HONORS THESIS
EQUATIONS FOR THE LOG CANONICAL MODEL OF HYPERPLANE ARRANGEMENT COMPLEMENTS

SHELBY COX

1. Introduction

The main objects of study in algebraic geometry are the sets of zeroes of finitely many polynomial equations, called algebraic varieties. Sometimes varieties are defined explicitly from given equations, but we can also consider algebraic varieties that arise implicitly from certain procedures, like constructing the variety of a moduli space. When we know the equations that define a variety, we better understand the variety: we can find singularities of the variety and study properties like projective normality. Therefore, in the cases where the equations need to be derived, it is important that we know as much as possible about them.

One classical way in which algebraic varieties can implicitly arise is the embedding of canonical curves. In particular, when the curve is not hyperelliptic, the linear system associated to the canonical class gives us an embedding of the curve into projective space. In this case, there are several results. For example, Petri's theorem tells us that for curves of genus greater than or equal to 4, the ideal of the embedding is generated by the degree 2 elements, except in certain special cases, where the ideal is generated in degrees 2 and 3.

More recently, mathematicians have begun to study the canonical ring and the log canonical model. The canonical ring is defined to be $\mathrm{R} = \bigoplus_{n\geq 0} H^0(X, nK_X)$, or global sections of tensor powers of the canonical line bundle. Taking the projective spectrum of R, denoted by Proj(R) gives the canonical model. A fundamental theorem of Birkar-Cascini-Hacon-McKernan is that the canonical ring is always finitely generated. Similarly, we can associate

Date: June 12, 2018.

to a variety X and a divisor of the variety, D, a log canonical algebra, defined in the same way, except that we allow our sections to have simple poles along the divisor.

Over the summer, I began researching the equations of the log canonical model of hyperplane arrangement complements through an REU with Professor Jenia Tevelev. Our original questions have led to numerous further questions in algebraic geometry and representation theory.

1.1. **Goals and Methodology.**

Let $\mathcal{A} = \{H_1, \ldots, H_m\} \subset \mathbb{P}^n$ be an arrangement of hyperplanes defined by the linear equations $f_1, \ldots, f_n$. Denote by $M(\mathcal{A})$ the complement of $\mathcal{A}$. Denote by $R(\mathcal{A})$ the log canonical algebra of $M(\mathcal{A})$. We know that $R(\mathcal{A})$ is generated in degree 1, that the degree 1 component is isomorphic to the top degree component of the Orlik-Solomon algebra and that there is an algorithm for computing a basis for $R(\mathcal{A})$ [1].

Despite these results, explicit equations for the ideal of the log canonical model are still not known. Although it would be unrealistic to expect to explicitly write the equations of the log canonical model for any given hyperplane arrangement complement, we would ultimately like to obtain a result similar to Petri's theorem for hyperplane arrangement complements, which tells us when the log canonical model is sufficiently "nice," i.e., we would like to know:

(Q1) *For what hyperplane arrangements is the log canonical algebra:* **(a)** *Koszul?*; *and when is the homogeneous ideal of the log canonical model* **(b)** *generated in degree 2?*; **(c)** *generated by a quadratic Groebner basis?*

In order to address this question computationally, we interpret the basis of $R(\mathcal{A})$ as a subset of the following meromorphic functions:

$$\left\{ \frac{1}{f_{i_1} \cdots f_{i_k}} \mid H_{i_1} \cap \cdots \cap H_{i_k} = \{0\} \right\}$$

For a given (sufficiently small) line arrangements, we can immediately answer (b) and (c) using Macaulay2 by defining a homomorphism and asking for a Groebner basis of the kernel:

$$\phi : \mathbb{Q}[T_1, \ldots, T_m] \to \mathbb{Q}(x_1, \ldots, x_n)$$

$$T_\alpha \mapsto \frac{1}{f_{\alpha_1} \cdots f_{\alpha_n}}$$

But before we can even begin to answer **Q1**, we need to find a basis for the log canonical algebra. It is a result of Orlik and Solomon [2] that the log canonical algebra is generated in degree 1, but they do not give any explicit generators. Brion and Vergne do give an explicit algorithm for finding generators in [1], but it is not clear if the relations we seek in **Q1** are easy to write down with the basis that the Brion-Vergne algorithm generates. In addition, a theorem of Zaslavsky states that there is a numerical equivalence between the number of bounded regions of a hyperplane arrangement and the dimension of the log canonical algebra. Therefore, one aspect of my project will be to answer the natural question:

(Q0) *Is there a natural one-to-one correspondence between basis elements of $R(\mathcal{A})$ and bounded regions of $\mathcal{A}$?*

We are especially interested in these questions for Coxeter arrangements, which are associated with root systems. Most of my research so far has centered on computing the log canonical model for Coxeter arrangements of type A, where we already know that the log canonical algebra is Koszul [3]. Using Macaulay2, I found that for $n \leq 4$ the ideal of the log canonical model of A_n has a quadratic Groebner basis. I also found a basis of the log canonical algebra for the braid arrangement from which quadratic equations of the log canonical model can be easily obtained. *Based on the findings of Keel-Tevelev* [3]*, we also wonder if these relations and the relations obtained by acting with S_{n+2} generate all equations of the ideal.*

In general though, computations for larger Coxeter arrangements, like E_8, are not feasible. Instead of trying to calculate the ideal directly, we can begin to explore the log canonical compactification of large arrangements through computations for subarrangements. One way we can obtain a subarrangement is by restricting the arrangement, i.e., we only consider the basis elements of the log canonical algebra which include the equation of a certain hyperplane. Through explicit computations for the hyperplane arrangements A_3 and A_4, I noticed that some generators of the A_4 ideal came directly from the generators of the ideal of A_3 by restricting the A_4 arrangement to specific hyperplanes, which leads to the following question:

(Q2): *How is the log canonical model of subarrangements related to the log canonical model of the whole arrangement? In particular, how are their homogeneous ideals related?*

In addition to the questions outlined above, I would like to further investigate the Orlik-Solomon algebra and how it relates to the log canonical algebra, explore the connection between the Whitehouse module and the log canonical algebra of the braid arrangement, and answer (Q1) for reflection arrangements.

Knowing the equations of the log canonical model will help mathematicians better understand and study its properties. This work is important because not much is known about the log canonical model, even for complements of hyperplane arrangements. We may also broaden the project to study all reflection arrangements, as classified by Shepherd and Todd in [4]. In this way, the project would contribute to a large body of existing information about reflection arrangements.

2. Background

2.1. **Definitions.** A hyperplane is a linear subspace of codimension 1. A hyperplane arrangement is a set of hyperplanes of the same dimension. We can consider a hyperplane arrangement in affine or projective and complex or real space. We have the following definitions related to hyperplane arrangements in $\mathbb{RP}^n$, though they can easily be generalized to $\mathbb{R}^n$, $\mathbb{C}^n$ or $\mathbb{CP}^n$:

Definition. Let $\mathcal{A} = \{H_i\}$ be an arrangement of hyperplanes in $\mathbb{RP}^n$.

(1) A **minimal region** of $\mathcal{A}$ is a connected component of the complement $\mathbb{RP}^n \setminus \mathcal{A}$. More generally, a **region** of $\mathcal{A}$ is a connected component of $\mathbb{RP}^n \setminus \mathcal{B}$, where $\mathcal{B} \subset \mathcal{A}$
(2) A **simplex** is a region of a subset of $n+1$ of the H_i. A **minimal simplex** is a region that is minimal and a simplex.
(3) We call $\mathcal{A}$ **simplicial** if all its minimal regions are simplices

(4) Fix a hyperplane H_∞, which is general with respect to $\mathcal{A}$. A region of $\mathcal{A}$ is **bounded with respect to** H_∞ if H_∞ does not intersect it.

(5) An affine hyperplane arrangement is **central** if all the hyperplanes in the arrangement pass through a common point (usually the origin).

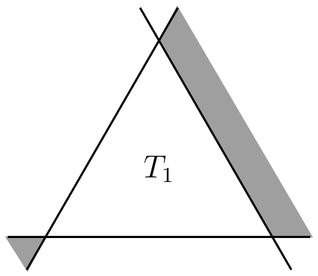

FIGURE 1. A simple line arrangement.

Example 2.1. The above arrangement has four minimal regions. One unbounded minimal region is shaded, and the only bounded minimal region is labeled T_1. This arrangement is simplicial.

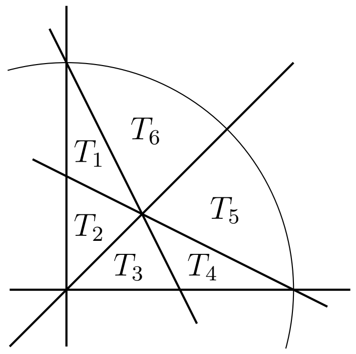

FIGURE 2. The arrangement A_3, with the minimal bounded regions labelled T_i.

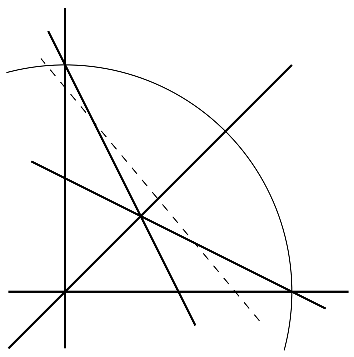

FIGURE 3. A_3 with a different choice of H_∞ (the dashed line).

Example 2.2. The above arrangement is the Coxeter arrangement A_3, with two different choices of H_∞. Note that A_3 is simplicial, with 6 minimal bounded regions, regardless of the choice of H_∞. The region made up of T_2 and T_3 is not a simplex.

The Log Canonical Algebra. Let D be a divisor and let (X, D) be a log canonical pair. Then the **log canonical algebra** $R(K_X + D)$ is:

$$R(K_X + D) := \bigoplus_{n \geq 0} H^0(X, n(K_X + D))$$

Let $\mathfrak{X}$ be a (wonderful) compactification of the complement of $\mathcal{A}$, $\mathbb{P}^n \setminus \mathcal{A}$, with normal crossing boundary. Denote by $p, l, \dots$ the points, lines, etc. to be blown-up, and by $\mathcal{H}$ the proper transform of H_i. Then the **canonical divisor** on $\mathfrak{X}$ is:

$$K_{\mathfrak{X}} = -(n+1)\mathcal{H} + (n-1)\sum_p E_p + (n-2)\sum_l E_l + \cdots$$

And the **boundary divisor** on $\mathfrak{X}$ is:

$$\Delta = \sum_{p,l,\dots} E + \sum_{i=1}^{r} \left(\mathcal{H} - \sum_{p \in H_i} E_p - \sum_{l \in H_i} E_l - \cdots \right)$$

And the **log canonical divisor** is the sum of the canonical divisor and the boundary divisor:

$$K_{\mathfrak{X}} + \Delta$$

Using $\mathfrak{X}$ as the compactification, we can define the **log canonical algebra** (LCA) of $\mathbb{P}^n \setminus \mathcal{A}$ to be:

$$R(\mathcal{A}) := \bigoplus_{n \geq 0} H^0(\mathfrak{X}, n(K_{\mathfrak{X}} + \Delta))$$

Remark. The log canonical algebra is independent of the choice of compactification of $\mathbb{P}^n \setminus \mathcal{A}$, hence the name "canonical."

Then the **log canonical model** (LC) is defined to be:

$$LC(\mathbb{P}^n \setminus \mathcal{A}) := \operatorname{Proj}(R(\mathcal{A}))$$

Remark 1. If $\mathfrak{X}$ is the wonderful compactification of $\mathbb{P}^n \setminus \mathcal{A}$, then "usually"

$$\mathfrak{X} = \operatorname{Proj}(R(\mathcal{A}))$$

Geometrically, this means that $K_{\mathfrak{X}} + \Delta$ is ample on the wonderful model.

In general, if $\mathfrak{Y}$ is a variety, D is a divisor on $\mathfrak{Y}$ and $S = \bigoplus_{n \geq 0} H^0(\mathfrak{Y}, nD)$, then $\mathrm{Proj}(S) = \mathfrak{Y}$ if and only if D is ample on $\mathfrak{Y}$.

For real line arrangements, the two possibilities for the log canonical model are described below.

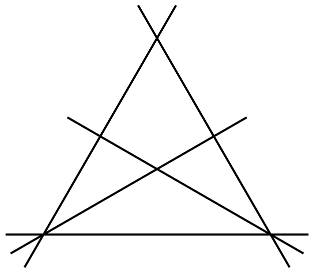

FIGURE 4. The complement of a line arrangement whose wonderful compactification is not the log canonical model.

Example 2.3. The wonderful compactification of the hyperplane arrangement complement in figure 4 is just the blow-up of $\mathbb{P}^2$ at two points, $\mathrm{Bl}_{p_1,p_2}\mathbb{P}^2$. But the log canonical model is the blow-up $\mathrm{Bl}_{p_1,p_2}\mathbb{P}^2$ with the line connecting p_1 and p_2, $\overline{p_1p_2}$, contracted. It is an (easy) exercise to check that the log canonical model of the above hyperplane arrangement complement is $\mathbb{P}^1 \times \mathbb{P}^1$.

Example 2.4. Suppose $\mathcal{A}$ is a line arrangement so that each line has 3 distinct points of intersection with other lines in the arrangement. Then $\mathfrak{X} = \mathrm{Proj}(R)$. Note that this is the only other possibility in $\mathbb{P}^2$.

2.2. **Previous Work.**

Theorem 1 (Orlik-Solomon Theorem). *For $\mathcal{A}$, f, $\mathfrak{X}$ as before,*

(1) $H^0(\mathfrak{X}, \Omega^n_{\mathfrak{X}}(log)) = H^n(M(\mathcal{A}); \mathbb{C})$
(2) $H^0(\mathfrak{X}, \Omega^n_{\mathfrak{X}}(log))$ is generated in degree 1, by $\frac{df}{f}$.

Remark 2. For a hyperplane arrangement $\mathcal{A} \subset \mathbb{P}^n$, we can also consider the corresponding central arrangement in affine space of dimension one greater, $\mathbb{A}^{n+1}$. The projectivization

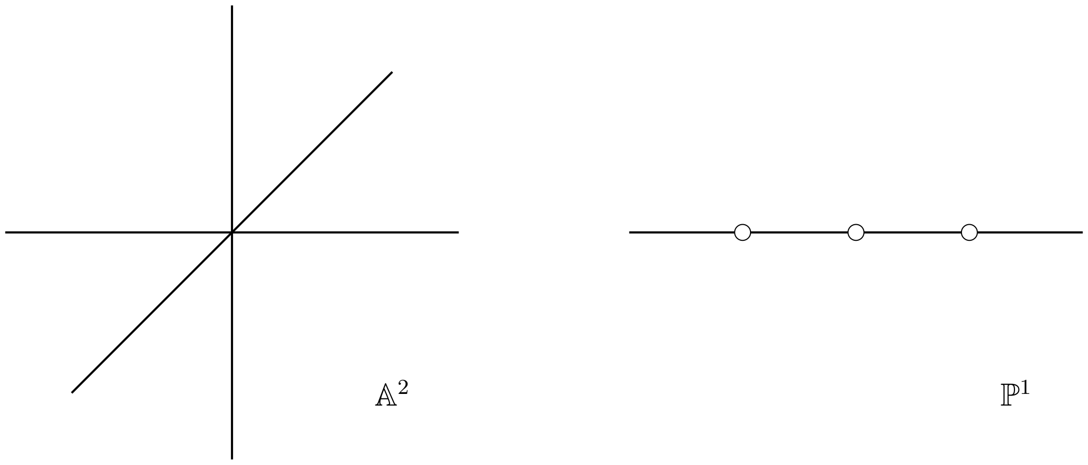

FIGURE 5. Projectivization is a $\mathbb{C}^*$-bundle

map is a $\mathbb{C}^*$-bundle, so the LCA of the projective hyperplane arrangement is the same as the LCA of the corresponding central affine arrangement.

This is important, for example, because the Orlik-Solomon theorem is stated in terms of 3-forms and there is no natural way to think of forms as sections on $\mathbb{P}^2$.

Remark. The first degree components of the LCA of a hyperplane arrangement complement, $R_1(\mathcal{A})$, is isomorphic to the top degree component of the Orlik-Solomon algebra of $M(\mathcal{A})$ [1].

Remark. For a line arrangement, the Orlik-Solomon theorem tells us that 3-forms with log poles corresponding to the triangles (simplices) of the arrangement generate the LCA (although, in most cases, this set is not linearly independent).

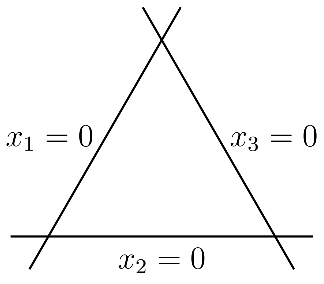

FIGURE 6. A very simple projective line arrangement.

Example 2.5. If $\mathcal{A} = \{x_1, x_2, x_3\}$ (as in the figure above), then $R(\mathcal{A})$ is generated by the 3-form:

$$\frac{dx_1}{x_1} \wedge \frac{dx_2}{x_2} \wedge \frac{dx_3}{x_3}$$

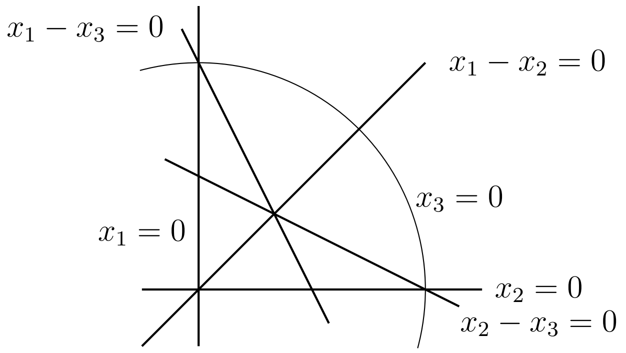

FIGURE 7. The projectivized A_3 arrangement.

Example 2.6. Figure 7 shows the Coxeter arrangement A_3, projectivized. The Orlik-Solomon theorem tells us that the 16 (6 choose 3, minus 4) 3-forms corresponding to non-degenerate triangles generate $R(A_3)$. In fact, we can do much better. That is, $R(A_3)$ is generated by just 6 of those 16 3-forms, which are linearly independent.

One way to pick generators for A_3 is to take the 3-forms corresponding to the smallest triangles in the picture:

$$\frac{df_1}{f_1} \wedge \frac{df_5}{f_5} \wedge \frac{df_6}{f_6}, \frac{df_1}{f_1} \wedge \frac{df_4}{f_4} \wedge \frac{df_5}{f_5}$$
$$\frac{df_2}{f_2} \wedge \frac{df_4}{f_4} \wedge \frac{df_6}{f_6}, \frac{df_2}{f_2} \wedge \frac{df_5}{f_5} \wedge \frac{df_6}{f_6}$$
$$\frac{df_3}{f_3} \wedge \frac{df_4}{f_4} \wedge \frac{df_5}{f_5}, \frac{df_3}{f_3} \wedge \frac{df_4}{f_4} \wedge \frac{df_6}{f_6}$$

Question: Do the forms corresponding to minimal bounded regions always generate $R(\mathcal{A})$?

2.3. **Primary Literature.** The following is a brief review of the primary literature important to my project.

([3]) **Keel-Tevelev**. In this paper, Keel and Tevelev state several important results for the log canonical algebra of the complement of the Coxeter arrangement, A_n. In particular, they relate it to the blow up of $\overline{M}_{0,n}$, the closure of the moduli space of curves of genus 0 with n marked points, and prove that the algebra is Koszul. They also describe implicitly the equations of the log canonical model. One of the goals of my project is to be able to write the equations down explicitly.

([1]) **Brion-Vergne**: In their paper, Brion and Vergne state the correspondence between the Orlik-Solomon Algebra and the Log Canonical algebra, which allows us to apply Zaslavsky's theorem to our problem. They also provide an algorithm for finding a basis of the top degree component of the Orlik-Solomon Algebra, which makes our problem computable.

In this paper, they also consider a larger algebra than the log canonical one, which may be useful to me later. If $\mathcal{A} = \{H_i\}$ is a hyperplane arrangement with its equations given by $\{f_i\}$, the larger algebra Brion and Vergne consider is generated by all fractions of the form $1/f_i$. In contrast, the first degree component of the log canonical algebra is generated by fractions with the product of three linear terms in the denominator.

([5]) **Zaslavsky's Theorem.** Zaslavsky's theorem is an important result about the dimension of the top degree component of the Orlik-Solomon algebra, which I can relate to my problem via the correspondence described in the previous subsection. If $\mathcal{A}$ is either a real affine or real projective hyperplane arrangement, then Zaslavsky's theorem states:

Theorem 2 (Zaslavsky)**.** *The dimension of the top degree component of the Orlik-Solomon Algebra of the complement of $\mathcal{A}$ is equal to the number of minimal bounded regions of the arrangement.*

Zaslavsky's proof uses advanced combinatorial methods; we can provide another proof using methods from algebraic geometry.

([2]) **Yuzvinsky.** Yuzvinsky's paper provides a survey of Orlik-Solomon algebras of hyperplane arrangement complements, which may be useful to me as I further explore the connection between the Orlik-Solomon Algebra and Log Canonical Algebra.

([4]) **Shepherd-Todd.** Shepherd and Todd give a classification of all hyperplane arrangements. Although most are too large for the log canonical model to be directly computed using Macaulay2, it would be interesting to find the log canonical model for different classes of hyperplane arrangements, as outlined in this paper. Other classifications of line arrangements are given by Hoge and Rohrle in [6].

3. Results

3.1. **An Alternate Proof of Zaslavsky's Theorem.**

Theorem 3 (Zaslavsky's Theorem)**.** *The number of minimal bounded regions of an arrangement $\mathcal{A}$ is equal to the number of basis elements of $R(\mathcal{A})$.*

Zaslavsky's original proof was a purely combinatorial argument, but we can also provide a proof using algebraic geometry. First, recall the Kawamata-Viehweg vanishing theorem:

Theorem 4 (Kawamata-Viehweg Vanishing Theorem)**.** *Let X be a smooth projective algebraic variety, and let $\Delta = \Delta_0 + A$ be a $\mathbb{Q}$-divisor of X so that*

(i) Δ_0 has coefficients c_i with $0 < c_i < 1$
(ii) Δ_0 has simple normal crossings
(iii) A is ample

Then $H^i(\omega_X(\Delta)) = 0$ for $i > 0$.

We can use the Kamawata-Viehweg vanishing theorem to prove the following lemma:

Lemma 4.1. *Suppose $\mathcal{A} = \{H_i\}_{i=1}^r$ is a hyperplane arrangement, and let $\mathfrak{X}$ be the wonderful compactification of its complement. Then $H^i(\mathfrak{X}, n(K_{\mathfrak{X}} + \Delta)) = 0$ for all $i > 0$.*

Proof. The boundary divisor of $\mathfrak{X}$ is

$$\Delta = \sum_i E_i + \sum_{i=1}^{r}(\mathcal{H} - \sum_{p \in H} E_p - \sum_{l \in H} E_l - \cdots)$$

Now let $\overline{A}$ be any ample divisor, and write:

$$\overline{A} := d\mathcal{H} - \sum m_i E_i$$

Let $k_p = \sum_{i=1}^{r} \mathcal{H}_i \cdot E_p$ and let $k = \max(k_i, r) = r$. Then take

$$c' := 1 - \frac{1}{2k+1}$$

$$c_i := 1 + (c' - 1)k_i - \delta m_i$$

$$\beta := (1 - c')r - \delta d$$

$$\delta := \frac{1 - c'}{\max(m_i) + 1}$$

Now define

$$A := \delta \overline{A} + \beta H$$

$$\Delta_0 := c_1 E_1 + \cdots + c_s E_s + \sum c'(H - \sum_p E_p)$$

Then, in fact,

$$\begin{aligned}\Delta_0 + A &= \sum_i c_i E_i + \sum c'(H - \sum_p E_p) + \delta A + \beta H \\ &= \Big[1 + (c' - 1)k_i - \delta m_i - c' k_i\Big] E_i + \Big[rc' + \delta d + (1 - c')r - \delta d\Big] H \\ &= (1 - k_i)E_i + rH \\ &= \Delta\end{aligned}$$

and

(i) Δ_0 has coefficients between 0 and 1. Obvious by the choice of c', c_i:

$$\begin{aligned} c' &= 1 - \frac{1}{2k+1} \in (0,1) \\ \iff \frac{1}{2k+1} &\in (0,1)\end{aligned}$$

which is true since $r > 0$.

$$\begin{aligned} c_i &= 1 + (c' - 1)k_i - \delta m_i \in (0,1) \\ \iff \delta m_i &- (c' - 1)k_i \in (0,1)\end{aligned}$$

which follows from the fact that:

$$0 < |\delta m_i| < (1-c')k_i < \frac{1}{2}$$

From right to left: the first inequality is obvious; the second inequality follows from the choice of δ and because $k_i \geq 1$ for each i; the last inequality is because of our choice of c'.

(ii) Δ_0 has simple normal crossings. This follows directly from the fact that we chose $\mathfrak{X}$ to be the wonderful compactificiation.

(iii) A is ample. Since $\overline{A}$ is ample by assumption and βH is NEF (since β is positive), A is ample by Kleiman's criterion.

Hence, by the Kamawata-Viehweg vanishing theorem, $H^i(\mathfrak{X}, n(K_{\mathfrak{X}} + \Delta)) = 0$ for all $i > 0$. □

Alternate Proof of Zaslavsky's Theorem. We can proceed using induction on the number of hyperplanes, and the deletion-restriction sequence. Let $\mathcal{A} = \{H_1, \ldots, H_r\} \subset \mathbb{P}^n$. Obviously the result holds if we take the fewest number of hyperplanes possible, $r = n+1$ (there is only one bounded region, and it must be a simplex).

Now suppose that $\mathcal{A}$ has $r+1$ hyperplanes. Then let $\mathcal{A}' = \mathcal{A} \setminus H_{r+1}$, and let $\overline{A} = \mathcal{A}\mid_{H_{r+1}}$. $\overline{H}_{r+1}$ be the proper transform of H_{r+1} in $\mathfrak{X}$, and let $\Delta' = \Delta - \overline{H}_{r+1}$.

One can show using the Kawamata-Viehweg vanishing theorem that $H^i(\mathfrak{X}, n(K_{\mathfrak{X}}+\Delta)) = 0$ for $i > 0$. So by adjunction, we have the short exact sequence:

$$0 \to H^0(\mathfrak{X}, K_{\mathfrak{X}} + \Delta') \to H^0(\mathfrak{X}, K_{\mathfrak{X}} + \Delta) \to H^0(\mathfrak{X}, K_{\overline{H}_{r+1}} + \Delta_{\overline{H}_{r+1}}) \to 0$$

since $H^i(\mathfrak{X}, K_{\overline{x}} + \Delta) = 0$ for all $i > 0$ by lemma 2.1. By our inductive assumption, $\dim(H^0(\mathfrak{X}, K_{\mathfrak{X}}+\Delta'))$ is equal to the number of bounded regions of $\mathcal{A}$ without the hyperplane H_{r+1}, and $\dim(\mathfrak{X}, K_{H_{r+1}} + \Delta_{H_{r+1}})$ is equal to the number of bounded regions that H_{r+1} intersects in $\mathcal{A}$. Every bounded region H_{r+1} intersects will give one new bounded region when we add H_{r+1} to the arrangement. Hence,

$$\dim(H^0(\mathfrak{X}, K_{\mathfrak{X}} + \Delta)) = \dim(H^0(\mathfrak{X}, K_{\mathfrak{X}} + \Delta')) + \dim(\mathfrak{X}, K_{H_{r+1}} + \Delta_{H_{r+1}})$$

is exactly the number of bounded regions of $\mathcal{A}$ after adding H_{k+1}. □

Proposition 4.1. *There is a canonical basis of the log canonical algebra given by the composition:*

$$H^0(M(\mathcal{A})) \longrightarrow H_n(\mathbb{RP}^n, \mathcal{A}) \longrightarrow H_n(\mathbb{CP}^n, \mathcal{A}_{\mathbb{C}}) \longleftarrow H^n(M_{\mathbb{C}}(\mathcal{A})) \longleftarrow H^n_{dR}(M_{\mathbb{C}}(\mathcal{A}))$$

That is, pick a hyperplane at infinity, H_∞; then the elements of $H^0(M(\mathcal{A}))$ corresponding to the bounded regions with respect to H_∞ give a basis (under the composition) of $H^n_{dR}(M_{\mathbb{C}}(\mathcal{A}))$, which in turn gives rise to a basis of the degree 1 part of the log canonical algebra.

Note. For the following two lemmas, we work in affine space. That is, choose one hyperplane of $\mathcal{A}$ to be the hyperplane at infinity, and use it to orient the resulting affine vector space.

Lemma 4.2. *Let A be a minimal bounded region of $M(\mathcal{A})$. If A is simplicial, with bounding equations $f_1, \dots, f_{n+1}$, then*

$$\frac{df_1}{f_1} \wedge \cdots \wedge \frac{df_{n+1}}{f_{n+1}}$$

is the form corresponding to A under the composition.

Proof. We can reduce to the case where the hyperplanes of $\mathcal{A} = \{H_i\}$ form a basis of $\mathbb{RP}^n$ and there is exactly one minimal bounded region, A. We need to show that the form corresponding to A under the maps in the proposition is

$$\omega = c\frac{df_1}{f_1} \wedge \cdots \wedge \frac{df_{n+1}}{f_{n+1}}$$

for some constant c.

Let

$$C = \big\{z = (z_1, \dots, z_{n+1}) \,\big|\, |z_i| = 1\big\} \subset \mathbb{C}^{n+1}$$

Note that C is dually associated with ω.

By basic complex analysis,

$$\int_C \frac{df_1}{f_1} \wedge \cdots \wedge \frac{df_{n+1}}{f_{n+1}} = c(2\pi i)^{n+1}$$

On the other hand, A and C are cycles intersecting transversely in a single point, so

$$A \cdot C = 1$$

So take $c = \frac{1}{(2\pi i)^{n+1}}$.

Since both vector spaces are one-dimensional, we conclude that ω is in fact the form corresponding to A. $\square$

Lemma 4.3. *Let A be any bounded region of $M(\mathcal{A})$, and let $\{T_i\}$ be any triangulation of A. Then the form*

$$\omega_{T_1} + \cdots + \omega_{T_k}$$

is the form corresponding to A under the composition.

Proof. It suffices to study the case when A can be triangulated by adding one hyperplane to the arrangement. In $\mathbb{RP}^2$, this reduces to studying the case when A is a quadrilateral, and adding one diagonal of A to the arrangement, illustrated below:

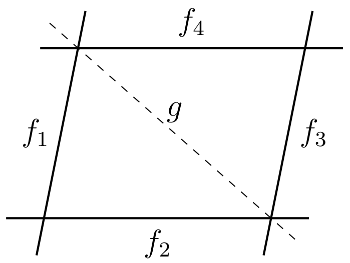

FIGURE 8. A triangulation of a quadrilateral in $\mathbb{P}^2$.

Let $\mathcal{A} = \{H_i\}$, and let H_g be the hyperplane we add to triangulate A. Then consider the following diagram:

$$\begin{tikzcd}
H_n(\mathbb{CP}^n, \mathcal{A}_{\mathbb{C}}) \arrow[d, "j_*"] & H^n(M_{\mathbb{C}}(\mathcal{A})) \arrow[l, "{[N']\frown}"'] \arrow[d, "r"] \\
H_n(\mathbb{CP}^n, \mathcal{A}_{\mathbb{C}} \cup H_g) & H^n(M_{\mathbb{C}}(\mathcal{A} \cup H_g)) \arrow[l, "{[N]\frown}"']
\end{tikzcd}$$

Where $[N']$ is an orientation class of $M_{\mathbb{C}}(\mathcal{A})$ and $[N]$ is the restriction of that class to $M_{\mathbb{C}}(\mathcal{A} \cup H_g)$. The horizontal maps are Lefschetz duality, the left vertical map is from the long exact sequence of the pair $(\mathbb{CP}, \mathcal{A}_{\mathbb{C}})$ and the right vertical map is restriction. It is easy to see that the diagram commutes.

Furthermore, the right column is part of a short exact sequence:

$$\begin{tikzcd}
0 \arrow[r] & H^k(M_{\mathbb{C}}(\mathcal{A})) \arrow[r, "r"] & H^k(M_{\mathbb{C}}(\mathcal{A} \cup H_g)) \arrow[r, "\alpha^{-1}\circ\tau"] & H^{k-1}(M_{\mathbb{C}}(\mathcal{A} \mid_{H_g})) \arrow[r] & 0
\end{tikzcd}$$

where α is the excision of $M(\mathcal{A}) \backslash Tub(\mathcal{A} \mid_{H_g})$ (where $Tub(\mathcal{A} \mid_{H_g})$ is a tubular neighborhood of $\mathcal{A} \mid_{H_g}$ in $\mathcal{A}$) and τ is the Thom isomorphism (this sequence is explained in detail in [2]). It follows that r is injective.

It follows from injectivity of r and the previous lemma that if there is a form corresponding to A, it must be the form $\omega_{T_1} + \omega_{T_2}$.

So we just have to show that the form $\omega_{T_1} + \omega_{T_2}$ has no pole along the new hyperplane, H_g. Choose some ordering on the vertices of A. It may be necessary to reorder the hyperplanes of A. If n is even, order the hyperplanes so that $H_{f_i} \cap H_{f_{n+3-i}} \cap H_g$ is a linear subspace of codimension 2. If n is odd, let f_{n+2} be the hyperplane which bounds both T_1 and T_2, and reorder the other hyperplanes so that $f_i \cap f_{n+3-i} \cap H_g$ is a linear subspace of codimension 2.

For now assume that n is even. The case when n is odd is very similar. Further order the f_i so that

$$\omega_{T_1} = \frac{dg}{g} \wedge \frac{df_1}{f_1} \wedge \cdots \wedge \frac{df_{n/2+1}}{f_{n/2+1}}$$

Then we must have

$$\omega_{T_2} = -\frac{dg}{g} \wedge \frac{df_{n+2}}{f_{n+2}} \wedge \cdots \wedge \frac{df_{n/2+2}}{f_{n/2+2}}$$

since adjacent simplices must naturally have opposite orientations.

Note that since $H_{f_i} \cap H_{f_{n+3-i}} \cap H_g$ is a linear subspace of codimension 2, the restriction of f_i to H_g, $\overline{f_i}$, is a multiple of f_{n+3-i} restricted to H_g. Hence,

$$\frac{d\overline{f_i}}{\overline{f_i}} = \frac{d\overline{f_{n+3-i}}}{\overline{f_{n+3-i}}}$$

Then,

$$\begin{aligned}
\mathrm{res}_g(\omega_{T_1} + \omega_{T_2}) &= \mathrm{res}_g(\frac{dg}{g} \wedge \frac{df_1}{f_1} \wedge \cdots \wedge \frac{df_{n/2+1}}{f_{n/2+1}}) + \mathrm{res}_g(\frac{dg}{g} \wedge \frac{df_{n+2}}{f_{n+2}} \wedge \cdots \wedge \frac{df_{n/2+2}}{f_{n/2+2}}) \\
&= \mathrm{res}_g(\frac{dg}{g} \wedge \frac{df_1}{f_1} \wedge \cdots \wedge \frac{df_{n/2+1}}{f_{n/2+1}}) - \mathrm{res}_g(\frac{dg}{g} \wedge \frac{df_1}{f_1} \wedge \cdots \wedge \frac{df_{n/2+1}}{f_{n/2+1}}) \\
&= 0
\end{aligned}$$

Hence, $\omega_{T_1} + \omega_{T_2}$ has no pole along H_g. This proves that $\omega_{T_1} + \omega_{T_2}$ is the form corresponding to A. □

3.2. **Equations of the Log Canonical Model.**

3.3. **Warm-Up.** To find equations for the log canonical model, we exploit the connection between forms and fractions. That is, if we associate to each form $\omega = \frac{df}{f}$ the fraction $frac(\omega) = \frac{1}{f}$, then the equations of log canonical model are given by the kernel of the homomorphism

$$\phi : k[T_1, \dots, T_l] \to k(x_1, \dots, x_n)$$

$$T_i \to frac(\omega_{T_1})$$

where ω_{T_i} are any basis of the log canonical algebra.

Example 3.1. Take numbers a, b, c and consider the homomorphism from the polynomial algebra in variables A, B, C to the field of rational functions $k(x)$ in one variable, which sends A to $1/(x-a)$, B to $1/(x-b)$, and C to $1/(x-c)$. Find generator(s) of the kernel of this homomorphism.

The following code in Macaulay2 answers the question:

```
R = QQ[A,B,C,a,b,c,MonomialOrder=>Lex];
```

```
S = frac(QQ[x,a,b,c]);


phi = map(S,R,{1/(x-a),1/(x-b),1/(x-c),a,b,c});
ker phi
```

with the output:

```
ideal (B*C*b - B*C*c - B + C, A*C*a - A*C*c - A + C, A*B*a - A*B*b - A
    + B)
```

Remark 3. I was unable to find a way to do this in Magma, SAGE or Mathematica. The kernel methods in those computer algebra softwares only find the kernel of a quotient homomorphism.

3.3.1. *Known Equations.* I was able to directly compute equations for the log canonical model in several cases. The code for these specific cases, and in the most general case, is provided in the appendix; I list only the equations here.

Example 3.2. A_3. The hyperplanes of A_3 are:

$$\{x_1, x_2, x_3, x_1 - x_2, x_2 - x_3, x_1 - x_3\}$$

The basis of the log canonical algebra corresponding to the minimal bounded regions is:

$$t_1 = \frac{1}{x_1(x_1 - x_3)(x_2 - x_3)}$$
$$t_2 = \frac{1}{x_1(x_1 - x_2)(x_2 - x_3)}$$
$$t_3 = \frac{1}{x_2(x_1 - x_2)(x_1 - x_3)}$$
$$t_4 = \frac{1}{x_2(x_2 - x_3)(x_1 - x_3)}$$
$$t_5 = \frac{1}{x_3(x_1 - x_2)(x_2 - x_3)}$$
$$t_6 = \frac{1}{x_3(x_1 - x_2)(x_1 - x_3)}$$

The kernel of the homomorphism has a Groebner basis:

$$t_3t_5 - t_3t_6 - t_4t_6$$

$$t_2t_4 - t_2t_5 + t_3t_6 + t_4t_6$$

$$t_1t_5 - t_2t_5 + t_2t_6$$

$$t_1t_4 - t_2t_5 + t_2t_6 + t_4t_6$$

$$t_1t_3 - t_2t_6 + t_3t_6$$

Example 3.3. A_4. The log canonical model of the complement of A_4 has a quadratic Groebner basis of 175 equations.

Example 3.4. B_3. The log canonical model of the complement of B_3 has a quadratic Groebner basis of 71 equations.

Remark 4. In all cases that I was able to directly compute, the equations of the log canonical model had a quadratic Groebner basis.

4. Coxeter Arrangements

I worked through several examples with Coxeter arrangements, most thoroughly with A_n and B_n:

$$A_n = \{x_i, x_i - x_j \mid 1 \leq i < j \leq n\}$$

$$B_n = \{x_i, x_i \pm x_j \mid 1 \leq i < j \leq n\}$$

Coxeter arrangements were the first examples I worked with because of their simplicity and symmetry. For example, we know that the dimension of R_1 of a Coxeter arrangement is equal to the product of its exponents. We can also use the Weyl action on Coxeter arrangements to find quadratic relations.

4.1. **A_n.** The Coxeter arrangement A_n is particularly special because we already know that it is Koszul [3] and because of its relation to the Whitehouse module, which is described in

Arrangement	Exponents + 1
A_n	$2, 3, \dots, n+1$
B_n	$2, 4, \dots, 2n$
C_n	$2, 4, \dots, 2n$
D_n	$2, 4, \dots, 2n-2, n$
E_6	$2, 5, 6, 8, 9, 12$
E_7	$2, 6, 8, 10, 12, 14, 18$
E_8	$2, 8, 12, 14, 18, 20, 24, 30$
F_4	$2, 6, 8, 12$
G_2	$2, 6$

FIGURE 9. Exponents of Coxeter Arrangements.

the next section. In this section, I describe some explicit bases for $R_1(A_n)$ and some of the equations for the log canonical model.

4.1.1. *Bases of R_1.* There are two bases of A_n that are particularly useful. The first is the basis corresponding to minimal bounded regions, which has a particularly simple form:

$$\left\{ \frac{\operatorname{sign}(\sigma)}{x_{\sigma(1)}(x_{\sigma(1)} - x_{\sigma(2)}) \cdots (x_{\sigma(n-1)} - x_{\sigma(n)})} \mid \sigma \in S_n \right\}$$

Linear Independence. Let T_i be the elements of the set above. Suppose that

$$\sum_{i=1}^{n!} \lambda_i T_i = 0$$

for some constants, λ_i. Clear denominators and evaluate the sum at the point $[1 : 0 : \cdots : 0]$. This shows that $\lambda_1 = 0$. Similar arguments show that $\lambda_i = 0$ for all i, hence the T_i are linearly independent.

Generating. As with the previous basis, it is obvious that this basis has the correct number of elements and is linearly independent, so it must also be generating.

We can choose a basis of A_3 via matrix:

$$\begin{array}{c|cc} & \frac{1}{x_1 - x_2} & \frac{1}{x_1 - x_3} \\ \hline \frac{1}{x_1} & T_0 & T_1 \\ \frac{1}{x_2} & T_2 & T_3 \\ \frac{1}{x_3} & T_4 & T_5 \end{array}$$

Each T_i has a factor of the fractions indicated beside the rows and columns, as well as a factor of $1/(x_2 - x_3)$. For example,

$$T_0 = \frac{1}{x_1(x_1 - x_2)(x_2 - x_3)}$$

In general we can choose a basis for A_n via the $n-1$ dimensional matrix with labels:

$$\frac{1}{x_1} \cdots \frac{1}{x_n}$$

$$\frac{1}{x_1 - x_2} \cdots \frac{1}{x_1 - x_n}$$

$$\vdots$$

$$\frac{1}{x_{n-1} - x_n}$$

Clearly, this gives a basis with $n!$ elements.

Linear Independence. First we prove directly that the basis chosen for A_3 is linearly independent, and then that the proposed basis for A_n is linearly independent by induction. Suppose that some linear combination of the given T_i is 0. That is,

$$\sum_{i=0}^{5} \lambda_i T_i = 0$$

Then clearing denominators, we have

$$\lambda_0 x_2 x_3 (x_1 - x_3) + \lambda_1 x_2 x_3 (x_1 - x_2) + \lambda_2 x_1 x_3 (x_1 - x_3) + \lambda_3 x_1 x_3 (x_1 - x_2)$$

$$+\lambda_4 x_1 x_2 (x_1 - x_3) + \lambda_5 x_1 x_2 (x_1 - x_2) = 0$$

Since the λ_i must work for any choice of x_i, we can take $x_1 = 0$. This leaves:

$$-\lambda_0 x_2 x_3^2 - \lambda_1 x_2^2 x_3 = 0$$

Then taking $x_2 = x_3 \neq 0$ implies $\lambda_0 = -\lambda_1$ and taking $x_2 = -x_3 \neq 0$ implies $\lambda_0 = \lambda_1$. Hence, we must have $\lambda_0 = \lambda_1 = 0$. Similar choices of x_i complete the proof for A_3.

Now in general, clear denominators and take any $x_i = 0$. The non-zero images are a basis for A_{n-1} and so by induction we are done.

Generating. The number of basis elements given by the labels is $n!$, and we know that $R_1(\mathcal{A})$ has dimension $n!$. Since we have already shown that the fractions are linearly independent, we conclude that they must also generate R_1.

4.1.2. *Quadratic Relations.* We can find equations for the log canonical models of A_3 and A_4 using Macaulay2, but for $n \geq 5$ the program crashes.

We can find some equations via the second basis provided in the previous subsection. We do this by taking 2×2 "minors" of the label matrix.

Example 4.1. $n = 3$. When $n = 3$, the relations come from minors of the matrix:

$$\begin{array}{c|cc|} & \frac{1}{x_1 - x_2} & \frac{1}{x_1 - x_3} \\ \frac{1}{x_1} & T_0 & T_1 \\ \frac{1}{x_2} & T_2 & T_3 \\ \frac{1}{x_3} & T_4 & T_5 \end{array}$$

Specifically, the relations we get are:

$$T_0T_3 - T_1T_2$$

$$T_2T_5 - T_3T_4$$

$$T_1T_4 - T_0T_5$$

Note that we found using Macaulay2 that the log canonical model of A_3 is generated by five equations, so the three above equations cannot form a basis of the log canonical model.

We can obtain more equations of the log canonical model by applying the S_{n+2} group action to the equations from the 2×2 minors of the matrix.

Example 4.2. $n = 3$

4.2. **B_n.** We can make similar observations for B_n. Specifically, the labels for B_n are:

$$\frac{1}{x_1}, \frac{1}{x_1 - x_j} \text{ for } i < j \leq n$$
$$\vdots$$
$$\frac{1}{x_n}$$

The proof that this is a basis of $R_1(B_n)$ is very similar to the proof to A_n, and I do not reproduce it here. An example of the basis when $n = 3$ is given below:

Example 4.3. $n = 3$

$$\begin{array}{c|ccc|} & \frac{1}{x_2} & \frac{1}{x_2+x_3} & \frac{1}{x_2-x_3} \\ \frac{1}{x_1} & T_1 & T_6 & T_{11} \\ \frac{1}{x_1+x_2} & T_2 & T_7 & T_{12} \\ \frac{1}{x_1-x_2} & T_3 & T_8 & T_{13} \\ \frac{1}{x_1+x_3} & T_4 & T_9 & T_{14} \\ \frac{1}{x_1-x_3} & T_5 & T_{10} & T_{15} \end{array}$$

That is, the basis we get is:

$$T_1 = \frac{1}{x_1x_2x_3} \qquad T_6 = \frac{1}{x_1(x_2+x_3)x_3} \qquad T_{11} = \frac{1}{x_1(x_2-x_3)x_3}$$
$$T_2 = \frac{1}{(x_1+x_2)x_2x_3} \qquad T_7 = \frac{1}{(x_1+x_2)(x_2+x_3)x_3} \qquad T_{12} = \frac{1}{(x_1+x_2)(x_2-x_3)x_3}$$
$$T_3 = \frac{1}{(x_1-x_2)x_2x_3} \qquad T_8 = \frac{1}{(x_1-x_2)(x_2+x_3)x_3} \qquad T_{13} = \frac{1}{(x_1-x_2)(x_2-x_3)x_3}$$
$$T_4 = \frac{1}{(x_1+x_3)x_2x_3} \qquad T_9 = \frac{1}{(x_1+x_3)(x_2+x_3)x_3} \qquad T_{14} = \frac{1}{(x_1+x_3)(x_2-x_3)x_3}$$
$$T_5 = \frac{1}{(x_1-x_3)x_2x_3} \qquad T_{10} = \frac{1}{(x_1-x_3)(x_2+x_3)x_3} \qquad T_{15} = \frac{1}{(x_1-x_3)(x_2-x_3)x_3}$$

Just like with the labels for A_n, we can use the labels above for B_n to find some of the equations for the log canonical model of B_n:

Example 4.4. $n = 3$. Referring to the matrix of T_i above, we get the relations:

$$T_i T_{i+6} - T_{i+1} T_{i+5}$$

I was unable to confirm whether or not these were a basis for the equations of $LC(B_n)$.

4.3. **The Whitehouse Module.** The Whitehouse module, W^n, is a free Lie Algebra in $n+2$ variables with an inner product (x, y) and exactly the following relations:

$$[x, y] = -[y, x] \tag{1}$$

$$[[x, y], z] + [[y, z], x] + [[z, x], y] = 0 \tag{2}$$

$$(x, [y, z]) = ([z, x], y) \tag{3}$$

We then consider the homogeneous degree 1 part of the algebra (i.e., elements with one of each variable, like $(x_1, [x_2, x_3])$) , which we denote by W_n^1. A basis of W_n^1 is given by the elements

$$(x_{n+2}, [\cdots [x_{n+1}, x_{\sigma(n)}], x_{\sigma(n-1)}], \cdots], x_{\sigma(1)}]),\ \sigma \in S_n$$

Note that there is a natural action of S_{n+2} on W_n given by permuting the x_i.

Example 4.5 (n=2)**.** A basis of W_2 is given by

$$(x_4, [[x_3, x_2], x_1]),\ (x_4, [[x_3, x_1], x_2])$$

The action of S_4 on W_2 is generated by transpositions, so it suffices to describe the action of the 6 transpositions. A full explanation of each calculation is provided in the appendix.

In summary,

$$(x_1\ x_2)\cdot A = B;\ (x_1\ x_2)\cdot B = A$$

$$(x_1\ x_3)\cdot A = -A;\ (x_1\ x_3)\cdot B = B-A$$

$$(x_1\ x_4)\cdot A = A-B;\ (x_1\ x_4)\cdot B = -B$$

$$(x_2\ x_3)\cdot A = A-B;\ (x_2\ x_3)\cdot B = -B$$

$$(x_2\ x_4)\cdot A = -A;\ (x_2\ x_4)\cdot B = B-A$$

$$(x_3\ x_4)\cdot A = B;\ (x_3\ x_4)\cdot B = A$$

4.4. **Relation to the Log Canonical Algebra of A_n.** Here, A_n refers to the Coxeter arrangement in $\mathbb{RP}^{n-1}$ given by the following equations:

$$\{x_i, x_i - x_j : 1 \le i < j \le n\}$$

I have proved that the following set is a basis of $R_1(A_n)$:

$$\left\{\frac{\operatorname{sign}(\sigma)}{x_{\sigma(1)}(x_{\sigma(1)}-x_{\sigma(2)})\cdots(x_{\sigma(n-1)}-x_{\sigma(n)})} \mid \sigma \in S_n\right\}$$

We can then consider an action of S_{n+2} on A_n given by permutations of the variables, the Cremona involution and one additional action.

4.4.1. *Cremona Involution.* The Cremona involution is obtained by inverting all the variables:

$$x_i \mapsto \frac{1}{x_i}$$

and then multiplying by a factor of

$$\frac{1}{\prod_i x_i^2}$$

It is easy to apply the Cremona involution in general:

$$\frac{1}{x_{\sigma(1)}(x_{\sigma(1)}-x_{\sigma(2)})\cdots(x_{\sigma(n-1)}-x_{\sigma(n)})} \mapsto \frac{(-1)^{n-1}}{x_{\sigma(n)}(x_{\sigma(1)}-x_{\sigma(2)})\cdots(x_{\sigma(n-1)}-x_{\sigma(n)})}$$

4.4.2. *Additional Action.* The addition action can be described as matrix multiplication by

$$\begin{bmatrix} 1 & & -1 \\ & \ddots & \vdots \\ & & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Example 4.6 (n=2)**.** Let

$$A = \frac{1}{x_1(x_1 - x_2)}, \quad B = \frac{1}{x_2(x_1 - x_2)}$$

be the basis of the log canonical algebra of A_2. Then we can compute the entire action of S^4:

(1) $(x_1\ x_2) \cdot A = -B$
(2) Cremona:
$$A \mapsto \frac{-1}{x_2(x_1 - x_2)} = -B$$
(3) Additional:
$$x_1 \mapsto x_1 - x_2$$
$$x_2 \mapsto -x_2$$
So,
$$A \mapsto \frac{1}{(x_1 - x_2)x_1} = A$$
$$B \mapsto \frac{1}{(-x_2)x_1} = A - B$$

References

[1] M. Brion and M. Vergne. Arrangement of hyperplanes I Rational functions and JeffreyKirwan residue. *Ann. Sci. cole Norm. Sup. 32*, 1999.

[2] S. Yuzvinskiĭ. Orlik-Solomon algebras in algebra and topology. *Uspekhi Mat. Nauk*, 56(2(338)):87–166, 2001.

[3] Sean Keel and Jenia Tevelev. Equations for $\overline{M}_{0,n}$. *Internat. J. Math.*, 20(9):1159–1184, 2009.

[4] G. C. Shephard and J. A. Todd. Finite unitary reflection groups. *Canad. J. Math.*, 6:274–304, 1954.

[5] Thomas Zaslavsky. Facing up to arrangements: face-count formulas for partitions of space by hyperplanes. *Mem. Amer. Math. Soc.*, 1(issue 1, 154):vii+102, 1975.

[6] Torsten Hoge and Gerhard Rhrle. Supersolvable reflection arrangements. *Proceedings of the American Mathematical Society*, 142(11):37873799, Apr 2014.

Appendix A. Code

A.1. **Brion-Vergne Algorithm.** In [1] (section 3.5), Brion and Vergne give an algorithm for finding a basis of the degree 1 component of the log canonical algebra, $R_1(M(\mathcal{A}))$, for $\mathcal{A}$ a hyperplane arrangement in $\mathbb{RP}^n$. The following is a brief discussion of the algorithm and an implementation of the algorithm in SAGE. I believe that with little or no modification, the given code could also be run in Python.

The algorithm is as follows:

1. Fix any ordering on the hyperplanes of $\mathcal{A} = \{H_i\}_{i=1}^k$ (in particular, the ordering does not have to be consistent with an ordering on monomials). Denote by f_i the equation of the hyperplane H_i.
2. An $n+1$-tuple, p, of hyperplanes in the arrangement corresponds to a basis element if the following two conditions are satisfied:
   (a) The elements of p form a basis of $\mathbb{RP}^n$. I will refer to this as the **basis condition**.
   (b) For each $1 \leq j \leq k$, the set
   $$\{H_i \in p : i \geq j\} \cup \{H_j\}$$
   is linearly independent. I will refer to this as the **second condition**.
3. Associate to each $(n+1)$-tuple, $(H_{i_1}, \ldots, H_{i_{n+1}})$, satisfying the above condition the form
   $$\frac{df_{i_1}}{f_{i_1}} \wedge \cdots \wedge \frac{df_{i_{n+1}}}{f_{i_{n+1}}}$$
   These forms generate $R_1(\mathcal{A})$.

Remark 5. The algorithm can be slightly computationally simplified by noting that no $(n+1)$-tuple without the first hyperplane in the arrangement will ever satisfy the second condition, since no $(n+2)$-tuple of hyperplanes can be linearly independent. Therefore it suffices to check only $(n+1)$-tuples which include the first hyperplane.

Example A.1. Recall that $A_3 = \{x_1, x_2, x_3, x_1 - x_2, x_2 - x_3, x_1 - x_3\}$. Fix the ordering of the hyperplanes to be the order in which they were listed. We begin by listing triples of hyperplanes which contain the first hyperplane and form a basis of $\mathbb{RP}^2$:

(1)	x_1	x_2	x_3
(2)	x_1	x_2	$x_2 - x_3$
(3)	x_1	x_2	$x_1 - x_3$
(4)	x_1	x_3	$x_1 - x_2$
(5)	x_1	x_3	$x_2 - x_3$
(6)	x_1	$x_1 - x_2$	$x_2 - x_3$
(7)	x_1	$x_1 - x_2$	$x_1 - x_3$
(8)	x_1	$x_2 - x_3$	$x_1 - x_3$

Now we need to check to see if each triple satisfies the second condition.

(1) There is nothing to check, so triple (1) satisfies the second condition.
(2) We have two sets to check: $\{x_3, x_2 - x_3\}$ and $\{x_1 - x_2, x_2 - x_3\}$. They are both linearly independent, so triple (2) satisfies the second condition.
(3) There are three sets to check: $\{x_3, x_1 - x_3\}$, $\{x_1 - x_2, x_1 - x_3\}$ and $\{x_2 - x_3, x_1 - x_3\}$. They are all linearly independent sets, so triple (3) satisfies the second condition.
(4) There is just one set to check: $\{x_2, x_3, x_1 - x_2\}$. Since it is linearly independent, triple (4) satisfies the second condition.
(5) There are two sets to check: $\{x_2, x_3, x_2 - x_3\}$ and $\{x_1 - x_2, x_2 - x_3\}$. The first of the sets is not linearly independent, so triple (5) does **not** satisfy the second condition.
(6) There are two sets to check: $\{x_2, x_1 - x_2, x_2 - x_3\}$ and $\{x_3, x_1 - x_2, x_2 - x_3\}$. They are both linearly independent, so triple (6) satisfies the second condition.

(7) There are three sets to check: $\{x_2, x_1 - x_2, x_1 - x_3\}$, $\{x_3, x_1 - x_2, x_1 - x_3\}$ and $\{x_2 - x_3, x_1 - x_3\}$. They are all linearly independent, so triple (7) satisfies the second condition.

(8) There are three sets to check: $\{x_2, x_2 - x_3, x_1 - x_3\}$, $\{x_3, x_2 - x_3, x_1 - x_3\}$ and $\{x_1 - x_2, x_2 - x_3, x_1 - x_3\}$. The last set is not linearly independent, so triple (8) **fails** the second condition.

We are left with the basis:

$$\frac{dx_1}{x_1} \wedge \frac{dx_2}{x_2} \wedge \frac{dx_3}{x_3}$$
$$\frac{dx_1}{x_1} \wedge \frac{dx_2}{x_2} \wedge \frac{d(x_2 - x_3)}{(x_2 - x_3)}$$
$$\frac{dx_1}{x_1} \wedge \frac{dx_2}{x_2} \wedge \frac{d(x_1 - x_3)}{(x_1 - x_3)}$$
$$\frac{dx_1}{x_1} \wedge \frac{dx_3}{x_3} \wedge \frac{d(x_1 - x_2)}{(x_1 - x_2)}$$
$$\frac{dx_1}{x_1} \wedge \frac{d(x_1 - x_2)}{(x_1 - x_2)} \wedge \frac{d(x_2 - x_3)}{(x_2 - x_3)}$$
$$\frac{dx_1}{x_1} \wedge \frac{d(x_1 - x_2)}{(x_1 - x_2)} \wedge \frac{d(x_1 - x_3)}{(x_1 - x_3)}$$

In my implementation of the algorithm, I defined several helper-methods (isBasis, sCond, poss, getInd) to simplify the main method (getGens). Here is a brief description of each method:

isBasis Takes two parameters: a list of hyperplanes p (in $\mathbb{RP}^n$) and an integer k. The integer k should be the minimum of $n + 1$ and the number of hyperplanes in p. The method creates a matrix whose rows are the elements of p. Returns **True** if the rank of the matrix is equal to k, *False* otherwise.

sCond Takes three parameters: a hyperplane arrangement L, a subarrangement p, and the list of indices of p in L, vp.

poss Takes two parameters: a list of hyperplanes, p, and an integer k. Returns all subsets of p with k elements, which include the first element of p. This method is used in getGens to get a list of $(n+1)$-tuples to check.

getInd Takes two parameters: a list of L and a sublist (not necessarily with the same order) l. Returns a list of the indices of elements of l in L.

getGens Takes one parameter: a list of hyperplanes S. Goes through a list of possible basis elements and checks the two conditions of the Brion-Vergne algorithm. Returns a list of basis elements (as lists of hyperplanes).

The following is the code for the implementation, ready to run in SAGE. An example of A_3 is given as an example.

```
import itertools


##set B = hyperplanes in the arrangement, vector format
##n is the dimension of the ambient space
##the example of A_3 is given here
B = [[1,0,0],[0,1,0],[0,0,1],[1,1,0],[0,1,1],[1,0,1]];


##takes a list of hyperplanes, p
##k is min of dimension or len(p)
def isBasis(p,k):
##checks if a given set of hyperplanes forms a basis of the space
    if matrix(p).rank() == k:
        return True
    return False


##given a set of hyperplanes L, in vector format and a subset of L, p,
##this method checks the second condition of the algorithm for p
def sCond(L,p):
    vp = getIndices(L,p)
    for y in range(len(vp)-1):
        for x in range(vp[y]+1,vp[y+1]):
            tocheck = p[y+1:]
            tocheck.append(L[x])
            ##if the set is linearly dependent
            if isBasis(tocheck,len(tocheck)) == False:
                ##then the given subset does not satisfy the second
                    condition
                return False
    return True


##takes a list p of hyperplanes, in RP^{k-1}
def poss(p,k):
##returns possible basis elements of the LCA
##from the list of hyperplanes, p
##k is the dimension of the ambient space
##since the first element of p needs to be in every basis element
##of the BV algorithm, we only look at subsets with it
    poss = []
    ##go through all subsets of size k, without the first element
    for l in itertools.combinations(p[1:],k-1):
        ##itertools returns tuples, so make the output a list first
        m = list(l)
        ##add the first element of p to the list
        m.append(p[0])
        ##add the possible basis element to poss
        poss.append(m)
    return poss
```

```
def getIndices(L,l):
##l a subset of L
##returns the indices of elements of l in L
    ind = []
    for x in l:
        ind.append(L.index(x))
    return ind

##S is a set of hyperplanes, given in vector form
def getGens(S):
        ##k will be dim + 1
        k = len(S[0])
        ##define an empty list to store the generators
    Gens = []
    ##go through all possible n+1 element subsets of S
    ##which include the first element of S
    for s in poss(S,k):
        i = getIndices(S,s)
        ##if the two conditions of the BV algorithm are satisfied
        if isBasis(s,k) and sCond(s,i,S):
            Gens.append(s)
    return Gens

print(getGens(B))
```

The output of the above code is the following:

```
[[[1 , 0, 0], [0 , 1, 0], [0 , 0, 1]] , [[1 , 0, 0], [0 , 1, 0], [0 , 1, −1]],
    [[1 , 0, 0], [0 , 1, 0], [1 , 0, −1]], [[1 , 0, 0], [0 , 0, 1], [1 , −1,
  0]] , [[1 , 0, 0], [1 , −1, 0], [0 , 1, −1]], [[1 , 0, 0], [1 , −1, 0] ,
  [1 , 0 , −1]]]
```

A.2. **Converting BV-output to M2.** The following code in Python converts the output of the Brion-Vergne algorithm implemented in the previous subsection to an m2 file which is ready to run in Macaulay2. It should be noted that although this program was designed to convert the output of the Brion-Vergne algorithm, it can also be used to create the m2 file for the next step of the calculation from any basis given as a list of lists of vectors (lists). To use the code, following these four steps:

Step 0. Copy and paste the code below into a text file, and save it with a .py extension.

Step 1. Create a text file (in the code, I call it raw-basis.txt) and paste the output of the previous algorithm in it. This code assumes that the text file is in the same working directory as the python code. You can also put in whatever basis of R_1, as long as it is formatted property. See note below.

IMPORTANT NOTE: When copying the output of the previous file, do not include the first two and last two brackets. For example, copy:

```
[1 , 0, 0], [0 , 1, 0], [0 , 0, 1]] , [[1 , 0, 0], [0 , 1, 0], [1 , 1, −1]],
  [[1 , 0, 0], [0 , 0, 1], [1 , 1, −1]
```

and **not**:

```
[[[1 , 0, 0], [0 , 1, 0], [0 , 0, 1]] , [[1 , 0, 0], [0 , 1, 0], [1 , 1, −1]],
    [[1 , 0, 0], [0 , 0, 1], [1 , 1, −1]]]
```

Step 2. Change the dimension (n, in the fourth line of code) to the appropriate number. Change the name of the file with the basis (line 8 of the code) to the appropriate file name.

Step 3. Run the code. The output is an m2 file in the working directory. It can be run in Macaulay2 as-is; more explanation is given in the next subsection.

```
import re


##dimension of the arrangement (#var - 1)
n = 3
##generate a list of strings to use as variables, x1,...,xn
var = ['x' + str(i+1) for i in range(n+1)]
##open the file with the raw basis output from SAGE program
textlines = open("raw-basis.txt", "r")
##and read the output to a string
stringlines = textlines.read()


##splits the raw basis string on special characters
freeliststr = re.split("\[([^[\]]*)\]",stringlines)


##replaces special characters with None
for i in range(len(freeliststr)):
        if freeliststr[i] == ', ' or freeliststr[i] == '], [':
                freeliststr[i] = None
##removes all None elements from freeliststr
freeliststr = list(filter(None,freeliststr))


#print(len(freeliststr))


##changes hyperplanes from vector format to equation format


planeList = []
for i in range(len(freeliststr)):
        sLine = freeliststr[i].split(', ')
        ##will replace zeta7 with z (not needed for real arrangements)
        ##more code needed for arrangements with more than one complex
            coefficient
        splitLine = [sLine[x].replace('zeta7', 'z') for x in range(len(
            sLine))]
        ##make a new string to store an equation
        iPlane = ''
        for k in range(len(splitLine)):
                iPlane += '(' + splitLine[k] + ')*' + var[k]
                if k != len(splitLine) - 1:
                        iPlane += '+'
        planeList.append(iPlane)


##group line equations in
ti = []
i = 0
count = 1
while i < len(freeliststr):
        newT = ''
        for j in range(n):
                if j == 0:
                        newT += 't' + str(count) + '␣=␣1/('
                newT += '(' + planeList[i+j] + ')'
                if j != n-1:
                        newT += '*'
                if j == n-1:
                        newT += ');'
        ti.append(newT)
        i += n
        count += 1
```

```
##open the file to write to
toWrite = open('code.m2', 'w')

##write in the M2 code
toWrite.write('Q = QQ \n')
toWrite.write('R = Q[x1..x' + str(len(ti)) + ',MonomialOrder=>Lex]; \n'
    )
toWrite.write('S = frac(Q[x1..x' + str(n+1) + ']); \n \n')
##then write in the fractions
for t in ti:
        toWrite.write("%s\n" % t)
##then the rest of the code
s = 't1'
for i in range(2,len(ti)+1):
        s = s + ', t' + str(i)
toWrite.write('\n' + 'phi = map(S,R,{' + str(s) + '}); \n')
toWrite.write('I = ker phi; \n')
toWrite.write('mingens gb I \n')
```

A.3. **Equations of the Log Canonical Model.** The following code is used to find equations for the log canonical model (the last step in the calculation!). The example below is the code for A_3 which is generated by the previous two steps in the algorithm.

The first chunk of code defines the polynomial rings we will work with. In the ring R, there should be one variable for each element of a basis of the log canonical algebra. The number of variables in the fraction ring S should be equal to the dimension of the ambient (affine) space or the dimension of the ambient projective space, plus one.

The second chunk of code defines the fractions that form a basis of the log canonical algebra. The final chunk of code defines a map, phi, which sends the variables of the polynomial ring R to the fractions we defined in the second chunk. Then we find the kernel of phi, I, and print the minimum generators of a Groebner basis of I, which are generators for the equations of the log canonical model.

Remark 6. Although finding a Groebner basis is usually computationally expensive, it does not appear to be as costly as finding the kernel of the map phi. That is, whenever the kernel can be found using Macaulay2, a Groebner basis for it can also be found.

```
Q = QQ
R = Q[x1..x6,MonomialOrder=>Lex];
S = frac(Q[x1..x4]);


t1 = 1/(((1)*x1+(0)*x2+(0)*x3)*((0)*x1+(1)*x2+(0)*x3)*((0)*x1+(0)*x2
    +(1)*x3));
t2 = 1/(((1)*x1+(0)*x2+(0)*x3)*((0)*x1+(1)*x2+(0)*x3)*((1)*x1+(0)*x2
    +(-1)*x3));
t3 = 1/(((1)*x1+(0)*x2+(0)*x3)*((0)*x1+(1)*x2+(0)*x3)*((0)*x1+(1)*x2
    +(-1)*x3));
t4 = 1/(((1)*x1+(0)*x2+(0)*x3)*((0)*x1+(0)*x2+(1)*x3)*((1)*x1+(-1)*x2
    +(0)*x3));
t5 = 1/(((1)*x1+(0)*x2+(0)*x3)*((1)*x1+(-1)*x2+(0)*x3)*((1)*x1+(0)*x2
    +(-1)*x3));
t6 = 1/(((1)*x1+(0)*x2+(0)*x3)*((1)*x1+(-1)*x2+(0)*x3)*((0)*x1+(1)*x2
    +(-1)*x3));


phi = map(S,R,{t1, t2, t3, t4, t5, t6});
I = ker phi;
mingens gb I
```

Remark 7. There are two things that we can do to make the program slightly more efficient.

(1) First, we can clear denominators, so that we are just working over polynomial rings. To do this automatically in the code, change line 48 of sage-to-m2.py to:

```
newT += 't' + str(count) + ' = cd/('
```

and add a new line to the m2 code which defines *cd* to be the product of all the hyperplanes in the arrangement.

(2) We can also work over a finite field instead of $\mathbb{Q}$. To do this, change line 35 of the sage-to-m2.py code to

```
iPlane += '(a^' + splitLine[k] + ')*' + var[k]
```

and change line 62 to

```
toWrite.write('Q = GF(k,Variable=>a); \n')
```

or make the appropriate changes in the code.m2 output file.

A.4. **Generating Arrangements from Group Actions.** Some line arrangements are generated by a subset of lines and some group action (for example, the Klein arrangement, given in [?]). The following code generates all the hyperplanes in the arrangement and outputs a list of the hyperplanes in vector form.

```
#takes a list of hyperplanes, L
#and a list of actions, G
#returns a list [g*l for each g in G, l in L]
def findNewLines(G,L):
    toReturn = []
    for a in G:
        for l in L:
            toReturn.append(a*l)
    return toReturn
```

```
##checks if two lines (as column matrices) are scalar multiples of each
    other
def sameLine(n,l):
    k = 0
    kset = False
    for x in range(l.nrows()):
        if l[x][0] != 0 and n[x][0] != 0:
            if not kset:
                k = (l[x][0])/(n[x][0])
                kset = True
            elif (l[x][0])/(n[x][0]) != k:
                return False
        if l[x][0] == 0 and n[x][0] != 0:
            if k != 0:
                return False
            if not kset:
                kset = True
        if l[x][0] != 0 and n[x][0] == 0:
            #12345 is just some random number
            if kset and k != 12345:
                return false
            if not kset:
                k = 12345
                kset = True
    return True

#checks if a list L contains a line equivalent to n
def containsLine(n,L):
    for l in L:
        if sameLine(n,l):
            return True
    return False


#will find all the lines in the arrangement
#arrgen is a list of lines that generate the arrangement
#actions are the group actions used to generate the arrangement
#guess is an upperbound on the number of lines to generate
def genLines(arrgen,actions,guess):
    #make a new list to hold all the lines
    lines = [l for l in arrgen]
    #while we have less than the upper bound on the number of lines
    while len(lines) < guess:
        #get some candidate new lines
        nl = findNewLines(gens,lines)
        count = 0
        for n in nl:
            #if the new line isn't in the list already, add it
            if not containsLine(n,lines):
                lines.append(n)
                count += 1
        #if we didn't add any lines, then we've finished the list!
        if count == 0:
            return lines
```

A.5. **Code and Output for Specific Arrangements.**

A.5.1. A_3. **Input:**

```
—#declare and print the polynomial algebra, ring of rational functions
```

```
R = QQ[T1,T2,T3,T4,T5,T6,MonomialOrder=>Lex];
S = frac(QQ[x1,x2,x3]);

--#declare necessary fractions
t1 = 1/(x1*(x1-x3)*(x2-x3));
t2 = 1/(x1*(x1-x2)*(x2-x3));
t3 = 1/(x2*(x1-x2)*(x1-x3));
t4 = 1/(x2*(x2-x3)*(x1-x3));
t5 = 1/(x3*(x2-x3)*(x1-x2));
t6 = 1/(x3*(x1-x3)*(x1-x2));

--#define the map from the polynomial algebra to the ring of rational
    functions
phi = map(S,R,{t1,t2,t3,t4,t5,t6});
--#find and name the kernel of the above map
I = ker phi;
--##find a Groebner basis for the kernel
mingens gb I
```

Output:

```
T3T5-T3T6-T4T6  T2T4-T2T5+T3T6+T4T6  T1T5-T2T5+T2T6  T1T4-T2T5+T2T6+T4T6
    T1T3-T2T6+T3T6
```

A.5.2. A_4. **Input:**

```
R = QQ[T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,
    T19,T20,T21,T22,T23,T24,MonomialOrder=>Lex];
S = frac(QQ[x1,x2,x3,x4]);

t1 = 1/(x1*(x1-x2)*(x2-x3)*(x3-x4));
```

```
t2 = 1/(x1*(x1-x2)*(x2-x4)*(x3-x4));
t3 = 1/(x1*(x1-x3)*(x2-x3)*(x2-x4));
t4 = 1/(x1*(x1-x3)*(x3-x4)*(x2-x4));
t5 = 1/(x1*(x1-x4)*(x2-x4)*(x2-x3));
t6 = 1/(x1*(x1-x4)*(x3-x4)*(x2-x3));

t7 = 1/(x2*(x1-x2)*(x1-x3)*(x3-x4));
t8 = 1/(x2*(x1-x2)*(x1-x4)*(x3-x4));
t9 = 1/(x2*(x2-x3)*(x1-x3)*(x1-x4));
t10 = 1/(x2*(x2-x3)*(x3-x4)*(x1-x4));
t11 = 1/(x2*(x2-x4)*(x1-x4)*(x1-x3));
t12 = 1/(x2*(x2-x4)*(x3-x4)*(x1-x3));

t13 = 1/(x3*(x1-x3)*(x1-x2)*(x2-x4));
t14 = 1/(x3*(x1-x3)*(x1-x4)*(x2-x4));
t15 = 1/(x3*(x2-x3)*(x1-x2)*(x1-x4));
t16 = 1/(x3*(x2-x3)*(x2-x4)*(x1-x4));
t17 = 1/(x3*(x3-x4)*(x1-x4)*(x1-x2));
t18 = 1/(x3*(x3-x4)*(x2-x4)*(x1-x2));

t19 = 1/(x4*(x1-x4)*(x1-x2)*(x2-x3));
t20 = 1/(x4*(x1-x4)*(x1-x3)*(x2-x3));
t21 = 1/(x4*(x2-x4)*(x1-x2)*(x1-x3));
t22 = 1/(x4*(x2-x4)*(x2-x3)*(x1-x3));
t23 = 1/(x4*(x3-x4)*(x1-x3)*(x1-x2));
t24 = 1/(x4*(x3-x4)*(x2-x3)*(x1-x2));

phi = map(S,R,{t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,t14,t15,t16,
    t17,t18,t19,t20,t21,t22,t23,t24});
```

```
I = ker phi;
```

Output:

Appendix B. Calculations

B.1. Whitehouse Module.

B.1.1. $n = 2$.

(1) $(x_1\ x_2) \cdot A = B$

(2) $(x_1\ x_3)$

$$\begin{aligned}
(x_1\ x_3) \cdot A &= (x_4, [[x_1, x_3], x_2]) && \text{by (1)} \\
&= -(x_4, [[x_3, x_1], x_2]) \\
&= -A \\
(x_1\ x_3) \cdot B &= (x_4, [[x_1, x_2], x_3]) && \text{by (2)} \\
&= (x_4, [[x_3, x_2], x_1]) - (x_4, [[x_3, x_1], x_2]) \\
&= B - A
\end{aligned}$$

(3) $(x_1\ x_4)$

$$\begin{aligned}
(x_1\ x_4)\cdot A &= (x_1, [[x_3, x_4], x_2]) && \text{by (3)}\\
&= ([x_2, x_1], [x_3, x_4]) && \text{by symmetry of the inner product}\\
&= ([x_3, x_4], [x_2, x_1]) && \text{by (3)}\\
&= (x_4, [[x_2, x_1], x_3]) && \text{by (2)}\\
&= (x_4, [[x_3, x_1], x_2]) - (x_4, [[x_3, x_2], x_1])\\
&= A - B\\
(x_1\ x_4)\cdot B &= (x_1, [[x_3, x_2], x_4]) && \text{by (3)}\\
&= ([x_4, x_1], [x_3, x_2]) && \text{by skew-symmetry of the bracket}\\
&= -([x_1, x_4], [x_3, x_2]) && \text{by (3)}\\
&= -(x_4, [[x_3, x_2], x_1])\\
&= -B
\end{aligned}$$

(4) $(x_2\ x_3)$

$$\begin{aligned}
(x_2\ x_3)\cdot A &= (x_4, [[x_2, x_1], x_3]) && \text{by (2)}\\
&= (x_4, [[x_3, x_1], x_2]) - (x_4, [[x_3, x_2], x_1])\\
&= A - B\\
(x_2\ x_3)\cdot B &= -B
\end{aligned}$$

(5) $(x_2\ x_4)$

$$\begin{aligned}
(x_2\ x_4)\cdot A &= (x_2, [[x_3, x_1], x_4]) && \text{by (3)} \\
&= ([x_4, x_2], [x_3, x_1]) && \text{by skew-symmetry of the bracket} \\
&= -([x_2, x_4], [x_3, x_1]) && \text{by (3)} \\
&= -(x_4, [[x_3, x_1], x_2]) && \\
&= -A &&
\end{aligned}$$

$$\begin{aligned}
(x_2\ x_4)\cdot B &= (x_2, [[x_3, x_4], x_1]) && \text{by (3)} \\
&= ([x_1, x_2], [x_3, x_4]) && \text{by symmetry of the inner product} \\
&= ([x_3, x_4], [x_1, x_2]) && \text{by (3)} \\
&= (x_4, [[x_1, x_2], x_3]) && \text{by (2)} \\
&= (x_4, [[x_3, x_2], x_1]) - (x_4, [[x_3, x_1], x_2]) && \\
&= B - A &&
\end{aligned}$$

(6) (x_3, x_4)

$$\begin{aligned}
(x_3, x_4) \cdot A &= (x_3, [[x_4, x_1], x_2]) && \text{by (3) twice} \\
&= ([x_1, [x_2, x_3]], x_4) && \text{by (1) twice} \\
&= ([[x_3, x_2], x_1], x_4) && \text{by symmetry of the inner product} \\
&= (x_4, [[x_3, x_2], x_1]) \\
&= B \\
(x_3, x_4) \cdot B &= (x_3, [[x_4, x_2], x_1]) && \text{by (3) twice} \\
&= ([x_2, [x_1, x_3]], x_4) && \text{by (1) twice} \\
&= ([[x_3, x_1], x_2], x_4) && \text{by symmetry of the inner product} \\
&= (x_4, [[x_3, x_1], x_2]) \\
&= A
\end{aligned}$$