \gamma_1, \gamma_2, \gamma_3, \gamma_4\rangle = \sum_{\alpha,\beta} \langle \gamma_1, \gamma_2, \alpha\rangle \eta^{\alpha,\beta} \langle \beta, \gamma_3, \gamma_4\rangle \tag{4.1}$$

ComputedCorrsOut ComputedCorrsOut($W, [x_1, \ldots, x_N]$, number of points k, genus g, $\mathbf{q}$, $\mathcal{H}_W$); [List of correlators we can compute with their values, (List of equations for correlators not completely determined, if any,) List of correlators about which we can't compute anything.]

`W::polynom, Vars::list, NumPoints::integer, genus::integer,`

`Charges::list, CurlyH::listlist`

Finds all possibly non-zero correlators of the given g; k for the given ring. Applies the pairing, index zero, and composition axioms to compute as much information as possible about these correlators; at present we can compute most 3-point correlators, but not much beyond that.

ChooseWithRepeats ChooseWithRepeats(Size of set, Length of tuple); List of all multisets with Length elements from 1 to Size, expressed as lists

`n::integer, k::integer`

Used to overproduce all the possible correlators; generally the input n will be $h = |\mathcal{H}_W|$, and k is the number of points for the correlator.

LBdegrees LBdegrees(genus g, $\mathbf{q}$,ExplicitCorrelator); List of line bundle degrees of the correlator

`genus::integer, Charges::list, Correlator::list`

Computes the line bundle degrees of the given correlator; the order of the degrees matches the variable order of the charges as input.

TestDegs TestDegs(List); Boolean

`K::list`

Returns true exactly if the list is composed entirely of integers

MakeCorrelators MakeCorrelators($\mathcal{H}_W$, number of points k); List of all k-point correlators from the basis $\mathcal{H}_W$

`CurlyH::listlist, NumPoints::integer`

Produces all k-point correlators from the given basis. The vast majority of these will be zero, and so this procedure should almost always be used to feed into something that will filter the results down.

AllIntList AllIntList(number of points k, genus g, $\mathbf{q}$, $\mathcal{H}_W$); List of correlators with integer line bundle degrees

`NumPoints::integer, genus::integer, Charges::list, CurlyH::listlist`

Gives you the list of all the correlators formed from the given basis which have all integer line bundle degrees. These are the only possible non-zero correlators, and so we get to basically ignore all the others.

AllNeg AllNeg(List); Boolean

`K::list`

Returns true exactly if all the items in the list are less than zero.

IndexZeroTest IndexZeroTest(List of line bundle degrees of a correlator, genus g); Boolean

`LBDsOfCorrelator::list, genus::integer`

Returns true exactly when the line bundle degrees meet the condition for the correlator being index zero. This is necessary, but not sufficient, as the correlator might contain a Raymond sector.

Dim Dim(genus g, $\mathbf{q}$, ExplicitCorrelator); Algebraic cohomological dimension of ExplicitCorrelator

`genus::integer, Charges::list, L::listlist`

Computes the algebraic cohomological dimension of the input correlator.

NeveuSchwartz NeveuSchwartz(ExplicitCorrelator); Boolean

`Correlator::listlist`

Returns true exactly when the input correlator has only NS insertions.

HasUnit HasUnit(ExplicitCorrelator, $\mathbf{1}$); Boolean

`Correlator::listlist, Unit::list`

Returns true exactly if the correlator has the specified sector as an insertion. This doesn't have to be used with the unit as the specified sector, but that's what's been most useful, and the context in which it's used inside other procedures.

SortedCorrelators SortedCorrelators(Number of points k, genus g, $\mathbf{q}$, $\mathcal{H}_W$); Prints to screen only, no output

`NumPoints::integer, genus::integer, Charges::list, CurlyH::lislist`

Separates the set of possibly non-zero correlators into lists based how and if they can be computed, and then prints these lists to screen. Largely obsolete with the development of

ComputedCorrsOut, but seeing this information might have some use with looking patterns in types of correlators.

HomZeroDimList HomZeroDimList(genus g, $\mathbf{q}$, List of correlators); Subset of input list which have homological dimension zero

`genus::integer, Charges::list, K::listlist`

Filters a list of correlators, and returns only those which have homological dimension zero, or equivalently, cohomological dimension $3g-3+k$.

CohoZeroDimList CohoZeroDimList(genus g, $\mathbf{q}$, List of correlators); Subset of input list which have cohomological dimension zero

`genus::integer, Charges::list, K::listlist`

Filters a list of correlators, and returns only those which have cohomological dimension zero, or equivalently, homological dimension $3g-3+k$.

CohoGoodResDimList CohoGoodResDimList(genus g, $\mathbf{q}$, List of correlators); Subset of input list which have cohomological dimension one less than expected, i.e. homological dimension $3g-3+k-1$

`genus::integer, Charges::list, K::listlist`

Filters a list of correlators, returning those that restrict well to the boundary, with homological dimension $3g-3+k-1$. This is used for composition axiom computations.

ZeroRestrictList ZeroRestrictList(Number of points k, genus g, $\mathbf{q}$, $\mathcal{H}_W$); Prints to screen only, no output

`Numpoints::integer, genus::integer, Charges::list, CurlyH::listlist`

Computes which correlators have integer line bundle degrees and cohomological degree zero, and for each you get a line printed with the correlator, if it's concave, if it's index zero, and the line bundle degrees.

GoodRestrictList GoodRestrictList(Number of points k, genus g, $\mathbf{q}$, $\mathcal{H}_W$); Prints to screen only, no output

`Numpoints::integer, genus::integer, Charges::list, CurlyH::listlist`

Computes which correlators have integer line bundle degrees and cohomological degree $3g - 3 + k - 1$, and for each you get a line printed with the correlator, if it's concave, if it's index zero, and the line bundle degrees.

GetCoordinates GetCoordinates($[\gamma_1, \gamma_2, \ldots, \gamma_m], \gamma$); $[a_1, a_2, \ldots, a_n]$ where $\gamma = \prod \gamma_i^{a_i}$

`BasisVectors::list, GroupElt::list`

Tries to find how to express the given group element γ as a product of the generators γ_i. Returns FAIL if such a representation is not possible. The procedure currently does not check if the generator set might be excessive, and thus the representation might not be unique.

CompactOutputFormatting CompactOutputFormatting($[\gamma_1, \gamma_2, \ldots, \gamma_m]$, Sector in program output form [degree, γ, polynomial]); polynomial $\mathbf{e}_{a_1,a_2,\ldots,a_n}$ where $\gamma = \prod \gamma_i^{a_i}$, unless the polynomial is 0 or 1

`BasisVectors::list, Sector::list`

This procedure converts the Maple syntax for imple ring elements into the **e** format, with respect to the input basis for the group G. If the monomial for the sector is 0 (the sector is NS) then there will be no polynomial, and in a very annoying piece of having to work around Maple trying to be helpful, when the monomial is 1, as happens in singularities with integral $\hat{c}$, I put the symbol @, since Maple with treat 1 as the understood coefficient. I regret the necessity, but it's a rare enough case that the other options were much worse for normal use.

BigOutputConversion BigOutputConversion(Expression, $[\gamma_1, \gamma_2, \ldots, \gamma_m], \mathcal{H}_W$); Expression using **e** notation

`TargetObject, BasisVectors::list, CurlyH::listlist`

Converts general A-model ring elements to the **e** notation.

BMultiply BMultiply$(c_1(\prod x_i^{a_i})\mathbf{q}_1, c_2(\prod x_i^{b_i})\mathbf{q}_2, W, [x_1, \ldots, x_N])$; $(c_1(\prod x_i^{a_i})\mathbf{q}_1 \star c_2(\prod x_i^{b_i})\mathbf{q}_2$

`FirstTerm::list, SecondTerm::list, W::polynom, Vars::list`

Performs B-model multiplication for simple B-model elements where $\mathbf{q}_1, \mathbf{q}_2 \in \mathcal{Q}_W$.

MultiplicationTable MultiplicationTable$(W, [x_1, \ldots, x_N], \mathcal{Q}_W)$; $[\mathbf{q}_i \star \mathbf{q}_j]$

`W::polynom, Vars::list, BBasis::listlist`

Orders $\mathcal{Q}_W$ by degree and returns a matrix which is the ring multiplication table. Primarily useful for when the B-model is not the Milnor ring.

BPairingMatrix BPairingMatrix$(W, [x_1, \ldots, x_N], \mathcal{Q}_W)$; $[\langle \mathbf{q}_i, \mathbf{q}_j \rangle]$

`W::polynom, Vars::list, BBasis::listlist`

Orders $\mathcal{Q}_W$ by degree and returns a matrix giving the values for the pairing. This is a small step past the multiplication table.

BCorrs BCorrs$(W, [x_1, \ldots, x_N], \mathcal{Q}_W)$; $\{\langle \mathbf{q}_i, \mathbf{q}_j, \mathbf{q}_k \rangle = \text{value}\}$

`W::polynom, Vars::list, BBasis::listlist`

Computes the values of all 3-point B-correlators, and returns the data as a set of equations. This is used internally to be matched up with the known A-model correlators to try to find isomorphisms explicitly.

ConvertExpressions ConvertExpressions(ExplicitCorrelator, $\mathcal{H}_W, \mathcal{Q}_W$);

Equation in B-side correlators involving coefficients $BtoA_{i,j}$

`Correlator::listlist, ABasis::listlist, BBasis::listlist`

For the given A-model correlator, returns an equation in B-model correlators involving linear coefficients for the B- to A-model isomorphism. Since we can explicitly find the 3-point B-correlators, this puts you just a step or two away from explicitly finding the isomorphism(s). After verifying that the gradings are compatible, we assume there is a vector space isomorphism between subspaces of a given degree, with $BtoA_{i,j}$ as the coefficients of the map.

FindSubsTargets FindSubsTargets(Equation, $\mathcal{H}_W, \mathcal{Q}_W$); Equation with all A-model correlators replaced with B-model correlators

`Correlator::listlist, ABasis::listlist, BBasis::listlist`

Switches out all occurences of AValue() in an equation (of limited depth of operations) and returns the same equation, now with B-side correlators. Used for restating Composition Axiom equations in the isomorphism finder.

IsomorphismFinder IsomorphismFinder(Output from ComputedCorrsOut for 3-pt genus-0 correlators for W, $\mathcal{H}_W, \mathcal{Q}_W, W, [x_1, \ldots, x_N]$); Equations for explicit isomorphisms from the given B-model basis to the given A-model Basis

`ComputedCorrsOutput::list, ABasis::listlist, BBasis::listlist,`

`W::polynom, Vars::list`

After generating data to compute the 3-point genus zero correlators on the A side using ComputedCorrsOut, Isomorphism Finder uses Maple's polynomial solving to look for explicit isomorphisms as a B- to A-model map. Be advised, this can be quite slow and memory intensive, but you get out explicit algebra isomorphisms, to the extent that data is available. There may be degrees of freedom remaining, though this can at least in part reflect underlying symmetries, and not a failure in finding isomorphisms.

MinimalGenSet MinimalGenSet(G); $[\gamma_1, \ldots, \gamma_m]$ where these γ_i are a minimal generating set for the input group

`Group::listlist`

Produces a minimal generating set for the input group. If the input is not a group, the procedure will produce a minimal generating set for the group generated by the input. For each i from 1 to N (the number of variables), we take a group element with θ_i at the minimal positive value. These elements are then sorted by decreasing order as group elements, and we give as the minimal generating set the first k of these elements, where k is the smallest value such that they generate the full group. If all N are insufficient to generate the input G, then we take G as a generating set, and return the minimal generating set for the group

G generates. This is assuredly not the most efficient way to reduce a linearly dependent generating set, though that is a possible use.

MakeGroup MakeGroup($[\gamma_1, \ldots, \gamma_m]$, polynomial, $[x_1, \ldots, x_N]$);
$[G, [\gamma_1, \ldots, \gamma_m]]$ where these γ_i are a minimal generating set for the input group
`Generators::list, W::polynom, Vars::list`
Gives the intersection of the group generated by the input and the symmetry of the input polylnomial. Inputting the zero polynomial will return the trivial group, while a non-zero constant polynomial will impose no restriction on the group generated by the input. The minimal generating set is found by the MinimalGenSet procedure.

Preliminary Preliminary($W, [x_1, \ldots, x_N]$); Output is a list, with three cases.
`W::polynom, Vars::list`
This procedure determines if the input polynomial has an irreducible isolated quasihomogeneous singularity at the origin. If the input fails to be isolated or quasihomogeneous, the output is the one element list [false]. If the singularity is a direct sum of singularities, the output is [true, List of summand singularities]; if the input is irreducible, the output is [true, [$\mathbf{q}$, G_W, a generating set for G_W]].

LGModels LGModels($W, [x_1, \ldots, x_N]$, (optional inputs)G_1, G_2, G_3, G_4);
[$\mathbf{q}$, $\mathcal{H}_{G_1}^{G_2}$, $\mathcal{Q}_{G_3}^{G_4}$]
```
W ::polynom, Vars :  list, ASectorsGroup ::list:= [[ ],1,Vars],
BSectorsGroup ::list:= [[ ],1,Vars], AInvarianceGroup ::list:= [[ ],1,Vars],
BInvarianceGroup ::list:= [[ ],1,Vars]
```
Produces bases for A- and B-side rings for the given singularity. The A-side defaults to the maximal abelian symmetry group of W, while the B-side defaults to the trivial group. The optional slots are independent of each other. Also note that entering any information for the optional slots will override the default approach of stopping short when presented with a reducible singularity. It is very useful to use this procedure in an assignment with

output suppressed from the screen, to have the information to feed to correlator computation procedures, and to avoid the (noticable) time needed to print the bases of very large rings to the screen.

SymGroup SymGroup($W, [x_1, \dots, x_N]$); [$\mathbf{q}$, G_W, a generating set for G_W]

`W::polynom, Vars::list`

Returns the charges of the polynomial, its maximal abelian symmetry group, and a generating set for said group.

Chapter 5. Maple Code

```
 restart, with(ListTools) : with(LinearAlgebra) : with(Groebner) :
with(Graph Theory) : with(combinat) :


IsolatedTest := proc (W::polynom, Vars::list)
    local JacobeanIdeal,LocusSolutions, LocusComponents;
    description "Finds the zero locus of the Jacobean ideal,
and tests if it contains the origin and is zero dimensional. ";
    JacobeanIdeal := convert( map(proc (a) options operator, arrow,
diff(W, a) end proc,Vars), set);
    LocusSolutions:= MakeUnique(solve(JacobeanIdeal, Vars));
    if not eval(W, [seq(k = 0, ‘in‘ (k, Vars))]) = 0 then
        print("Hypersurface in specified variables does not include origin");
        return false
    end if;
    if not member([seq(k = 0, ‘in‘ (k, Vars))],LocusSolutions) then
        return false
    else return true
    end if
end proc:


DegRead := proc (Wj, Vars::list)

    local TuplelnProgress, N, i;

    description "This rips the coefficents off a monomial

to give an integer list.";

    N:= nops(Vars);

    TuplelnProgress:= NULL;

    for i to N do

        TuplelnProgress:= Flatten([TuplelnProgress, degree(Wj, op(i, Vars))])

    end do;

    return TuplelnProgress

end proc:


GetMatrix :=proc(W, Vars)

    # builds the matrix B, where rows correspond to monomials,

# so B_i,j is the power of variable X_j in the ith monomial

    if type(W, ‘+‘) then

        return Matrix([seq(DegRead(op(W), Vars) , i = 1..nops( W))]);

    else return Matrix([DegRead(W, Vars)]);

    end if;

end proc:


GroupInvariantsA := proc (SectorMilnorData::list, Generators::set, Vars::list)

    local FixLocusBasis, i, k, FixedPlaces, RamondList, ActionImage;

    description "Takes in the Milnor ring for sector and returns those

invariant under the group action.

When applied to a B model, this gives

the determinant-twisted group action.";
```

```
    if nops(SectorMilnorData)= 2 then
            # This happens exactly when the fixed locus is trivial;
# the 2 elements are the group element and 0.
        return SectorMilnorData
    end if;
    RamondList:= SectorMilnorData[2];
# this is the basis for the sector's Milnor ring
    FixedPlaces:= SortOutZeros(SectorMilnorData[1]) [2];
    for i to nops(Generators) do
        RamondList := select(proc (item) options operator, arrow,
type (ListTools[ DotProduct] (DegRead(item,Vars),
            convert(Generators[i],list)) + add(Generators[i] [j], j
= FixedPlaces), integer) end proc,RamondList)
                # This is just the definition of the group action
    end do;
    return seq([SectorMilnorData[1], RamondList[k]], k = 1 .. nops(RamondList))
end proc:


Reducibility := proc (W::polynom, Vars::list)
    local Connectors, i, j, Edges, n, GraphPieces, TensorPieces, Vertices;
    description "Determines if polynomial can be written as a direct sum,
and if so returns the summands. Does not check if the pieces are
            singularities.";
    n:= nops(Vars);
    Edges:= { };
    Vertices:= [ ];


        # We're going to make a graph with a vertex for each variable,
        # and edges connecting pairs of variables
        # which occur together in a term of W
    for i to n do
        if diff(W, Vars[i]) != 0 then
            Vertices:= [op(Vertices), Vars[i]]
        end if
    end do;
    Connectors := select(proc (item) options operator, arrow,
  type (item,‘*‘) endproc, W);
    for i to n-1 do
        for j from i + 1 to n do
            if diff(Connectors, Vars[i], Vars[j]) != 0 then
# This identifies if there are any terms containing both variables i,j
                Edges:= ‘union‘(Edges,{{Vars[i], Vars[j]}})
            end if
        end do
    end do;
    GraphPieces := ConnectedComponents(Graph(Vertices, Edges));

        # The components of the graph correspond to the summands of W.
    if 1 < nops( GraphPieces) then
        TensorPieces := seq([W-(eval(W, [seq(k = 0,‘in‘ (k, GraphPieces[l]))])),
GraphPieces[l]], l = 1 .. nops(GraphPieces));
        return [false, TensorPieces]
    else return [true]
    end if
end proc:

AddGeneration := proc (InList::list, Charge::list, cHat, tupleposition)
```

```
    local NewGeneration, dim, inSize, listposition, chargeMult, chargeBit;
    description "Adds all possible powers of a new variable to
 the input list of monomials (in integer list form).
 Is used exclusively inside of the TrialElts procedure for generating
a monomial basis for a Milnor ring with maximum grading cHat";
    NewGeneration := [ ];
    inSize := nops( InList);
    dim := nops(Charge);
    for listposition to inSize do
        for chargeMult from 0
         while chargeMult*Charge[tupleposition]
                           + InList[listposition][2] < cHat do
            chargeBit:= subsop(tupleposition = chargeMult, [seq(0, k=1 .. dim)]);
            NewGeneration:= [op(NewGeneration),
                        [InList[listposition][1] + chargeBit,
chargeMult*Charge[tupleposition] + InList[listposition][2]]]
         end do
    end do;
    return NewGeneration
end proc:

TrialElts := proc (Charge::list, cHat, JacobIdealBasis, Vars::list)
    local ListlnProg, NewGeneration, PrevGeneration, tupleposition, dim, starter;
    description "Builds a list of all monomials with degree at most cHat,
reduced mod the given basis. The list is guaranteed not to contain
equivalent elements, but if your value for cHat
 is too low, you will be missing elements, and won't have a basis";
    dim := nops(Charge);
    starter:= [[[seq(0, k = 1 .. dim)], 0]];
    NewGeneration := starter;
    ListInProg:= NewGeneration;
    for tupleposition to dim do
        NewGeneration:=
            AddGeneration(NewGeneration, Charge, cHat, tupleposition);
        NewGeneration := map(proc (item) options operator, arrow,
[NormalForm(Exponentiate(item[1], Vars), JacobIdealBasis,plex(op(Vars))),
 item[2]] endproc, NewGeneration);
        NewGeneration:= remove(proc (item) options operator, arrow,
 evalb(item[1] = 0 or type(item [1], '+')) end proc, NewGeneration);
        NewGeneration:= map(proc (item) options operator, arrow,
[DegRead( item [1], Vars) ,item[2]] endproc, NewGeneration);
        ListInProg:= MakeUnique([op(ListInProg), op(NewGeneration)])
    end do;
    gc();
    return ListInProg
end proc:

NotDivisibleBy := proc (TrialElt::list, Criterion::list, dim::integer)
    local i;
    description "Returns true exactly when TrialElt is NOT
divisible by Criterion, where both TrialElt and Criterion
are monomials expressed as integer lists. ";
         for i to dim do
        if TrialElt[i] - Criterion[i] < 0 then
            return true
        end if
    end do;
    return false
```

```
end proc:


Exponentiate := proc (tuple, Vars)

    local InProg, k;

    description "Converts an integer list to a monomial, Vars[k]^(tuple[k])";

    InProg:= [ ];

    for k to nops(Vars) do

        InProg:= [op(InProg), Vars[k]^tuple[ k]]

    end do;

    return mul( i ‘in‘ (i, InProg))

end proc:


MilnorBasis := proc (W::polynom, Vars::list, RawBasis)

    local JacobIdeal, B, NearBasis;

    description "Finds a basis for a Milnor ring given a larger basis.

Used to get a basis for the Milnor ring of a polynomial restricted

to a fixed locus, given the basis for the full polynomial’s Milnor ring";

    if W = 0 then

        return [ ]

    else

        JacobIdeal := convert(map(proc (a) options operator, arrow,

                    diff(W, a) end proc, Vars), set);

        B := Basis(JacobIdeal,plex(op(Vars)));

        NearBasis:= MakeUnique(NormalForm(RawBasis, B, plex(op(Vars))));

        NearBasis:= map(DegRead,‘minus‘( {op(NearBasis)}, {0}), Vars);

        NearBasis:= convert( map(Exponentiate, NearBasis, Vars), list);

        return NearBasis

    end if

end proc:
```

```
TempWBasis:= proc (g, W, basisList, q, Vars)
    local temporaryW,rawTemporaryBasis, tempCharges, i, goodTempBasis;
    description "Gives a Milnor basis for the fixed locus of each group element,
 and the charges for the fixed variables.
Neveu-Schwartz elements give the basis as 0, to distinguish them";
    temporaryW:= W;
    rawTemporaryBasis:= basisList; tempCharges := convert(q, list);
    goodTempBasis:= [ ];

        # Here we find the restriction of W to the fixed locus
    for i to nops(Vars) do
        if not g[i] = 0 then
            temporaryW:= eval(temporaryw, Vars[i] = 0);
            rawTemporaryBasis:= remove(has, rawTemporaryBasis, Vars[i])
        end if
    end do;
    for i to nops(Vars) do
        if DegRead( temporaryw, Vars)[i] < 1 then
            tempCharges[i] := 0
        end if
    end do;
    if temporaryW = 0 then
        return [g, 0]
    else
        goodTempBasis:= MilnorBasis(temporaryw, Vars, rawTemporaryBasis);
        return [g, goodTempBasis, tempCharges]
    end if
endproc:
```

```
TransposeW:= proc (w, Vars::list)
    local listForm, i,j, Wtr;
    description "Returns the transpose singularity
for those singularities which are invertible.";
    Wtr:= 0;
    if not type(W, ‘+‘) then
        return W
    elif not nops(W) = nops(Vars) then
        print("Not an invertible singularity");
        return NULL
    else
        listForm:= map(DegRead, [op(W)], Vars);
        for i to nops(Vars) do
           Wtr:= Wtr+ Exponentiate([seq(listForm[j,i], j=1.. nops(Vars))], Vars)
        end do;
        return Wtr
    end if
end proc:

SortOutZeros := proc (L)
    local i, Nonzero, Zero;
    description "For a given input, returns a list of where the input entries
are NON-zero, and then where input entries are zero";
    Nonzero:= [ ];
    Zero := [ ];
    for i to nops( L) do
        if L[i] = 0 then
            Zero := [op( Zero), i]
        else Nonzero := [op(Nonzero), i]
        end if
    end do;
    return [Nonzero, Zero]
end proc:


IndexZeroSort:= proc (L::list)
    local i, Zero, Neg2;
    description "For computing index zero correlators:
returns where the list (presumably of line bundle degrees) has 0's and -2's,
and then the third component returns true exactly if there is one of each,
which is the only time we can (easily) compute the degree of the Witten map.";
    Neg2:= [ ];
    Zero := [ ];
    for i to nops(L) do
        if L[i] = 0 then
            Zero := [op( Zero), i]
        elif L[i] = - 2 then
            Neg2:= [op(Neg2), i]
        end if
    end do;
    return [Zero,Neg2, evalb(nops(Zero) = 1 and nops(Neg2) = 1)]
endproc:


IndexZeroValuesOut:= proc (W::polynom, Vars::list, genus::integer,
             Charges, Correlators::listlist)
    local NSCorrelators, IZCorrelators,TempData, position,
             KnownCorrelatorValues, UnknownSet;
    description "Computes all elementary index zero correlators
```

```
from the given list (those with a Witten map from
a 1-dim space to a 1-dim space).";
    KnownCorrelatorValues:= { };
    UnknownSet:= { };
    NSCorrelators:= select(NeveuSchwartz, Correlators);
    IZCorrelators:= select(proc (corr) options operator, arrow,
IndexZeroTest(LBdegrees(genus,Charges, corr), genus) endproc,
 NSCorrelators, genus);
    for position to nops(IZCorrelators) do
        TempData:=
 IndexZeroSort(LBdegrees(genus, Charges, IZCorrelators[position]), genus);
        if TempData[3] then
             KnownCorrelatorValues:= `union`(KnownCorrelatorValues,
                   {IZCorrelators[position] =
                   -degree(diff(W, Vars[TempData[2]]),Vars[TempData[1]])})
             else UnknownSet:= `union`(UnknownSet, {IZCorrelators[position]})
        end if
    end do;
    return [KnownCorrelatorValues, UnknownSet]
end proc:


RamPairingOut := proc (W::polynom, Vars::list,
                 Charges::list, Correlators::listlist)
    local position, CurrentReducedCorrelator, JacobIdeal, B, H, Hess,
CorrVal, FixedVars, RamondCorrelators, KnownCorrelatorValues,
 UnknownSet, TempW, TempVars, i;
    description "Computes all correlators determined by the pairing.
This is only useful for 3 point correlators. ";
    KnownCorrelatorValues:= { };
    UnknownSet:= { };

    RamondCorrelators:= remove(NeveuSchwartz, Correlators);

    for position to nops(RamondCorrelators) do

        CurrentReducedCorrelator:= remove(proc (item) options operator, arrow,

evalb(item[2] = Charges) end proc, RamondCorrelators[position]);


            # We can reduce correlators which contain

            # the identity of the ring (namely J),

# so we're going to throw out the J inclusions,

            # and work with the reduced correlator.

# For cases where we reduce it to a two point correlator,

            # we can find the value through the pairing.


        if nops(CurrentReducedCorrelator) = 2

              and nops(RamondCorrelators[position]) = 3 then

 # i.e. Is this a 3-point correlator that reduces to the pairing?

            FixedVars := SortOutZeros(CurrentReducedCorrelator[1, 2])[2];

            TempW:= W;

            for i to nops(Vars) do

                if not `in`(i, FixedVars) then

                    Temp W:= eval(Temp W, Vars[i] = 0)

                end if

            end do;

            JacobIdeal := convert(map(proc(a) options operator, arrow,

                      diff(TempW, a) endproc, Vars[FixedVars]), set);

            B := Basis(JacobIdeal, plex(op(Vars[FixedVars])));

            H := LinearAlgebra[Determinant](

                      VectorCalculus[Hessian](W,Vars)[FixedVars, FixedVars]);

            Hess := NormalForm(H, B, plex(op( Vars[FixedVars])));
```

```
                # now we can directly compute the pairing
            if
ListTools[DotProduct](
        DegRead(mul(CurrentReducedCorrelator[i,3], i=1..2),Vars),
Charges) != add(1-2*k, `in`(k, Charges[FixedVars])) then
                     KnownCorrelatorValues:=
`union`(KnownCorrelatorValues,{RamondCorrelators[position] = 0})
           else
               CorrVal := NormalForm(mul(
                      CurrentReducedCorrelator[i, 3], i = 1 .. 2)*
                        SectorMu(CurrentReducedCorrelator[1,2], Charges),
                            B, plex(op(Vars[FixedVars])));
                 KnownCorrelatorValues := `union`(KnownCorrelatorValues,
     {RamondCorrelators[position] = CorrVal/ Hess})
            end if;
        elif nops(RamondCorrelators[position])
                   = nops(CurrentReducedCorrelator) then
            UnknownSet :=
                 `union`(UnknownSet, {RamondCorrelators[position]})
        end if
    end do;
    UnknownSet:=
          `minus`(convert(Correlators, set), map(lhs, KnownCorrelatorValues));
    return [KnownCorrelatorValues,UnknownSet]
end proc:


ComputedFourPtCorrsOut := proc(W::polynom, Vars::list, genus::integer,
           Charges, Correlators::listlist)
```

```
local kappa, psi, DegSt, nu, istarDelta, i, CurrentCorr, LBD, node,
       b, mark, Listq, Lambda, Theta, Denominator, KnownCorrelatorValues,
       UnknownSet;
description "Finds the values of as many four-point correlators as
        currently possible, without using the composition axiom. ";
KnownCorrelatorValues:= { };
UnknownSet:= { };
Listq := convert( Charges, list);
Denominator := mul( k, `in`(k, DegRead(W,Vars)));
if type (W, `+`) and nops( Vars) < nops( W) then
    return "Not invertible singularity"
else
    if igcd(op(map(denom,Charges)))
              != igcd(op(map(denom,Charges)), Denominator) then
        Denominator := igcd(op(map(denom, Charges)))
    end if
end if;


    # As ugly as this is, it's just direct computation
    # of concave 4-point correlators.
kappa := 1 / Denominator;
psi := 1 / Denominator;
DegSt:= 1 / Denominator;
for nu to 3 do
    istarDelta[nu] := 1 / (Denominator* r[nu])
end do;
if `not`(nops(Correlators[1]) = 4) then
    return "Four point only"
else
```

```
        for CurrentCorr in Correlators do
           if `in`(0,`union`(op(map(proc (item) options operator, arrow,
                          convert(op(2, item), set) endproc, CurrentCorr)))) then
              UnknownSet:= `union`(UnknownSet, {CurrentCorr})
            else
              LBD := LBdegrees(genus, Charges, CurrentCorr);
              if AllNeg(LBD) then
                  for node to 3 do
                     b[node] := map(frac, [seq(8, i = 1 .. nops(Charges))]
                           + Listq -CurrentCorr[1, 2]-CurrentCorr[node+ 1,2])
                  end do;
        Lambda[CurrentCorr]:= simplify((map(proc (x) options operator, arrow,
 (1/2)*X^2 - (1/2)*x + 1/12 end proc, Charges) * kappa
   -add(map(proc (x) options operator, arrow, 1/12- (1/2) *x* (1-x) endproc,
   CurrentCorr[mark, 2]), mark = 1 .. 4)*psi
     + add(r[node]*map(proc (x) options operator, arrow,
       1/12- (1/2) * x * (1- x) end proc, b[node])
              * istarDelta[node], node = 1 .. 3))/ DegSt);
 KnownCorrelatorValues:= `union`(KnownCorrelatorValues,
                  {CurrentCorr = add(k,`in`(k, Lambda[CurrentCorr]))})
              else
                  UnknownSet := `union`(UnknownSet, {CurrentCorr})
              end if;
            end if;
         end do;
     end if;
    return [KnownCorrelatorValues, UnknownSet]
end proc:
```

```
CompositionAxiom := proc (W::polynom, Vars::list, genus::integer,
               Charges::list, Correlators::listlist)
    local position, RamondSector, JacobIdeal, B, H, Hess,
        PossiblePartnerMonomials,  PartnerMonomial, FixedVars,
        PartnerSector, Complements, FourPtClassCandidates,
        PreDecompositions, FourPtChoice, DecompLefts, Decomp,
        PartnerCorrelator, Summands, RamondPieces, EtaMatrix, InverseEta,
        AlphasMatrix, BetasMatrix, Equations, UnknownSet, TempW, i;
    description "Uses the composition axiom to find equations for
3-point, genus zero correlators to connect them to a computable
4-point class from the boundary. Will likely be extended
to larger correlators in future versions. ";
    FourPtClassCandidates := { };
    Equations := { };
    UnknownSet := { };
    for position to nops(Correlators) do
        if nops(Correlators[position]) = 3 and
               nops(remove(proc (item) options operator, arrow,
               item [3] = 0 end proc, Correlators[position])) = 1 then
   # We pick out three-point correlators with exactly one Ramond insertion
   # Then we look at the fixed locus so we can find the sectors that
   # will have non-zero pairing with the aforementioned Ramond sector


           RamondSector:= op(remove(proc (item) options operator, arrow,
                     item [3] = 0 end proc, Correlators[position]));
           FixedVars := SortOutZeros(RamondSector[2])[2];
           JacobIdeal := convert(map(proc (a) options operator, arrow,
                     diff(W, a) endproc, Vars), set);
           B := Basis(JacobIdeal, plex(op(Vars)));
```

```
          H := LinearAlgebra[Determinant](VectorCalculus[Hessian]
                         (W,Vars)[FixedVars, FixedVars]);
          Hess := NormalForm(H, B, plex(op(Vars)));
          PossiblePartnerMonomials :=
                  remove(proc (item) options operator, arrow,
 NotDivisibleBy(item, DegRead(RamondSector[3], Vars), nops(Vars))
end proc, map(DegRead, [op(H)], Vars));
         PossiblePartnerMonomials := map(proc (item) options operator, arrow,
 Exponentiate(item - DegRead(RamondSector[3], Vars), Vars) end proc,
 PossiblePartnerMonomials);
         PossiblePartnerMonomials:= remove(proc (item) options operator, arrow,
 evalb(NormalForm(item*RamondSector[3], B, plex(op(Vars))) = 0) end proc,
PossiblePartnerMonomials);


           PartnerMonomial := PossiblePartnerMonomials[1];
   # So I don't actually go through all the possibilities for partners,
   # I just take one. Really, it's fast enough that you could probably
   # go through all of them,  but I don't know that it wouldn't be
   #  largely redundant information.


           PartnerSector:=
     [2 * iota( map( frac, [seq(8, i = 1 .. nops(Vars))]
         - RamondSector[2]), Charges),
     map(frac, [seq(8, i= 1 .. nops(Vars))] - RamondSector[2]), PartnerMonomial];


           Complements := select(proc (item) options operator, arrow,
evalb(member(PartnerSector, item) and
nops(remove(proc (Corr) options operator, arrow,
evalb(Corr = PartnerSector) end proc, item))= 2) endproc, Correlators);
```

```
        # We find all the 3-pt correlators which contain the partner sector


            FourPtClassCandidates:= map(proc (item) options operator, arrow,
 [op(select(proc (sector) options operator, arrow,
     evalb( sector[ 3] = 0) end proc
 Correlators[position])), op(select(proc (sector) options operator, arrow,
 evalb( sector[ 3] = 0) end proc, item))] endproc, Complements);
    # We find all the possible four-point classes built from
    # the two NS insertions of the original 3-point correlator,
    # and the two NS sectors from a complementary
    # 3-point correlator, that is, a 3-point correlator with two NS insertion
    # and a Ramond sector which has a possibly non-zero pairing
    # with the Ramond sector or the first correlator.


            FourPtClassCandidates:= select(proc (item) options operator, arrow,
 IndexZeroSort(LBdegrees(0, Charges, item), 0) [ 3 ] end proc,
                CohoGoodResDimList( 0, Charges, FourPtClassCandidates));
# This filters it down to index zero 4-point classes
$ which we can explicitly compute.


            if nops(FourPtClassCandidates) = 0 then
                UnknownSet:= `union`(UnknownSet, {Correlators[position]})
            else
                for FourPtChoice in FourPtClassCandidates do
                    PreDecompositions := {{{FourPtChoice[1], FourPtChoice[2]},
                                    {FourPtChoice[3], FourPtChoice[4]}},
                    {{FourPtChoice[1], FourPtChoice[3]},
                         {FourPtChoice[2], FourPtChoice[4]}},
                     {{FourPtChoice[1], FourPtChoice[4]},
                         {FourPtChoice[2], FourPtChoice[3]}}};
           # There are three ways to decompose the 4-point class,
           # and this explicitly constructs them.


                Summands := [ ];
                Decomp := op(select(proc (item) options operator, arrow,
     evalb(item [ 1] = `minus`(convert(Correlators[position], set),
            {RamondSector})
     or item [ -1] =
      `minus`(convert( Correlators[position], set),
              {RamondSector})) end proc,
         PreDecompositions));
        # We pick out decompositions of the 4-point class
        # which use the NS sectors from the original 3-pt correlator


    DecompLefts := select(proc (item) options operator, arrow,
        evalb({op(map(proc (Corr) options operator, arrow,
              Corr[2] end proc, item))} =
        {Decomp[1,-1,2], Decomp[1,1,2], map(frac, [seq(8, k= 1 .. nops(Vars))]
          - Decomp[1,1,2] - Decomp[1,-1,2] + Charges)}) end proc, Correlators);
 # Picks out all 3-point correlators which contain
 # the pair of insertions from Decomp.
 # In particular, thistells us all the possible decorations of the edge, namely,
 # the elements from the
 # map(frac, [seq(8, k= 1 .. nops(Vars))]
 #        - Decomp[1,1,2] - Decomp[1,-1,2] + Charges) sector
 # Now that we all the possible ways the edge can be decorated,
 # we can apply the Composition axiom and just compute.
```

```
   if NeveuSchwartz(DecompLefts[1]) then
       PartnerCorrelator:= op(select(proc (item) options operator, arrow,
       evalb({op(map(
proc (Corr) options operator, arrow, Corr[2] end proc, item))}
       = {Decomp[-1,-1,2], Decomp[-1,1,2],
             map(frac, [seq( 8, k = 1 .. nops(Vars))]
       Decomp[-1,1,2]-Decomp[-1,-1,2] + Charges)}) end proc, Correlators));
       Summands:= [op(Summands),
             AValue(DecompLefts[ 1]) * AValue(PartnerCorrelator) ]
    else
                        RamondPieces:= map(proc (Corr) options operator, arrow,
 add(sector[3], `in`(sector, Corr)) endproc, DecompLefts);
                        FixedVars := SortOutZeros(RamondSector[2])[2];
                        Temp W:= W;
                        for i to nops(Vars) do
                            if not `in`(i, FixedVars) then
                                TempW:= eval(TempW, Vars[i] = 0)
                            end if
                        end do;
             JacobIdeal := convert( map(proc (a) options operator, arrow,
 diff(TempW, a) endproc, Vars[FixedVars]), set);
                        B := Basis(JacobIdeal, plex(op(Vars[FixedVars])));
                        H := LinearAlgebra[Determinant](VectorCalculus[Hessian]
             (W,Vars)[FixedVars, FixedVars]);
                        Hess := NormalForm(H, B, plex(op(Vars[FixedVars])));

                        # We compute the pairing matrix
EtaMatrix := Matrix(nops(RamondPieces), proc (i,j) options operator, arrow,
    floor((1 + ListTools[DotProduct]
      (DegRead(RamondPieces[i]*RamondPieces[j],Vars), Charges))
        /(1 +add(1-2*k, `in`(k, Charges[FixedVars]))))
         * SectorMu(RamondSector[2], Charges)
         * NormalForm(RamondPieces[i]*RamondPieces[j], B, plex(op(Vars)))
         * floor((1 + add(1-2*k,`in`(k, Charges[FixedVars])))
       / (1 + ListTools[DotProduct](
 DegRead(RamondPieces[ i]*RamondPieces[j], Vars), Charges)))
   / Hess end proc);
     AlphasMatrix := Matrix([map(AValue, DecompLefts)]);
      BetasMatrix := Vector(map(proc (object) options operator, arrow,
                A Value( op( object) ) end proc,
      map(proc (Corr) options operator, arrow,
 select(proc (item) options operator, arrow,
      evalb( {op( map(proc (Corr) options operator, arrow,
 Corr[2 .. 3] endproc, item))}
       = {[Decomp[-1,-1,2],0],[Decomp[-1,1,2],0],
           [map(frac, [seq(8, k= 1 .. nops(Vars))]
          -Decomp[-1,1,2]-Decomp[-1,-1,2] + Charges), Corr]})
              endproc, Correlators)
                end proc, RamondPieces)));
  InverseEta := MatrixInverse(EtaMatrix);
  Summands := [op( Summands), op(
                   convert(AlphasMatrix.lnverseEta.BetasMatrix, list))]
                    end if;
         Equations:= `union`(Equations,{add(summand, `in`(summand, Summands))
           = rhs(IndexZeroValuesOut(
                  W, Vars, genus, Charges, [FourPtChoice])[1,1])})
                end do;
            end if;
        else UnknownSet:= UnknownSet union {Correlators[position]};
        end if;
    end do;
    return [Equations,UnknownSet];
    gc( )
end proc:


ComputedCorrsOut := proc (W::polynom, Vars::list, NumPoints,
genus::integer, Charges, CurlyH::listlist)
    local NSCorrs, RamondCorrs, ConcaveCorrs, IndexZeroCorrs, ListlnProg,
 ListInProgB, Degs,IsCCBoolean, PairRamond, LeftoverRamond, LeftoverNS, k,
 KnownCorrelatorValues, UnknownSet, NewData, CompositionData, Eqn, TempSolns,
 OutputEqns, SolvedEqns, SimplifiedFormEqns;
    description "Returns as much data as possible on the values
 for the specified size and genus of correlator.
 For 3-point, genus-zero correlators, this will be enough to verify
the isomorphism with the appropriate Milnor ring. ";
    KnownCorrelatorValues := { };
    UnknownSet := { };
    ConcaveCorrs := [ ];
    IndexZeroCorrs := [ ];
    LeftoverNS := [ ];
    ListInProg := selectremove(NeveuSchwartz,
HomZeroDimList(genus, Charges, AlllntList(NumPoints, genus, Charges, CurlyH)));
    ListInProgB := [[ ], [ ]];
    NSCorrs := ListlnProg[1];
    RamondCorrs := ListlnProg[2];
    ListInProgB:= selectremove(HasUnit, RamondCorrs, Charges);
    if NumPoints = 3 then
```

```
        PairRamond := ListInProgB[1]
    else
        PairRamond := [ ]
    end if;
    LeftoverRamond := ListInProgB[2];
    Degs := map[3](LBdegrees, genus, Charges, NSCorrs);
    IsCCBoolean := map(AIlNeg, Degs);
    for k to nops(NSCorrs) do
        if IsCCBoolean[k] then
            ConcaveCorrs:= [op(ConcaveCorrs), NSCorrs[k]]
        elif IndexZeroTest(Degs[k], genus) then
            IndexZeroCorrs := [op(IndexZeroCorrs), NSCorrs[k]]
        elif not HasUnit(NSCorrs[k],Charges) then
            LeftoverNS:= [op(LeftoverNS), NSCorrs[k]]
        end if
    end do;
    if 0 < nops(PairRamond) then
        NewData := RamPairingOut(W, Vars, Charges, PairRamond);
        KnownCorrelatorValues := `union`(KnownCorrelatorValues, NewData[1]);
        UnknownSet := `union`(UnknownSet, NewData[2])
    end if;
    UnknownSet := `union`(UnknownSet, convert(LeftoverRamond, set));
    if NumPoints = 3 and 0 < nops( ConcaveCorrs) then
        KnownCorrelatorValues := `union`(KnownCorrelatorValues,
map( proc (item) options operator, arrow, item = 1
         end proc, convert(ConcaveCorrs, set))
    elif NumPoints = 4 and 0 < nops(ConcaveCorrs) then
        NewData := ComputedFourPtCorrsOut(W, Vars, genus, Charges, ConcaveCorrs);
        KnownCorrelatorValues:= `union`(KnownCorrelatorValues, NewData[ 1]);
        UnknownSet:= `union`(UnknownSet, NewData[2])
    else UnknownSet:= `union`(UnknownSet, convert(ConcaveCorrs, set))
    end if;
    if 0 < nops(IndexZeroCorrs) then
        NewData := IndexZeroValuesOut(W, Vars, genus, Charges, IndexZeroCorrs);
        KnownCorrelatorValues:= `union`(KnownCorrelatorValues, NewData[1]);
        UnknownSet:= `union`(UnknownSet, NewData[2])
    end if;
    if 0 < nops(LeftoverNS) then
        UnknownSet = `union`(UnknownSet,convert(LeftoverNS, set))
    end if;
    if `not`(genus = 0 and NumPoints = 3) then
        return [KnownCorrelatorValues, UnknownSet]
    else
        if nops( UnknownSet) = 0 then
            CompositionData:= [{ }, { }];
        else
            CompositionData := CompositionAxiom(W, Vars, 0,
                                                Charges, [op(UnknownSet)]);
        end if;
        SolvedEqns := { };
        OutputEqns := { };
        SimplifiedFormEqns := { };
        for Eqn in simplify(eval(`minus`(CompositionData[1], SolvedEqns),
                map(proc (item) options operator, arrow,
 AValue( lhs( item)) = rhs( item) end proc, KnownCorrelatorValues))) do
            if type(op(-1, sort(lhs(Eqn))), rational) then
                SimplifiedFormEqns:= `union`(SimplifiedFormEqns,
{lhs(Eqn) - op(-1, sort(lhs( Eqn) )) = rhs(Eqn) - op(-1, sort(lhs(Eqn)))})
```

```
            else
                SimplifiedFormEqns:= ‘union‘(SimplifiedFormEqns, {Eqn})
            end if
        end do;
        return [KnownCorrelatorValues,
simplify(‘union‘(SimplifiedFormEqns, OutputEqns)), CompositionData[2]]
    end if
end proc:


> ChooseWithRepeats:= proc (n::integer, k::integer)
    local i, j, L, LP;
    description "Chooses all k-combos with repeats from 1 to n.";
    option remember;
    LP:= [ ];
    if 0 < k then
        L := Choose WithRepeats(n, k-l);
        if L = [ ] then
            for j to n do
                LP:= [op(LP), [j]]
            end do;
        else
            for i to nops(L) do
                for j from L[i, -1] to n do
                    LP := [op( LP), [op( L[ i]), j]]
                end do;
            end do;
        end if;
    end if;
    return LP
```

```
end proc:


cHat := proc (g::list, Charges::list)

    return add(1-2*Charges[i], i = SortOutZeros(g)[2])

endproc:


sPlus := proc (g::list, Charges::list)

    return cHat( [seq(0, i = 1 .. nops(Charges))], Charges) - cHat(g, Charges)

end proc:


sMinus:= proc (g::list, Charges::list)

    return add(2*g[i]-1, i = SortOutZeros(g)[1])

end proc:


kaufmannS:= proc (g::list, Charges::list)

    return (1/2) * sPlus(g, Charges) + (1/2) * sMinus(g, Charges)

end proc:


kaufmannSbar := proc (g::list, Charges::list)

    return (1 / 2) * sPlus(g, Charges) - (1/2) * sMinus(g, Charges)

end proc:


Rshift := proc (g::list, Charges::list)

    return (1/2) * cHat(g, Charges)

end proc:


ngamma := proc (g:: list)

    description "Ngamma is the number of variable fixed by a group element,

hence the number of zeros in the given list";
```

```
    return nops(SortOutZeros(g)[2])
end proc:


iota := proc (g::list, Charges::list)
    return kaufmannS(g, Charges) + Rshift(g,Charges)
end proc:


SectorMu := proc (g::list, Charges::list)
    local fixedVars;
    description "The dimension of the Milnor ring of
the group element's fixed locus.";
    FixedVars := SortOutZeros(g)[2];
    if nops(fixedVars) = 0 then
        return 0
    else
        return mul(1/Charges[i] -1, i = FixedVars)
    end if
end proc:


LBdegrees := proc (genus:: integer, Charges::list, Correlator::list)
    local SectorList;
    description "Computes the line bundle degrees of a given correlator.";
    SectorList:= map(proc (item) options operator, arrow,
op(2, item) end proc, Correlator);
    return [seq(Charges[i] * (2 * genus-2 + nops(Correlator))
         - add(g[i], g = SectorList), i = 1 .. nops(Charges))]
endproc:


TestDegs := proc (K::list)
```

```
    description "Returns true exactly when fed a list of only integers.";
    return andmap( type, K, integer)
end proc:


MakeCorrelators := proc (CurlyH::list, NumPoints::integer)
    local preList;
    description "Makes a list of all possible correlators of the specified size
from the given basis CurlyH.
Note that most will be zero, and will be filtered out  later.";
    preList := ChooseWithRepeats(nops(CurlyH), NumPoints);
    return map(proc (Corr) options operator, arrow,
sort(map(proc (item) options operator, arrow, op(item, CurlyH) endproc, Corr),
        proc (a, b) options operator, arrow,
   evalb( a[1] < b[1] or a[1] = b[1] and degree(a[3]) < degree(b[3]))
end proc) end proc, preList)
end proc:


AllIntList := proc (NumPoints, genus, Charges, CurlyH)
    local RawList, Degs, IsIntBoolean, AllIntDegreesList, k;
    description "Produces a list of all correlators which have
only integer line bundle degrees.
Only these correlators are potentially non-zero.";
    option remember;
    RawList := MakeCorrelators(CurlyH, NumPoints);
    AllIntDegreesList := [ ];
    Degs := map[3](LBdegrees, genus, Charges, RawList);
    IsIntBoolean := map(TestDegs, Degs);
    for k to nops(Degs) do
        if IsIntBoolean[k] then
```

```
            AllIntDegreesList:= [op(AllIntDegreesList), RawList[k]]
        end if
    end do;
    return AllIntDegreesList
end proc:

AllNeg := proc (K:: list)
    description "Returns true exactly if all items in the list
are less than zero.";
    return andmap(proc (item) options operator, arrow,
 evalb(item < 0) end proc, K)
end proc:

IndexZeroTest:= proc (LBDsOfCorrelator:: list, genus)
    description "IF fed an Neveu-Schwartz correlator,
will tell you if it's Index Zero.";
    return evalb(add( i, i = LBDsOfCorrelator)
                            = (genus-1)*nops(LBDsOfCorrelator))
end proc:

Dim := proc (genus:: integer, Charges, L::list)
    local SectorList;
    description "Algebraic Cohomological dim of correlator";
    SectorList := map(proc (item) options operator, arrow,
 op(2, item) endproc, L);
    return (genus-1) * cHat([seq(0, i = 1 .. nops( Charges))], Charges)
      + add(kaufmannS(g, Charges) + (1/2)*cHat(g, Charges), g = SectorList)
end proc:
```

```
NeveuSchwartz := proc (correlator)
    local Monomials;
    description "Returns true exactly if the given correlator
 has only Neveu-Schwartz sectors.";
    Monomials := convert(map(proc (item) options operator, arrow,
 op(3, item) endproc, correlator), set);
    return evalb(Monomials = {0})
endproc:


HasUnit := proc (correlator::list, unit:: list)
    local SectorList;
    description "Returns true exactly if the correlator
contains the unit of the ring.";
    SectorList := map(proc (item) options operator, arrow,
                                     op(2, item) end proc, correlator);
    return member(unit, SectorList)
end proc:


SortedCorrelators:= proc (NumPoints, genus, Charges, CurlyH)
    local NSCorrs, RamondCorrs, ConcaveCorrs, IndexZeroCorrs,
ListInProg, ListInProgB, Degs, IsCCBoolean, PairRamond,
LeftoverRamond, ReducNS, LeftoverNS, k;
    description "Sorts all potentially non-zero correlators of the
 given size and genus by how (and if) the can be computed directly.";
    ConcaveCorrs := [ ];
    IndexZeroCorrs := [ ];
    ReducNS := [ ];
    LeftoverNS := [ ];
    ListInProg := selectremove(NeveuSchwartz,
         HomZeroDimList(genus, Charges,
                  AllIntList(NumPoints, genus, Charges, CurlyH)));
    ListInProgB := [[ ], [ ]];
    NSCorrs := ListInProg[1];
    RamondCorrs := ListInProg[2];
    ListInProgB := selectremove(HasUnit, RamondCorrs, Charges);
    if NumPoints = 3 then
        PairRamond := ListInProgB[1]
    else
        PairRamond := [ ]
    end if;
    LeftoverRamond := ListInProgB[2];
    Degs := map[3](LBdegrees, genus, Charges, NSCorrs);
    IsCCBoolean := map(AllNeg, Degs);
    for k to nops(NSCorrs) do
        if IsCCBoolean[k] then
            ConcaveCorrs := [op( ConcaveCorrs) , NSCorrs[k]]
        elif IndexZeroTest(Degs[k], genus) then
            IndexZeroCorrs:= [op(IndexZeroCorrs) ,NSCorrs[k]]
        elif not HasUnit(NSCorrs[k], Charges) then
            LeftoverNS := [op(LeftoverNS), NSCorrs[k]]
        end if
    end do;
    print( ‘Ramond correlators determined by pairing ‘,op(PairRamond));
    print( ‘Other Ramond correlators ‘* op(LeftoverRamond));
    print( ‘Concave (NS) correlators ‘ op(Con cave Corrs)) ;
    print( ‘Index Zero (NS) correlators ‘,op(IndexZeroCorrs));
    print( ‘Other Neveu-Schwartz correlators ‘* op(LeftoverNS))
end proc:
```

```
HomZeroDimList:= proc (genus:: integer, Charges, K:: list)
    local i, trips;
    description "Find those correlators that have cohoDim=3 g-3+k, or HomDim=0.
 These are the ones that have a potentially non-zero
        correlator without descendants. ";
    trips:= [ ];
    for i to nops(K) do
        if Dim (genus, Charges, K[i]) = 3 * genus- 3 + nops(K[i]) then
            trips := [op( trips), K[i]]
        end if
    end do;
    return trips
end proc:

CohoZeroDimList:= proc (genus:: integer, Charges, K::list)
    local i, trips;
    description "Find those correlators that have cohoDim = 0,
 or HomDim = 3g - 3 + k. ";
    trips:= [ ];
    for i to nops(K) do
        if Dim (genus, Charges, K[i]) = 0 then
            trips:= [op(trips), K[i]]
        end if
    end do;
    return trips
end proc:

CohoGoodResDimList := proc (genus:: integer, Charges, K::list)
```

```
    local i, trips;
    description "Find those correlators that have cohoDim = 1 less than expected,
  or HomDim = 3g-3+k-1. These are the ones that restrict nicely to the boundary";
    trips:= [ ];
    for i to nops(K) do
        if Dim (genus, Charges, K[i]) = 3*genus-4 + nops(K[ i]) then
            trips:= [op(trips), K[i]]
        end if
    end do;
    return trips
end proc:


ZeroRestrictList := proc (NumPoints::integer, genus:: integer,
                                        Charges::list, CurlyH::list)
   local i, LBD, ZeroResList, CcBoolean, IndZeroBoolean;
   description "Lists correlators with coho dim=0, integral line bundle degrees,
  and indicates if concave. ";
    ZeroResList := CohoZeroDimList(genus, Charges,
                             AllIntList(NumPoints, genus, Charges, CurlyH));
    if evalb(nops(ZeroResList) = 0) then
        print( 'none found ')
    end if;
    for i to nops(ZeroResList) do
        CcBoolean := false;
        IndZeroBoolean := false;
        LBD := LBdegrees(genus, Charges, ZeroResList[i]);
        if NeveuSchwartz (ZeroResList[i]) and AllNeg(LBD) then
            CcBoolean:= true
        end if;
        if IndexZeroTest(LBD, genus) then
            IndZeroBoolean := true
        end if;
        printf("\n Correlator \%a concave \%a index zero \%a LB degrees \%a",
                          ZeroResList[ i], CcBoolean, IndZeroBoolean, LBD)
    end do
end proc:


GoodRestrictList := proc (NumPoints::integer, genus:: integer,
                                        Charges::list, CurlyH::list)
    local i, LBD, GoodResList, CcBoolean, IndZeroBoolean;
    description "Lists correlators with coho dim = 3g-3+n-l,
 integral line bundle degrees, and indicates if concave. ";
    GoodResList := CohoGoodResDimList(genus, Charges,
                          AllIntList(NumPoints, genus, Charges, CurlyH));
    if evalb(nops(GoodResList) = 0) then
        print( 'none found ')
    end if;
    for i to nops(GoodResList) do
        CcBoolean := false;
        IndZeroBoolean := false;
        LBD := LBdegrees(genus, Charges, GoodResList[i]);
        if NeveuSchwartz(GoodResList[i]) and AllNeg(LBD) then
            CcBoolean := true
        end if;
        if IndexZeroTest(LBD, genus) then
            IndZeroBoolean := true
        end if;
        printf("\n Correlator \%a concave: \%a index zero: \%a LB degrees \%a",
                        GoodResList[i], CcBoolean, IndZeroBoolean, LBD)
    end do
end proc:

> GetCoordinates:= proc (BasisVectors, GroupElt:: list)
    local Modulus, BasisMatrix, ModGroupElt, Solution;
    description "Finds how to express the given group element
 as a linear combination of the specified ordered list of generators. ";
    Modulus := ilcm(op(map(proc (item) options operator, arrow,
 ilcm(op(map(denom, item))) endproc, BasisVectors)));
    ModGroupElt := Modulus * GroupElt;
    BasisMatrix := Matrix([op(map(proc (item) options operator, arrow,
 Modulus*item end proc, BasisVectors}), ModGroupElt]);
    BasisMatrix := LinearAlgebra[Modular]
        [Mod](Modulus, Transpose(BasisMatrix), integer[ ]);
    try Solution := Linear Algebra[Modular]
        [LinearSolve](Modulus, BasisMatrix, 1, inplace = false)
    catch "matrix is singular":
        print("Given group element is not generated by
the input set of basis vectors");
        return FAIL
    end try;
    return op (convert(Solution, list)[1 .. nops(BasisVectors)])
end proc:

CompactOutputFormatting := proc (BasisVectors, Sector:: list)
    description "Returns the compact notation version of the specified sector.
 Group element is given as a tuple with respect to
the ordered set of generators given.";
```

```
    if Sector [3] = 0 then

        return e[GetCoordinates(BasisVectors, Sector [2])]

    elif Sector [3] = 1 then

        return '@' * e[GetCoordinates(BasisVectors, Sector [2])]

# This is for when we need the monomial 1 from a nontrivial Milnor ring.

    else

        return Sector[3] * e[GetCoordinates(BasisVectors, Sector[2])]

    end if

end proc:


BigOutputConversion := proc (TargetObject, BasisVectors, ABasis::listlist)

    local SectorConversions;

    description "Returns the target object with all A-model sectors

 expressed in the more compact notation.

Note this is specific to a given ordered set  of generators for the group.";

    SectorConversions := map(proc (item) options operator, arrow,

 item = CompactOutputFormatting(BasisVectors, item) end proc, ABasis);

    return eval(TargetObject, SectorConversions)

end proc:


BMultiply:= proc (FirstTerm, SecondTerm, W:: polynom, Vars:: list)

    local ReducFirstMonomial, ReducSecondMonomial, Gamma,

 FixedVars1, FixedVars2, Fix1and2, Hess1and2, TempW, JacobIdeal,

 B, H, Hess, GroupProd, GroupProdZeros, i, ProductHess,

GroupProdZerosList, FinalMonomial;

    description "Performs multiplication of elements for an orbifold B-model. ";

    if FirstTerm = 0 or SecondTerm = 0 then

        return 0

    end if;
```

```
   FixedVars1 := SortOutZeros(FirstTerm[2])[2];
   TempW:= W;
   for i to nops(Vars) do
       if not `in`(i, FixedVars1) then
           TempW:= eval(TempW, Vars[i] = 0)
       end if
   end do;
   JacobIdeal := convert(map(proc (a) options operator, arrow,
diff(TempW, a) endproc, Vars[FixedVars1]), set);
   B := Basis(JacobIdeal, plex(op(Vars)));
   H := LinearAlgebra[Determinant](VectorCalculus[Hessian]
                                  (W, Vars)[FixedVars1, FixedVars1]);
   Hess := NormalForm(H, B, plex(op(Vars)));
   if FirstTerm[3] = 0 then
       ReducFirstMonomial := 1
   else
       ReducFirstMonomial := NormalForm(FirstTerm[3], B, plex(op(Vars)))
   end if;
   FixedVars1 := convert(FixedVars1, set);
   FixedVars2 := SortOutZeros(SecondTerm[2])[2];
   TempW:= W;
   for i to nops(Vars) do
       if not `in`(i, FixedVars2) then
           TempW:= eval(TempW, Vars[i] = 0)
       end if
   end do;
   JacobIdeal := convert(map(proc (a) options operator,
arrow, diff(TempW, a) endproc, Vars[FixedVars2]), set);
   B := Basis(JacobIdeal, plex(op(Vars)));
   H := LinearAlgebra[Determinant](VectorCalculus[Hessian]
                                  (W, Vars)[FixedVars2, FixedVars2]);
   Hess := NormalForm(H, B, plex(op(Vars)));
   if SecondTerm[3] = 0 then
       ReducSecondMonomial := 1
   else
       ReducSecondMonomial := NormalForm(SecondTerm[3], B, plex(op(Vars)))
   end if;
   FixedVars2:= convert(FixedVars2, set);
   Fix1and2 := convert(`intersect`(FixedVars1, FixedVars2), list);
   TempW:= W;
   for i to nops(Vars) do
       if not `in`(i, Fix1and2) then
           TempW:= eval(TempW, Vars[i] = 0)
       end if
   end do;
   JacobIdeal := convert(map(proc (a) options operator, arrow,
diff(TempW, a) end proc, Vars[Fix1and2]), set);
   B := Basis(JacobIdeal, plex(op(Vars)));
   H := LinearAlgebra[Determinant](VectorCalculus[Hessian]
                                            (W, Vars)[Fix1and2, Fix1and2]);
   Hess1and2 := NormalForm(H, B, plex(op(Vars)));
   GroupProdZerosList := SortOutZeros(map(frac, FirstTerm[2]
                                              + SecondTerm[2]))[2];
   GroupProdZeros := convert( GroupProdZerosList, set);
   if nops(`union`(`union`(GroupProdZeros, FixedVars1), FixedVars2))
                                    < nops(Vars) then
       return 0
   else
```

```
        TempW:= W;
        for i to nops(Vars) do
            if not `in`(i, GroupProdZerosList) then
                TempW:= eval(TempW, Vars[i] = 0)
            end if
        end do;
        JacobIdeal := convert( map(proc (a) options operator, arrow,
 diff( TempW, a) end proc, Vars) , set);
        B := Basis(JacobIdeal, plex( op( Vars)));
        H := LinearAlgebra[Determinant](VectorCalculus[Hessian]
(TempW, Vars)[GroupProdZerosList, GroupProdZerosList]);
        ProductHess := NormalForm(H, B, plex(op(Vars)))
    end if;
    if nops(GroupProdZerosList) = 0 and nops(Fix1and2) = 0 then
        Gamma := ProductHess / Hess1and2
    elif nops(GroupProdZerosList) = 0 then
        Gamma := ProductHess*nops(Fix1and2)/Hess1and2
    elif nops( Fix 1 and2) = 0 then
        Gamma := ProductHess / (Hess1and2 * nops( GroupProdZerosList))
    else Gamma := ProductHess*nops(Fix1and2)/(Hess1and2*nops(GroupProdZerosList))
    end if;
    FinalMonomial := Gamma*ReducFirstMonomial*ReducSecondMonomial;
    for i to nops(Vars) do
        if not `in`(i, GroupProdZerosList) then
            FinalMonomial := eval(FinalMonomial, Vars[i] = 0)
        end if
    end do;
    if NormalForm(FinalMonomial, B, plex( op( Vars))) = 0 then
        return 0
    else
        return [FirstTerm[1] + SecondTerm[1],
           map(frac, FirstTerm[2] + SecondTerm[2]),
             NormalForm(FinalMonomial, B, plex(op( ars)))]
    end if
end proc:

MultiplicationTable := proc (W:: polynom, Vars::list, BBasis::listlist)
    local m, MultTable, OrderedBBasis;
    description "Returns a matrix which is
a multiplication table for an orbifold B-model.";
    m := nops(BBasis);
    OrderedBBasis:= sort(BBasis, proc (a, b) options operator, arrow,
 evalb(a[1] < b[1] or a[1] = b[1] and degree(a[3]) < degree(b[3])) end proc);
    MultTable := Matrix(m, proc (i,j) options operator, arrow,
 BMultiply(OrderedBBasis[i], OrderedBBasis[j], W, Vars) end proc);
    return MultTable
end proc:

BPairingMatrix := proc (W:: polynom, Vars::list, BBasis::listlist)
   local m, MultTable, OrderedBBasis;
   description "Returns a matrix displaying the pairing
                for the Milnor ring given.";
   m := nops(BBasis);
   OrderedBBasis := sort(BBasis, proc (a, b) options operator, arrow,
 evalb( a[1] < b[1] or a[1] = b[1] and degree( a[3]) < degree( b[3])) end proc);
    MultTable := Matrix(m, proc (i,j) options operator, arrow,
 BMultiply(OrderedBBasis[i], OrderedBBasis[j], W, Vars) end proc);
    return map(proc (item) options operator, arrow,
 floor (item [1]/ OrderedBBasis[-1, 1]) * item [-1]
        / Exponentiate(DegRead(item [-1],Vars), Vars) end proc, MultTable)
end proc:


BCorrs := proc (W:: polynom, Vars::list, BBasis::listlist)
    local m, NewProduct, CorrsSet, CorrsProducts, OrderedBBasis,
            i, j, MaxDegree, OutputSet;
    description "Returns the correlators for the orbifold B-model
with the given basis. This allows the construction of all possible isomorphisms
 from the B-model  to the A-model using the IsomorphismFinder command. ";
    CorrsSet:= { };
    m := nops(BBasis);
    OrderedBBasis := sort(BBasis, proc (a, b) options operator, arrow,
 evalb( a[1] < b[1] or a[1] = b[1] and degree(a[3]) < degree(b[3])) end proc);
    MaxDegree := OrderedBBasis[-1,1];
    CorrsSet:= select(proc (item) options operator, arrow,
 evalb( add(k[ 1], `in` (k, item)) = MaxDegree) end proc,
        MakeCorrelators(BBasis, 3));
    CorrsProducts:= map(proc (item) options operator, arrow,
       BMultiply(BMultiply(item[1], item[2], W, Vars), item [3], W, Vars)[3]
          end proc, CorrsSet);
    OutputSet:= {seq(CorrsSet[i] = CorrsProducts[i]
        / Exponentiate(DegRead(CorrsProducts[i], Vars),Vars),
             i = 1 .. nops(CorrsSet))};
    return OutputSet
end proc:


> ConvertExpressions:= proc (Correlator::listlist,
                                 ABasis::listlist, BBasis::listlist)
    local OrderedABasis, OrderedBBasis, ConversionMatrix, listPlace,
CorrPlace, CorrCoeffs, dim, BVec, BTerms, BuildCorrLists, CurrentCorrList,
term, TermMultiset, sectorFactor, i, NormOrderedADegrees, NormOrderedBDegrees;
    description "Converts A-model correlators into B-model correlators,
where BtoA_i,j are the coefficients of a linear map
from the B basis to the A basis.
Note this forces the map to respect the grading,
and makes it easy to see that you get isomorphisms. ";
    dim := nops(ABasis);
    OrderedABasis := sort(ABasis, proc (a, b) options operator, arrow,
 evalb(a[1] < b[1] or a[1] = b[1] and degree(a[3]) < degree(b[3])) end proc);
    OrderedBBasis := sort(BBasis, proc (a, b) options operator, arrow,
 evalb(a[1] < b[1] or a[1] = b[1] and degree(a[3]) < degree(b[3])) end proc);
    NormOrderedADegrees := map(proc (item) options operator, arrow,
 item[1] endproc, OrderedABasis);
    NormOrderedBDegrees := map(proc (item) options operator, arrow,
 2 * item[1] end proc, OrderedBBasis);
    if not NormOrderedADegrees = NormOrderedBDegrees then
        print("The given bases do not have compatible gradings.");
        return NULL
    end if;
    ConversionMatrix := Matrix(dim, proc (i,j) options operator, arrow,
 BtoA [i, j] * floor ((1 + OrderedABasis[i,1]) / (1 + OrderedABasis[j, 1]) )
        * floor((1 + OrderedABasis[j, 1])/(1 + OrderedABasis[j,1])) endproc);
    CorrCoeffs := [];
    BVec := Vector[row]([seq(V[i], i = 1 .. dim) ]);
    for listPlace to dim do
        for CorrPlace to nops(Correlator) do
            if Correlator[CorrPlace] = OrderedABasis[listPlace] then
```

```
                CorrCoeffs:= [op(CorrCoeffs),
                      BVec.ConversionMatrix.UnitVector(listPlace, dim)]
            end if
        end do
    end do;
    BTerms := expand(mul(i ‘in‘(i, CorrCoeffs)));
    if type(BTerms,‘+‘) then
        BTerms := map(proc (item) options operator, arrow,
item/Exponentiate(DegRead(item, [seq(V[i], i = 1 .. dim)]),
[seq(V[i], i = 1 .. dim)]) end proc, [op(BTerms)], [seq(V[i], i = 1 .. dim) ])
    else BTerms := [BTerms/ Exponentiate(
                              DegRead(BTerms, [seq(V[i], i = 1 .. dim)]),
                                [seq(V[i], i = 1 .. dim)])]
    end if;
    BuildCorrLists:= [ ];
    for term in BTerms do
    TermMultiset := convert(term, multiset);
    CurrentCorrList:= [ ];
    for sectorFactor in TermMultiset do
        if not type(sectorFactor[1], rational) then
            for i to sectorFactor[2] do
                CurrentCorrList:= [op(CurrentCorrList), op( 1, sectorFactor[1])]
            end do
        end if
    end do;
    BuildCorrLists := [op(BuildCorrLists), sort( CurrentCorrList)]
    end do;
    BTerms := map(proc (item) options operator, arrow,
eval(item, BtoA[1, 1] = 1) end proc,BTerms);
```

```
    return add(i `in`(i,seq(BTerms[i] * BValue(OrderedBBasis[BuildCorrLists[i]]),
                              i = 1 .. nops(BTerms))))
end proc:


FindSubsTargets := proc (Eqn::equation, ABasis::listlist, BBasis::listlist)
    local Parts, SubsEqns;
    description "Finds all occurances of AValue() in an equation
and prepares an equation for substituting in the relevant B-model correlator.";
    Parts:= `union`( `union`(convert(select(proc (item) options operator, arrow,
 op(0, item) = AValue end proc,
        map(op, [op( Ihs(Eqn))])), set),
convert(select(proc (item) options operator, arrow,
        op(0, item) = AValue end proc, [op(lhs(Eqn))]), set)),
        convert(select(proc (item) options operator, arrow, op(0, item)
        = AValue end proc, map(op, map(op, [op(lhs(Eqn))]))), set));
    SubsEqns:= map(proc (item) options operator, arrow,
 item = ConvertExpressions(op(item) , ABasis, BBasis) endproc, Parts);
    return SubsEqns
end proc:


IsomorphismFinder := proc (CompCorrsOutput::list,
ABasis::listlist, BBasis::listlist, W:: polynom, Vars::list)
    local KnownCorrs, CompositionSubs, CompositionCorrs, TotalEqns,
 BValues, Parts, Divided, BtoACoeffs, k;
    description "Given two bases, we first check that the gradings match,
 and then find all possible linear maps which respect the grading
and respect the A-model multiplication.
 Note that multi-dimensional Ramond sectors give degrees of freedom,
 and so there may be many solutions given.
 BtoA_i,j is the projection of the image of the ith Bmodel basis element,
 ordered by degree, onto the jth A-model basis element, again ordered by degree.
 Zero can occur as a value for BtoAJ,j,
 but only in multi-dimensional Ramond sectors. ";
    if not nops(ABasis) = nops(BBasis) then
        print("Dimensions do not match");
        return NULL
    end if;
    KnownCorrs := map(proc (item) options operator, arrow,
 ConvertExpressions(lhs(item), ABasis, BBasis) = rhs(item) end proc,
         CompCorrsOutput[1]);
    CompositionSubs := map(proc (item) options operator, arrow,
 op(FindSubsTargets(item, ABasis, BBasis)) end proc, CompCorrsOutput[2]);
    BValues:= map(proc (item) options operator, arrow,
 BValue(lhs(item)) = rhs(item) end proc, BCorrs(W, Vars, BBasis));
    CompositionCorrs := subs(CompositionSubs, CompCorrsOutput[2]);
    TotalEqns:= simplify(eval(‘union‘(KnownCorrs, CompositionCorrs), BValues));
    BtoACoeffs:= { };
    Parts:= map(op, TotalEqns);
    for k to 3 do
        Parts := remove(type, Parts, rational);
        Divided := selectremove(proc (item) options operator, arrow,
 evalb(op(0, item) = BtoA) endproc, Parts);
        BtoACoeffs:= ‘union‘(BtoACoeffs, Divided[1]);
        Parts:= map(op, Divided[2])
    end do;
    BtoACoeffs:= convert(BtoACoeffs, list);
    return solve(TotalEqns, BtoACoeffs)
end proc:
```

```
MinimalGenSet := proc (Group::listlist)
    local CompMins, position, PreBasisList, SoFar, TriedGens,
 SpanOfTriedGens, Excess;
    description "Finds a minimal generating set for the group given.
 If the input is not a group, it will return a minimal generating set
 for the group generated  by the input data. ";
    CompMins:= { };
    if nops( Group) = 1 then
        return convert( Group, set)
    end if;
    for position to nops( Group[ 1 ]) do
       # CompMins is for component minimums.
       # We're going to collect a set of group elements by taking one
       # with the minimum phase in a given position,
       #  as  position sweeps through all the possibilites
        if not CompMins = convert(Group, set) then
            CompMins :=
             `union`(CompMins, {sort(remove(proc (item) options operator, arrow,
                               `in`(item, CompMins) endproc, Group),
 proc (a, b) options operator, arrow,
 evalb(op(position, a) < op(position, b)  and 0 < op(position, a)
               or 0 < op(position, a)  and op(position, b) = 0)
                 or evalb(op(position, a) = op(position, b)
                   and add(i `in`(i,a)) < add(j,`in`(j, b))) end proc)[1]})
        end if
    end do;
    PreBasisList := sort(convert(CompMins, list), proc (a, b)
       options operator, arrow,
            max(map(denom, b)) < max(map(denom, a)) endproc);
    TriedGens := { };
    SpanOfJriedGens:= {[seq(0, i = 1 .. nops( Group[1]))]};
    while nops(SpanOfJriedGens) < nops(Group) and 0 < nops(PreBasisList) do
        TriedGens:= `union`(Tried Gens, {PreBasisList[1]});
        SpanOfTriedGens:= `union`(map(proc (item) options operator, arrow,
 seq(map(frac, k*PreBasisList[1] + item),
            k= 0 .. ilcm(op(map(denom,PreBasisList[1]))) - 1) end proc,
                  SpanOfTriedGens));
        PreBasisList := remove(proc (item) options operator, arrow,
 member(item, SpanOfTriedGens) end proc, PreBasisList)
    end do;
    if `not`(SpanOfTriedGens = convert(Group, set)) then
        print("The input to MinimalGenSet is not an additive group mod 1");
        Excess:= `minus`(convert( Group, set), SpanOfTriedGens);
        if 0 < nops(Excess) then
            TriedGens := MakeGroup(convert( union`(Excess, TriedGens), list), 1,
                               [seq(x[i], i = 1 .. nops(Group[1]))])[2]
        end if
    end if;
    return TriedGens
end proc:


MakeGroup := proc (Generators::list, W::polynom, Vars::list)
    local Subgps, RelatorsLeft, SubGpPlace, CombinedlnProg,
            BigGroup, ResultGroup;
    description "Returns the group with the given generators,
and obeying the relations expressed by setting each term of W to be the identity,
 along with a minimal generating set for this group.
```

```
Setting W = 1 imposes no relations beyond being additive mod 1,
and W=0 gives the trivial group. ";
   if type(W, ‘+‘) then
       RelatorsLeft:= map(DegRead, {op(W)},Vars)
   else
       RelatorsLeft:= [DegRead(W, Vars)]
   end if;
   Subgps := map(proc (item) options operator, arrow,
{seq( map( frac, k* item), k = 1 .. ilcm(op(convert(map(denom, item), list))))}
         endproc, Generators);
   while 1 < nops(Subgps) do
       CombinedlnProg:= {seq(seq(i+j, ‘in‘(i, Subgps[1])), ‘in‘(j, Subgps[2]))};
       Subgps := [CombinedlnProg, op(Subgps[3 .. -1])]
   end do;
   BigGroup := op(Subgps);
   while 0 < nops(RelatorsLeft) do
       BigGroup := remove(proc (item) options operator, arrow,
           not type(ListTools[DotProduct](convert(item,list),
                        RelatorsLeft[1]), integer) end proc, BigGroup);
       RelatorsLeft := RelatorsLeft[2 .. -1]
   end do;
   if nops(Big Group) = 0 then
       Big Group := [[seq(0, c = 1 .. nops(Vars))]]
   end if;
   ResultGroup := MakeUnique(convert(map(convert,
     map(proc (item) options operator, arrow,
         map(proc (number) options operator, arrow,
            frac(8 + number) end proc, item) end proc, Big Group),list),list));
   gc( );
```

```
    return ResultGroup, MinimalGenSet(ResultGroup)
end proc:


Preliminary := proc (W::polynorn, Vars::list, Cheat::string := "Off")
    local SymGpData, q, ReducW;
    description "This will check to see if W has
 an isolated quasihomogeneous singularity at the origin
 with uniquely determined weights for the variables,
and if it is irreducible";
    if nops(W) < nops(Vars) or type(W, `*`) and 1 < nops(Vars) then
        print("The polynomial does not have
          uniquely determined weights for the given list of variables");
        return [false]
    end if;
    if Cheat = "Off" then
        if not IsolatedTest(W, Vars) then
            print("The polynomial does not have
              an isolated singularity at the origin");
            return [false]
        end if;
    else
        print("Cheat mode ON -- Beware for Non-Isolated-ness!");
    end if;
    SymGpData := SymGroup( W, Vars);
    q := SymGpData[1];
    if not convert(convert(map(proc (item) options operator, arrow,
 0 < item and item < 1 end proc, q), list), 'and') then
        print("The weights do not all lie between 0 and 1.");
        return [false]
    end if;
    gc( );
    ReducW:= Reducibility(W, Vars);
    if ReducW[1] then
        return [true, op(SymGpData)]
    else
        return [op(ReducW), op(SymGpData)]
    end if
end proc:

> LGModels:= proc (W::polynom, Vars::list, ASectorsGroup::list:= [[ ],1,Vars],
 BSectorsGroup::list:= [[ ],1,Vars], AInvarianceGroup::list := [[ ],1,Vars],
 BInvarianceGroup::list:= [[ ],1,Vars], Cheat:: string := "Off")
    local q, Listq, MaxGGenerators, mu, B, cHat, dim, PrelimData, JacobIdeal,
 NearBasis, H, Hess, MaxG, ASectorsG, AInvarianceG, AGenerators,
 AFixedLociBases, BSectorsG, BInvarianceG, BGenerators, BFixedLociBases,
 CurlyH, OrbifoldB, OrderOfG, OrderOfAInvar, OrderOfASectors, PossibleBGen,
 OrderOfJ,  BSideMaxGroup, BGenMax, RejectReducible;
    description "Computes the orbifold A- and B- models for a given singularity.
It first checks if the polynomial is has an isolated
quasihomogeneous singularity at the origin,
checks if it is irreducible,
and procedes to find the symmetry group,
 the charges, and bases for the A-model and B-model rings.";
    dim := nops(Vars);
    PrelimData := Preliminary(W, Vars, Cheat);
    RejectReducible :=
         evalb(ASectorsGroup = [[ ], 1, Vars]
                   and AInvarianceGroup = [[ ], 1, Vars]
```

```
                  and BSectorsGroup = [[ ], 1, Vars]
                  and BInvarianceGroup = [[ ], 1, Vars]);
   if not PrelimData[1] and 1 < nops(PrelimData) then
       print("The given singularity is the tensor product
                         of the following singularities:",
                PrelimData[2 .. -4]);
       if RejectReducible then
           return NULL
       end if;
   elif not PrelimData[1] then
       print("The given input is not a polynomial with an
           isolated quasihomogeneous singularity at the origin");
       return NULL
   end if;

   q := PrelimData[-3];
   Listq := convert( q, list);
   mu := mul(1 / q[i]-1, i= 1 .. dim);
   cHat:= add( 1-2 * q[i], i = 1 .. dim);
   JacobIdeal := convert( map(proc (a) options operator, arrow,
diff( W, a) end proc, Vars) , set);
   B:= Basis(JacobIdeal, plex(op(Vars)));
   H:= LinearAlgebra[Determinant](VectorCalculus[Hessian](W, Vars));
   Hess:= NormalForm(H, B, plex(op(Vars)));
   NearBasis:= map(proc (item) options operator, arrow,
Exponentiate(item[1], Vars) end proc, TrialElts(Listq, cHat, B, Vars));

   if Cheat = "Off" then
       MaxG := PrelimData[-2];
```

```
        MaxGGenerators := PrelimData[-1];
    else MaxG, MaxGGenerators:= MakeGroup( [Listq], 1, Vars) :
        print("Cheat mode ON -- declaring G=<J>");
    end if;

    OrderOfG := nops(MaxG);
    OrderOfJ:= ilcm(op(convert(map(denom, q), list)));
    print( "Generating Set for Maximal Abelian Symmetry Group:",
                     MaxGGenerators) ;
    PossibleBGen :=
             convert(MinimalGenSet(select(proc (item) options operator,arrow,
             evalb(frac(add(k, `in`(k, item))) = 0) end proc, MaxG)),list);
    print("Possible B-side generators: ", convert(PossibleBGen, set));
    BSideMaxGroup, BGenMax:= MakeGroup(PossibleBGen, W, Vars);
    if ASectorsGroup[ 1] = [ ] then
        ASectorsG := MaxG
    else
        ASectorsG := select(proc (item) options operator, arrow,
 member(item, MaxG) end proc,
           MakeGroup([
op(ASectorsGroup[1]), Listq], ASectorsGroup[2], ASectorsGroup[3])[1])
    end if;
    if AInvarianceGroup[1] = [ ] then
        AInvarianceG := ASectorsG;
        AGenerators := MinimalGenSet(ASectorsG)
    else
        AInvarianceG, AGenerators := MakeGroup(op(AInvarianceGroup))
    end if;
    if not member(Listq, AInvarianceG) then
```

```
        AInvarianceG := select(proc (item) options operator, arrow,
 member( item, MaxG) end proc,
            MakeGroup([op(AInvarianceGroup[1]), Listq],
                   AInvarianceGroup[2], AInvarianceGroup[3])[1]);
        AGenerators := MinimalGenSet(AInvarianceG);
        print("Invalid input for A-invariance group;
replacing by the join of <J> and the given group's intersection
 with the maximal abelian group:", AGenerators)
    end if;


    if BSectorsGroup[1] = [ ] then
        BSectorsG:= [[seq(0, i = 1 .. nops( Vars))]]
    else
        if Cheat = "BSide" then
            BSectorsG:= MakeGroup(op(BSectorsGroup))[1];
        else
            BSectorsG := select(proc (item) options operator, arrow,
 member(item, BSideMaxGroup) end proc, MakeGroup(op(BSectorsGroup))[1])
        end if;
    end if;
    if BInvarianceGroup[1] = [ ] then
        BInvarianceG := BSectorsG
    else
        if Cheat = "BSide" then
            BInvarianceG := MakeGroup(op(BInvarianceGroup))[1];
        else
            BInvarianceG := select(proc (item) options operator, arrow,
 member(item, BSideMaxGroup) end proc, MakeGroup(op(BInvarianceGroup))[1])
        end if;
    end if;


    BGenerators := MinimalGenSet(BInvarianceG);

    AFixedLociBases := map(TempWBasis, ASectorsG, W, NearBasis, q, Vars);

    BFixedLociBases := map(TempWBasis, BSectorsG, W, NearBasis, q, Vars);

    print("Charges= ", Listq);

    print("cHat= ", cHat, "mu= ", mul(1 / q-1, q in Listq));

    print( "Index of G over J = ", OrderOfG / OrderOfJ);

    CurlyH := map(proc (item) options operator, arrow,

 [2 * iota(item[1], Listq), op(item)] end proc,

map(GroupInvariantsA, AFixedLociBases, AGenerators, Vars));

    OrbifoldB := map(proc (item) options operator, arrow,

 [max(0, ListTools[DotProduct](DegRead(item[2], Vars), Listq))

        + (1/2) * sPlus(item [1], Listq), op(item)] end proc,

map(GroupInvariantsA, BFixedLociBases, BGenerators, Vars));

    printf( "h = \%a \t", nops(CurlyH)); printf( "m = \%a \t", nops(OrbifoldB));

    CurlyH := sort(CurlyH, proc (a, b) options operator, arrow,

 evalb(a[1] < b[1] or a[1] = b[1] and degree(a[3]) < degree(b[3])) end proc);

    OrbifoldB := sort(OrbifoldB, proc (a, b) options operator, arrow,

 evalb(a[1] < b[1] or a[1] = b[1] and degree(a[3]) < degree(b[3])) end proc);

    gc( );

    return [Listq, CurlyH, OrbifoldB]

end proc:


> SymGroup := proc(w:: polynom, Vars:: list)

    local dim, RowDim, BMatrix, StdBasis, PossGenerators, SubMatrices,

 AllOnes, Charges, GoodGenerators, PosGen2, FracVec, GpInProg,

TempSubgp, GeneratorsLeft, UsedGen;

    dim := nops(Vars);
    BMatrix := GetMatrix(W, Vars);

    RowDim:= RowDimension(BMatrix);

    FracVec := Vector(dim, fill = 1);

    AllOnes := Vector(RowDim, fill = 1);

    StdBasis:= [seq(Vector(dim, shape = scalar[k, 1]), k = 1 .. dim) ];

    SubMatrices := select(item -> evalb(Rank(item) = dim),

                                  [seq(BMatrix[k], k in choose(RowDim, dim))]);


# Here we take all full-rank submatrices of B

    if nops(SubMatrices) = 0 then

        print("Input polynomial does not have

 uniquely determined weights for quasihomgeneity");

        return [false];

    end if;


    Charges := LinearSolve(BMatrix, AllOnes);

    PossGenerators :=

        convert(Flatten([seq(map(LinearSolve, SubMatrices, k),

                              k in StdBasis)]), set);

    PosGen2:= map(object-convert(object, Vector),

         map(vect->map(frac, vect),

             map(item->convert(item + RowDimPracVec, list), PossGenerators)));

# We take all distinct generator candidates

# We tapdance by going to lists and back

# because Maple is stupid about recognizing vectors

# as being the same when in a list or set.


    PosGen2 := PosGen2 union {Charges};

# This is too brutish: there should be a decent way to do this without
# forcing <J>. This could induce errors of life if there are polynomials
# weird enough.

    GoodGenerators := select(object->
            andmap( entry-> type(entry, integer), BMatrix.object), PosGen2);
# we keep only the generator candidates
# that satisfy all monomials as group relators

    GoodGenerators :=
           convert(map(item -> convert(item, list), GoodGenerators), list);
    GoodGenerators := sort(GoodGenerators, (a, b) ->
                  evalb(ilcm(op(map(denom, a))) > ilcm(op(map(denom, b)))));
# now we sort the possible generators by order as group elements

    GeneratorsLeft:= GoodGenerators;
    GplnProg := {[seq(0, k = 1..dim)]};
    UsedGen := [ ];
    while nops( GeneratorsLeft) > 0 do
        TempSubgp := {seq( map( frac, k*GeneratorsLeft[1]),
           k = 1 .. ilcm (op(convert(map(denom, GeneratorsLeft[1]), list))))};
# make cyclic group for next unused generator
        UsedGen := [op(UsedGen) , GeneratorsLeft[1]];
# add this new generator to the list of generators we've used
        GpInProg:= {seq(seq(map(frac, i + j), i in TempSubgp), j in GpInProg)};
        GeneratorsLeft:= remove(item-> member(item, GpInProg), GeneratorsLeft);
# remove anything that would be redundant (rom the list of unused generators
    end do;
    return [Charges, convert( GplnProg, list), UsedGen]:
end proc:
```

Bibliography

[1] P. Berglund and T. Hubsch. A generalized construction of mirror manifolds. *Nuclear Physics B*, 393, 1993.

[2] H. Fan, T. Jarvis, and Y. Ruan. The witten equation, mirror symmetry, and quantum singularity theory. *arXiv:0712.4021*, 2007.

[3] M. Krawitz. *FJRW Rings and Landau-Ginzburg Mirror Symmetry.* PhD thesis, University of Michigan, 2010.

[4] M. Krawitz, N. Priddis, P. Acosta, N. Bergin, and H. Rathnakumara. Fjrw rings and mirror symmetry. *Communications in Mathematical Physics*, 296(1):145–174, 2010.

[5] M. Kreuzer and H. Skarke. On the classification of reflexive polyhedra. *Comm. Math. Physi.*, 185:495–508, 1997.

[6] H. Skarke. Reflexive polyhedra and their applications in string and f-theory. *arXiv:hep-th/0002246v1*, 2000.

[7] V. Batyrev. Dual polyhedra and mirror symmetry for calabi-yau hypersurfaces in toric varieties. *arXiv:alg-geom/9310003v1*, 1993.

[8] E. Witten. Phases of n=2 theories in two dimensions. *Nuclear Physics B*, 403:159–222, 1993.

[9] M. Kreuzer and H. Skarke. On the classification of quasihomogeneous functions. *Comm. Math. Physi.*, 150(1):137–147, 1989.

[10] V.I. Arnold, S.M. Gusein-Zade, and A.N. Varchenko. *Singularities of Differentiable Maps.* Birkhauser, 1985.