A calculus for monomials in Chow group $A^{n-3}(n)$

Jiayue Qi

DK Statusseminar

2019.09.27.

basic setting

- Let $n \in \mathbb{N}$, $n \geq 3$, set $N := \{1, \dots, n\}$.
- A partition (I, J) of N where both cardinality of I and J are at least 2 is called a **cut** (of M_n).
- This talk focus on the Chow ring of M_n, where M_n is the moduli space of stable n-pointed curves of genus zero.
- Denote $\delta_{I,J}$ as the class of a cut subvariety $D_{I,J}$ of M_n.
- We will not focus on the details of M_n, what is important for this talk is the properties of this Chow ring.
- We denote the Chow ring of M_n as $A^*(n)$.

properties of $A^*(n)$

- It is a graded ring, we have $A^*(n) = \bigoplus_{k=0}^{n-3} A^k(n)$; and these homogeneous components are defined as Chow groups (of M_n). Here, for instance, we say $A^r(n)$ is a **Chow group of dimension** r.
- Fact1: $A^r(n) = \{0\}$ for $r > n-3$.
- Fact2: $A^{n-3}(n) \cong \mathbb{Z}$, we denote this isomorphism as $\int : A^{n-3}(n) \longrightarrow \mathbb{Z}$.
- $\{\delta_{I,J} \,|\, \{I,J\} \text{ is a cut}\}$ is a set of generators for $A^1(n)$; hence they are also generators for $A^*(n)$.
- For simplicity, we call them **generators** in the later text.
- $\prod_{i=1}^{n-3} \delta_{I_i,J_i}$ can be viewed as an element in $A^{n-3}(n)$ since we are in a graded ring.
- Quadratic relations between the generators.
- Linear relations between the generators.

Keel's quadratic relation

Among the generators of $A^*(n)$, $\delta_{I_1,J_1} \cdot \delta_{I_2,J_2} = 0$ and we say these two generators fulfill **Keel's quadratic relation** if the following conditions hold:

- $I_1 \cap I_2 \neq \emptyset$;
- $I_1 \cap J_2 \neq \emptyset$;
- $J_1 \cap I_2 \neq \emptyset$;
- $J_1 \cap J_2 \neq \emptyset$.

Easy example: When $n = 5$, $\delta_{12|345} \cdot \delta_{13|245} = 0$ but $\delta_{12|345}$ and $\delta_{123|45}$ does not fulfill this relation.

Keel's linear relation

Denote $\epsilon_{ij|kl} := \sum_{i,j\in I,k,l\in J} \delta_{I,J}$. Then we have the equality relations $\epsilon_{ij|kl} = \epsilon_{il|kj} = \epsilon_{ik|jl}$, we call it **Keel's linear relation**.

Example

When $n = 6$, we have $\epsilon_{12|35} = \epsilon_{13|25} = \epsilon_{15|23}$, i.e.,

$$\delta_{12,3456} + \delta_{124,356} + \delta_{126,345} + \delta_{1246,35}$$

$$= \delta_{13,2456} + \delta_{134,256} + \delta_{136,245} + \delta_{1346,25}$$

$$= \delta_{15,2346} + \delta_{145,236} + \delta_{156,234} + \delta_{1456,23}$$

motivation

- Many problems from yesterday's rigidity workshop can be reduced to computation of $\int \prod_{r=1}^{n-3} \epsilon_{i_r j_r | k_r l_r}$, subproblem of which is to compute $\int \prod_{r=1}^{n-3} \delta_{I_r, J_r}$.
- Denote $T := \prod_{r=1}^{n-3} \delta_{I_r, J_r}$, we define the **value of** T to be $\int (\prod_{r=1}^{n-3} \delta_{I_r, J_r})$.
- A easy case is when two factors of the monomial fulfill Keel's quadratic relation; we simply get value zero because of Keel's quadratic relation.
- What if this is not the case?
- Now we only need to consider the monomials $T := \Pi_{i=1}^{n-3} \delta_{I_i, J_i}$ where no two factors fulfill Keel's quadratic relation; we call this type of monomials **tree monomial** since there is a one-to-one correspondence between these monomials and *loaded tree with n labels and k edges*. We come to the definition of these trees now.

loaded tree

A **loaded tree with n labels and k edges** is a tree (V, E) together with a labeling function h from V to the power set of N such that the following conditions hold:

- Non-empty labels $\{h(v)\}_{v \in V}$ form a partition of N;
- Number of edges is k and here multiple edges are allowed;
- $\deg(v) + |h(v)| \geq 3$ holds for every $v \in V$.

loaded tree

See some examples of loaded trees. (check with definitions)

Figure: This is a loaded tree with 5 labels and 2 edges.

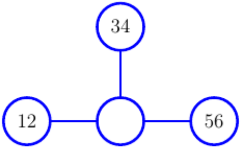


Figure: This is a loaded tree with 6 labels and 3 edges.

monomial of a given tree

- We define the **monomial of a given loaded tree** as the following:
- For each edge we collect the labels on one side of it to form I and labels on the other side of it to form J. And we say (I, J) is the corresponding cut for this edge.
- The monomial of this given loaded tree is $\prod_{i=1}^{n-3} \delta_{I_i,J_i}$; each edge of the tree contributes to the monomial a factor $\delta_{I,J}$ if (I, J) is the corresponding cut for this edge.
- It is well-defined and each loaded tree has a unique monomial representation.

monomial of a given tree

Figure: This is a loaded tree with 5 labels and 2 edges, the corresponding tree of tree monomial $\delta_{12|345} \cdot \delta_{123|45}$.

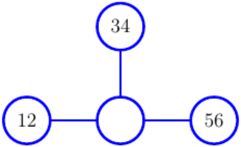


Figure: This is a loaded tree with 6 labels and 3 edges, the corresponding tree of tree monomial $\delta_{34|1256} \cdot \delta_{12|3456} \cdot \delta_{56|1234}$.

one-to-one correspondence

We claim that any monomial of a given loaded tree is actually a tree monomial; and every tree monomial uniquely represents a loaded tree.

Theorem

There is a one to one correspondence between tree monomials $T = \prod_{i=1}^{m} \delta_{I_i,J_i} (1 \leq m \leq n-3)$ and loaded trees with n labels and m edges. We call the corresponding tree of a tree monomial **tree of the given tree monomial**.

one-to-one correspondence

Proof.

- Prove by induction on m.
- Base case: $m = 1$, $T = \delta_{I_1,J_1}$. We define its corresponding tree simply as a tree with two vertices and one edge connecting them, setting two labeling sets of the vertices as I_1 and J_1, respectively. Obviously this tree is a loaded tree with n labels and 1 edge and its monomial is exactly T.
- Assume the statement holds for all $m \leq k$ $(1 \leq k \leq n-3)$.
- When $T = \prod_{i=1}^{k+1} \delta_{I_i,J_i}$, we define its corresponding tree as the following:

□

one-to-one correspondence

Proof.

- First collect these I_i, J_i $(1 \leq i \leq k+1)$ together in a set C (which can be a multi-set).
- Then pick any element $x \in C$ such that x has minimum cardinality; assume (x, y) is the cut (for M_n).
- Define $T_1 := \frac{T}{\delta_{x,y}}$, obviously it is still a tree monomial. By induction, there is a unique loaded tree $LT_1 = (V_1, E_1, h_1)$ with n labels and k edges representing T_1.
- Then all nodes of x must be together in $h_1(v)$ for some $v \in V_1$; otherwise, there will be another factor of T fulfilling Keel's quadratic relation with $\delta_{x,y}$ and this contradicts with the fact that T is a tree monomial.
- Then there are two cases: (1) $x = h_1(v)$; (2) $x \subsetneq h_1(v)$.

□

one-to-one correspondence

Proof.

- First case: $x = h_1(v)$, since x has minimal cardinality in set C, v must be a leaf and its adjacent edge corresponds to cut (x, y). In this case, we simply add one more multiplicity to this edge. Denote this new tree as LT.
- Second case: $x \subsetneq h_1(v)$. Add a new vertex u with labelling set x and one more edge uv connecting v and u; denote this new tree as $LT = (V, E, h)$.
- It is not hard to verify that in both cases LT is a loaded tree with n labels and $k + 1$ edges and the monomial of LT is just the product of $\delta_{x,y}$ and the monomial of LT_1, i.e., $T_1 \cdot \delta_{x,y}$, which is exacty T. In this way, we proved the uniqueness.
- By induction, the statement holds.

□

one-to-one correspondence

- From the proof above, we can extract an algorithm for constructing a loaded tree of the given tree monomial.
- However, the mutiplicity issue of edges can be simplified a bit.
- We can set that set C in the algorithm to be a normal set.
- The multiplicity of edges can be considered after the tree structure is constructed easily.
- (illustrate the example on the blackboard)
- We call it **tree algorithm**.

one-to-one correspondence

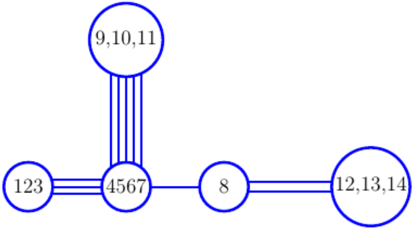


Figure: This is the corresponding loaded tree of the given monomial.

value of a loaded tree

- Goal: calculate $\int(T)$ for any tree monomial T
- Recall: $\int$ represents the isomorphism from $A^{n-3}(n)$ to $\mathbb{Z}$
- Because of this one-to-one correspondence, now we define **value of a loaded tree** as $\int(T)$, where T is the corresponding monomial of this loaded tree.

value of a loaded tree

- Goal: calculate $\int(T)$ for any tree monomial T
- Recall: $\int$ represents the isomorphism from $A^{n-3}(n)$ to $\mathbb{Z}$
- Because of this one-t-one correspondence, now we define **value of a loaded tree** as $\int(T)$, where T is the corresponding monomial of this loaded tree.
- Goal: Given a loaded tree with n labels and $n-3$ edges, we want to calculate its value.

special case

Theorem

If all factors are distinct in $T := \prod_{i=1}^{n-3} \delta_{I_i,J_i}$*, then* $\int(T) = 1$*. We call this type of tree monomial* **clever monomial** *and its corresponding loaded tree* **clever tree**.

Remark

For clever trees, we know that they have value 1*. What about non-clever trees? Let's see the following example for a general idea.*

Recall Keel's linear relation

Example

When $n = 6$, we have $\epsilon_{12|35} = \epsilon_{13|25} = \epsilon_{15|23}$, i.e.,

$$\delta_{12,3456} + \delta_{124,356} + \delta_{126,345} + \delta_{1246,35}$$
$$= \delta_{13,2456} + \delta_{134,256} + \delta_{136,245} + \delta_{1346,25}$$
$$= \delta_{15,2346} + \delta_{145,236} + \delta_{156,234} + \delta_{1456,23}$$

Remark

From the exmaple above we easily see that we can replace some $\delta_{I,J}$, say $\delta_{12|3456}$, by $\epsilon_{13|25} - (\epsilon_{12|35} - \delta_{12|3456})$. Basicly we can replace $\delta_{I,J}$ by a sum of $(2^{n-3} - 1)$ many $(\pm)\delta_{I',J'}$.

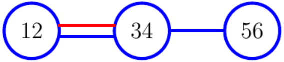
main idea behind

Example

- Given: $\delta^2_{12|3456} \cdot \delta_{1234|56}$
- corresponding tree see below
- use Keel's linear relation:

$$\delta^2_{12|3456} \cdot \delta_{1234|56} = \delta_{12|3456} \cdot \delta_{1234|56} \cdot (\epsilon_{13|25} - \delta_{124|356} - \delta_{126|345} - \delta_{1246|35})$$

- After cancellations caused by Keel's quadratic relation, we get $\delta^2_{12|3456} \cdot \delta_{1234|56} = -\delta_{12|3456} \cdot \delta_{1234|56} \cdot \delta_{124|356}$.
- obtain tree value/monomial value: -1



main idea behind

- For simpler monomials we can try to replace those higher powered factors using Keel's linear relation.
- And hopefully finally get a sum of clever monomials (maybe with a negative sign).
- Then the number of of clever monomials should be the absolute value of given monomial.
- Based on this idea, we have an algorithm for calculus for all tree monomials in $A^{n-3}(n)$.

sketch of the algorithm

- Input: a loaded tree with n labels and $n - 3$ edges
- Output: a natural number
- Transfer the loaded tree to a semi-redundancy tree.
- Calculate the sign of the tree value.
- Construct a redundancy forest from the semi-redundancy tree.
- Apply a recursive algorithm to this redundancy forest, obtaining the absolute tree value.
- Product of the sign and absolute value gives us tree value.
- We call it **forest algorithm**.
- Now we explain these terminologies.

semi-redundancy tree

- Given: loaded tree $LT = (V, E, h)$.
- Define a weight function $w : V \cup E \longrightarrow \mathbb{N}$ as the following:
- For any $v \in V$, $w(v) := \deg(v) + |h(v)| - 3$.
- Note that here in the degree of v, multiple edges are counted only once. And from the definition of loaded tree we know the weight of any vertex must be non-negative.
- For any $e \in E$, $w(e) :=$multiplicity of $e - 1$. Then we see the weight of any edge is also non-negative.
- semi-redundancy tree (of LT) $SRT := (LT, w)$.

semi-redundancy tree

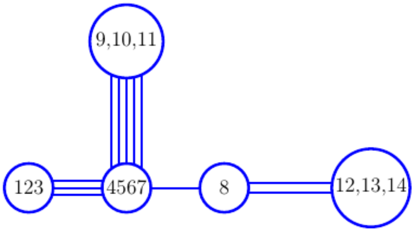


Figure: This is a loaded tree LT with 14 labels and 11 edges.

Let's figure out its semi-redundancy tree!

semi-redundancy tree

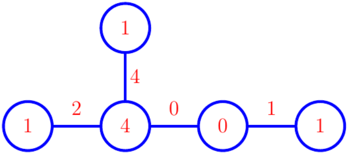


Figure: This is the semi-redundancy tree of the loaded tree LT, where the weight of vertices and edges are tagged in red. For simplicity we ommit labels for vertices here.

sign of the tree value

- Given a semi-redundancy tree $SRT = (LT, w)$.
- Let S be the sum of vertex weight (or edge weight) of LT.
- Sign of the tree value of loaded tree LT is $(-1)^S$.
- It's not hard to verify that weight sum of edges and of vertices are the same.

sign of the tree value

$$
\begin{aligned}
\sum_{v\in V} w(v) &= \sum_{v\in V} (\deg(v) + |h(v)| - 3) \\
&= \sum_{v\in V} \deg(v) + \sum_{v\in V} |h(v)| - 3\cdot|V| \\
&= 2\cdot|E| + n - 3\cdot|V| \\
&= 2\cdot|E| + n - 3\cdot|E| - 3 \\
&= n - 3 - |E| \\
\sum_{e\in E} w(e) &= \sum_{e\in E} (multiplicity(e) - 1) \\
&= \sum_{e\in E} multiplicity(e) - |E| \\
&= n - 3 - |E|
\end{aligned}
$$

Note that here in $|E|$ multiple edges are counted only once.

sign of the tree value

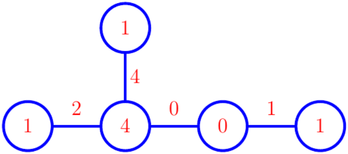


Figure: This is the semi-redundancy tree of the loaded tree LT, where the weight of vertices and edges are tagged in red. For simplicity we ommit labels for vertices here.

Sum of vertex weight $S = 1 + 4 + 1 + 0 + 1 = 7$, so the sign of LT value is $(-1)^7 = -1$.

redundancy forest

- How do we transfer a semi-redundancy tree (LT, w) (assume $LT = (V, E, h)$) to a redundancy forest?
- Replace each edge by a length-two edge with a new vertex connecting them which has the same weight as the replaced edge.
- Then we obtain the redundancy tree (of loaded tree LT) $RT := (V \cup E, E_1, h, w)$.
- Union of subtrees of RT such that no vertex has weight zero is the redundancy forest of LT.

redundancy forest

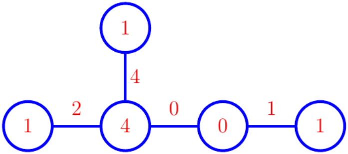


Figure: This is the semi-redundancy tree of the loaded tree LT, where the weight of vertices and edges are tagged in red. For simplicity we ommit labels for vertices here.

Let's figure out its redundancy forest!

redundancy forest

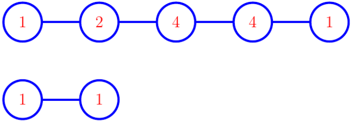


Figure: This is the redundancy forest *RF* of loaded tree *LT*, which contains two trees and the weight of vertices of *RF* are tagged in red.

Let's figure out how to apply the recursive algorithm to obtain the absolute value!

absolute value

- Let $RF = (V, E, h, w)$ be the redundancy forest of a loaded tree LT.
- We define the value of RF as the following:
- Pick any leaf of this forest, say $l \in V$, denote the unique parent of l as l_1.
- If $w(l) > w(l_1)$, return 0 and terminate the algorithm; otherwise, remove l from RF and assign weight $(w(l_1) - w(l))$ to l_1, replacing its previous weight. Denote the new forest as RF_1.
- Value of RF is the product of binomial coefficient $\binom{w(l_1)}{w(l)}$ and the value of RF_1.
- Base cases: whenever we reach a degree-zero vertex, if it has non-zero weight, return 0 and terminate the algorithm; otherwise, return 1.
- Product of absolute value of the corresponding redundancy forest of LT and sign of its tree value gives us the value of LT.

absolute value

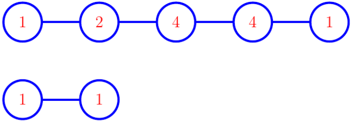
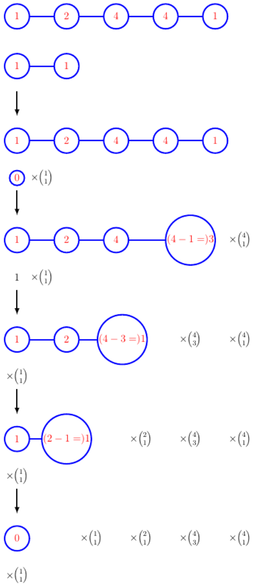


Figure: This is the redundancy forest *RF* of loaded tree *LT*, which contains two trees and the weight of vertices of *RF* are tagged in red.

Let's figure out how to apply the recursive algorithm to obtain the absolute value!

absolute value



tree value

- Finally we get the absolute value of RF as $1 \times \binom{1}{1} \times \binom{2}{1} \times \binom{4}{3} \times \binom{4}{1} \times \binom{1}{1} = 32$.
- Combining with the sign -1, we obtain the value of LT as -32.

forest algorithm

- Input: a loaded tree with n labels and $n-3$ edges
- Output: a natural number
- Transfer the loaded tree to a semi-redundancy tree.
- Calculate the sign of the tree value.
- Construct a redundancy forest from the semi-redundancy tree.
- Apply a recursive algorithm to this redundancy forest, obtain the absolute tree value.
- Prduct of the sign and absolute value gives us tree value.
- Implemented in Python; based on forest algorithm, computation of $\int \prod_{r=1}^{n-3} \epsilon_{i_r j_r | k_r l_r}$ is also implemented in Python.

well-definedness; termination

- Not hard to verify that at every step it does not matter from which leaf we start and base cases are well-defined. Hence forest algorithm is well-defined.
- Input is a tree with $(n-2)$ vertices maximally, the redundancy forest can have at most $(2n-5)$ vertices.
- The recursive algorithm strictly reduces the number of vertices by 1 in each step, obtaining a proper sub-forest.
- Hence the algorithm terminates and is well-defined.

correctness

Conjecture

Forest algorithm is correct.

Thank You