A tree-based algorithm on monomials in the Chow group of zero cycles in the moduli space of stable pointed curves of genus zero

Jiayue Qi *†

Abstract

The Chow group of zero cycles in the moduli space of stable pointed curves of genus zero is isomorphic to the integer additive group. With the help of a tree representation of the monomial, we are able to provide two different methods for computing this integer: one is algebraic, the other pictorial — the latter can be seen as an equivalent characterization of the former. Also, this tree representation enables us to further expedite the algorithm, via considering an extra property of the tree. With our result, we figure out the value of a specific type of monomials. In addition, we prove the equivalence between a multinomial coefficient identity and the theorem for the value of that specific type of monomials, using this graphical algorithm.

1 Introduction

Chow ring is a vital tool in intersection theory. In the Chow ring of some ambient variety, each subvariety is assigned to a class in this ring, which generalizes the degree of a curve in projective space of dimension two. For a standard reference in intersection theory, see [1]. See [2, Chapter 1] for more detailed explanations on Chow rings.

Multiplication of elements in the Chow ring corresponds to the intersection of cycles — formal sums of subvarieties — in the ambient variety. For a projective variety $X$ of dimension $k$, the Chow ring $A^*(X)$ is the direct sum of $k+1$ groups, each of them is composed of cycles of a fixed dimension. The group consists of cycles of codimension $r$ is usually denoted by $A^r(X)$, while $A^r(X)$ is always the zero group when $r$ is bigger than the dimension of $X$. In particular, the Chow group $A^k(X)$ is the group of cycles of dimension zero and, it is isomorphic to the integer additive group.

In this paper, we work in the Chow ring of a specific variety, namely the moduli space of stable $n$-pointed curves of genus zero, denoted by

*Doctoral Program “Computational Mathematics” W1214, Johannes Kepler University Linz.
†Research Institute for Symbolic Computation, Johannes Kepler University Linz.
\( M_n \). It is a smooth irreducible projective compactification of stable \( n \)-pointed genus zero curves. This moduli space was originally introduced by Knudsen and Mumford in their series of papers \([3], [4] \) and \([5] \). We denote the Chow ring of it by \( A^*(n) \) instead of \( A^*(\mathcal{M}_n) \), for simplicity. The paper \([6] \) is a very good reference on this Chow ring and can be helpful for a better understanding of the background of the tools that we will mainly use in our paper. We will explain soon the relation between their work and ours. The Chow rings of moduli spaces of curves were first defined by Mumford in \([7] \). In that paper, they defined the Chow ring of moduli space of curves of a general degree. Also, they worked out completely the case when the genus of stable curves is two. Faber Carel worked in the Chow rings of moduli space of stable curves of genus 3 and 4 in \([8] \) and \([9] \), respectively. They determined completely the situation for genus three, while results were gained partially for genus four. Izadi Elham continued on this path the investigation in \([10] \) for the genus five case. In \([11] \), the author presented the Chow ring of the moduli space of curves of genus zero. They provide a specific basis for the Chow groups and the intersection pairings between the basis. They consider the same ambient ring as we do, their paper may help if one wants to know more properties of our ambient ring, in addition to those considered by us.

From the general property of a Chow ring, we can directly get the following properties of our ambient ring. The moduli space \( \mathcal{M}_n \) has dimension \( n - 3 \). Hence we have

\[
A^*(n) = \bigoplus_{r=0}^{n-3} A^r(n),
\]

where \( A^r(n) \) is the Chow group of codimension \( r \). When \( r > n - 3 \), we have \( A^r(n) = \{0\} \). To be more precise, we work in the Chow group of codimension \( n - 3 \) — which is the group of cycles of dimension zero — and consider the isomorphism from this group to the integer additive group. We denote by \( f : A^{n-3}(n) \to \mathbb{Z} \) this isomorphism. In this paper, we focus on the following problem: Compute the corresponding integer under this isomorphism for any monomial in Chow group \( A^{n-3}(n) \) is a subproblem of the target problem. Later on, we actually invented two algorithms for this subproblem — one will be presented in this paper, and another one, actually more efficient, can be checked in \([13] \). One thing we should confess is that our new algorithms do not seem faster than what they provided in \([12] \). Nevertheless, we find this subproblem and the corresponding algorithms interesting on their own, which also may have further applications. Hence, we consider this subproblem on its own and try to write down what we discovered in this winding but exciting journey.
As is mentioned in the last paragraph, the algorithm that we will introduce in this paper is not more efficient than the algorithm — which is called forest algorithm — presented in [13], while both solve the same problem. However, we value the idea of this algorithm, not only because it is the bridge for us to reach the forest algorithm, but we see more potential of it as well. Especially we praise the idea of vertex-splitting, which is an essential step of the algorithm and will be explained more into details in Section 4.2. When we tried to prove the base case of the forest algorithm, the idea of vertex-splitting was essential. What is worthwhile to mention is that, in the proof of that base case, an identity about multinomial coefficient is needed — we manage to prove it as well. Hence, the algorithm led to another work of us, namely the discovery of this identity. We find the identity as well as its proof fascinating. There is a connection between our algorithm and this identity on the multinomial coefficient. Furthermore, we prove the equivalence between this base case and the identity, which gives the identity an equivalent characterization. Or maybe we should say that the identity hides some structural information of our main algorithm, algebraically. For detailed story on it, see Section 7.

In addition but not the least, the idea of viewing algebraic reduction as tree generating process stands on its own. Starting from the equivalent tree-representation of the given monomial, we complete the characterization of the monomial reduction process within the tree representations and we prove that these two reductions are theoretically equivalent. However, the tree-version reduction is more efficient in many situations. Moreover, it allows us to introduce a necessary condition for the monomial to have value zero, which we call balancing condition in the paper. This condition sufficiently improve the efficiency of the algorithm. Pure algebraic reduction is not capable of doing this. This equivalent characterization, namely viewing the algebraic reduction as some operations on the trees, is also the main spirit that we would like to convey.

In order to explain the basic idea of our characterization, we need to introduce some specific properties of the ambient ring. Let \( n \geq 3 \) be an integer and denote by \( N \) the set \( \{1, \ldots, n\} \). A bipartition \( \{I, J\} \) of \( N \) is called a cut if both \( I \) and \( J \) have cardinality bigger than one — then, we say that \( I \) and \( J \) are parts of this cut. For every cut \( \{I, J\} \), there is a hypersurface \( D_{I, J} \) in the ambient variety \( \mathcal{M}_n \). We denote by \( \delta_{I, J} \) the corresponding element of \( D_{I, J} \) in the Chow ring \( A^*(n) \) — note that \( \delta_{I, J} = \delta_{J, I} \) and \( D_{I, J} = D_{J, I} \). In [6], Keel Sean introduced a linear relation ( [6, Section 4, Theorem 1.(2)]) and a quadratic relation ( [6 Section 4, Theorem 1.(3)]) between the generators \( \delta_{I, J} \). This two relations are the main tools used by us and are called Keel’s linear relation and Keel’s quadratic relation in our paper, respectively.

These elements \( \delta_{I, J} \) generate the Chow group of codimension one, i.e., \( A^1(n) \). They also generate the whole ring, of course, when used as ring generators. Since we are working in a graded ring, it is not hard to see that the monomial \( \prod_{i=1}^{n-3} \delta_{I_i, J_i} \) is an element in the Chow group of codimension \( n - 3 \) and any monomial in this group should look like so — the product of \( n - 3 \) generators. Recall that the problem we want to solve is to compute the (integral) value of such monomials.
We solve the problem via the following steps. First, we transfer the monomial to its equivalent tree-representation. See [14, Section 2.2] and [13, Theorem 0.2.] for the one-to-one correspondence between a monomial in $A^*(n)$ and its tree representation. Second, we apply a vertex-splitting operation on this tree, collect all possible outputs. This gives us a set of new trees, each of them corresponds to some degree-reduced-by-one monomial. These monomials are exactly the summand monomials we obtain on the right hand side of the equation when we do the reduction algebraically. Next, we check if each of these new trees fulfills the balancing condition, we exclude those do not — since they have value zero. For more detailed explanation of the balancing condition, see Section 5.3. Then, we apply the vertex splitting process recursively to these new trees. We have a specific principle how the vertex splitting operation should be done; when this is obeyed, it is guaranteed that after finitely many steps, all the obtained monomials are squarefree. Another theorem (Theorem 3.4) tells us that squarefree monomial always has value one. Thence, we are able to directly “count” the absolute value of the given monomial accordingly. The sign depends on the number of recursive steps: being odd leads to the negative sign, while even gives the positive.

The structure of this paper is as follows. In Section 2, we introduce the Keel’s quadratic relation between the generators of the ambient ring. Also we introduce the tree representation of the monomials in the Chow group of cycles of dimension zero. In Section 3 we illustrate the algebraic reduction of any monomial in the ambient group with several examples. In the next section, we focus on one step of our algorithm, namely the vertex-splitting process. Then, in Section 4 we introduce the complete graphical algorithm, or maybe better called tree-based algorithm, which characterize the algebraic reduction in a graphical view. In Section 5 we complete the missing proofs from earlier context. In the end, we introduce an application of the main result in Section 7 which may lead to a deeper understanding of our algorithm.

2 Loaded trees

There are linear and quadratic relations between the generators of the ring $A^*(n)$. We say that the two generators $\delta_{I_1,J_1}$, $\delta_{I_2,J_2}$ (or the pair $(\delta_{I_1,J_1}, \delta_{I_2,J_2})$) fulfill Keel’s quadratic relation if the following four conditions hold:

1. $I_1 \cap I_2 \neq \emptyset$;
2. $I_1 \cap J_2 \neq \emptyset$;
3. $J_1 \cap I_2 \neq \emptyset$;
4. $J_1 \cap J_2 \neq \emptyset$.

And in this case, we have $\delta_{I_1,J_1} \cdot \delta_{I_2,J_2} = 0$. For example, when $n = 5$, $(\delta_{12,345}, \delta_{13,245})$ fulfills Keel’s quadratic relation but $(\delta_{12,345}, \delta_{23,145})$ does not, also then we have $\delta_{12,345} \cdot \delta_{13,245} = 0$. This fact comes directly from [6, Section 4, Theorem 1.(3)], we just use a different notation. Note that the index are abbreviated a bit: $\delta_{12,345}$ refers to $\delta_{(1,2),(3,4,5)}$ for instance. We will also use this abbreviation in the later context.
Inspired by this property, we know that if any two factors of the monomial fulfill this relation, the integral value of the given monomial will then be zero. So, the problem is solved in this case. Now we only need to focus on those monomials where no two factors fulfill this quadratic relation. We call those monomials tree monomials, since there is a one-to-one correspondence between these monomials and a type of tree (see Theorem 2.2), which we call loaded tree.

**Definition 2.1** ([13], Definition 0.1). A loaded tree with \( n \) labels and \( k \) edges is a tree \((V,E)\) together with a labeling function \( h : V \rightarrow 2^N \) and an edge multiplicity function \( m : E \rightarrow \mathbb{N}^+ \) such that the following three conditions hold:

1. \( \{h(v)\}_{v \in V, h(v) \neq \emptyset} \) form a partition of \( N \) — elements in \( N \) are called the labels of \( T \);
2. \( \sum_{e \in E} m(e) = k \);
3. For every \( v \in V \), \( \deg(v) + |h(v)| \geq 3 \), note that here multiple edges are only counted once for the degree of its incident vertex.

We say that two loaded trees are of the same type if they have the same number of labels and the same number of edges as well.

Note that when we want to verify if some given tree is a loaded tree (of any type) or not, we only need to verify the third condition in Definition 2.1, since the other two are just there to specify the type of the tree.

We define the monomial of a given loaded tree as follows. Since it is a tree, we will obtain two components when we remove any edge \( e \). We collect the labels in each component and form the sets \( I \) and \( J \), respectively. We say that \( \{I, J\} \) is the corresponding cut for edge \( e \), and that \( \delta_{I,J} \) is the corresponding factor for edge \( e \). Because of the third condition in Definition 2.1, one can check that no two edges can have the same corresponding cut. Therefore, given a loaded tree, there is a one-to-one correspondence between the edge set and the set of their corresponding cuts (or, the set of their corresponding factors). Hence, we can represent an edge by its corresponding cut. The monomial of a given loaded tree is \( \prod_{m=1}^{\delta_{I,J}} \delta_{I,J} \), where \( \delta_{I,J} \)s are the corresponding factors for the edges, and \( m \) refers to the number of edges of the given tree, where edges are counted with multiplicities. We see that once the loaded tree is given, its monomial is then uniquely determined. For multiple edges, its corresponding factor in the monomial has the power same as its multiplicity. We see in Figure 1 and Figure 2 for two examples of loaded trees and their monomials.

In [14], the authors introduce the one-to-one correspondence between the set of cuts and the phylogenetic tree. In the next theorem, we refer to the same result, while the only difference is that we allow some cuts to appear more than once; that is to say, we consider a multi-set of cuts. Consequently, the corresponding phylogenetic tree differs from that in [14] in the sense that we allow multiple edges; and the multiplicities of edges equal to the occurrences of cuts. In addition, the tree is called differently, in our paper loaded trees, while in [14] phylogenetic trees.
Figure 1: This is a loaded tree with 5 labels and 2 edges. $LT = (V, E, h, m)$, where $V = \{v_1, v_2, v_3\}$, $E = \{e_1, e_2\}$, $h(v_1) = \{1, 2\}$, $h(v_2) = \{3\}$, $h(v_3) = \{4, 5\}$, $m(e_1) = m(e_2) = 1$. Its monomial is $\delta_{12,345} \cdot \delta_{123,45}$.

Figure 2: This is a loaded tree with 6 labels and 4 edges. $LT = (V, E, h, m)$, where $V = \{v_1, v_2, v_3, v_4\}$, $E = \{e_1, e_2, e_3\}$, $h(v_1) = \{1, 2\}$, $h(v_2) = \emptyset$, $h(v_3) = \{5, 6\}$, $h(v_4) = \{3, 4\}$, $m(e_1) = m(e_3) = 1$, $m(e_2) = 2$. Its monomial is $\delta_{12,3456} \cdot \delta_{34,1256} \cdot \delta_{2,56,1234}$.

**Theorem 2.2.** [14, Section 2.2] There is a one-to-one correspondence between tree monomials $M = \prod_{i=1}^{k} \delta_{I_i, J_i}$ and loaded trees with $n$ labels and $k$ edges, where $I_i \cup J_i = N$ for all $1 \leq i \leq k$.

**Remark 2.3.** Note that there is an extremal case for each $n \geq 3$, namely the tree containing only one vertex with labeling set $N = \{1, \cdots, n\}$. Such a loaded tree has no edges, is an element in the Chow group $A^0(n)$. We call its corresponding monomial the empty monomial, denote it by $\emptyset$; it is also a tree monomial. We see that a loaded tree with $n$ labels and $k$ edges represents a tree monomial in the Chow group $A^k(n)$. We often denote the corresponding loaded tree of a given tree monomial $M$ as $T_M$; and the corresponding tree monomial of a given loaded tree $T$ as $M_T$.

We have an algorithm realizing this correspondence, namely converting from the monomial to loaded tree, which we call tree algorithm in [13]. But we will not go into details on this algorithm here. We also do not consider here the case where the number of edges is three less than the number of labels. In the algebraic language, a proper loaded tree represents some tree monomial in $A^{n-3}(n)$ for some $n \in \mathbb{N}^+$. Given a proper loaded tree, we want to calculate its value. This is another way of expressing our goal.

In the sequel, we introduce the weight function of a loaded tree, which is usually denoted by $w$. This concept will be helpful in stating our al-
The weight function \( w : V \cup E \rightarrow \mathbb{N} \) is defined as:

\[
w(v) := \deg(v) + |h(v)| - 3 \quad \text{for all } v \in V \quad \text{and} \quad w(e) := m(e) - 1 \quad \text{for all} \quad e \in E.
\]

One additional remark is that: it is not hard to verify the following identity about the weight function for a proper loaded tree — we call it weight identity:

\[
\sum_{v \in V} w(v) = \sum_{e \in E} w(e).
\]

Up to here one may wonder: how indeed can we calculate the value of a proper monomial? We are still missing one important tool. Now is the stage time for Keel’s linear relation, to help us out of the sludge.

### 3 Linear reduction

In this section, we describe how to figure out the value of a given proper monomial, using Keel’s linear relation and Keel’s quadratic relation, via algebraic reductions. Keel’s linear relation was originally proved in [6, Theorem 1.(2)]. We state exactly the same content as follows, but in different notations.

**Fact 3.1 (Keel’s linear relation, [6] Theorem 1.(2)).** Denote by \( \epsilon_{ij} | kl : = \sum_{i,j \in I, k,l \in J} \delta_{I,J} \). Then we have the equality relations \( \epsilon_{ij} | kl = \epsilon_{il} | kj = \epsilon_{ik} | jl \). We call it Keel’s linear relation.

Let us see a concrete example on it.

**Example 3.2.** 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 3.3.** From the example above we see that we can substitute some \( \delta_{I,J} \), say \( \delta_{12,3456}, \) by \( \epsilon_{13}|25 - (\epsilon_{12}|35 - \delta_{12,3456}) \). Basically we can replace \( \delta_{I,J} \) by a sum of \((2^{n-3} - 1)\) many \((\pm)\delta_{I',J'}\)s.

It is possible that the follow theorem is known, but because of the lack of proper reference, we will prove it for the sake of completeness. We postpone the proof to Section 6 since our proof method will need some
more tools that will be introduced later on — and of course, those tools are independent from this theorem. For now, we continue our story by accepting this result.

**Theorem 3.4.** If all factors are distinct in the tree monomial $M := \Pi_{i=1}^{n-3} \delta_{I_i,J_i}$, where $I_i \cup J_i = N$ for all $1 \leq i \leq n - 3$. Then $\int(M) = 1$ and we call this type of tree monomial clever monomial and its corresponding loaded tree clever tree.

**Remark 3.5.** Notice that clever trees are by definition proper. For clever trees, we know that they have value one. What about non-clever trees? We would like to use Keel’s linear relation to reduce them, hoping to get one occurrence of $\delta$ together with $\epsilon$ to finally get a sum of clever monomials (maybe with a negative sign). Let us see a concrete example for a better understanding of this idea.

**Example 3.6.** Given $\delta_{12,3456}^2 \cdot \delta_{1234,56}$, which is a tree monomial. We want to calculate its value. Now, apply Keel’s linear relation, and replace one occurrence of $\delta_{12,3456}$ by $\epsilon_{13|25} - (\epsilon_{12|35} - \delta_{12,3456}):$

\[
\delta_{12,3456}^2 \cdot \delta_{1234,56} = \delta_{12,3456} \cdot \delta_{1234,56} \cdot (\epsilon_{13|25} - \delta_{124,356} - \delta_{126,345} - \delta_{1246,35}).
\]

Since any summand $\delta_{I,J}$ of $\epsilon_{13|25}$ has $1,3 \in I$ and $2,5 \in J$. Therefore, it together with $\delta_{12,3456}$ fulfills Keel’s quadratic relation and we obtain that $\delta_{12,3456} \cdot \delta_{I,J} = 0$. Consequently, we have $\delta_{12,3456} \cdot \epsilon_{13|25} = 0$. Hence we have:

\[
\delta_{12,3456}^2 \cdot \delta_{1234,56} = \delta_{12,3456} \cdot \delta_{1234,56} \cdot (-\delta_{124,356} - \delta_{126,345} - \delta_{1246,35}).
\]

One can check that the two pairs ($\delta_{1234,56}, \delta_{126,345}$) and ($\delta_{1234,56}, \delta_{1246,35}$) fulfill Keel’s quadratic relation. Hence both products are zero. Therefore, we have

\[
\delta_{12,3456}^2 \cdot \delta_{1234,56} = \delta_{12,3456} \cdot \delta_{1234,56} \cdot \delta_{124,356}.
\]

We get one clever monomial with a negative sign, so the tree value (monomial value) is $-1$.

The process is to replace one factor in the given monomial via some Keel’s linear relation and then expand it into a sum of monomials. Then cancel all the terms that will lead to zero because of Keel’s quadratic relation. This process is called a linear reduction. In the sequel, we see an example where the linear reduction is needed more than once, in order to finally get a sum of clever monomials (maybe with a negative sign).

**Example 3.7.** Let $M := \delta_{123,4567}^2 \cdot \delta_{12345,67} \in A^4(7)$ be the given monomial. We use Keel’s linear relation for $n = 7$, replacing $\delta_{123,4567}$ via $\epsilon_{14|26} = \epsilon_{14|26}$. Then, we obtain

\[
\delta_{123,4567}^2 \cdot \delta_{12345,67} = \delta_{123,4567} \cdot \delta_{12345,67} \cdot (\epsilon_{14|26} - (\epsilon_{12|46} - \delta_{123,4567})).
\]

Then we see that each summand of $\epsilon_{14|26}$ fulfills Keel’s quadratic relation together with $\delta_{123,4567}$, hence we have $\epsilon_{14|26} \cdot \delta_{123,4567} = 0$. Hence we have:

\[
\delta_{123,4567}^3 \cdot \delta_{12345,67} = \delta_{123,4567} \cdot \delta_{12345,67} \cdot (-(\epsilon_{12|46} - \delta_{123,4567}))
\]

\[
= \delta_{123,4567}^2 \cdot \delta_{12345,67} \cdot (-(\epsilon_{12|46} - \delta_{123,4567}) - \delta_{127,3456} - \delta_{1235,467} - \delta_{1237,456} - \delta_{1257,346} - \delta_{12357,46}).
\]
Then we need to exclude those summands of $\epsilon_{12|46}$ which fulfill Keel’s quadratic relation with any factor(s) of $M$ – here it refers to $\delta_{12345,67}$ and $\delta_{123,4567}$. After the exclusion, we obtain that

$$\delta^3_{123,4567} \cdot \delta_{12345,67} = \delta^2_{123,4567} \cdot \delta_{12345,67} \cdot (-\delta_{12,34567} - \delta_{1235,467})$$

(2)

Then we need to detect those factors in the summands of $\epsilon$.

We obtain:

$$\delta^2_{123,4567} \cdot \delta_{12345,67} \cdot \delta_{12,34567} = \delta_{123,4567} \cdot \delta_{12345,67} \cdot \delta_{12,34567} \cdot (\epsilon_{13|46} - (\epsilon_{13|46} - \delta_{123,4567}))$$

(3)

Then, for the same reason, we can omit $\epsilon_{14|36}$. Hence we have

$$\delta^2_{123,4567} \cdot \delta_{12345,67} \cdot \delta_{12,34567} = \delta_{123,4567} \cdot \delta_{12345,67} \cdot \delta_{12,34567} \cdot (\epsilon_{13|46} - (\epsilon_{13|46} - \delta_{123,4567}))$$

(4)

Now we need to detect those factors in the summands of $\epsilon_{13|46}$ which fulfills Keel’s quadratic relation together with any factor among $\delta_{123,4567}$, $\delta_{12345,67}$ and $\delta_{12,34567}$. After the cancellations, only one of them is left.

$$\delta^2_{123,4567} \cdot \delta_{12345,67} \cdot \delta_{12,34567} = \delta_{123,4567} \cdot \delta_{12345,67} \cdot \delta_{12,34567} \cdot (\epsilon_{13|46} - (\epsilon_{13|46} - \delta_{123,4567}))$$

(5)

We can see that we obtain a clever monomial with a negative sign, the value of which is $-1$. For the other monomial on right hand side of Equation (2) — $\delta^2_{123,4567} \cdot \delta_{12345,67} \cdot \delta_{1235,467}$ — we use $\epsilon_{12|45} = \epsilon_{14|25}$ to reduce the powered factor. Then we get

$$\delta^2_{123,4567} \cdot \delta_{12345,67} \cdot \delta_{1235,467} = \delta_{123,4567} \cdot \delta_{12345,67} \cdot \delta_{1235,467} \cdot (\epsilon_{14|25} - (\epsilon_{12|45} - \delta_{123,4567}))$$

(6)

Here we obtain a clever monomial with negative sign, the value of which is $-1$. Substituting these two values back to Equation (2) we obtain that the value of the given monomial $M := \delta^3_{123,4567} \cdot \delta_{12345,67}$ is 2.
Remark 3.8. From the above two examples, we see clearly at least one thing: whenever we replace one occurrence of $\delta_{I,J}$ by some $\epsilon_{ijkl} - (\epsilon_{ij|kl} - \delta_{I,J})$, we can directly omit $\epsilon_{ik|jl}$, since any summand of it fulfills the Keel’s quadratic relation with $\delta_{I,J}$ and there is at least one occurrence of $\delta_{I,J}$ still left in the remaining part of the monomial. Hence from now on we will only say that we replace $\delta_{I,J}$ by $-(\epsilon_{ij|kl} - \delta_{I,J})$.

Also, because of the same reason, whenever we do one-time linear reduction, we obtain a negative sign on the right hand side. Therefore, how many times of linear reduction we use decides the sign for the value of the given monomial — odd times gives a negative sign while even times leads to a positive sign. In the sequel, we only need to worry about how to obtain the absolute value of the given monomial.

4 Tree-version linear reduction

Non-squarefree tree monomials are in one-to-one correspondence with non-clever trees. After we do linear reductions to a given tree monomial, the obtained monomials in the expanded form on the right hand side of the equation are all tree monomials — because all monomials containing a pair of factors fulfilling Keel’s quadratic relation have been cancelled out during the reduction process. Because of the one-to-one correspondence between tree monomials and loaded trees, this process can parallelly be viewed as a tree transforming/generating process. This is an equivalent process of the (algebraic) linear reduction. We call it the tree-version linear reduction.

In this section, we will describe it into details. Then, in Section 5 we prove that this tree-based algorithm is indeed an equivalent characterization of the algebraic linear reduction described in Section 3. We view the algebraic reduction as some operations on the loaded tree corresponding to the given monomial. Now we will try to explain the tree-version linear reduction step by step — translating the algebraic language to graphical language.

- Non-squarefree tree monomials corresponds precisely to loaded trees with multiple edge(s).
- In the linear reduction process, first we pick some $\delta_{I,J}$ that has power higher than one in the given monomial. In the graphical language, this operation means that we pick a multiple edge $e$ whose corresponding factor is $\delta_{I,J}$, i.e., whose corresponding cut is $\{I, J\}$.
- Then we decide on which Keel’s linear relation $\epsilon_{ijkl} = \epsilon_{ikjl}$ to use, in order to replace the factor $\delta_{I,J}$. In graphical language, it says that we pick $i, j \in I$ and $k, l \in J$.

We need to pause here, so as to explain this quadruple choosing step. In the algebraic reduction, it seems that we are allowed to choose whatever quadruple for Keel’s linear relation as long as we can replace the factor — we will see later on that it is not true. We have a specific requirement in the pictorial algorithm — we require a specific way of choosing the quadruple, namely a proper choice.
4.1 Proper choice of the Keel’s linear quadruple

Actually we can pick any quadruple \((i, j, k, l)\) such that \(i, j \in I\) and \(k, l \in J\) for Keel’s linear relation; the correctness is guaranteed in theory. We call such a quadruple Keel’s linear quadruple. However, we want to make sure that no summand in \(- (\epsilon_{ij|kl} - \delta_{I,J})\) divides \(M\). We call it summand distinction property of the quadruple \((i, j, k, l)\) or of the set \(\{i, j, k, l\}\), simply because it means any summand in \(- (\epsilon_{ij|kl} - \delta_{I,J})\) is distinct from all factors of \(M\). Why do we need this property?

Suppose that we want to replace one occurrence of \(\delta_{I_1,J_1}\) in \(M = \delta_{I_1,J_1} \cdots \delta_{I_q,J_q}\) by \(- (\epsilon_{ij|kl} - \delta_{I_1,J_1})\) and \((i, j, k, l)\) fulfills the summand distinction property. W.l.o.g., assume that \(- (\epsilon_{ij|kl} - \delta_{I_1,J_1}) = \sum_{s=1}^{t} \delta_s\). Then we obtain the following equation:

\[
\delta_{I_1,J_1} \cdots \delta_{I_q,J_q} = \delta_{I_1,J_1}^{p-1} \cdots \delta_{I_q,J_q} \cdot (- (\epsilon_{ij|kl} - \delta_{I_1,J_1}))
\]

\[= \delta_{I_1,J_1}^{p-1} \cdots \delta_{I_q,J_q} \cdot \sum_{s=1}^{t} \delta_s \]

\[= \delta_{I_1,J_1}^{p-1} \cdots \delta_{I_q,J_q} \cdot \delta_1 + \cdots + \delta_{I_1,J_1}^{p-1} \cdots \delta_{I_q,J_q} \cdot \delta_t\]

Now we focus on any monomial on right hand side of the equation, say \(M_r := \delta_{I_1,J_1}^{p-1} \cdots \delta_{I_q,J_q} \cdot \delta_r\). Since \(\delta_r\) does not divide \(M\), \(\delta_r\) is distinct from all factors of \(M\). Therefore, no factor can have a higher power after the replacement. Then, we can really guarantee that after finitely many times linear reduction, we can indeed obtain a (maybe negative) sum of clever monomials, which, in value, is equal to the given monomial. The analysis above also tells us that we should also obey this rule when doing the algebraic reduction, otherwise we may not finish the reduction (until a sum of only clever monomials) after finitely many steps.

How can we guarantee that the chosen quadruple satisfies summand distinction property? By a proper choice of the Keel’s linear quadruple. In order to explain what is this “proper choice”, we need the concept of cluster first.

**Definition 4.1.** We say that \(cl \subset N\) is a cluster of vertex \(v\) if and only if one of the following two conditions holds:

1. \(cl\) is a one-element-subset of \(h(v)\).

2. \(cl\) is the collection of labels in one component when we remove vertex \(v\) and all its incident edges.

Note that here \(h\) denotes the labeling function of the loaded tree to which \(v\) belongs. If a cluster has cardinality one, we say that it is a singleton; otherwise, we say that it is a proper cluster.

**Remark 4.2.** We observe that collecting all parts of cuts of a loaded tree gives us exactly the collection of proper clusters of all vertices.

**Remark 4.3.** It is not hard to check that the above two cases are disjoint for any loaded tree. When a cluster of vertex \(v\) fulfills the first condition, it contributes one to the cardinality of \(h(v)\). When a cluster of vertex \(v\) fulfills the second condition, it contributes one to the degree of \(v\). Recall the expression \(\text{deg}(v) + |h(v)|\) in the third item of Definition 2.1. For a
vertex, each adjacent edge corresponds to a cluster of it, and each of its labels corresponds to a cluster of it as well.

For a better idea of this definition, let us see what are the clusters for vertices of the loaded tree in Figure 2.

- Clusters for $v_1$: $\{1\}$, $\{2\}$, $\{3, 4, 5, 6\}$.
  - Singletons for $v_1$: $\{1\}$, $\{2\}$.
  - Proper clusters for $v_1$: $\{3, 4, 5, 6\}$.
- Clusters for $v_2$: $\{1, 2\}$, $\{3, 4\}$, $\{5, 6\}$.
  - Singletons for $v_2$: none.
  - Proper clusters for $v_2$: $\{1, 2\}$, $\{3, 4\}$, $\{5, 6\}$.
- Clusters for $v_3$: $\{1, 2, 3, 4\}$, $\{5\}$, $\{6\}$.
  - Singletons for $v_3$: $\{5\}$, $\{6\}$.
  - Proper clusters for $v_3$: $\{1, 2, 3, 4\}$.
- Clusters for $v_4$: $\{1, 2, 5, 6\}$, $\{3\}$, $\{4\}$.
  - Singletons for $v_4$: $\{3\}$, $\{4\}$.
  - Proper clusters for $v_4$: $\{1, 2, 5, 6\}$.

Now we are prepared for the concept of a “proper choice” of Keel’s linear quadruple. Assume w.l.o.g. that when we remove edge $e = \{v_1, v_2\}$, vertex $v_1$ is in the component where all labels collected to be $I$ and $v_2$ is in the component where all labels collected to be $J$. We call the corresponding components $\text{Component-}I$ and $\text{Component-}J$, respectively. We choose the quadruple $(i, j, k, l)$ such that $i, j \in I$ are from two distinct clusters of $v_1$ and $k, l \in J$ are from two distinct clusters of $v_2$. We call this way of choosing $i, j, k, l$ a proper choice. And we call the corresponding quadruple $(i, j, k, l)$ a proper quadruple of the edge $e$, or of the cut $\{I, J\}$; we call $\{i, j, k, l\}$ a proper quadruple set of $e$ or of $\{I, J\}$. Note that we are always able to choose a proper quadruple for any edge of some loaded tree, since apart from the cluster connected by edge $e$ to $v_2$ (or $v_1$), $v_1$ (or $v_2$) has at least two more clusters, by the third condition of Definition 2.1.

Let us continue with focusing on the loaded tree in Figure 2. For this loaded tree, suppose that we want to replace one occurrence of edge $e_2$ (i.e., the cut $\{\{1, 2, 3, 4\}, \{5, 6\}\}$). We should choose $i, j$ from $\{1, 2, 3, 4\}$ and $k, l$ from $\{5, 6\}$ for Keel’s linear reduction. We see that $\{1, 3, 5, 6\}$ is a proper choice, but neither $\{1, 2, 5, 6\}$ nor $\{3, 4, 5, 6\}$ is a proper choice. We claim that any proper quadruple fulfills the summand distinction property. We leave the proof of this to Section 6. For now, we continue by accepting this statement.

So when we have a loaded tree, we should decide on which multiple edge to reduce, then we should decide on a proper quadruple to do the reduction. In the sequel, we introduce the next step: how to directly tell which loaded trees or tree monomials will be generated on the right hand side of the equation in the algebraic linear reduction (see Section 3) but using a graphical method. For this, we need to introduce the concept of vertex splitting.
4.2 Vertex-splitting

Before we can explain this concept, we need to define branches of a vertex first. Let \( v \) be a vertex in loaded tree \( T \). Removing vertex \( v \) but not its adjacent edges gives us \( \deg(v) \) many parts. Each part is a structure of a tree but lacking a vertex. We make \( \deg(v) \) many copies of vertex \( v \), and concatenate it to each of these parts at the place where a vertex is missing. Then we obtain \( \deg(v) \) many trees, we call them branches of \( v \). We call the copy of \( v \) the special vertex in each of these trees. We say that a branch with the special vertex \( v \) is attached to some vertex \( v' \) (of some tree \( T' \)) if we add an extra edge \( e' \) between \( v \) and \( v' \), and then contract it (two vertices merged into a new vertex \( v_1 \)), and then set the labeling set of \( v_1 \) to be the labeling set of \( v' \).

Note that each branch corresponds to a cluster of \( v \). Each cluster of \( v \) corresponds either to a branch of \( v \) or to a label of \( v \). The above statement is just to make things precise in a formal sense. However, in our concrete operation, we can think of the branch simply as the structure of a tree but lacking that special vertex. And when we attach a branch to some vertex \( v' \) of another tree \( T' \), we can just view it as to concatenate \( T' \) to the branch at vertex \( v' \), to the endmost of that branch where a vertex is missing. From now on, for convenience, we will use this simplified concept of branch and branch-attaching.

In order to better understand the concept, we see an example. Let us turn our focus back to vertex \( v_2 \) in loaded tree in Figure 2. See Figure 3 for its branches. See Figure 4 for an example of branch-attaching. Now we are prepared for the vertex splitting process.

We explain this process as a series of operations on a loaded tree. Recall that the weight function is defined as \( \deg(v) + |h(v)| - 3 \) for each vertex \( v \).

1. Input: a loaded tree \( T_1 = (V_1, E_1, h_1, m_1) \); a proper quadruple set
Then we have Remark 4.5. Let $T$ be the weight of vertex $v_1$ with the corresponding cut $\{I,J\}$. 

2. Output: NULL or a loaded tree $T_2 = (V_2, E_2, h_2, m_2)$. 

3. If both $v_1$ and $v_2$ have zero-weight, return NULL. Otherwise, pick one of the incident vertices of $e$ with non-zero weight — assume w.l.o.g. that vertex $v_1$ is chosen. Let $I$ be so that $v_1$ is in Component-$I$ of $T_1$. Naturally, $v_2$ is in Component-$J$. 

4. Replace $v_1$ by two vertices $v_1'$ and $v_1''$ with a single edge $e' = \{v_1', v_1''\}$ connecting them. 

5. Set the labeling sets of $v_1'$ and that of $v_1''$ so that $h_2(v_1') \cup h_2(v_1'') = h_1(v_1) \cap (Q \cap I) \subset h_2(v_1)$; 

   (**) However, note that if $v_1$ has no other branches except for the ones that contain any label in $Q \cap I$ or edge $e$, then $|h_2(v_1')| \geq 1$ must hold. 

6. Arrange the branches of $v_1$ in the following way: 

   Among the branches of $v_1$, those containing any label in $Q \cap I$ are attached to $v_1'$ and the one that contains labels in $Q \cap J$ is attached to $v_1''$ — it is not hard to check that the two labels in $Q \cap J$ are in the same branch of $v_1$. The branch containing $e$ should be modified slightly: the multiplicity of edge $e$ in this branch gets reduced by one, then gets attached to $v_1''$. 

   For any other branches of $v_1$ in $T_1$, it can be either attached to $v_1'$ or $v_1''$. 

   (***) However, note that if $h_2(v_1') = \emptyset$, then we must attach at least one branch to $v_1''$. 

Remark 4.4. The requirements (***) and (***') are there to guarantee that vertex $v_1''$ gets at least one label or at least one branch (other than the one contains edge $e$). In this way, we guarantee that $v_1''$ fulfills the third condition in Definition 2.1. Since we require that the cluster of each of the two labels in $Q \cap I$ is either attached to $v_1'$ (as a branch), or put into the labeling set of $v_1'$ (as a singleton). Also we have $\{v_1', v_1''\} \in E(T_2)$. Hence vertex $v_1'$ has at least three clusters, it also fulfills the third condition in Definition 2.1. Therefore, $T_2$ is a loaded tree. 

Remark 4.5. Let $w_1$ be the weight function for $T_1$ and $w_2$ be that for $T_2$. Then we have 

$$w_2(v_1') + w_2(v_1'') = (\deg(v_1') + |h_2(v_1')| - 3) + (\deg(v_1'') + |h_2(v_1'')| - 3)$$

$$= (\deg(v_1') + \deg(v_1'')) + (|h_2(v_1')| + |h_2(v_1'')|) - 6$$

$$= (\deg(v_1) + 2) + |h_1(v_1)| - 6$$

$$= \deg(v_1) + |h_1(v_1)| - 4$$

$$= \deg(v_1) + |h_1(v_1)| - 3 - 1$$

$$= w_1(v_1) - 1$$

We see from the above reasoning that the vertex-splitting process indeed require the weight of vertex $v_1$ to be non-zero; otherwise, it cannot be split. Actually, if there is a multiple edge $e = \{v_1, v_2\}$ in some proper
loaded tree, both \( v_1 \) and \( v_2 \) being weight-zero leads to the value of this tree being zero. Therefore, if none of the weights is non-zero, we can immediately output zero as the wanted value. We will prove this conclusion in Section 6. This is the reason why we assume that \( v_1 \) has non-zero weight in the input statement. If \( v_1 \) has zero weight, while \( v_2 \) has non-zero weight, we can simply rename the vertices.

**Remark 4.6.** It is not hard to see that \( T_1 \) and \( T_2 \) have the same set of labels and the same number of edges. Hence, \( T_2 \) is of the same type with \( T_1 \). Thence when \( T_1 \) is proper, \( T_2 \) is also proper. After these analysis, we see that the above stated process is an algorithm — it terminates, and returns a loaded tree or NULL. We say that in this algorithm, vertex \( v_1 \) is split into vertices \( v'_1 \) and \( v''_1 \).

Because of the summand distinction property, the weight sum of edges of \( T_2 \) is always one less than that of \( T_1 \). Therefore, if we recursively apply this process, then after finitely many steps, we will obtain only clever tree. This observation provides us an idea on calculating the value of a given tree monomial.

In the above algorithm, output is just one loaded tree. We observe that we actually have some freedom at several steps:

1. If both \( v_1 \) and \( v_2 \) have non-zero weight, then we can split any of them.
2. We could also have some freedom on how to set up the labeling function for \( v'_1 \) and \( v''_1 \), respectively — as long as condition \((\ast)\) holds.
3. Also, we have some freedom on the arrangements of branches of \( v'_1 \) and those of \( v''_1 \) — as long as condition \((\ast\ast)\) holds.

When we consider all these freedom, and collect all the possibly generated loaded trees, we obtain the tree-version linear reduction algorithm. This algorithm does the same thing and should give us the same result with the algebraic reasoning which is introduced in Section 3 — after a slight modification. We postpone the correctness proof of it to Section 6.

### 5 Reduction algorithm

#### 5.1 Tree-version linear reduction algorithm

In this subsection, we explain the tree-version linear reduction algorithm, see Algorithm 1.

**Remark 5.1.** Note that we could also obtain NULL in some steps, in that case, we simply do not add anything to \( ST \). We modify the input loaded tree into a proper tree monomial \( M \), the output into the negative formal sum of all corresponding monomials of the loaded trees in \( ST \). And add one step at the very beginning — using tree algorithm, transferring \( M \) to \( T_M \), one step at the very end — transferring loaded trees in \( ST \) into their corresponding monomials. Then the above algorithm does the same thing and should give us the same result with the algebraic reasoning/algorithm which is introduced in Section 3. By Theorem 2.2, the function of Algorithm 1 is equivalent to the modified version stated above.
Algorithm 1: tree-version linear reduction

**input:** a proper loaded tree $T_M$ (the corresponding loaded tree of proper monomial $M$); a multiple edge $e = \{v_1, v_2\}$ (with corresponding cut $\{I, J\}$ of $T_M$; a proper quadruple set $\{i, j, k, l\}$ of $e$, or equivalently, of $\{I, J\}$ such that $i, j \in I$ and $k, l \in J$.

**output:** loaded trees whose corresponding monomials are the ones that survive after the (algebraic) linear reduction on $M$ which uses Keel’s linear reduction on the relation $\epsilon_{ij|kl} = \epsilon_{ik|jl}$.

1. $w_1 \leftarrow$ weight of $v_1$;
2. $w_2 \leftarrow$ weight of $v_2$;
3. If $w_1 = 0$ and $w_2 = 0$ then return $\emptyset$

ST $\leftarrow$ the set of all loaded trees that can be obtained from $T_M$ with edge $e$ and the proper quadruple set $\{i, j, k, l\}$, after applied to the vertex splitting algorithm — either by splitting vertex $v_1$, or by splitting vertex $v_2$ — in the set ST;

return ST

Remark 5.2. Note that our algorithm is applicable also to non-proper monomials (trees). However, in our problem, the input is always a proper monomial (tree), and we always obtain one or more proper monomials (loaded trees) after each linear reduction.

Now we see an example for a better understanding of the above algorithm, as well of the vertex splitting algorithm stated earlier.

Example 5.3. Given a proper loaded tree $T_1$ as in Figure 5. Let us follow the above vertex-splitting process, see what we will obtain. We pick a multiple edge $e_2$ and a proper quadruple set $\{1, 7, 5, 6\}$. We calculate the weight of its incident vertices, find out that the weight of $v_3$ is zero while that of $v_2$ is 2 (non-zero). Hence we can only choose $v_2$ to split, and note that $v_2$ is in Component-$\{1, 2, 3, 4, 7, 8\}$ of $T_1$.

We should bipartition the labeling set of $v_2$ such that

$$\{7, 8\} \cap (\{1, 2, 3, 4, 7, 8\} \cap \{1, 7, 5, 6\}) = \{7\} \subset h_2(v'_2)$$

holds. So if we follow the instructions in Algorithm 1 we have two options in this step:

1. $h_2(v'_2) = \{7\}$ and $h_2(v''_2) = \emptyset$;
2. $h_2(v'_2) = \{7, 8\}$ and $h_2(v''_2) = \emptyset$.

Then, based on the above two choices, we distribute the branches of $v_2$ to be the branches of $v'_2$ or those of $v''_2$. From the principle we know that the branch containing 1 should be attached to $v'_2$ and the branch containing 5 or 6 should be attached to $v''_2$; edge multiplicity of $e_2$ should be reduced by one. The remaining branch — the branch containing labels
Figure 5: This a proper loaded tree $T_1$ with 8 labels and 5 edges. We want to reduce edge $e_2$ with the proper quadruple set $\{1, 7, 5, 6\}$, via splitting the vertex $v_2$.

3 and 4 can be attached to either $v'_2$ or $v''_2$ in the first label-distribution option; but it must be attached to $v''_2$ in the second label-distribution option because of the (***) requirement. Hence, we obtain in total three loaded trees, after applying Algorithm 1.

Figure 6 shows these loaded trees. It is not hard to check that each of them is still a loaded tree, and is of the same type with $T_1$. Note that vertices in $T_1$ and those in the new trees do not necessarily have the same names/symbols. We keep most of them the same, just to make it easier to compare and understand the vertex-splitting process. This also applies to some later context or examples.

Figure 6: The set of these three loaded trees is the output of Algorithm 1 - applied to the loaded tree $T_1$ with multiple edge $e_2$ (in Figure 5) and the proper quadruple set $\{1, 7, 5, 6\}$. The edge which gets reduced is marked in green. Vertex $v_2$ is split into $v'_2$ and $v''_2$. The new edge is denoted as $e'$. Each of them is a loaded tree with 8 labels and 5 edges.

In the sequel, we look back on our examples in Section 3. We will apply the tree-version linear reduction algorithm and see if we will obtain the same result as if we apply the algebraic method.

**Example 5.4.** See Figure 7 for the corresponding loaded tree $T$ of monomial $\delta_{12,3456} \cdot \delta_{1234,56}$ in Example 3.6. Now we want to apply the tree-version linear reduction algorithm to it. We pick the multiple edge $e_1$ to reduce. Its corresponding cut is $\{\{3, 4, 5, 6\}, \{1, 2\}\}$. The proper linear reduction quadruple set we pick here is $\{3, 5, 1, 2\}$. By an easy calculation we know that the weight of $v_1$ is zero and that of $v_2$ is 1. Therefore, we can only split vertex $v_2$ — no freedom of choice here. And $v_2$ is in Component-$\{3, 4, 5, 6\}$ of $T$.

First we split vertex $v_2$ into $v'_2$ and $v''_2$ such that $\{3, 4\} \cap (\{3, 4, 5, 6\} \cap \{3, 5, 1, 2\}) = \{3\} \subset h_2(v'_2)$. And observe that $v_2$ has only two branches,
Figure 7: This is the corresponding loaded tree of monomial \( M = \delta_{12,3456}^2 \cdot \delta_{1234,56} \). We want to reduce edge \( e_1 \) with the proper quadruple set \( \{3, 5, 1, 2\} \).

Figure 8: This is the only loaded tree that can be obtained after applying the tree-version linear reduction algorithm to the loaded tree in Figure 7 with edge \( e_1 \) and the quadruple set \( \{3, 5, 1, 2\} \). The new edge is denoted by \( e' \).

one contains 5, the other contains edge \( e_1 \). Therefore, by condition (⋆), \( h_2(v_2') \) should contain at least one label. Hence we have that \( h_2(v_2') = \{4\} \).

We see that there is also no freedom of choice in this step.

The next step is to arrange the branches of \( v_2 \) to be the branches of \( v_2' \) or those of \( v_2'' \). The branch containing \( e_1 \) should be modified — multiplicity of edge \( e_1 \) should get reduced by one — and then gets attached to \( v_2'' \). The branch containing label 5 should be attached to \( v_2' \). We see that there is also no freedom of different options in this step. We obtain only one loaded tree, which is also a clever tree.

Hence we only need one time linear reduction for calculating the value of \( T \), the sign for the result is then \((-1)^1 = -1\). In Figure 8 we see the output tree. We can easily obtain that the corresponding monomial of this new tree is \( \delta_{12,3456} \cdot \delta_{124,356} \cdot \delta_{1234,56} \), which coincides with the result we obtain in Example 3.6 with the algebraic reasoning.

In the sequel, we apply the tree-version linear reduction algorithm to the monomial in Example 3.7. Let us see if we can save some labor, comparing to the algebraic method in Section 3.

Example 5.5. See Figure 9 for the corresponding loaded tree \( T \) of monomial \( M := \delta_{123,4567}^2 \cdot \delta_{12345,67} \) in Example 3.7. This is a proper tree.

The edge \( e_1 = \{v_1, v_2\} \) to be reduced is the corresponding edge of cut \( \{(1, 2, 3), (4, 5, 6, 7)\} \). We pick 1, 2 from two distinct clusters of \( v_1 \) and 4, 6 from two distinct clusters of \( v_2 \).

We see that we can either split \( v_1 \) or \( v_2 \) since they both have non-zero weight. When we split vertex \( v_1 \), there is only one loaded tree \( T_1 \) that can be obtained, see the loaded tree in Figure 10a. When we split vertex \( v_2 \), there is only one loaded tree \( T_2 \) that can be obtained, see the loaded tree in Figure 10b. Hence after we apply one time tree-version linear reduction algorithm to \( T \), we obtain in total two loaded trees: \( T_1 \) and \( T_2 \). Their corresponding monomials are \( \delta_{12,34567} \cdot \delta_{123,4567} \cdot \delta_{12345,67} \) and \( \delta_{12,34567}^2 \cdot \delta_{123,4567} \cdot \delta_{12345,67} \), respectively. This coincides with the resulting monomials in Equation 2.
Figure 9: This is the corresponding loaded tree $T$ of monomial $\delta_{1234,567}^3 \cdot \delta_{12345,67}$. We want to reduce edge $e_1$ with the proper quadruple set \{1, 2, 4, 6\}.

Figure 10: The set of these two loaded trees is the output set of Algorithm 1 applied to tree $T$ in Figure 9 with edge $e_1$ and quadruple set \{1, 2, 4, 6\}. When we choose to split vertex $v_1$, we obtain the loaded tree in Figure 10a. When we choose to split vertex $v_2$, we obtain the loaded tree in Figure 10b.

We see that in the above two examples, we obtain exactly the same result as when we did it via algebraic approach; however, much labor saved. Note that we should choose the Keel’s quadruple set coinciding with the one used in the algebraic reasoning, each time; then we will always get the same result with the algebraic algorithm. This is an equivalent characterization of the algebraic linear reduction, in a more visualized way. This characterization naturally also leads to a complete algorithm that is equivalent to the whole process of the algebraic linear reduction until only clever monomials are in the expansion on right hand side of the equation. This is called reduction chain algorithm and will be introduced in the upcoming section.

5.2 Reduction chain algorithm

Tree-version linear reduction algorithm tells us what monomials are on the right hand side of the equation after one-time linear reduction, using the Keel’s linear relation. However, as mentioned earlier, we want to know the value of any proper tree monomial (loaded tree).

A very natural idea is that we can apply Algorithm 1 recursively on the obtained set of loaded trees, after finitely many steps, we should obtain a set of clever trees. The cardinality of the obtained set should give us the absolute value of the given loaded tree. And from Remark 3.8 we know that the sign is just $-1$ to the power of number of recursions. Correctness of this algorithm is guaranteed by the correctness of tree-version linear reduction algorithm. We call this process reduction chain algorithm. This is an equivalent characterization of the algebraic algorithm for computing the value of the given monomial described in Section 3, using graphical language. We state this process as follows.

1. Input: a proper loaded tree $T_M$ (of which the corresponding mono-
Figure 11: This is the only loaded tree $T'_1$ one can obtain after applying Algorithm 1 to tree $T_1$ in Figure 10a with edge $e_1$ and quadruple set $\{4, 6, 1, 3\}$. Edge $e_1$ gets reduced, vertex $v_2$ gets split into vertices $v''_2$ and $v'_2$. And $e'_1$ denotes the newly-generated edge.

Now, let us see an example for a better idea. We continue with Example 5.5.

Example 5.6. We want to apply the above stated algorithm to $T$ in Figure 9, we already obtain $A_1$ in Example 5.5. The set $A_1$ is the set of $T_1$ and $T_2$, when we choose $e_1$ and $\{3, 5, 1, 2\}$ to do the reduction. Then we continue with loaded tree $T_1$ in Figure 10a, the edge to be reduced is evidently (the corresponding edge of cut) $\{\{1, 2, 3\}, \{4, 5, 6, 7\}\}$. We pick a proper linear reduction quadruple set $\{4, 6, 1, 3\}$. We can only split $v_2$ from the weight information. Then we carry out the vertex-splitting process, there is only one loaded tree that can be obtained, see Figure 11. We see that its monomial coincides with the right hand side of Equation 5.

Now we continue with loaded tree $T_2$, the edge to be reduced is evidently (the corresponding edge of cut) $\{\{1, 2, 3\}, \{4, 5, 6, 7\}\}$. We pick a proper linear reduction quadruple set $\{1, 2, 4, 5\}$. We can only split $v_1$ from the weight information. Then we carry out the vertex-splitting process, there is only one loaded tree that can be obtained, see Figure 12. We see that its monomial coincides with the right hand side of Equation 5. We obtain exactly the same result as in Example 3.7!

Therefore, we only need two times recursions, in order to obtain a set of clever loaded trees. And the output is $A_2 = \{T'_1, T'_2\}$. The value of the given monomial $M := \delta_{123,4567} \cdot \delta_{12345,67}$ is $(-1)^2 \cdot 2 = 2$. This coincides with the result in Example 5.7.
Figure 12: This is the only loaded tree $T'_2$ one can obtain after applying Algorithm 1 to tree $T_2$ in Figure 10b with edge $e_1$ and quadruple set $\{1, 2, 4, 5\}$. Edge $e_1$ gets reduced, vertex $v_1$ gets split into vertices $v'_1$ and $v''_1$. And $e'_1$ denotes the newly-generated edge.

Figure 13: This is the only loaded tree $T''_1$ one can obtain after applying Algorithm 1 to tree $T'_1$ on the leftmost loaded tree of Figure 6 with edge $e_3$ and quadruple set $\{1, 7, 3, 4\}$. Edge $e_3$ gets reduced and is marked in green, vertex $v'_2$ gets split into vertices $v'_2$ and $v''_2$. And $e'_3$ denotes the newly-generated edge. Notice that this is a clever tree.

Next, let us continue with Example 5.3. We will apply the reduction chain algorithm to the loaded tree $T_1$ in Figure 5 and see how many clever trees can we obtain.

**Example 5.7.** In Example 5.3, we already obtain all the loaded trees after one time recursion. These loaded trees are shown in Figure 6. Now we apply Algorithm 1 on each of them. However, we see that both incident vertices of the only multiple edge are zero-weighted in the rightmost loaded tree. From Remark 4.4 we know that this tree has value zero. Therefore, we can omit it. We only need to consider the first and the second loaded trees --- denote them as $T'_1$ and $T''_1$, respectively.

First, consider $T'_1$. We pick $e_3$ to reduce and the proper quadruple set $\{3, 4, 1, 7\}$. Manifestly, we can only use vertex $v'_2$ to split. After Algorithm 1 we obtain only one loaded tree $T''_1$ in the output set. This loaded tree is shown in Figure 13. It is a clever tree. Then let us consider $T''_1$. We pick edge $e_3$ to reduce and a proper quadruple set $\{3, 4, 5, 8\}$. We leave it to the reader to check the result --- we should obtain only one clever tree as well in the output set. Then we union these two sets and obtain the output set for the reduction chain algorithm. Since we did two times recursion, the sign should be positive. Therefore, the value for tree $T_1$ in Figure 5 is 2.

In the above example, we already see that after the linear reduction, some loaded trees in the output set have value zero anyway. We can
directly omit those trees since they do not contribute for the value we want to count. In this way, we can also gain some efficiency. **Balancing condition** is a necessary condition for a loaded tree to have value zero. In the next subsection, we introduce this condition. Thanks to the tree representation of the monomials, we can well-define this concept.

### 5.3 Balancing condition

Although those monomials/trees obtained after tree-version linear reduction algorithm are already precisely the ones that are on right hand side of the equation in algebraic expressions, we want to further improve the algorithm, so that there is more efficiency. We will introduce an edge-cutting lemma, which offers one more filter to judge whether a monomial has value zero — in addition to Keel’s quadratic relation. And it is the first filter for the zeroness property of a loaded tree — though only a necessary one. Before this, we need to introduce the concept of **edge-cutting**.

Let $T$ be a proper loaded tree with some weight-zero/single edge $e = \{v_1, v_2\}$. Then we can construct two new loaded trees $T_1$, $T_2$ by cutting off this weight-zero edge and adding one more label to the two incident vertices of this edge, respectively. We call this operation **edge-cutting operation** on edge $e$. Let us see an example on it.

**Example 5.8.** Consider the loaded tree $T$ in Figure 9, we see that $e_2$ is a single edge. We can cut it off, add one extra label, say $\star$, to vertex $v_2$, and one extra label, say $\diamond$, to vertex $v_3$. Then we obtain two new loaded trees, see Figure 14.

In the above example, we see that after applying edge-cutting operation on a given proper loaded tree, we obtain two new loaded trees. And they are even proper. Naturally one may wonder, if this is always the case. So, yes, this operation keeps the property of being a loaded tree; but it does not keep the property of being proper.

**Lemma 5.9.** Let $T$ be a loaded tree and $T_1$ and $T_2$ are the two trees obtained from an edge-cutting operation executed on the edge $e$ (whose corresponding cut is $\{I, J\}$) of $T$. Then, both $T_1$ and $T_2$ are loaded trees.

**Proof.** $T_1$ is a loaded tree with labeling set $I \cup \{\star\}$, $T_2$ is a loaded tree with labeling set $J \cup \{\diamond\}$. The only thing that we need to verify is the third condition of Definition 2.1 for the two incident vertices of $e$ in $T_1$ and

---

**Figure 14:** These are the two loaded trees that are obtained after an edge-cutting operation on loaded tree $T$ in Figure 9. The loaded tree in Figure 14a has 6 labels and 3 edges. The labeling sets of the two vertices are $\{1, 2, 3\}$ and $\{4, 5, \star\}$, respectively. The loaded tree in Figure 14b has 3 labels and 0 edges. The labeling set of the only vertex is $\{\diamond, 6, 7\}$. 

---

![Figure 14](image-url)
As for the properness, we can see a counter-example. Consider the rightmost tree in Figure 4. It is a loaded tree with 8 labels and 5 edges; evidently proper. We can cut the edge $e'$. Then it is not hard to check that we obtain a loaded tree with 5 labels and 1 edge; and another loaded tree with 5 labels and 3 edges. Both of them are not proper. Later on, with the help of this being “improper” property, we can derive the filter that is claimed at the beginning of this section.

Next, we introduce the main statement of the section. We call it “edge-cutting lemma”, although it is stated as a proposition — since the author views it interesting on its own.

**Proposition 5.10** (edge-cutting lemma). Let $T$ be a loaded tree and $T_1$ and $T_2$ are the two trees obtained from an edge-cutting operation executed on edge $e = \{v_1, v_2\}$ of $T$. Then, we have that $|f(T)| = |f(T_1)| \cdot |f(T_2)|$.

**Proof.** Let $\{I, J\}$ be the corresponding cut of $e$ and w.l.o.g. assume that $v_1$ is in Component-$I$ of $T$ and $v_2$ is in Component-$J$. Also, assume that the extra label added to $v_1$ is $\ast$ and the one added to $v_2$ is $\diamondsuit$.

Let $M = M_1 \cdot \delta_{I,J} \cdot M_2$ be the corresponding monomial of $T$, where $M_1$ contains all corresponding factors of edges in Component-$I$; $M_2$ contains all corresponding factors of edges in Component-$J$ and $\delta_{I,J}$ is the corresponding factor for edge $e$. From the reduction chain algorithm, we know that after finitely many times (algebraic) linear reduction (applied to $M$), $M_1$ can be represented as a sum of, say $m_1$ many, clever monomials — maybe with a negative sign; so does $M_2$, say it is a sum of $m_2$ many clever monomials — maybe with a negative sign. Then, after an elementary expansion of the polynomial, $M$ can be represented as a sum of $m_1 \cdot m_2$ many clever monomials (maybe with a negative sign). Formally, and algebraically, we will have

$$M = (M_1 + M_2 + \cdots + M_{m_1}) \cdot \delta_{I,J} \cdot (M'_1 + M'_2 + \cdots + M'_{m_2})$$

$$= \sum_{1 \leq i \leq m_1, 1 \leq j \leq m_2} M_i \cdot \delta_{I,J} \cdot M'_j,$$

after these finitely many reductions. From the reduction chain algorithm, we know that each summand in this equation is a clever monomial. And the absolute value of $M$ (or $T$) should be $m_1 \cdot m_2$, i.e., $|f(T)| = m_1 \cdot m_2$. Then, it is sufficient to show that $|f(T_1)| = m_1$ and $|f(T_2)| = m_2$.

Let $M = \overrightarrow{M_1} \cdot \delta_{I,J} \cdot \overrightarrow{M_2}$ be the monomial representation of any loaded tree $T$ that is generated at some recursion round. Let $\delta_{I_1,J_1}$ be any factor of $\overrightarrow{M_1}$ — let $\overrightarrow{e} = \{\overrightarrow{v_1}, \overrightarrow{v_2}\}$ be its corresponding edge. W.l.o.g., assume that $\overrightarrow{v_1}$ is in Component-$I_1$ of $T$ and the unique path from $\overrightarrow{v_1}$ to edge $e$ (corresponding edge of cut $\{I, J\}$, or factor $\delta_{I,J}$) is incident with $\overrightarrow{v_2}$. Then, from Keel’s quadratic relation, we obtain that $J \subset J_1$. Hence, all labels in $J$ always appear together (as a subset of one part of the corresponding cut) in any factor in $\overrightarrow{M_1}$-part. Consequently, in each intermediate tree, for any vertex $v$ in Component-$I$, all labels in $J$ belong to the same cluster of $v$. Therefore, whenever we split some vertex in Component-$I$, there exists at
most one label from $J$ in any proper quadruple. And this property remains true no matter which recursion step we are at during the reduction chain algorithm.

We replace each occurrence of $J$ (as a subset of some cut, say $J_1$ of factor $\delta_{I_1,J_1}$) in $M_1$ by $*$, and denote the new monomial by $M'_1$. One can see that $M'_1$ is actually the monomial of $T_1$. Whenever a quadruple containing any label in $J$ is chosen for the linear reduction, we only need to replace that label by $*$; if the chosen quadruple does not contain any label in $J$, then we leave it unchanged. By this replacement, we see that the whole execution process of reduction chain algorithm induces an execution of the reduction chain algorithm on $T_1$. Parallelly, in algebraic view (language), it is the following process: We do this replacement for all factors through all those linear reductions, until the representation of $M_1$ is a sum of clever monomials (maybe with a negative sign). Then, we actually obtain a linear reduction series for $M'_1$. And we simultaneously obtain the representation of $M'_1$ by a sum of clever monomials (maybe with a negative sign)! Therefore, we have that $|\mathcal{I}(T_1)| = m_1$ and $|\mathcal{I}(T_2)| = m_2$.

Corollary 5.11. Let $T$ be a proper tree with a single edge $e$ and $T_1$, $T_2$ be two loaded trees obtained from $T$ via an edge-cutting operation on edge $e$. If $T_1$ is not proper, then $\mathcal{I}(T) = 0$.

Proof. Assume that $T$ has $n$ labels and $n - 3$ edges, $T_1$ has $n_1$ labels and $k_1$ edges, $T_2$ has $n_2$ labels and $k_2$ edges. It is not hard to obtain that $n_1 + n_2 = n + 2$ and $k_1 + k_2 = n - 4$. Therefore, we obtain that $n_1 + n_2 = k_1 + k_2 - 6$. Hence $T_1$ is proper if and only if $T_2$ is proper. Suppose w.l.o.g. that $k_1 > n_1 - 3$. Then we know that $T_1$ corresponds to a monomial in the Chow group $A^{k_1}(n_1)$. From the fact stated in Section 1 we have that $A^{k_1}(n_1) = \{0\}$ and hence $\mathcal{I}(T_1) = 0$. By Proposition 5.10, we obtain that

$$|\mathcal{I}(T)| = |\mathcal{I}(T_1)| \cdot |\mathcal{I}(T_2)| = 0 \cdot |\mathcal{I}(T_2)| = 0.$$

If both $T_1$ and $T_2$ in above stated process are proper, we say that $T$ is balanced or fulfills the balancing condition with respect to edge $e$. A loaded tree is balanced if and only if it is balanced with respect to any single edge.

Actually, the edge-cutting operation is defined locally. If we do this operation on each single edge of loaded tree $T$, then no matter in which sequence we choose to cut those edges, we always obtain a same set of loaded trees. With the similar reasoning, edge-cutting lemma has a generalized version; so does Corollary 5.11. We conclude them in the following lemma, and we omit the proof.

Lemma 5.12. Let $T$ be a loaded tree and $T_1, \cdots, T_n$ are the trees obtained from edge-cutting operations on all single edges of $T$. Then, we have that

$$|\mathcal{I}(T)| = |\mathcal{I}(T_1)| \cdot |\mathcal{I}(T_2)| \cdots |\mathcal{I}(T_n)|.$$

And if $T_i$ is not proper for any $1 \leq i \leq n$, $|\mathcal{I}(T)| = 0$.

The next result holds consequentially.

24
Theorem 5.13. Unbalanced loaded trees have value zero.

Proof. By Lemma 5.12 straightforward. □

Now let us look back on the vertex-splitting process. Whenever we split a vertex, one single edge $e'$ is generated. Therefore, in the tree-version linear reduction algorithm, we can further filter the loaded trees in the output set — those that are not balanced w.r.t. edge $e'$ can be directly removed from the set, since zero does not contribute anything to the value calculation. In this sense, we can always check the local balanceness (w.r.t. edge $e'$) of the generated loaded tree. Furthermore, we can check the (overall) balanceness of the generated trees and omit the unbalanced ones. In this way, more efficiency is brought into the algorithm. See Algorithm 2 for an optimized version of Algorithm 1.

Algorithm 2: optimized tree-version linear reduction

input : a proper loaded tree $T_M$ (corresponding loaded tree of the proper monomial $M$); a multiple edge $e = \{v_1, v_2\}$ (with corresponding cut $\{I, J\}$) of $T_M$; a proper quadruple set $\{i, j, k, l\}$ of $e$, or equivalently, of $\{I, J\}$.

output: balanced loaded trees whose corresponding monomials are the ones that survive after the linear reduction on $M$ which uses Keel’s linear reduction on the relation $\epsilon_{ij|kl} = \epsilon_{ik|jl}$

1. $w_1 \leftarrow$ weight of $v_1$;
2. $w_2 \leftarrow$ weight of $v_2$;
3. if $w_1 = 0$ and $w_2 = 0$ then
   1. return $\emptyset$
4. end if

$ST \leftarrow$ the set of all balanced loaded trees that can be obtained from $T_M$ with edge $e$ and the proper quadruple set $\{i, j, k, l\}$, after applying the vertex splitting algorithm — either by splitting vertex $v_1$, or by splitting vertex $v_2$ — in the set $ST$;

return $ST$

As an example, we consider the loaded tree $T_1$ in Figure 5. After the first recursion round of Algorithm 1, we obtain three loaded trees as shown in Figure 6. However, if we check the balanceness of them, we see that the rightmost tree is not balanced with respect to edge $e'$. Hence it is not balanced, has value zero. The output of optimized tree-version linear reduction algorithm will just contain the two loaded trees on the right for this round.

In some other cases, this condition can help filter out a lot of loaded trees. One may realize already that this condition can be much more complicated if we consider it for the monomials. But of course, the corresponding monomial of an unbalanced tree also has value zero. Hence this also shows the strength of the tree representation. Only when we consider the monomial reductions in the view point of loaded trees, can we make use of the balancing condition, and gain much more efficiency for the computing.
To integrate the balancing checking to reduction chain algorithm, we only need to substitute the occurrences of tree-version linear reduction algorithm by the optimized tree-version linear reduction algorithm. With this, we conclude this section. In the next section, all the undone proofs will be settled.

6 The missing proofs

In this section, we will settle down all the missing proofs that are postponed in the earlier sections. After going through the previous context, we collect in total four missing proofs:

1. Theorem 3.4.
2. Any proper quadruple fulfills the summand distinction property.
3. If a proper loaded tree has a multiple edge $e = \{v_1, v_2\}$ where the weights of $v_1, v_2$ are both zero, then the tree has value zero.
4. Correctness of the tree-version linear reduction algorithm.

We will first prove Theorem 3.4. The theorem can be restated as follows:

**Theorem 6.1.** A clever tree has value one.

**Proof.** First, we consider the situation in the Chow ring $A^*(3)$. This ring is exactly the Chow group $A^0(3)$, which is isomorphic to the integer additive group. There is only loaded tree $T$ in this ring, which consists of a single vertex with three labels 1, 2 and 3. Hence the group $A^0(3)$ is generated by this single clever tree (or by the empty monomial). Combining with the fact that $A^0(3)$ is isomorphic to $\mathbb{Z}$, we obtain that the value of $T$ is either 1 or $-1$. We choose it to be one. And note that $T$ is the only clever tree that has no edges.

Let $T_1$ be any other clever tree — then it has at least one edge. Any edge of $T_1$ is a single edge. By the edge-cutting lemma, we can apply the edge-cutting operation on each of its edge, at the end obtaining $|V(T)|$ many clever trees with three labels, where $V(T)$ denotes the vertex set of $T$. Since renaming or permuting the labels does not influence the integral value of a loaded tree, we know that the value of $T_1$ is the product of $|V(T)|$ many 1, hence it has value one.

**Remark 6.2.** Note that we can also choose $-1$ as the value of $T$. Then the value monomial in $A^{n-3}(n)$ is just the product of $-1$ and the value of it when we choose 1 for the value of $T$ — since there are in total two automorphisms on the integer additive group.

Then let us focus on the algebraic reduction, try to prove the second item above. We will need some help from the tree representation of the (tree) monomial for the proof.

**Theorem 6.3.** Let $M$ be a proper non-clever tree monomial and $\delta_{I,J}$ the (power-higher-than-one) factor to be reduced, let $Q = \{i, j, k, l\}$ be a proper quadruple set such that $i, j \in I$, $k, l \in J$ hold. Then, no summand in $-(\epsilon_{ij}k_l - \delta_{I,J})$ divides $M$. 

26
Proof. Assume that after the linear reduction, we obtain the representation of \( M \) as the negative sum (see Remark [3.8]) of several tree monomials: 
\[ M = - \sum_{r=1}^{k} M_r, \] 
where any \( M_r \) (\( 1 \leq r \leq k \)) is a monomial. Let \( M_e \) be any summand on the right hand side. Let \( e = \{v_1, v_2\} \) be the corresponding multiple edge of cut \( \{I, J\} \) — w.l.o.g. assume that \( v_1 \) is in Component-\( I \) — and \( T_M = (V, E, h, m) \) the corresponding proper loaded tree for \( M \).

It is obvious from the algebraic point of view that after the linear reduction, all the other factors except for \( \delta_{ij} \) stay unchanged, we only need to argue that the new factor \( \delta_{ij} \) (corresponding to edge \( e' \)) in \( M_e \) is distinct from all the factors in \( M \). Since \( \delta_{ij} \) is a summand in 
\[ -(\epsilon_{ijkl} - \delta_{ij}), \] 
we see that it is also distinct from \( \delta_{ij} \), by Fact 3.1. In the graphical language: we only need to argue that it cannot be the same edge with any other edges in tree \( T_M \). Note that \( \delta_{ij} \) is a summand in 
\[ -(\epsilon_{ijkl} - \delta_{ij}), \] 
hence we have w.l.o.g. \( i, j \in I' \) and \( k, l \in J' \).

When we do a proper choice of the quadruple set \( \{i, j, k, l\} \), there are in total six different cases. Denote by \( h(v) := h(v_1) \cup h(v_2) \).

1. \( i, j \notin h(v), k, l \notin h(v) \).
   
   Since \( i, j \) are from two distinct clusters, say \( I_1 \) and \( I_2 \), of \( v_1 \), we see that \( \delta_{ij} \) cannot coincide with any edge in Component-\( I \). Because of the symmetry of \( i, j \) and \( k, l \) in this case, \( \delta_{ij} \) also cannot coincide with any edge in Component-\( J \). Also, we see that \( I_1, I_2 \subset I \).
   
   Suppose that \( e' \) is not incident with \( e \) (in \( T_{M_{v_1}} \)), then there exists another edge \( e'' \) (with corresponding cut is \( \{I'', J''\} \)) on the path from \( e' \) to \( e \). Note that \( e'' \) should also be an edge in \( T_{M_{v_1}} \). Then we see that \( I' \not\subset I'' \not\subset I \). Since there is no proper cluster containing both \( I_1, I_2 \), which is a proper subset of \( I \) in \( T_M \), we obtained a contradiction. Therefore, \( e' \) is incident with \( e \).

2. \( i, j \notin h(v), k \notin h(v) \).
   
   The same reasoning as in item 1. tells us that \( e' \) cannot coincide with any edge in Component-\( I \). Denote by \( J_1 \) the cluster of \( k \) in \( T_M \).
   
   We know that \( J_1 \subset J' \) since \( k, l \in J' \). Hence \( e' \) cannot be coincident with any edge in Component-\( J \). A similar reasoning with item 1. proves that \( e' \) is incident with \( e \).

3. \( i, j \notin h(v), k, l \in h(v) \).
   
   Since \( k, l \in J' \) but not in any other cluster of \( v_2 \) (except for \( J \)), \( e' \) is distinct from all edges in \( T_M \). A similar reasoning with item 1. proves that \( e' \) is incident with \( e \).

4. \( i, k \notin h(v), j, l \in h(v) \).

5. \( i \notin h(v), j, k, l \in h(v) \).

6. \( i, j, k, l \in h(v) \).

Proof for case 4., 5., 6. can be obtained analogously via the analysis of the first three items. We conclude that \( Q \) fulfills the summand distinction property. \( \square \)

From the above proof, we actually gained more than needed. We can see that the set of labels (can be seen from the index of factors in \( M \),
or $M_r$) stay unchanged. Also the number of factors in $M$ is the same as that in $M_r$: only one occurrence of $\delta_{i,j}$ is replaced by $\delta_{i',j'}$. Hence $T_M$ and $T_{M_r}$ are loaded trees of the same type. The proper loaded tree $T_{M_r}$ differs from $T_M$ only in the following aspects:

1. Multiplicity of edge $e$ is reduced by one.
2. A new edge $e'$ (incident to $e$) with multiplicity one is generated.
3. All other edges stay unchanged: both the corresponding cut and the multiplicity stay unchanged.

In the sequel, we prove the correctness of tree-version linear reduction algorithm. We will keep using the notations from the above analysis.

**Theorem 6.4.** Tree-version linear reduction algorithm is correct.

**Proof.** First, we will show that any loaded tree $T_{M_r}$ of some summand $M_r$ (with respect to the Keel’s linear relation $\epsilon_{ij|kl} = \epsilon_{il|kj} = \epsilon_{ik|jl}$) can be obtained by a vertex splitting process on $T_M$ with respect to the proper quadruple set $\{i,j,k,l\}$.

In order to keep all the other edges (except for $e$) unchanged, it is necessary to have step 6. in vertex splitting process; otherwise, once the structure of any branch (of $v_1$) is changed, some edge (and corresponding cut) in this branch will change accordingly. In order to generate the new edge $e'$ incident with $e$, it is necessary to have step 4. And in order to make sure that we obtain a loaded tree as well, it is necessary to have step 5. Since $i,j \in I \cap I'$ and $k,l \in J \cap J'$, the specific requirements in Step 5. and Step 6. are also necessary. From Remark 4.5 we already see that Step 3. is necessary. Hence, the vertex splitting process is necessary for obtaining a monomial on right hand side after one time linear reduction.

In the sequel, we prove that any loaded tree $T_2$ obtained by a vertex splitting process on $T_1$ with respect to the proper quadruple set $\{i,j,k,l\}$ corresponds to some monomial $M_r$ in $M = -\sum_{i=1}^k M_r$ after an algebraic linear reduction using the Keel’s linear relation $\epsilon_{ijk|kl} = \epsilon_{ij|kj} = \epsilon_{ik|jl}$, where $M = M_{T_1}$ is the corresponding monomial of input (proper) loaded tree $T_1$.

First, we see that all edges of $T_2$ — compare to $T_1$, the input of vertex splitting process — stay unchanged except for $e$; but a new edge $e'$ incident to $e$ is generated. In algebraic language: all factors stay unchanged (compare $M_{T_1}$ with $M_{T_2}$) except for one occurrence of $\delta_{i,j}$, which is replaced by $\delta_{i',j'} \neq \delta_{i,j}$; we know that $i,j \in I \cap I'$ and $k,l \in J \cap J'$. Hence we see that $\delta_{i',j'}$ is a summand in $-(\epsilon_{ijk|kl} - \delta_{i,j})$. Therefore, $M_{T_2} = M_r$ for some $1 \leq r \leq k$.

From the correctness of tree-version linear reduction algorithm, we naturally obtain the following proposition.

**Proposition 6.5.** If a proper loaded tree has a multiple edge $e = \{v_1, v_2\}$ where the weights of $v_1$, $v_2$ are both zero, then the tree has value zero.

**Proof.** In this case, when we want to replace one occurrence of the corresponding factor of $e$ to do the linear reduction, we see that it is not possible to split any adjacent vertices of $e$ via the vertex splitting process. Hence we obtain NULL via the vertex splitting process. By Theorem 6.4. }

28
we know no monomial can survive the Keel’s quadratic relation, i.e., all of them contain some pair of generators fulfilling Keel’s quadratic relation. Hence all the monomials we obtain on right hand side after linear reduction have integral value zero. Therefore, the value of the given loaded tree is zero.

7 An application: sun-like trees

In this section, we introduce a specific type of proper trees, and compute their values. This computation can also serve as the base case for the proof of forest algorithm correctness ([13, Theorem 0.5.]).

7.1 Problem

We say a proper loaded tree is sun-like if it has domination number equal to one — there exists a vertex $v$ such that all other vertices are neighbors of it — and all adjacent vertices of $v$ have weights zero, all edges have positive weights. We call this vertex $v$ the middle vertex. Let $k$ be the weight for the middle vertex and $m_1, \cdots, m_r$ the weights for its incident edges, respectively. By the weight identity for proper loaded trees, we know that $k = \sum_{i=1}^{r} m_i$. See Figure 15 for a visualization.

The specific problem that we want to handle in this section is: prove that the absolute value of a sun-like proper loaded tree — using the above notations — is $(k m_1, \cdots, m_r)$. It can be formulated as follows:

**Theorem 7.1.** Let $T = (V, E, h, m)$ be a sun-like proper loaded tree with $v$ its middle vertex. Let $e_1, \cdots, e_r$ be the edges of $T$, where $e_i = \{v, v_i\}$ for $1 \leq i \leq r$. Denote by $w$ the weight function for $T$, assume that $w(v) = k$ and $w(e_i) = m_i \geq 1$ for $1 \leq i \leq r$. Then we have $|\int(T)| = (k m_1, \cdots, m_r)$.

Before the proof, we need to introduce an identity on multinomial coefficient, which will help us in the proof later on.

7.2 An identity on multinomial coefficient

For any $r$-many positive-integer parameters $m_1, m_2, \cdots, m_r$, define $s := \sum_{i=1}^{r} m_i$. Denote a set of $r$-many indeterminates as $X := \{x_1, x_2, \cdots, x_r\}$. 

![Figure 15](image-url)
Define $T := \{ B \mid B \subset X, x_1 \in B \}$ and

$$B := \{ (B_1, B_2) \mid B_1 \in T, B_2 = X \setminus B_1 \}. $$

Define $g : X \to \{ m_1 - 1, m_2, \cdots, m_r \}$ by $g(x_i) := m_1 - 1$ if $i = 1$ and $g(x_i) := m_i$ otherwise. Here note that the set $\{ m_1 - 1, m_2, \cdots, m_r \}$ may be a multiset in form but we just consider it as a normal set, in practice. For convenience in the later writing, we introduce the following notation.

Define for $B \subset X$,

$$S(B) := \sum_{x \in B} g(x), \quad \left( \frac{S(B)}{B} \right) := \frac{S(B)!}{\prod_{x \in B} (g(x)!)}. $$

Based on the above preparation, the identity we want to introduce can be formulated as follows.

**Theorem 7.2.**

$$\left( \begin{array}{c} s \\ m_1, m_2, \cdots, m_r \end{array} \right) = \sum_{(B_1, B_2) \in B} \left( \begin{array}{c} s - r + 1 \\ S(B_2) - |B_2| \end{array} \right) \left( \begin{array}{c} S(B_1) \\ B_1 \end{array} \right) \left( \begin{array}{c} S(B_2) \\ B_2 \end{array} \right),$$

where $|B_2|$ refers to the cardinality of $B_2$.

The proof for this identity is postponed to Section 7.5 in order not to distract our main rhythm. In the next section, we provide the proof of Theorem 7.1 using the above identity.

### 7.3 Proof for the value of sun-like tree

- **Base case:** We prove by induction on $k$. When $k = 0$, since it is proper and all weights of the edges are positive, we know that in this case the tree has no edge. By Theorem 6.1, we know that it has value 1, which coincides with the wanted formula since $0! = 1$.

- **Proof idea for the general case — apply Algorithm 2:** When $k \geq 1$, by Remark 3.8 and Algorithm 2, we know that the absolute value of $T$ equals to the sum of the values of balanced loaded trees obtained in the output of Algorithm 2 when applied to $T$ with a multiple edge and a proper quadruple. This indicates our proof idea: we choose a multiple edge and a proper quadruple, then consider what balanced loaded trees can be generated after the vertex-splitting process. Since in this process, one new weight-zero edge is generated and an old edge’s weight is reduced by one, hence the edge weight sum, or equivalently the vertex weight sum, is reduced by one. For any tree in the output, we can cut-off the zero-weighted edge generated in the vertex-splitting process. Then for the obtained two loaded trees, we can apply the induction hypothesis.

- **Get prepared for the input data:** We choose edge $e_1$ to reduce. In order to choose a proper quadruple, we should first figure out how many labels does the middle vertex $v$ have. By Definition 2.1 third condition, we have $k = \deg(v) + |h(v)| - 3$. Clearly $\deg(v) = r$ and we also have $k = \sum_{i=1}^{r} m_i \geq r$. Hence

$$|h(v)| = k - \deg(v) + 3 = k - r + 3 \geq 3.$$
Hence $v$ has at least three labels, denote by $i, j$ two of them. Since vertex $v_1$ has degree one and weight zero, it is not hard to reason that it has two labels, denote them by $k, l$. We see that $\{i, j, k, l\}$ is a proper quadruple set for the multiple edge $e_1$. And obviously we can only split vertex $v$, since $v_1$ has value zero. We will use as input these data for Algorithm 2.

- **Arranging the labels — part one:** Suppose that the statement is true for all cases when the middle vertex has weight less or equal to $k - 1$, where $k \geq 1$, now let us consider the case when the middle vertex has weight $k$. Let $T_1 = (V_1, E_1, h_1, m_1)$ be any loaded tree in the output. We split vertex $v$ into $v'$ and $v''$, denote by $e'$ the edge $\{v', v''\}$. By Step 5. of vertex-splitting, we know that $i, j \in h_1(v')$. Hence, we have the freedom on the arrangements of all other labels of $v$ to be the labels of either $v'$ or $v''$. How many choices do we have? The answer will be revealed a bit later.

- **Arranging the branches:** Also, we have the freedom for arranging the branches. Denote by $B_1$ the set of branches for $v''$ and $B_2$ for that of $v'$. Denote by $x_1, \ldots, x_r$ the branches corresponding to $e_1, \ldots, e_r$, respectively. Let $B := \{x_1, \ldots, x_r\}$, we see that $B = B_1 \cup B_2$. By Step 6. of vertex-splitting, we know that $x_1 \in B_1$, since this branch contains labels $k$ and $l$. For each bipartition of $B$ into $B_1$ and $B_2$ such that $x_1 \in B_1$, we need to consider the distribution of labels so that the obtained tree is balanced. For this, we need to introduce several notations, as to express the arrangements of branches.

- **A formal model for the arrangements of branches:** Define a function $g : B \to \{m_1 - 1, m_2, \ldots, m_r\}$ by $x_1 \mapsto m_1 - 1, x_i \mapsto m_i$ for $2 \leq i \leq r$. Define $S(A) := \sum_{x \in A} g(x)$ and

\[
\binom{S(A)}{A} := \frac{S(A)!}{\prod_{x \in A} (g(x))!}
\]

Let $B_1$ be any arrangement of branches for vertex $v''$. Then $B_2 = B \setminus B_1$.

- **Balancing condition considered:** Since we require loaded tree $T_1$ to be balanced, it is also balanced with respect to edge $e'$. After we cut-off the edge $e'$, the tree containing $v'$ should be proper, denote it by $T_1'$; so does the tree containing $v''$, denote it by $T_1''$ — the weight identity should hold for both trees. From this, we obtain that the weight of $v'$ in $T_1'$ is $S(B_2)$ and that the weight of $v''$ in $T_1''$ is $S(B_1)$. Since the edge-cutting operation does not change the weight of vertex, we know that $w_1(v') = S(B_2)$ and $w_1(v'') = S(B_1)$. And obviously, degree of $v'$ is $|B_2| + 1$, while that of $v''$ is $|B_1| + 1$, since we need to count edge $e'$ once also for their degree, respectively.

- **Arranging the labels — part two:** Now we can figure our how many labels should we distribute to $v'$. In $T_1$:

\[
w_1(v') = \deg(v') + |h_1(v')| - 3.
\]

31
Therefore, we have:

\[ |h_1(v')| = w_1(v') - \deg(v') + 3 \]
\[ = S(B_2) - (|B_2| + 1) + 3 \]
\[ = S(B_2) - |B_2| + 2 \]

However, we already know that \( i, j \in h_1(v') \). Therefore, we should distribute \( S(B_2) - |B_2| \) many labels of \( v \) to vertex \( v' \), in the vertex-splitting process. Then naturally the remaining labels went to the labeling set of vertex \( v'' \).

**The induction step:** Then we cut off the single edge \( e_1' \) in \( T_1' \), obtaining two smaller proper trees \( T_1'' \) and \( T_1''' \), since \( T_1 \) is balanced. By Remark \( 4.5 \) we have \( w_1(v') + w_1(v'') = w(v) - 1 \). Hence \( 0 \leq w_1(v') < w(v) = k \) and \( 0 \leq w_1(v'') < w(v) = k \). Therefore, we can use the induction hypothesis on \( T_1'' \) and \( T_1''' \) if \( m_1 - 1 \neq 0 \).

**Special case for the induction step:** When \( m_1 - 1 = 0 \), we can cut off edge \( e_1 \) in \( T_1'' \), then apply the induction hypothesis for the same weight for vertex \( v'' \) — since edge-cutting does not influence the weights of the two adjacent vertices of the cut-off edge. After this edge-cutting, we obtain a single vertex with three labels (the two of which are \( k \) and \( l \)) — this loaded tree has value one. Then by the edge-cutting lemma (Proposition \( 5.10 \)), we see that the value of \( T_1'' \) equals to the value of the tree containing \( v'' \) after cutting off \( e_1 \). Then we can use the induction hypothesis on this smaller tree. However, since

\[
\begin{pmatrix}
S \\
 s_1, \ldots, s_p
\end{pmatrix} = \begin{pmatrix}
S \\
 0, s_1, \ldots, s_p
\end{pmatrix}
\]

for \( S = \sum_{i=1}^p s_i \) and \( s_i \in \mathbb{N}^+ \), we see that our induction hypothesis can also apply in this case — the value of \( T_1'' \) is not influenced by whether \( m_1 - 1 \) is zero or not.

**Express the value of each loaded tree in the output:** For each given \( B_1 \), among all the \( k - r + 3 \) labels of \( v, i, j \) are doomed to belong to \( v' \). We should choose \( S(B_2) - |B_2| \) many labels for \( v' \), from \( k - r + 1 \) many labels of \( v' \). Then by the edge-cutting lemma, we know that for this arrangement, the value of the obtained balanced proper loaded tree is the product of the values of two smaller trees, after cutting off the edge \( e_1' \) — which is exactly \( (S(B_2)) \cdot (S(B_1)) \).

**Summing over these values:** That is to say, whenever the arrangement of the branches is fixed, because of the balancing condition requirement, the label distribution is also fixed. There are \( (S(B_2) - |B_2|) \) many distributions of labels, up to the permutations of labels. Since permutation or renaming the labels does not influence the value of the loaded tree, we know that for any two such permutations, the two trees have the same value: \( (S(B_2)) \cdot (S(B_1)) \) — the product of the values of the two smaller trees. Summing over the different arrangements of the branches, we obtain that the sum of
the values of all the trees in the output is precisely:

$$\sum_{(B_1, B_2) \in B} \left( \begin{array}{c} k - r + 1 \\ S(B_2) - |B_2| \end{array} \right) \cdot \left( \begin{array}{c} S(B_1) \\ B_1 \end{array} \right) \cdot \left( \begin{array}{c} S(B_2) \\ B_2 \end{array} \right).$$

- **Show-time for the identity**: Hence the only thing that is needed for our proof is the following identity:

$$\sum_{(B_1, B_2) \in B} \left( \begin{array}{c} k - r + 1 \\ S(B_2) - |B_2| \end{array} \right) \cdot \left( \begin{array}{c} S(B_1) \\ B_1 \end{array} \right) \cdot \left( \begin{array}{c} S(B_2) \\ B_2 \end{array} \right) = \left( \begin{array}{c} k \\ m_1, \ldots, m_r \end{array} \right).$$

With Theorem 7.2 we conclude the proof.

### 7.4 Equivalent characterization

In the last section, we see that we can prove Theorem 7.1 using Theorem 7.2. In this section, we show that we can also prove the identity given that Theorem 7.1 holds.

Let us reconsider the proof steps in Section 7.3. All the steps until the last do not depend on the identity, therefore we can still use those analysis for the proof in this section. We keep using the notation from last section, suppose that Theorem 7.1 holds. Then we know that if we apply Algorithm 2 to $T$ with multiple edge $e_1$ and corresponding quadruple set $\{i, j, k, l\}$ same as in the last section, the sum of the values of output loaded trees can be expressed as:

$$\sum_{(B_1, B_2) \in B} \left( \begin{array}{c} k - r + 1 \\ S(B_2) - |B_2| \end{array} \right) \cdot \left( \begin{array}{c} S(B_1) \\ B_1 \end{array} \right) \cdot \left( \begin{array}{c} S(B_2) \\ B_2 \end{array} \right).$$

Correctness of Algorithm 2 tells us that the above sum equals to $\left( \begin{array}{c} k \\ m_1, \ldots, m_r \end{array} \right)$. That is to say, we have

$$\left( \begin{array}{c} k \\ m_1, m_2, \ldots, m_r \end{array} \right) = \sum_{(B_1, B_2) \in B} \left( \begin{array}{c} k - r + 1 \\ S(B_2) - |B_2| \end{array} \right) \cdot \left( \begin{array}{c} S(B_1) \\ B_1 \end{array} \right) \cdot \left( \begin{array}{c} S(B_2) \\ B_2 \end{array} \right).$$

Hence, Theorem 7.4 is an equivalent characterization of the identity. However, in order to show their equivalence, Algorithm 2 or the vertex-splitting process plays the essential role. We believe that the identity indicates some complicated structural information of the vertex-splitting process, in an algebraic way. In the next section, we prove this identity.

### 7.5 Proof of the identity

Continuing with the notations in Section 7.2, we give a combinatorics proof for Theorem 7.2 in the sequel — which then also leads to the correctness of both Theorem 7.2 and Theorem 7.4.

Let us first recall the notations, note that we slightly modify the notations, so that they serve well for our proof — namely we add an index $r$ for many of them, indicating that we are considering $r$ many sums for
the multinomial coefficient. We will see later on that this index is helpful. We will also add some comments on why these notations were used in the identity, to show some original idea behind.

- \( m_1, m_2, \ldots, m_r \): \( r \)-many positive-integer parameters.
- \( s_r := \sum_{i=1}^r m_i \).
- \( X_r := \{x_1, x_2, \ldots, x_r\} \): a set of \( r \)-many indeterminates. This set is introduced so that we can consider all the bipartitions of the values \( \{m_1-1, m_2, \ldots, m_r\} \). In this way, we are able to formally go through all combinations.
- \( T_r := \{B \mid B \subset X_r, x_1 \in B\} \). The elements in \( T_r \) indicates one part of the bipartition and we always put \( x_1 \) in it, so as to avoid repetition.
- \( B_r := \{ (B_1, B_2) \mid B_1 \in T_r, B_2 = X_r \setminus B_1 \} \). This set is exactly the collection of all the bipartition of \( X_r \).
- \( g_r : X_r \to \{m_1-1, m_2, \ldots, m_r\} \), \( x_1 \mapsto m_1-1, x_i \mapsto m_i \) for \( i \neq 1 \). This function is introduced for the sake of the next two notations, mainly because the value for \( m_1 \) is reduced by one.
- \( S(B) := \sum_{x \in B} g_r(x) \), for \( B \subset X_r \). This is just the normal sum of \( m_i \) for \( 1 \leq i \leq r \), except that \( m_1 \) is replaced by \( m_1 - 1 \) as a summand — this is also why we need the function \( g_r \).

For \( B \subset X_r \), Note that this notation is just a generalized definition of multinomial coefficient.

Besides these notations from earlier, we still need several more, so as to present our proof properly.

- Define
  \[ S_r := \{ (P_1, P_2, \ldots, P_r) \mid \bigcup_{i=1}^r P_i = \{1, 2, \ldots, s_r\}, |P_i| = m_i \} . \]

With this set, we collect all partitions of the set \( \{1, 2, \ldots, s_r\} \) into \( r \) parts such that the \( i \)-th part has cardinality \( m_i \).

- Let \( L_r := \{2, 3, \ldots, r\} \). These elements are special elements in \( \{1, 2, \ldots, s_r\} \). Later on we will see why or how they are special, in the definition of function \( \varphi_r \). The next two notations are also there to serve the definition of function \( \varphi_r \).

- For \( A \subset \{1, 2, \ldots, r\} \), define \( P_A := \bigcup_{i \in A} P_i \). \( P_A \) is the union of the part which has index in \( A \).

- For \( A \subset \{1, 2, \ldots, r\} \), define \( X_A := \{x_i \mid i \in A\} \). \( X_A \) collect the indeterminate that has index in \( A \).

Let us see an example, so that we do not get lost among the ocean of notations.

**Example 7.3.** Given \( r = 3 \), the following facts are already clear:
Example 7.4. \( \varphi \) our running example. Algorithm 3. We will prove that it is indeed an algorithm later on.

Algorithm 3 with the input \((\cdots)\) which may help to have a taste of the identity. We leave it to the readers to check that the identity holds in this example, for instance, then we also obtain the following facts:

- \( s_3 = \sum_{i=1}^{3} m_i = 2 + 2 + 1 = 5 \). Now we know that 2, 3 are considered special among 1, 2, 3, 4, 5.
- \( g_3 : X_3 \to \{1, 2\} \) is defined as \( g_3(x_1) = m_1 - 1 = 2 - 1 = 1, \ g_3(x_2) = m_2 = 2 \) and \( g_3(x_3) = m_3 = 1 \).
- Take \( B = \{x_2, x_3\} \subset X_3 \) for instance, then \( S(B) = g_3(x_2) + g_3(x_3) = m_2 + m_3 = 2 + 1 = 3 \).

Note that in this case \( S(B) \) is just the sum of \( m_2 \) and \( m_3 \), since \( x_1 \notin B \).

- Take \( B = \{x_2, x_3\} \subset X_3 \) for instance, then

\[
\left( \frac{S(B)}{B} \right) = \frac{S(B)!}{\prod_{x \in B} (g_3(x)!)} = \frac{3!}{g_3(x_2) \cdot g_3(x_3)} = \frac{6}{m_2 \cdot m_3} = \frac{6}{2 \cdot 1} = 3.
\]

Note that in this case, \( \left( \frac{S(B)}{B} \right) \) is just the normal multinomial coefficient \( \binom{m_2 + m_3}{m_2, m_3} \), since \( x_1 \notin B \).

- \( S_3 \) is the set of all partitions \((P_1, P_2, P_3)\) of the set \(\{1, 2, 3, 4, 5\}\) into three parts \(P_1, P_2, P_3\) such that \( |P_1| = m_1 = 2, \ |P_2| = m_2 = 2 \) and \( |P_3| = m_3 = 1 \).
- Take \( A = \{1, 2\} \subset \{1, 2, 3\} \) for instance, then \( P_A = P_1 \cup P_2 \) for some \((P_1, P_2, P_3) \in S_3\).

We leave it to the readers to check that the identity holds in this example, which may help to have a taste of the identity.

Now we define a function \( \varphi_r : S_r \to T_r, (P_1, \cdots, P_r) \mapsto B \) by Algorithm 3. We will prove that it is indeed an algorithm later on.

For a better understanding, let us see how is this function defined in our running example.

Example 7.4. \( \varphi_3 : S_3 \to T_3, (P_1, P_2, P_3) \mapsto B \in T_3 \). Let us go through Algorithm 3 with the input \((P_1, P_2, P_3) = (\{1, 3\}, \{4, 5\}, \{2\})\).

1. Input: \((P_1, P_2, P_3) = (\{1, 3\}, \{4, 5\}, \{2\})\).
2. Initial values: \( B = \{x_1\}, A_0 = \{1\}, i = 1.\)
Algorithm 3: function $\varphi_r$

input : $(P_1, \ldots, P_r) \in S_r$.
output: $B \in T_r$.

$B \leftarrow \{x_1\}$;
$A_0 \leftarrow \{1\}$;
$i \leftarrow 1$;

for $i = 1$ do
    $A_i \leftarrow L_r \cap P_{A_{i-1}}$;
    if $A_i = \emptyset$ then
        return $B$
    end if
    else
        $B = B \cup X_{A_i}$;
        $i = i + 1$
    end if
end for

3. Recall that $L_3 = \{2, 3\}$. First loop:

$A_1 = L_3 \cap P_{A_0} = L_3 \cap P_1 = \{2, 3\} \cap \{1, 3\} = \{3\} \neq \emptyset$.

Hence $B = \{x_1\} \cup X_{A_1} = \{x_1\} \cup \{x_3\} = \{x_1, x_3\}$,
$i = 1 + 1 = 2$.

4. Second loop:

$A_2 = L_3 \cap P_{A_1} = \{2, 3\} \cap P_2 = \{2, 3\} \cap \{2\} = \{2\} \neq \emptyset$.

Hence $B = \{x_1, x_3\} \cup \{x_2\} = \{x_1, x_2, x_3\}$,
$i = 2 + 1 = 3$.

5. Third loop:

$A_3 = \{2, 3\} \cap P_{A_2} = \{2, 3\} \cap P_2 = \{2, 3\} \cap \{4, 5\} = \emptyset$.

Return $B = \{x_1, x_2, x_3\}$.

6. Output: $B = \{x_1, x_2, x_3\}$.

Proposition 7.5. In the above defined process (Algorithm 3), $A_i \cap A_j = \emptyset$ for all $i \neq j$.

Proof. When $i = 0, j \neq 0$, we have $A_0 \cap A_j = \emptyset$ since $A_j \subset L_r, A_0 = \{1\}$ and $1 \notin L_r$. Suppose $A_i \cap A_j \neq \emptyset$ when $i, j > 0$. W.l.o.g., assume $i < j$.

since $A_i := L_r \cap P_{A_{i-1}}$, we obtain that $L_r \cap P_{A_{i-1}} \cap P_{A_{j-1}} \neq \emptyset$, hence $A_{i-1} \cap A_{j-1} \neq \emptyset$. Repeating the similar process, after finite steps we reach a situation where $A_0 \cap A_{j-i} \neq \emptyset$. This is a contradiction and we see this from the beginning part of this proof. \qed
Algorithm 3 we know that
∪
A
X
From Remark 7.6 we know that there exists
F
and
B
Proposition 7.9. If
proposition.
For any
formulated as the following lemma.
and
basic property of any mapping, we obtain
for all
i
Remark 7.6. Since
|L_r| < \infty, A_i \subset L_r \text{ for all } 1 \leq i \leq r, \text{ and } A_i \cap A_j = \emptyset \text{ for all } i \neq j, \text{ there must exist } i \in \mathbb{N}^+ \text{ such that } A_i = \emptyset. \text{ Therefore, this process terminates. And it is well-defined — once the input is given, the output is uniquely determined via the process and clearly } B \in T_r — \text{ thence it is indeed an algorithm.}

**Proposition 7.7.** Function \( \varphi_r : S_r \to T_r \) is a surjection.

**Proof.** For any \( B \in T_r \), define \( (P_1, \ldots, P_r) \) as follows:

- Define \( Q := \{ i \mid i \in L_r, x_i \in B \} \). Since \( |Q| < \infty \), we can list its elements as: \( q_1, \ldots, q_t \), assuming that \( |Q| = t \).
- Let \( P_1 := \{ q_1 \}, P_{i+1} = \{ q_{i+1} \} \) for \( 1 \leq j \leq t - 1 \).
- Now, we already defined all \( P_i \) when \( i \in L_r \) and \( x_i \in B \) except for \( P_t \), and we have defined \( P_1 \) as well.
- Let \( P_i = \{ i \} \) if \( i \in L_r \) but \( x_i \notin B \). Let \( P_t := \{ 1, \ldots, s_r \} \setminus \cup_{i \notin t} P_i \).

It is clear that \( 1 \in P_t \), hence all parts defined above are non-empty. Also, it is not hard to see that they are indeed a partition of \( 1, \ldots, s_r \) into \( r \) parts. Hence the input is well-defined. Furthermore, one can check that \( \varphi_r(P_1, \ldots, P_r) = B \).

We see that \( \varphi_r \) is a well-defined surjective function. Then, using a basic property of any mapping, we obtain

\[
\bigcup_{B \in T_r} \varphi_r^{-1}(B) = S_r
\]

and

\[
|S_r| = \sum_{B \in T_r} |\varphi_r^{-1}(B)| = \sum_{(B,X \in B)} |\varphi_r^{-1}(B)|.
\]

In order to prove the identity, we only need to show one thing and it is formulated as the following lemma.

**Lemma 7.8.** For any \( B_1 \in T_r \), define \( B_2 := X_r \setminus B_1 \), then

\[
|\{ x \in S_r \mid \varphi_r(x) = B_1 \}| = \left( \begin{array}{c} s_r - r + 1 \\ s_r - r + 1 \end{array} \right) \left( S(B_1) \begin{array}{c} S(B_2) - |B_2| \\ B_1 \\ B_2 \end{array} \right).
\]

In order to prove the above lemma, we need to introduce the following proposition.

**Proposition 7.9.** If \( \varphi_r(P_1, \ldots, P_r) = B_1 \) for some \( (P_1, \ldots, P_r) \in S_r \) and \( B_1 \in T_r \); denote \( B_2 := X_r \setminus B_1 \). Then \( P_{F_{B_1}} \cap L_r = F_{B_1} \setminus \{ 1 \} \), where \( F_{B_0} := \{ i \mid x_i \in B \} \). Consequently, we have \( P_{F_{B_2}} \cap L_r = F_{B_2} \) and \( |P_{F_{B_2}} \cap L_r| = |B_2| \).

**Proof.** From Remark 7.6 we know that there exists \( k \in \mathbb{N}^+ \) such that \( A_k = \emptyset \). Let \( t := k - 1 \), we claim that \( \cup_{i=0}^t A_i = F_{B_1} \). It is clear from Algorithm 3 that \( B_1 = \cup_{i=0}^t X_{A_i} \), which is equivalent to \( F_{B_1} = \cup_{i=0}^t A_i \). Hence we know that \( F_{B_1} \setminus \{ 1 \} = \cup_{i=0}^t A_i \). We only need to show \( P_{F_{B_1}} \cap L_r = \cup_{i=0}^t A_i \).

For any \( m \in \cup_{i=0}^t A_i \), there exists \( 1 \leq j \leq t \) such that \( m \in A_j \). From Algorithm 3 we know that \( A_j := L_r \cap P_{A_j-1} \). Hence \( m \in L_r \). And \( X_{A_{j-1}} \subset B_1 \) is equivalent to \( A_j \subset F_{B_1} \), which implies \( P_{A_{j-1}} \subset P_{F_{B_1}} \).
Hence \( m \in P_{A_{j-1}} \subset P_{F_{B_1}} \). We obtain that \( m \in P_{F_{B_1}} \cap L_r \). For any \( m \in P_{F_{B_1}} \cap L_r \), equivalently we have \( m \in \bigcup_{i=0}^t A_i \cap L_r = \bigcup_{i=1}^t A_i \cap L_r \).

Since \( A_i \subset L_r \) for any \( i \neq 1 \). We obtain that \( m \in \bigcup_{i=1}^t A_i \).

Since \( P_{F_{B_1}} \cup P_{F_{B_2}} = \{ 1, \ldots, s_r \} \) and \( L_r \subset \{ 1, \ldots, s_r \} \), we know that \((P_{F_{B_1}} \cap L_r) \cup (P_{F_{B_1}} \cap L_r) = L_r \). Therefore

\[
P_{F_{B_2}} \cap L_r = L_r \setminus (P_{F_{B_1}} \cap L_r) = L_r \setminus (F_{B_1} \setminus \{ 1 \}) = F_{B_2}.
\]

Then, \( |P_{F_{B_2}} \cap L_r| = |B_2| \) follows naturally, since \( |F_{B_2}| = |B_2| \).

Given \( B_1 \in T_r \), by Proposition 1.9, we know that \( P_{F_{B_1}} \cap L_r = F_{B_1} \setminus \{ 1 \} \), hence the special elements in \( P_{F_{B_1}} \) are fixed — so do the special elements in \( P_{F_{B_2}} \) since \( P_{F_{B_2}} \cap L_r = F_{B_2} \), where \( B_2 := X_r \setminus B_1 \).

Let \( K_r := \{ 1, \ldots, s_r \} \). Also, it is evident that \( P_{F_{B_1}} \cup P_{F_{B_2}} = K_r \).

Inspired by Proposition 1.9, in order to figure out what configurations in \( S_r \) have value \( B_1 \) under \( \varphi_r \), for some given \( B_1 \in T_r \), we view the problem in the following way.

Given \( B_1 \in T_r \), we know that the special elements in \( P_{F_{B_1}} \) are fixed. We only need to choose a proper amount of elements in \( K_r \setminus L_r \) and put them into \( P_{F_{B_1}} \). We call the elements in \( K_r \setminus L_r \) non-special. We need to choose \( |P_{F_{B_1}}| - (|B_1| - 1) = S(B_1) = |B_1| + 1 \) many elements from 

\[
|K_r \setminus L_r| = s_r - |L_r| = s_r - r + 1
\]

many elements, and put them in the group of \( P_{F_{B_1}} \). There are

\[
\binom{s_r - r + 1}{S(B_1) - |B_1| + 1}
\]

many ways to do so. Since \((S(B_1) - |B_1| + 1) + (S(B_2) - |B_2|) = S(B_1) + S(B_2) - (|B_1| + |B_2|) + 1 = s_r - r + 1\), we can also say that there are

\[
\binom{s_r - r + 1}{S(B_2) - |B_2|}
\]

many ways to arrange the non-special elements. Considering the definition of \( \varphi_r \), we see that no matter how we arrange the elements in \( P_{F_{B_2}} \), the value of \( \varphi_r \) is not influenced. Therefore, there are \( \binom{S(B_2)}{B_2} \) many configurations for the elements in \( P_{F_{B_2}} \). As for the arrangements in \( P_{F_{B_1}} \), they need to obey certain rules in order to guarantee that the value of \( \varphi_r \) is \( B_1 \).

From the analysis above, the first and third coefficients in the equation in Lemma 1.8 are both explained well in a combinatorics way. In order to prove this lemma, we only need to prove that given \( B_1 \in T_r \), the number of configurations for the elements in \( P_{F_{B_1}} \) is exactly \( \binom{S(B_1)}{B_1} \). Recalling the definition of \( \binom{S(B_1)}{B_1} \), one can see that it is equivalent to proving the following proposition.

**Proposition 7.10.** \( f_k(m_1, m_2, \ldots, m_k) = \left( \frac{m_1^{s_k-1}}{m_1}, \ldots, m_k \right), k \in \mathbb{N}^+, m_i \in \mathbb{N}^+, \right) \) \( f_k : (\mathbb{N}^+)^k \rightarrow \mathbb{N} \),

\( (m_1, m_2, \ldots, m_k) \mapsto |\{(P_1, P_2, \ldots, P_k) \in S_k \mid \varphi_k(P_1, P_2, \ldots, P_k) = X_k\}|. \)
Before we approach the proof, we need to introduce a known identity on multinomial coefficients and we will need it in later.

**Lemma 7.11.** For all \( n, m, k_1, \ldots, k_m \in \mathbb{N} \) with \( k_1 + \cdots + k_m = n \), \( n \geq 1 \) and \( m \geq 2 \), we have

\[
\binom{n}{k_1, \ldots, k_m} = \sum_{i=1}^{m} \binom{n-1}{k_1, \ldots, k_i-1, \ldots, k_m}.
\]

**Proof of Proposition 7.10.** Prove by induction. When \( k = 1 \), \( L_1 = \emptyset \), for any \( m_1 \in \mathbb{N}^+ \), we have

\[
|\{(P_1) \in S_1 \mid \varphi_1((P_1)) = \{x_1\}\}| = 1 = \binom{s_1 - 1}{m_1 - 1}
\]

since \( s_1 = m_1 \) in this case. Assume that the proposition holds whenever the number of parameters is less or equal to \( k - 1 \), where \( k \geq 2 \).

When the number of parameters is \( k \), we start the inner induction on \( s_k \). Obviously \( s_k \geq k \). When \( s_k = k \), we know that

\[
m_1 = m_2 = \cdots = m_k = 1.
\]

In the configurations that is mapped to \( X_k \) under \( \varphi_k \), we can choose any element in \( L_k \) for \( P_1 \), say \( i_1 \); there are \( |L_k| = k - 1 \) many possibilities. Then we can choose an element in \( L_k \setminus \{i_1\} \) for \( P_1 \), and so on. Until we choose the element \( i_{k-1} \in L_k \) for \( P_{i_{k-2}} \). Then we already arranged \( k - 2 \) many parts, then the only remaining part \( P_{i_{k-1}} \) can only be \( \{1\} \). In total there are \((k - 1)!\) many configurations. Hence we have

\[
f_k(m_1, m_2, \ldots, m_k) = (k - 1)! = \binom{k}{1, \ldots, 1} = \binom{1}{0, 1, \ldots, 1},
\]

which equals to

\[
\binom{s_k}{m_1 - 1, m_2, \ldots, m_k}.
\]

Assume that the proposition holds whenever the sum of these parameters is less or equal to \( s_k - 1 \), where we can assume \( s_k - 1 \geq k \), i.e., \( s_k \geq k + 1 \). When the sum of these parameters equals \( s_k \), left hand side of the proposition is \( f_k(m_1, m_2, \ldots, m_k) \), by definition we want to count the number of configuration that is mapped to \( X_k \) under \( \varphi_k \). We focus on the distribution of the element \( 1 \in K_k \). Since \( 1 \notin L_k \), it does not influence the value of \( \varphi_k \) on any configuration. So in the case when \( m_i \geq 2 \) for all \( 1 \leq i \leq k \), there are \( k \)-many cases for the distribution of \( 1 \), i.e., it can belong to any part \( P_i \) for \( 1 \leq i \leq k \). Hence in this case we obtain the following identity:

\[
f_k(m_1, m_2, \ldots, m_k) = f_k(m_1 - 1, m_2, \ldots, m_k) + f_k(m_1, m_2 - 1, \ldots, m_k) + \cdots + f_k(m_1, m_2, \ldots, m_k - 1).
\]
Now we can apply the induction hypothesis on the sum of the parameters. Then we obtain the following equation:

\[ f_k(m_1, m_2, \ldots, m_k) = \frac{s_k - 2}{m_1 - 2, m_2, \ldots, m_k} \]

Then by Lemma 7.11, we obtain

\[ f_k(m_1, m_2, \ldots, m_k) = \frac{s_k - 1}{m_1 - 1, m_2, \ldots, m_k} \]

If \( m_i = 1 \) for some \( i \neq 1 \). When we put 1 into \( P_i \), the problem can be reduced to counting the number of corresponding configurations of \( P_j \) for \( j \neq i \), since \( \frac{1}{1} \notin L_k \). Therefore, in this case, considering the distribution of the element 1 gives us the following identity:

\[ f_k(m_1, m_2, \ldots, m_k) = f_k(m_1 - 1, m_2, \ldots, m_k) + f_k(m_1, m_2 - 1, m_3, \ldots, m_k) + f_k(m_1, m_2, m_3 - 1, m_4, \ldots, m_k) + \ldots + f_k(m_1, m_2, \ldots, m_k - 1). \]

By induction hypothesis on \( k \) we obtain

\[ f_{k-1}(m_1, \ldots, m_{i-1}, m_i + 1, \ldots, m_k) \]

\[ = \left( \frac{s_k - m_i - 1}{m_1 - 1, \ldots, m_{i-1}, m_i + 1, \ldots, m_k} \right) \]

\[ = \left( \frac{s_k - 1}{m_1 - 1, \ldots, m_{i-1}, m_i + 1, \ldots, m_k} \right) \]

\[ = \left( \frac{s_k - 2}{m_1 - 1, \ldots, m_{i-1}, 0, m_i + 1, \ldots, m_k} \right) \]

\[ = \left( \frac{s_k - 2}{m_1 - 1, \ldots, m_{i-1}, m_i - 1, m_i + 1, \ldots, m_k} \right). \]

Substituting back this term, we get the same recurrence for \( f_k(m_1, \ldots, m_k) \) as in the case where \( m_i \geq 2 \) for \( 1 \leq i \leq k \). By Lemma 7.11 we as well obtain

\[ f_k(m_1, m_2, \ldots, m_k) = \left( \frac{s_k - 1}{m_1 - 1, m_2, \ldots, m_k} \right). \]

With the same idea, it is not hard to prove that the statement holds however many parameters except for \( m_1 \) equals one. If \( m_1 = 1 \), from the definition of function \( f_k \) and \( \varphi_k \), we know that \( 1 \notin P_1 \). Hence considering the distribution of the element 1, the recurrence
formula becomes

\[ f_r(m_1, m_2, \ldots, m_r) = f_r(m_1, m_2 - 1, \ldots, m_r) + \cdots + f_r(m_1, m_2, \ldots, m_r - 1). \]

Then by induction hypothesis on the sum of the parameters, we obtain

\[ f_k(m_1, m_2, \ldots, m_k) = \left( \begin{array}{c} s_k - 2 \\ m_1 - 1, m_2 - 1, \ldots, m_k \end{array} \right) + \cdots + \left( \begin{array}{c} s_k - 2 \\ m_1 - 1, m_2, \ldots, m_k - 1 \end{array} \right) \]

Now we can apply Lemma 7.11 and then obtain

\[ f_k(m_1, m_2, \ldots, m_k) = \left( \begin{array}{c} s_k - 1 \\ m_2, \ldots, m_k \end{array} \right) = \left( \begin{array}{c} s_k - 1 \\ 0, m_2, \ldots, m_k \end{array} \right) = \left( \begin{array}{c} s_k - 1 \\ m_1 - 1, m_2, \ldots, m_k \end{array} \right). \]

By induction, the proposition holds.

With this, we finished proving Theorem 7.2.

8 Conclusion

In this paper, we introduce an equivalent characterization of the algebraic reduction algorithm for computing the integral value of monomials in the Chow group of zero cycles in the moduli space of stable curves, in a graphical view. This characterization naturally can serve as an algorithm, which is also more efficient in many cases, compare to the algebraic one. Furthermore, making use of this tree-representation, we introduce the balancing condition, which brings much more efficiency to the algorithm and leads to an optimized version of the calculus.

Then, we introduce the application of our algorithm on a specific type of monomials. With the help of our main result, the equivalence between an identity on multinomial coefficient and the value of that specific type of monomials is shown, which further reveals the potential strength of our result. Also, the identity indicates some structural information of the tree-based algorithm, in an algebraic way. In the end, we give the proof of that identity, which completes the story.
9 Acknowledgement

The research was funded by the Austrian Science Fund (FWF): W1214-N15, project DK9.

I sincerely thank Josef Schicho for suggesting me to consider the Laman graph realization counting problem, which then leads to the whole journey on this path. Also, I am truly grateful to Josef Schicho for introducing to me the background knowledge of the problem and the basic properties of the ambient ring etc., providing me with the result of Theorem 3.4 and inspiring me with the idea of adding two extra labels to the adjacent vertices of the edge during the edge-cutting operation — which is the essential point for passing on the properness property to the generated trees.

I am genuinely grateful to Cristian-Silviu Radu for providing me with the rather detailed suggestions on how to formulate the whole algorithm properly, step by step — which to a great extend leads to the completion of this manuscript. Also, I thank Cristian-Silviu Radu for the suggestion on adding on the reduction chain algorithm, so as to give a complete algorithm; this makes the write-up more elegant and integral. Last but not the least, I genuinely thank Cristian-Silviu Radu for helping me formulate the identity and as well its proof in a mathematically proper way.

I am very much grateful to Matteo Gallet for inspiring me with the viewpoint on how to formulate a good introduction — which to a great extent improve the quality of the first section. Also, I sincerely thank Matteo Gallet for answering to me the general questions in intersection theory or specifically about the ambient ring, offering me quite some valuable and detailed suggestions for general mathematical writings.

The birth of Section 7.4 — which not only enhances our comprehension of the main result of the paper but also makes the last section more complete and elegant — was inspired by another discussion with Dongsheng Wu, I thank him a lot for it. Also, I am sincerely thankful to Dongsheng Wu for the instructive discussions on the proof of the identity, specifically for providing me with the main idea of the proof of Proposition 7.10.
References

[1] Fulton William. Intersection theory. Springer Science & Business Media, Volume 2, 2013.

[2] Eisenbud David and Harris Joe. 3264 and all that: A second course in algebraic geometry. Cambridge University Press, 2016.

[3] Knudsen Finn, and Mumford David. The Projectivity of the moduli space of stable curves I: Preliminaries on “det” and “Div”. Mathematica Scandinavica, 39.1 (1977), 19-55.

[4] Knudsen Finn F. The projectivity of the moduli space of stable curves, II: The stacks $M_{g,n}$. Mathematica Scandinavica, 52.2 (1983), 161-199.

[5] Knudsen Finn F. The Projectivity of the Moduli Space of Stable Curves, III: The Line Bundles on $M_{g,n}$, and a proof of the Projectivity of $M_{g,n}$ in Characteristic 0. Mathematica Scandinavica (1983), 200-212.

[6] Keel Sean. Intersection theory of moduli space of stable n-pointed curves of genus zero. Transaction of the American Mathematical Society, 330 (1992), no. 2, 545-574.

[7] Mumford David. Towards an enumerative geometry of the moduli space of curves. Arithmetic and geometry, Birkhäuser, Boston, MA, 1983. 271-328.

[8] Faber Carel. Chow Rings of Moduli Spaces of Curves I: The Chow Ring of $\mathcal{M}_3$. Annals of Mathematics (1990): 331-419.

[9] Faber Carel. Chow rings of moduli spaces of curves II: Some results on the Chow ring of $\mathcal{M}_4$. Annals of Mathematics (1990): 421-449.

[10] Izadi Elham. The Chow ring of the moduli space of curves of genus 5. The moduli space of curves, pp. 267-303. Birkhäuser Boston, 1995.

[11] Tavakol Mehdi. The Chow ring of the moduli space of curves of genus zero. Journal of Pure and Applied Algebra, Volume 221, Issue 4, April 2017, Page 757-772.

[12] Gallet Matteo, Grassegger Georg, Schicho Josef. Counting realizations of Laman graphs on the sphere. The Electronic Journal of Combinatorics, Volume 27, Issue 2 (2020).

[13] Qi Jiayue. A calculus for monomials in Chow group of zero cycles in the moduli space of stable curves. DK-Report, No. 2020-11, September 2020.

[14] Qi Jiayue, Schicho Josef. Five Equivalent Ways to Describe Phylogenetic Trees. [arXiv:2011.11774] preprint.