Some Generalizations of Classical Integer Sequences Arising in Combinatorial Representation Theory

Sasha Verona Malone

Western Kentucky University, sasham@fastmail.com

SOME GENERALIZATIONS OF CLASSICAL INTEGER SEQUENCES ARISING IN COMBINATORIAL REPRESENTATION THEORY

A Thesis
Presented to
The Faculty in the Department of Mathematics
Western Kentucky University
Bowling Green, Kentucky

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science

By
Sasha Verona Malone

December 2020

SOME GENERALIZATIONS OF CLASSICAL INTEGER SEQUENCES
ARISING IN COMBINATORIAL REPRESENTATION THEORY

Date Recommended November 19, 2020

Molly Dunkum

Molly Dunkum, Director of Thesis

[illegible]

Robert Donnelly

[illegible]

Dominic Lanphier

Tom Richmond

Thomas Richmond

[illegible]

Associate Provost for Research and Graduate Education

To my real family.

Amor vincit omnia. Vires in numeris.

CONTENTS

LIST OF FIGURES

SOME GENERALIZATIONS OF CLASSICAL INTEGER SEQUENCES ARISING IN COMBINATORIAL REPRESENTATION THEORY

Sasha Verona Malone December 2020 90 Pages

Directed by: Molly Dunkum, Rob Donnelly, Dominic Lanphier, and Tom Richmond

Department of Mathematics Western Kentucky University

There exists a natural correspondence between the bases for a given finite-dimensional representation of a complex semisimple Lie algebra and a certain collection of finite edge-colored ranked posets, laid out by Donnelly, et al. in, for instance, [Don03]. In this correspondence, the Serre relations on the Chevalley generators of the given Lie algebra are realized as conditions on coefficients assigned to poset edges. These conditions are the so-called *diamond, crossing, and structure relations* (hereinafter *DCS relations.*) New representation constructions of Lie algebras may thus be obtained by utilizing edge-colored ranked posets. Of particular combinatorial interest are those representations whose corresponding posets are distributive lattices.

We study two families of such lattices, which we dub the *generalized Fibonaccian lattices* $L_A^{Fib}(n+1,k)$ and *generalized Catalanian lattices* $L_C^{Cat}(n,k)$. These respectively generalize known families of lattices which are DCS-correspondent to some special families of representations of the classical Lie algebras A_{n+1} and C_n. We state and prove explicit formulae for the vertex cardinalities of these lattices; show existence and uniqueness of DCS-satisfactory edge coefficients for certain values of n and k; and report on the efficacy of various computational and algorithmic approaches to this problem. A Python library for computationally modeling and "solving" these lattices appears as an appendix.

CHAPTER 1

INTRODUCTION

The abstract describes our overall motivation, direction, and goals. Here we describe the organization of this thesis.

Chapter 2 provides the necessary background for the sequel, assuming an undergraduate-level knowledge of group theory, graph theory, and linear algebra. In particular, we introduce the basics of lattice theory, of representation theory, and of the theory of Lie algebras.

In Chapter 3, we present Donnelly's notion of a *supporting graph*, which makes explicit the aforementioned correspondence between posets and representations, following [Don03]. We furnish several examples of such poset-representation correspondences. In particular, we define the *generalized Fibonaccian lattices* $L_A^{Fib}(n+1,k) = \mathscr{F}(n+1,k)$ and illustrate some examples. We also reproduce an explicit formula for $|\mathscr{F}(n+1,k)|$ from [DDMN20] which specializes to the result that $|\mathscr{F}(3,k)|$ is the Fibonacci number F_{2n+2}.

In Chapter 4, we introduce the *generalized Catalanian lattices* $L_C^{Cat}(n,k) = \mathscr{C}(n,k)$ and illustrate some examples. We also prove a nice, explicit, and apparently new formula for $|L_C^{Cat}(n,k)| = \mathscr{C}(n,k)$. Finally, we demonstrate uniqueness of DCS-satisfactory edge coefficients for lattices in this family.

Chapter 5 introduces two well-known combinatorial constructs and proposes a bijection between them. The *topside peakless Motzkin paths* (TPMPs) are lattice paths obeying some simple constraints; we show that the number of TPMPs of length n is given by the nth peakless Motzkin number. *Littlewood-Richardson tableaux* are generalizations of Young tableaux which appear prominently in enumerative combinatorics. We conjecture that Littlewood-Richardson tableaux of a certain shape P_n/Q_n are equinumerous with length-n TPMPs, and provide some empirical evidence to support this claim.

Chapter 6 gives an overview of the algorithms and Python code used to produce the illustrations and some of the foregoing results.

CHAPTER 2

PRELIMINARIES

This chapter introduces the mathematics necessary to read Chapter 3. The main results draw from combinatorial lattice theory and the representation theory of Lie algebras.

2.1 Some aspects of combinatorial lattice theory

In this section we present a number of standard definitions and notions concerning posets and lattices, in particular the *diamond-colored distributive lattices* (DCDLs). DCDLs provide the setting for the graph-theoretic arguments and algorithms appearing in the sequel.

The primary references for this section are [Don18] and [Sta97].

Definition 2.1.1. Let P be a set. A *partial order* is a binary relation $\leqslant$ on P satisfying the following axioms for all $a, b, c \in P$.

1. $\leqslant$ is *reflexive*: $a \leqslant a$;
2. $\leqslant$ is *transitive*: $a \leqslant b$, $b \leqslant c \implies a \leqslant c$; and
3. $\leqslant$ is *antisymmetric*: $a \leqslant b, b \leqslant a \implies a = b$.

A set P equipped with a partial order $\leqslant$ is called a *partially ordered set*, or just *poset*.

Definition 2.1.2. A *total order* on a set P is a partial order $\leqslant$ in which, for all $a, b \in P$, either $a \leqslant b$ or $b \leqslant a$.

Definition 2.1.3. Let P be a poset. If $a, b \in P$, and for any $x \in P$, $a \leqslant x, x \leqslant b \implies x \in \{a, b\}$, we say b *covers* a and write $a \to b$.

Definition 2.1.4. To any poset L we may associate a directed graph H, called a *Hasse diagram*, whose vertex set is L and whose edge set is given by the covering relation; that is, there is an edge $a \to b$ in H only if $a \to b$ in L.

In the sequel we shall speak of lattices and their Hasse diagrams interchangeably.

Definition 2.1.5. Given two posets P, Q, with respective orders $\leqslant_P$, $\leqslant_Q$, we may define their *Cartesian product* $P \boxtimes Q$ as a poset in its own right by equipping the set $P \boxtimes Q = \{(p, q) : p \in P, q \in Q\}$ with the *product order* $\leqslant_{P\boxtimes Q}$, which is defined so that $(p_1, q_1) \leqslant_{P\boxtimes Q} (p_2, q_2)$ iff $p_1 \leqslant_P p_2$ and $q_1 \leqslant_Q q_2$.

Note that $(p_1, q_1) \to (p_2, q_2)$ in $P \boxtimes Q$ iff $p_1 \to p_2$ in P and $q_1 = q_2$ or vice versa; that is, the Hasse diagram of the product is the Cartesian product of the Hasse diagrams of its components.

Definition 2.1.6. An element of a poset $m \in P$ is *maximal* if $m \leqslant x \implies x = m$, and *maximum* if $x \leqslant m$ for all $x \in P$. The dual notions (*minimal*, *minimum*) are apparent. We represent the maximum (minimum) element of the poset P, if it exists, by $\top_P$ ($\bot_P$.)

Definition 2.1.7. A finite poset is said to be *ranked* if there exists a function $\rho : P \to \mathbb{N} \cup \{0\}$ such that

1. for any $a, b \in L$, $\rho(b) - \rho(a) = 1$ whenever $a \to b$;
2. if $x \in P$ is minimal, $\rho(x) = 0$.

For any natural number n, the nth *rank* of a ranked poset (P, ρ) is the preimage $\rho^{-1}[n]$.

Definition 2.1.8. The *height* of a finite ranked poset is $h(L) = \sup_{x\in L} \rho(x)$.

Definition 2.1.9. The *rank-generating function* of a ranked poset is the formal power series

$$RGF(L;q) = \sum_{x \in L} q^{\rho(x)} = \sum_{n=0}^{\infty} \left|\rho^{-1}[n]\right| q^n.$$

Definition 2.1.10. Given a poset P, a *weight function* $\omega : P \to \mathbb{Z}^n$ assigns to each poset element x an n-tuple of integer *weights* $\omega(x) = [\omega_1(x), \quad \ldots, \quad \omega_n(x)]$.

Definition 2.1.11. For a given poset P and weight function $\omega : P \to \mathbb{Z}^n$, the *weight-generating function* of a poset is the formal Laurent series

$$WGF(L;\mathbf{z}) = \sum_{x \in L} \mathbf{z}^{\omega(x)} = \sum_{x \in L} \prod_{i=1}^{n} z_i^{\omega_i(x)}.$$

Definition 2.1.12. An *order ideal* (sometimes *down-set* or *lower set*) of a poset P is a set $I \subseteq P$ such that for any $x \in P$, $y \in I$, if $x \leqslant y$ then $x \in I$.

In finite posets, order ideals can be uniquely identified with their maximal elements; if $I \subseteq P$ is an order ideal, $M \subseteq I$ its set of maximal elements, then $I = \bigcup_{m \in M} \{x \in P : x \leqslant m\}$.

Definition 2.1.13. A *lattice* L is a partially ordered set such that for any $a, b \in L$,

1. the set $\{x \in L : x \leqslant a, x \leqslant b\}$ has a maximum element $a \wedge b$, and
2. the set $\{x \in L : a \leqslant x, b \leqslant x\}$ has a minimum element $a \vee b$.

The symbols $\wedge$, $\vee$ are read as “meet” and “join” respectively.

Definition 2.1.14. A lattice L is *distributive* if for any $a, b, c \in L$, we have $(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$, or equivalently, $(a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c)$; that is, if meet distributes over join, or vice versa.

Definition 2.1.15. Let P be a poset. The poset $J(P)$ is the set of all order ideals of P ordered by containment.

Proposition 2.1. For any poset P, $J(P)$ is a distributive lattice.

Proof. To show $J(P)$ is a lattice, let $I, J \subseteq P$ be order ideals of P. We claim $I \vee J = I \cup J$ and $I \wedge J = I \cap J$.

First we show that $I \cup J$, $I \cap J$ are actually order ideals. Suppose $x \in I \cup J$ and $y \leqslant x$. Without loss of generality assume $x \in I$. Then since I is an order ideal, $y \in I \subset I \cup J$.

Similarly, suppose $x \in I \cap J$ and $y \leqslant x$. Then $x \in I$ and $x \in J$, so $y \in I$, $y \in J$. Thus $y \in I \cap J$.

Now we must show $I \cup J = I \vee J$. Suppose K is an order ideal of P such that $I \subset K$, $J \subset K$. Then clearly $I \cup J \subset K$. Thus $I \cup J$ is the minimum element of the set $\{K \in J(P) : K \leqslant I, K \leqslant J\}$.

Similarly, if K is an order ideal of P such that $K \subset I$, $K \subset J$, then $K \subset I \cap J$. Thus $I \cap J$ is the maximum element of the set $\{K \in J(P) : I \leqslant K, J \leqslant K\}$.

Therefore $J(P)$ is a lattice. Showing that $J(P)$ is distributive amounts to proving that intersections distribute over unions and vice versa, which is well-known. $\square$

In fact, a sort of converse holds; any finite distributive lattice is representable as $J(P)$ for some poset P.

Definition 2.1.16. An element $x \in L$ is said to be *join-irreducible* if $x \neq \min L$ and x cannot be written as $u \vee v$, where $u, v \neq x$.

Proposition 2.2. In a finite lattice L, an element x is join-irreducible iff it covers exactly one element.

Proof. If: By contraposition; if x covers distinct elements $u, v \in L$, then $x = u \vee v$, so x is not join-irreducible. If x covers zero elements of L, then $x = \min L$ and so is not join-irreducible.

Only if: Suppose x covers exactly one element $y \in L$. Then clearly $x \neq \min L$. Now suppose contrariwise that $x = u \vee v$, where $u, v \neq x$. Since $u, v \leqslant x$, we must have $u, v \leqslant y$. But then $u \vee v = y$, which is a contradiction. $\square$

Definition 2.1.17. We denote the subposet of all join-irreducible elements of L, with order induced by L, by $j(L)$.

The following theorem is called the Fundamental Theorem for Finite Distributive Lattices (or *FTFDL*) by [Sta01].

Theorem 2.1 (Birkhoff)**.** For any finite poset P, $j(J(P)) \approx P$. For any finite distributive lattice L, $J(j(L))$ is order isomorphic to L.

The "function"[1] J maps order ideals of P to elements in L; the "function" j goes the other way. Thus, for $I \subseteq P$, $x \in L$, we can consider $J(I) \in L$ and $j(x) \subseteq P$.

The preceding theorem assures that every finite distributive lattice is ranked, viz. $\rho(x) = |j(x)|$. It follows that for any two elements $x, y \in L$ with $x \leqslant y$, $\rho(y) - \rho(x) = |j(y) \backslash j(x)|$.

Colors and components

Definition 2.1.18. A *k-coloring* of a poset P is a function $c : P \to (\mathbb{N} \cap [1, k])$. We say P is *colored by c*.

Definition 2.1.19. Let a poset P be colored by some function c, and consider the Hasse diagram H of $J(P)$ as a directed graph. This graph has an edge from order ideal I to order ideal J iff $J = I \cup \{v\}$ for some $v \in P$. Thus we may naturally color the edge $I \to J$ by $c(v)$. When we regard the edges of $J(P)$ as being colored (by c) in this way, we shall call it a *diamond-colored distributive lattice*, or just a *DCDL.*

Definition 2.1.20. A *diamond* of a distributive lattice is the sublattice induced by four elements w, x, y, z such that $w \to x \to z$, $w \to y \to z$. In the Hasse diagram, we say the edge $w \to x$ is *parallel to $y \to z$*. Similarly, $w \to y$ is parallel to $x \to z$. (See Figure 2.1.)

[1] We stop short of defining these as functions for technical issues; strictly speaking, the "set" of all posets is actually a proper class.

$$\begin{tikzcd}
& z & \\
x \arrow[ur] & & y \arrow[ul] \\
& w \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Figure 2.1: A typical diamond.

We may regard parallelism as being naturally transitive; then the parallel edges of any DCDL fall into “monochromatic” equivalence classes.

Proposition 2.3. In a DCDL, parallel edges have the same color.

Proof. Let L be a DCDL colored by c, and let w, x, y, z be the vertices of a diamond, identified as in the above definition. Consider the order ideals $j(w), j(z)$. The set $j(z) \backslash j(w) \subset P$ has two elements u, v. Without loss of generality, write $j(x) = j(w) \cup \{u\}$ and $j(y) = j(w) \cup \{v\}$. Then $j(z) = j(x) \cup \{v\} = j(y) \cup \{u\}$. Thus the edges $w \to x$ and $y \to z$ have color $c(u)$ while the other two edges have color $c(v)$. $\square$

Definition 2.1.21. Let L be a DCDL whose edges are colored by $[n]$, and let $k \in [n]$. A *k-monochromatic component* of L or just *k-component* is a subset $C \subset L$ such that

1. C is a subposet of L under the induced order;
2. if $x \to y$ in C, then $x \to y$ in L;
3. C is maximal with respect to the property that all edges have color k.

Next, we prove the (nontrivial) fact that any k-component of a DCDL has a distributive lattice structure. The following definition, which describes the “preimage” under J of a k-component, will be necessary.

Definition 2.1.22. Let P be a poset with vertices colored by $[n]$, and let $k \in [n]$. A *k-subordinate* of P is a subset $S \subseteq P$ such that

1. S is a subposet of P under the induced order;
2. all elements of S have color k;
3. there exist order ideals $I, J \subset P$ where no maximal element of I has color k and no minimal element of $P \backslash J$ has color k and $S = J \subset I$.

We say that I and J are *bounding ideals* for S; I is *lower bounding* and J is *upper bounding.*

The next proposition establishes a one-to-one correspondence between k-subordinates of a poset P and k-components of its corresponding lattice $L = J(P)$.

Proposition 2.4. Let $L = J(P)$ be a DCDL. Let k be an edge or vertex color.

1. Let C be a k-component of L. Then C is a (distributive) sublattice of L, $S = \top_C \backslash \bot_C$ is a k-subordinate of P with bounding ideals $\top_C$ and $\bot_C$, and $C \approx J(S)$.
2. Let S be a k-subordinate of P with lower bounding ideal I and upper bounding ideal J. Then I and J are the minimum and maximum elements of some k-component $C \subset L$, and $j(C) \approx S$.

Proof. See [Don18], Proposition 13 and Theorem 14. $\square$

Proposition 2.5. Let C be a k-component of L. Then C can be written as a Cartesian product of chains if and only if its corresponding k-subordinate $S \subseteq j(L)$ can be written as a disjoint union of chains.

Proof. If: Suppose $S \subseteq P = j(L)$ can be written as a disjoint union of chains: $S = \bigsqcup_{i=1}^{n} C_i$. We make the following claims.

1. $C \approx J(S) = J(\bigsqcup_{i=1}^{n} C_i)$. This follows from the FTFDL and the preceding proposition.
2. $J(\bigsqcup_{i=1}^{n} C_i) = \boxtimes_{i=1}^{n} J(C_i)$. To see this, note that every order ideal of $\bigsqcup_{i=1}^{n} C_i$ corresponds uniquely to a choice of at most one element from each C_i.

3. Each $J(C_i)$ is a chain with $|J(C_i)| = |C_i| + 1$.

Only if: Since the correspondences between order ideals of S and choices of elements from each C_i and between S and C are both one-to-one, this proof reverses. $\square$

Incremental lattices

This section characterizes distributivity in a way more genial to our needs.

Definition 2.1.23. The *n-dimensional integer lattice* $\mathbb{Z}^n$ is the set of all integer n-tuples ordered elementwise.

If $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^n$, then $\mathbf{x} \vee \mathbf{y}$ (resp. $\mathbf{x} \wedge \mathbf{y}$) is given by the elementwise maximum (resp. minimum) of $\mathbf{x}$ and $\mathbf{y}$.

Lemma 2.1. Any sublattice of $\mathbb{Z}^n$ is distributive.

Proof. It suffices to show that, for $x, y, z \in \mathbb{Z}$, we have $\min\{\max\{x, y\}, \max\{x, z\}\} = \max\{x, \min\{y, z\}\}$. By symmetry we need only check the cases $x \leqslant y \leqslant z$, $y \leqslant x \leqslant z$, $y \leqslant z \leqslant x$. This verification is left to the reader. $\square$

Definition 2.1.24. A lattice L is *incremental* if there is an injective homomorphism $\phi : L \to \mathbb{Z}^n$ for some n. We say that L *embeds into* $\mathbb{Z}^n$ and call the function ϕ an *embedding.*

Proposition 2.6. A finite lattice is incremental if and only if it is distributive.

Proof. If. Suppose L is a finite distributive lattice. By the FTFDL, L is the lattice of order ideals of some poset $P = j(L)$. Write the vertex set of P as a union of disjoint chains, say $P = \bigcup_{i=1}^n C_i = \bigcup_{i=1}^n \{c_{i,k}\}_k$, where $c_{i,1} \to c_{i,2} \to \dots$

Now define $\phi : L \to \mathbb{Z}^n$ as follows: for each $x \in L$ and $i = 1, \dots, n$, the ith component of $\phi(x)$ is $\phi(x)_i = |C_i \cap j(x)|$. It is apparent that this value is the maximum $k = 1, 2, \dots$ for which $c_{i,k} \in j(x)$, or 0 if C_n and $j(x)$ are disjoint.

Since $P = j(L)$ is finite, ϕ is well-defined. We make the following claims.

1. ϕ is injective. Suppose $\phi(x) = \phi(y)$; then the maximal elements of $j(x)$ and $j(y)$ are the same, so $x = y$.
2. ϕ is a homomorphism. Let $x, y \in L$. Then for any i,

$$\begin{aligned}\phi(x \vee y)_i &= |C_i \cap j(x \vee y)| \\ &= |C_i \cap (j(x) \cup j(y))| \\ &= |(C_i \cap j(x)) \cup (C_i \cap j(y))| \\ &= \max\{|C_i \cap j(x)|, |C_i \cap j(y)|\} \\ &= \max\{\phi(x)_i, \phi(y)_i\}.\end{aligned}$$

   An analogous argument shows that $\phi(x \wedge y)_i = \min\{\phi(x)_i, \phi(y)_i\}$.

Only if. Suppose L is a finite incremental lattice. Let $\phi : L \to \mathbb{Z}^n$ be an injective homomorphism. Suppose contrariwise that L is not distributive. Then there exist $x, y, z \in L$ so that $(x \vee y) \wedge (x \vee z) \neq x \vee (y \wedge z)$. Since ϕ is injective, we must have $\phi((x \vee y) \wedge (x \vee z)) \neq \phi(x \vee (y \wedge z))$. Since ϕ is a homomorphism, we have $(\phi(x) \vee \phi(y)) \wedge (\phi(x) \vee \phi(z)) \neq \phi(x) \vee (\phi(y) \wedge \phi(z))$. But $\phi(L)$ is a sublattice of $\mathbb{Z}^n$, and so is distributive; so this is a contradiction. □

Therefore it suffices to study the distributive lattices in Chapter 3 as sublattices of $\mathbb{Z}^n$. Doing so gives us a nice characterization of their join-irreducible elements.

Proposition 2.7. Let L be a finite distributive lattice, and let $\phi : L \to \mathbb{Z}^n$ be an embedding. Write ϕ as a vector $[\phi_1, \phi_2, \ldots, \phi_n]$, where $\phi_k : L \to \mathbb{Z}$. Say $x \in L$ is *k-decrementable* if there is another element $y \in L$ such that $\phi_i(x) = \phi_i(y)$ if $i \neq k$ and $\phi_k(x) = \phi_k(y) + 1$. Then $x \in L$ is join-irreducible if and only if it is k-decrementable for exactly one k.

Proof. $x \in L$ is k-decrementable for exactly one k if and only if it covers exactly one element. □

Corollary 2.1. With the above setup, let $\phi_k(L) = \{m_k, m_k + 1, ..., M_k\}$ be the image of ϕ_k in $\mathbb{Z}$. For each $l \in \phi_k(L)$, let $L_{k,l}$ be the sublattice of l formed by those elements that have kth coordinate l. Let $\bot_{k,l} = \min L_{k,l}$. Then

$$j(L) = \bigcup_{k=1}^{n} \bigcup_{l=m_k+1}^{M_k} \{\bot_{k,l}\}.$$

Proof. First, we claim that any element $\bot_{k,l}$ covers at most one element. Suppose contrariwise that $\bot_{k,l}$ covers two elements; then it is decrementable in two positions k, k'. But since L is distributive, decrementing k' gives an element $\bot'_{k,l} \leqslant \bot_{k,l}$, and by construction $\bot'_{k,l} \in L_{k,l}$, which contradicts the minimality of $\bot_{k,l}$.

Obviously $x_{k,m_k} = \bot$, which is not join-irreducible. Thus the right-hand set is included in the left-hand set.

By the proposition, any join-irreducible element $y \in L$ is k-decrementable for exactly one k. We then have $y = \bot_{k,\phi_k(y)}$, and $\phi_k(y) > m_k$. Thus the left-hand set is included in the right-hand set. $\square$

Definition 2.1.25. Let C be a chain-factorizable lattice; i.e., let $C = \prod_{i=1}^{n} C_n$ where the C_i are chains. Let $\phi = [\phi_1, \phi_2, \ldots, \phi_n]$ be an embedding of C into $\mathbb{Z}^n$, and let $\phi_k(C) = \{m_k, m_k + 1, \ldots M_k\}$ be the image of ϕ_k in $\mathbb{Z}$. Then

1. the *k-slice through j* is the set $C_{k,j} := \{c \in C : \phi_k(c) = j\}$, and
2. if $j \in \{m_k, M_k\}$ (that is, j is extremal,) $C_{k,j}$ is called a *face* of C.

Faces of chain-factorizable lattices (specifically, chain-factorizable *monochromatic components*) play an important role in Chapter 3.

2.2 Lie theory

A full exposition of the theory of Lie algebras and their representations is well beyond the scope of this paper. We provide here only the necessary definitions and theorems to read §3.

We loosely follow [Hum72]. An introduction to the subject more suitable for advanced undergraduates is [EW06].

Definition 2.2.1. A *Lie algebra* is a vector space $\mathfrak{g}$ together with a *bracket* operation $[\cdot,\cdot] : \mathfrak{g} \times \mathfrak{g} \to \mathfrak{g}$ satisfying the following axioms for each $x, y, z \in \mathfrak{g}$ and scalars a, b:

1. *bilinearity:* $[ax + by, z] = a[x, z] + b[y, z]$;
2. *anticommutativity:* $[x, y] = -[y, x]$;
3. the *Jacobi identity:* $[x, [y, z]] + [y, [z, x]] + [z, [x, y]] = 0$.

When no confusion can result, we shall sometimes write the bracket operator $[X, Y]$ as just $[XY]$, omitting the comma.

A familiar example of a Lie algebra is the vector space $\mathbb{R}^3$ with the cross product as bracket. We leave the verification of the above axioms in this case to the reader. We shall now produce some more relevant examples.

Definition 2.2.2. The *general linear algebra* over an abstract vector space V, written $gl(V)$, is the algebra of linear endomorphisms over V with function composition as multiplication. For $n \in \mathbb{N}$ and a field $\mathbb{F}$, we identify $gl(V)$ with the algebra of $n \times n$ matrices over V by fixing a basis.

Proposition 2.8. Any nonempty subset of $gl(V)$ closed under the *commutator* operation defined by $[X, Y] := XY - YX$ forms a Lie algebra, with the commutator as bracket.

Proof. Clearly the commutator is bilinear, viz.

$$\begin{aligned}
[aX + bY, Z] &= (aX + bY)Z - Z(aX + bY) \\
&= aXZ + bYZ - aZX + bZY \\
&= a(XZ - ZX) + b(YZ - ZY) \\
&= a[XZ] + b[YZ],
\end{aligned}$$

and anticommutative, viz. $[XY] = XY - YX = -[YX]$.

Thus we have to show that arbitrary elements X, Y, Z of $gl(V)$ obey the Jacobi law $[X,[YZ]] + [Y,[ZX]] + [Z,[XY]] = 0$. We calculate:

$$\begin{aligned}
& [X,[YZ]] + [Y,[ZX]] + [Z,[XY]] \\
= \quad & [X,(YZ - ZY)] + [Y,(ZX - XZ)] + [Z,(XY - YX)] \\
= \quad & X(YZ - ZY) - (YZ - ZY)X \\
& + Y(ZX - XZ) - (ZX - XZ)Y \\
& + Z(XY - YX) - (XY - YX)Z \\
= \quad & XYZ - XZY - YZX + ZYX \\
& + YZX - YXZ - ZXY + XZY \\
& + ZXY - ZYX - XYZ + YXZ \\
= \quad & 0. \qquad \square
\end{aligned}$$

Thus we can furnish examples of Lie algebras simply by producing collections of matrices closed under the bracket.

Classical Lie algebras

The theory of general Lie algebras is quite rich. For our purposes it will suffice to limit our field of view to the following four families, which for historical reasons are collectively referred to as the *classical* Lie algebras.

1. The special linear algebra A_n

   The *trace* of a square matrix $M = (m_{ij})_{1 \leqslant i,j \leqslant n}$ is the sum of its diagonal elements $\operatorname{tr} M = \sum_{i=1}^{n} m_{ii}$.

   Proposition 2.9. Let M be an $n \times n$ matrix. Then $\operatorname{tr} M$ is invariant under changes of basis.

Proof. Let B be an invertible $n \times n$ matrix, and let $M_B = B^{-1}MB$. Denote the (ij) entry of M by $[M]_{ij}$. We have

$$\begin{aligned}
\operatorname{tr} M_B &= \sum_{i=1}^{n}[B^{-1}MB]_{ii} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n}[B^{-1}]_{ij}[MB]_{ji} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n}[B^{-1}]_{ij}\sum_{k=1}^{n}[M]_{jk}[B]_{ki} \\
&= \sum_{k=1}^{n}\sum_{j=1}^{n}[M]_{jk}\sum_{i=1}^{n}[B]_{ki}[B^{-1}]_{ij} \\
&= \sum_{k=1}^{n}\sum_{j=1}^{n}[M]_{jk}[I_n]_{kj} \\
&= \sum_{k=1}^{n}\sum_{j=1}^{n}[M]_{jk}\delta_{jk} \\
&= \operatorname{tr} M.
\end{aligned}$$

□

Proposition 2.10. The set of $n \times n$ matrices with zero trace forms a Lie algebra.

Proof. Let M, M' be $n \times n$ matrices.[2] Then

$$\begin{aligned}
\operatorname{tr}[MM'] &= \operatorname{tr}(MM' - M'M) \\
&= \sum_{i=1}^{n}(MM' - M'M)_{ii} \\
&= \sum_{i=1}^{n}(MM')_{ii} - (M'M)_{ii} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}M'_{ji} - \sum_{i=1}^{n}\sum_{j=1}^{n} M'_{ij}M_{ji} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}M'_{ji} - \sum_{j=1}^{n}\sum_{i=1}^{n} M_{ji}M'_{ij} \\
&= \sum_{i=1}^{n}(MM')_{ii} - \sum_{j=1}^{n}(MM')_{jj} = 0.
\end{aligned}$$

□

[2]It suffices to show that if two matrices each have zero trace, so does their commutator. In fact the commutator of any two matrices has zero trace, which is what we actually show.

Definition 2.2.3. The Lie algebra of $(n+1) \times (n+1)$ matrices with zero trace is called the *special linear algebra of rank* n and denoted A_n.

2. The orthogonal algebras B_n, D_n

Definition 2.2.4. A matrix M is *skew-symmetric* if $M^\top = -M$.

Proposition 2.11. Let S be a symmetric matrix. Then the set of all matrices M such that SM is skew-symmetric forms a Lie algebra.

Proof. Suppose $(SM)^\top = -SM$, $(SN)^\top = -SN$. We need to show that $(S[MN])^\top = -S[MN]$. We have

$$\begin{aligned}
(S[MN])^\top &= (SMN)^\top - (SNM)^\top \\
&= N^\top M^\top S^\top - M^\top N^\top S^\top \\
&= N^\top M^\top S - M^\top N^\top S \\
&= N^\top(-SM) - M^\top(-SN) \\
&= -N^\top SM + M^\top SN \\
&= (SN)M - (SM)N = -S[MN].
\end{aligned}$$
□

Definition 2.2.5. If $n = 2k+1$ is odd, let $S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & I_k \\ 0 & I_k & 0 \end{bmatrix}$. Then the set of all matrices M such that SM is skew-symmetric forms the *odd orthogonal Lie algebra of rank* k, denoted B_k.

Definition 2.2.6. If $n = 2k$ is even, let $S = \begin{bmatrix} 0 & I_k \\ I_k & 0 \end{bmatrix}$. Then the set of all matrices M such that SM is skew-symmetric forms the *even orthogonal Lie algebra of rank* k, denoted D_k.

3. The symplectic algebra C_n

Definition 2.2.7. A $2n \times 2n$ matrix M is *symplectic* if the matrix $SM = \begin{bmatrix} \mathbf{0}_n & \mathbf{I}_n \\ -\mathbf{I}_n & \mathbf{0}_n \end{bmatrix} M$ is symmetric.

Proposition 2.12. The set of $2n \times 2n$ symplectic matrices forms a Lie algebra, called the *symplectic algebra of rank* n and denoted C_n.

Proof. Suppose M, N are $2n \times 2n$ matrices such that $SM = (SM)^\top$, $SN = (SN)^\top$. Then

$$\begin{aligned} S[MN] &= SMN - SNM \\ &= (SM)^\top N - (SN)^\top M \\ &= M^\top S^\top N - N^\top S^\top M \\ &= -M^\top SN + N^\top SM \\ &= -M^\top (SN)^\top + N^\top (SM)^\top \\ &= -(SNM)^\top + (SMN)^\top \\ &= (SMN - SNM)^\top = (S[MN])^\top. \end{aligned}$$ □

Ideals and simplicity

Definition 2.2.8. Given a Lie algebra $\mathfrak{g}$, a subalgebra $\mathfrak{h}$ is an *ideal* of $\mathfrak{g}$ provided that for any $h \in \mathfrak{h}$, $g \in \mathfrak{g}$, $[gh] \in \mathfrak{h}$.

Definition 2.2.9. A function $\phi : \mathfrak{g} \to \mathfrak{g}'$ is a *homomorphism of Lie algebras* provided that it is linear and it preserves the bracket.

The usual homomorphism theorems hold, in particular the following:

Theorem 2.2. Let $\phi : \mathfrak{g} \to \mathfrak{g}'$ be a homomorphism of Lie algebras. Then $\ker \phi = \{x \in \mathfrak{g} : \phi(x) = 0\}$ is an ideal of $\mathfrak{g}$.

Proof. Left to reader. □

Definition 2.2.10. A Lie algebra $\mathfrak{g}$ is said to be *simple* if its only ideals are 0 and $\mathfrak{g}$ (and it is not the one-dimensional algebra.)

Definition 2.2.11. A Lie algebra is said to be *semisimple* if it can be written as a direct sum of simple Lie algebras.

For the purposes of this paper, it will suffice to think of semisimple Lie algebras as being well-behaved. The examples we consider in the sequel are all semisimple.

Toral subalgebras and roots

We take a short recess here to discuss certain combinatorial-geometric objects called *root systems.* This discussion, while appearing somewhat unmotivated, turns out to be crucial to understanding the structure of semisimple Lie algebras.

Definition 2.2.12. Consider $\mathbb{R}^n$ as a vector space with the standard (Euclidean) inner product. The *normal hyperplane* to $v \in \mathbb{R}^n$, denoted $v^\perp$, is the set of all vectors orthogonal to v.

Proposition 2.13. For any $v \in \mathbb{R}^n$, there exists a unique endomorphism r_v such that

1. $r_v(w) = w$ for every $w \in v^\perp$, and
2. $r_v(v) = -v$,

which we shall call the *reflection through* $v^\perp$.

Proof. Let B be a basis of $\mathbb{R}^n$ containing v. Orthogonalizing B with respect to v gives a basis $B' = (v, w_1, \dots, w_{n-1})$. Note $v^\perp$ is a $(n-1)$-dimensional vector space spanned by $B' \backslash \{v\}$. Let r_v be the endomorphism whose matrix relative to B' is $\operatorname{diag}(-1, 1, \dots, 1)$. Then it is easily checked that $r_v(w) = w$ for $w \in v^\perp$ and $r_v(v) = -v$. □

The above proof makes clear that $r_v^2 = I_n$, as befits its name. We now give an explicit formula for r_v.

Proposition 2.14. For any $v \in \mathbb{R}^n$, $r_v(w) = w - 2\operatorname{proj}_v w$.

Proof. Without loss of generality, take w relative to the basis B'. Then

$$r_v = \operatorname{diag}(-1, 1, \ldots, 1) = I_n - 2\operatorname{diag}(1, 0, \ldots, 0);$$

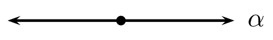
$$r_v(w) = w - 2\operatorname{diag}(1, 0, 0, 0)w = w - 2\operatorname{proj}_v w. \qquad \square$$

The number $\langle v, w\rangle = 2\frac{\|\operatorname{proj}_v w\|}{\|v\|} = 2\frac{v\cdot w}{v\cdot v}$ denotes the scalar multiple of v separating w and $r_v(w)$.

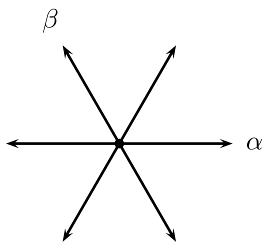
We are now ready for the central definition.

Definition 2.2.13. Let $\mathbb{R}^n$ be a vector space with the usual inner product. Then a finite spanning set of vectors $\Phi \subset \mathbb{R}^n$ is said to be a *root system* if

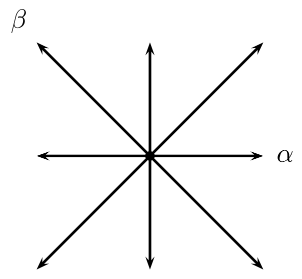
1. $\mathbf{0} \notin \Phi$.
2. For any $\alpha \in \Phi$, the only other scalar multiple of α belonging to Φ is $-\alpha$.
3. For any $\alpha \in \Phi$, $r_\alpha(\Phi) = \Phi$; that is, Φ is symmetric about α.
4. For any $\alpha, \beta \in \Phi$, $\langle\alpha, \beta\rangle$ is an integer.



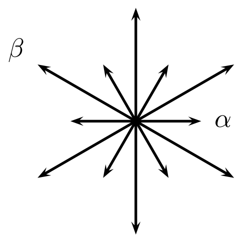
(a) A_1, rank 1



(b) A_2, rank 2



(c) B_2, rank 2



(d) G_2, rank 2

Figure 2.2: Some root systems, their ranks, and their bases.

Definition 2.2.14. A *base* of a rank-n root system Φ is a subset $\Delta \subset \Phi$ such that

1. Δ is a basis for $\mathbb{R}^n$;
2. the nonzero basis coefficients, relative to Δ, of any $\phi \in \Phi$ are either all positive (in which case ϕ is called a *positive root*) or all negative.

For completeness we mention the following fact:

Theorem 2.3. Every root system has a base.

Proof. See [Hum72] §10.1. □

The usual bases of each root system are indicated above with the labels α and β.

At this point we return to our discussion of Lie algebras, which for the remainder of this section we will take to be finite-dimensional over a field $\mathbb{F}$.

Definition 2.2.15. The *center* of a Lie algebra $\mathfrak{g}$ is the set

$$Z(\mathfrak{g}) := \{z \in \mathfrak{g} : [zg] = 0 \quad \forall g \in \mathfrak{g}\}.$$

Proposition 2.15. Any subalgebra of $Z(\mathfrak{g})$, including $Z(\mathfrak{g})$ itself, is an ideal of $\mathfrak{g}$.

Proof. Left to reader. □

Lemma 2.2. If $\mathfrak{g}$ is a simple Lie algebra, then $Z(\mathfrak{g}) = \{0\}$.

Proof. Let $z \in Z(\mathfrak{g})$. $\operatorname{span} z$, the one-dimensional subalgebra spanned by z, is an ideal of $\mathfrak{g}$. Since $\mathfrak{g}$ is not one-dimensional (by definition,) $\operatorname{span} z \neq \mathfrak{g}$. But since $\mathfrak{g}$ is simple, this implies $\operatorname{span} z = \{0\}$. Thus $z = 0$. □

Proposition 2.16. If $\mathfrak{g}$ is a semisimple Lie algebra, then $Z(\mathfrak{g}) = \{0\}$.

Proof. Write $\mathfrak{g}$ as a direct sum of simple algebras, say $\mathfrak{g} = \bigoplus_{k=1}^{n} \mathfrak{g}_i$. Any $z \in Z(\mathfrak{g})$ can then be written as $z = \sum_{k=1}^{n} z_i$, where $z_i \in \mathfrak{g}_i$.

We claim that, in fact, $z_i \in Z(\mathfrak{g}_i)$. To see this, note that for any $g \in \mathfrak{g}_i$, $0 = [zg] = \sum_{k=1}^{n} [z_k g]$. Since $\mathfrak{g}_i$ and $\mathfrak{g}_j$ are orthogonal if $i \neq j$, we have $[z_i g] = 0$.

Since $\mathfrak{g}_i$ is simple, $Z(\mathfrak{g}_i) = \{0\}$, so $z_i = 0$. Thus $z = 0$. □

Definition 2.2.16. For a subalgebra $\mathfrak{h}$ of $\mathfrak{g}$, define the *dual space* $\mathfrak{h}^*$ to be the space of linear functionals $\alpha : \mathfrak{h} \to \mathbb{F}$. For any $\alpha \in \mathfrak{h}^*$, define $\mathfrak{g}_\alpha := \{x \in \mathfrak{g} : [hx] = \alpha(h)x \quad \forall h \in \mathfrak{h}\}$, and define $\Phi := \{\alpha \in \mathfrak{h}^* : \alpha \neq 0, \mathfrak{g}_\alpha \neq 0\}$.

If $\mathfrak{g} = \bigoplus_{\alpha \in \Phi \cup \{0\}} \mathfrak{g}_\alpha$, the subalgebra $\mathfrak{h}$ is called *toral.* Additionally, if this condition is met,

1. elements of Φ are called *roots* of $\mathfrak{g}$ (relative to $\mathfrak{h}$;)
2. for any $\alpha \in \Phi$, $\mathfrak{g}_\alpha$ is called a *root space*;
3. the equality $\mathfrak{g} = \bigoplus_{\alpha \in \Phi \cup \{0\}} \mathfrak{g}_\alpha$ is called the *root space decomposition* of $\mathfrak{g}$ (relative to $\mathfrak{h}$.)

At this point the reader may be forgiven for finding this definition intimidating. We defang it thus.

Example 2.1. Consider $\mathfrak{g} = A_1$, the Lie algebra of 2×2 matrices over $\mathbb{R}$ with zero trace. We claim that the subalgebra of diagonal matrices $\mathfrak{h} = \operatorname{diag} A_1 = \operatorname{span}(1, -1)$ is toral. To show this, we calculate the functionals $\alpha \in \mathfrak{h}^*$ for which $\mathfrak{g}_\alpha$ is not the zero algebra.

Since $\mathfrak{h}$ is one-dimensional, an element $\alpha \in \mathfrak{h}^*$ is just a mapping $h \mapsto ch$, where $c \in \mathbb{R}$. So we look for scalars k and $x \in \mathfrak{g}$ such that $[hx] = khx$, where

$$\begin{aligned} khx &= k \begin{bmatrix} d & \\ & -d \end{bmatrix} \begin{bmatrix} a & b \\ c & -a \end{bmatrix} \\ &= k \begin{bmatrix} ad & bd \\ -cd & ad \end{bmatrix}, \\ [hx] = hx - xh &= \begin{bmatrix} d & \\ & -d \end{bmatrix} \begin{bmatrix} a & b \\ c & -a \end{bmatrix} - \begin{bmatrix} a & b \\ c & -a \end{bmatrix} \begin{bmatrix} d & \\ & -d \end{bmatrix} \\ &= \begin{bmatrix} 0 & 2bd \\ -2cd & 0 \end{bmatrix}. \end{aligned}$$

By inspection, if $k \notin \{0, \pm 2\}$, there are no nonzero satisfactory elements x; and

1. if $k = 0$, any satisfactory x has the form $\operatorname{diag}(a, -a)$;
2. if $k = 2$, any satisfactory x has the form $\begin{bmatrix} 0 & b \\ 0 & 0 \end{bmatrix}$;
3. if $k = -2$, any satisfactory x has the form $\begin{bmatrix} 0 & 0 \\ c & 0 \end{bmatrix}$.

Thus, $\Phi = \{2, -2\}$. (Remember, 0 is never a root.) Observe that $\bigoplus_{\alpha \in \{0\} \cup \Phi} \mathfrak{g}_\alpha = \mathfrak{g}$. Thus $\mathfrak{h}$ is toral in $\mathfrak{g}$, with root spaces $\mathfrak{g}_{\pm 2}$ spanned by $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$ respectively.

Proposition 2.17. Let $\mathfrak{h}$ be a toral subalgebra of $\mathfrak{g}$, and let $k, l \in \mathfrak{h}$. Then $[kl] \in Z(\mathfrak{g})$.

Proof. Let $\alpha \in \Phi$ be a root, and let $\mathfrak{g}_\alpha$ be its corresponding (nonzero) root space. Fix a basis $\{g_{\alpha,i}\}$ of $\mathfrak{g}_\alpha$. Then by the Jacobi identity, we have

$$[[k, l], g_{\alpha,i}] + [[l, g_{\alpha,i}], k] + [[g_{\alpha,i}, k], l] = 0.$$

We focus just on the latter two terms. Since α is a root, $[h, g_{\alpha,i}] = \alpha(h)g_{\alpha,i}$ for any $h \in \mathfrak{h}$, and we have

$$\begin{aligned}
[[l, g_{\alpha,i}], k] + [[g_{\alpha,i}, k], l] &= [\alpha(l)g_{\alpha,i}, k] - [\alpha(k)g_{\alpha,i}, l] \\
&= \alpha(l)[g_{\alpha,i}, k] - \alpha(k)[g_{\alpha,i}, l] \\
&= -\alpha(l)\alpha(k)g_{\alpha,i} + \alpha(k)\alpha(l)g_{\alpha,i} = 0.
\end{aligned}$$

Thus $[[k, l], g_{\alpha,i}] = 0$. Since this is true for any choice of α, $[[k, l], g] = 0$ for any $g \in \mathfrak{g}$. Therefore $[k, l] \in Z(\mathfrak{g})$. □

Corollary 2.2. A toral subalgebra $\mathfrak{h}$ of a semisimple Lie algebra $\mathfrak{g}$ is abelian.

At this point, we assume the following theorem.

Theorem 2.4 (Cartan semisimplicity criterion)**.** Let $\mathbb{F}$ be an algebraically closed field of characteristic zero. Let $\mathfrak{g}$ be a Lie algebra over $\mathbb{F}$.

Then $\mathfrak{g}$ is semisimple if and only if there exists a bilinear form $\kappa : \mathfrak{g} \times \mathfrak{g} \to \mathbb{F}$ such that

1. κ is *symmetric*: $\kappa(x, y) = \kappa(y, x)$ for all $x, y \in \mathfrak{g}$;
2. κ is *nondegenerate*: If $\kappa(x, y) = 0$ for all $y \in \mathfrak{g}$, then $x = 0$;
3. $\kappa([xy], z) = \kappa(x, [yz])$ for all $x, y, z \in \mathfrak{g}$.

If such a form κ (called the *Cartan-Killing form* or sometimes just the *Killing form*) exists, $\mathfrak{g}$ has a nontrivial toral subalgebra. If a toral subalgebra $\mathfrak{h}$ of $\mathfrak{g}$ is *maximal*, that is, not properly contained in any other toral subalgebra, we have $\mathfrak{h} = \mathfrak{g}_0$, so that

$$\mathfrak{g} = \mathfrak{h} \oplus \bigoplus_{a \in \Phi} \mathfrak{g}_\alpha.$$

Such a maximal toral subalgebra is called a *Cartan subalgebra*, or just *CSA*.

Proof. See [Hum72]. □

We now present some consequences. The next theorem lists some properties enjoyed by semisimple Lie algebras which will be needed shortly.

Theorem 2.5. Let $\mathbb{F}$ be an algebraically closed field of characteristic zero, $\mathfrak{g}$ a semisimple Lie algebra over $\mathbb{F}$, and $\mathfrak{h}$ a CSA of $\mathfrak{g}$. Then the following are true.

1. The restriction of κ to $\mathfrak{h} \times \mathfrak{h}$ is nondegenerate.
2. For each $\alpha \in \Phi$, there is a unique nonzero $t_a \in \mathfrak{h}$ such that $\alpha(h) = \kappa(h, t_\alpha)$ for all $h \in \mathfrak{h}$.
3. The dual space $\mathfrak{h}^*$ is ($\mathbb{F}$-)spanned by Φ.
4. $\alpha \in \Phi$ if and only if $-\alpha \in \Phi$.
5. For each $\alpha \in \Phi$ and each $x \in \mathfrak{g}_\alpha$, $y \in \mathfrak{g}_{-\alpha}$, we have $[xy] = \kappa(x, y)t_\alpha$. If $x \neq 0$, there exists y such that $\kappa(x, y) \neq 0$, and vice versa.

Proof. See [Hum72] §8. $\square$

Finally, we state that the roots indeed form a root system.

Theorem 2.6. Let $\mathfrak{g}$ and $\mathfrak{h}$ be as above. For roots α, β, let $\langle\alpha, \beta\rangle := \kappa(t_\alpha, t_\beta)$. Then the following are true.

1. $\langle\alpha, \alpha\rangle > 0$.
2. The only scalar multiples of α belonging to Φ are α and $-\alpha$.
3. $\dfrac{2\langle\beta, \alpha\rangle}{\langle\alpha, \alpha\rangle} \in \mathbb{Z}$.
4. $\beta - \dfrac{2\langle\beta, \alpha\rangle}{\langle\alpha, \alpha\rangle}\alpha \in \Phi$.
5. $\langle\alpha, \beta\rangle \in \mathbb{Q}$.

Proof. See [Hum72] §8. $\square$

We have the following important corollary which describes in detail the structure of semisimple Lie algebras.

Corollary 2.3. Let $\alpha \in \Phi$. Then $\mathfrak{g}_\alpha \oplus \mathfrak{g}_{-\alpha} \oplus [\mathfrak{g}_\alpha, \mathfrak{g}_{-\alpha}] \approx A_1$.

Proof. Let $x \in \mathfrak{g}_\alpha$. By consequence 5 of Theorem 2.5, we can choose $y \in \mathfrak{g}_{-\alpha}$ such that $\kappa(x, y) = 2/\langle\alpha, \alpha\rangle$.

Set $h = [xy] = \kappa(x, y)t_\alpha = \dfrac{2}{\langle\alpha,\alpha\rangle}t_\alpha$. Observe that

$$\begin{aligned}[hx] = \left[\frac{2}{\langle\alpha,\alpha\rangle}t_\alpha, x\right] = \frac{2}{\langle\alpha,\alpha\rangle}\cdot[t_\alpha, x] = \frac{2}{\langle\alpha,\alpha\rangle}\cdot\alpha(t_\alpha)x &= \frac{2}{\langle\alpha,\alpha\rangle}\cdot\kappa(t_\alpha, t_\alpha)x \\ &= \frac{2}{\langle\alpha,\alpha\rangle}\cdot\langle\alpha,\alpha\rangle x = 2x.\end{aligned}$$

Similarly, $[hy] = -2y$. Therefore $\operatorname{span}_{\mathbb{F}}\{x, y, h\} \approx A_1$.

We claim now that x spans $\mathfrak{g}_\alpha$. Let x' be another element of $\mathfrak{g}_\alpha$. By the Jacobi identity, we have

$$\begin{aligned}[h,[x,x']] &= -[x,[x',h]] - [x',[h,x]] \\ &= -[x, -\alpha(h)x'] - [x', \alpha(h)x] \\ &= \alpha(h)[x,x'] - \alpha(h)[x',x] \\ &= \alpha(h)[x,x'] + \alpha(h)[x,x'] \\ &= 2\alpha(h)[x,x'] = (2\alpha)(h)[x,x'].\end{aligned}$$

However, by consequence (2) of the preceding theorem, 2α is not a root, so $[x, x'] = 0$. Consider now $[x', y] = \kappa(x', y)t_\alpha$. By the Jacobi identity we have

$$\begin{aligned}0 = [0,y] = [[x,x'],y] &= -[[x',y],x] - [[y,x],x'] \\ &= -[\kappa(x',y)t_\alpha, x] - [-h, x'] \\ &= -\kappa(x',y)[t_\alpha, x] - [-h, x'] \\ &= -\kappa(x',y)\kappa(t_\alpha,t_\alpha)x + 2x',\end{aligned}$$

and $-\kappa(x', y)\kappa(t_\alpha, t_\alpha) \neq 0$. Thus x' is a scalar multiple of x, so $\operatorname{span}_{\mathbb{F}} x = \mathfrak{g}_\alpha$. Similarly, we have $\operatorname{span}_{\mathbb{F}} y = \mathfrak{g}_{-\alpha}$. Finally, by consequence 5 of Theorem 2.5, $\operatorname{span}_{\mathbb{F}} h = [\mathfrak{g}_\alpha, \mathfrak{g}_{-\alpha}]$.

Putting these together, we have that x, y, and h span $\mathfrak{g}_\alpha, \mathfrak{g}_{-\alpha}$, and $[\mathfrak{g}_\alpha, \mathfrak{g}_{-\alpha}]$ respectively. So we have $\mathfrak{g}_\alpha \oplus \mathfrak{g}_{-\alpha} \oplus [\mathfrak{g}_\alpha, \mathfrak{g}_{-\alpha}] \approx A_1$. □

Theorem 2.7. Φ is a root system in the sense of Definition 2.2.13.

Proof. Let $E_{\mathbb{Q}} := \operatorname{span}_{\mathbb{Q}} \Phi$. By consequences 1 and 5 of Theorem 2.6, $\langle \cdot, \cdot \rangle : E_{\mathbb{Q}} \times E_{\mathbb{Q}} \to \mathbb{Q}$ is a symmetric, positive-definite bilinear form. Let E be the extension of $E_{\mathbb{Q}}$ to a vector space over $\mathbb{R}$. The inner product $\langle \cdot, \cdot \rangle$ then naturally extends to an inner product on $\mathbb{R}$.

Furthermore, we have $E = \operatorname{span}_{\mathbb{R}} \Phi$. Also:

1. Φ spans the Euclidean space $E = \mathbb{R}^{|\Phi|}$, by construction;
2. $0 \notin \Phi$, by definition;
3. the only scalar multiples of $\alpha \in \Phi$ belonging to Φ are $\pm\alpha$, by consequence 2 of Theorem 2.6;
4. for $\alpha, \beta \in \Phi$, $r_\alpha(\beta) := \beta - \frac{2\langle \beta, \alpha \rangle}{\langle \alpha, \alpha \rangle}\alpha \in \Phi$, by consequence 4 of Theorem 2.6;
5. for $\alpha, \beta \in \Phi$, $\frac{2\langle \beta, \alpha \rangle}{\langle \alpha, \alpha \rangle} \in \mathbb{Z}$, by consequence 3 of Theorem 2.6. □

So, from a semisimple Lie algebra (and a choice of Cartan subalgebra,) we have obtained a root system. It turns out that the root system so obtained depends only on the choice of Lie algebra. See [Hum72].

The next sections explore further the correspondence between semisimple Lie algebras and root systems.

Cartan matrices

Definition 2.2.17. Let Φ be a rank-n root system with base $\Delta = \{\phi_k\}_{k=1}^n$. The *Cartan matrix* of Φ is the $n \times n$ matrix whose ijth entry is given by $\langle \phi_i, \phi_j \rangle$.

Since every root system corresponds to a semisimple Lie algebra, we can also speak of the *Cartan matrix of a Lie algebra.* In the next theorem, we will see that the Cartan matrix encodes crucial information that connects a given root system to its associated semisimple Lie algebra.

Example 2.2. From before, we know that the root system Φ associated to A_2 has rank 2. If we take the roots α, β in the given diagram to be unit vectors, their components are $\alpha = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \beta = \begin{bmatrix} -1/2 \\ \sqrt{3}/2 \end{bmatrix}$. We compute $\langle \alpha, \alpha \rangle = \langle \beta, \beta \rangle = 2$, $\langle \alpha, \beta \rangle = \langle \beta, \alpha \rangle = -1$. So the Cartan matrix of A_2 is $\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$.

Some general representation theory

Definition 2.2.18. A *representation* of a Lie algebra is a homomorphism $\phi : \mathfrak{g} \to gl(V)$; that is, a linear map that preserves the bracket.

Given a representation $\phi : \mathfrak{g} \to gl(V)$, we make the representing space V into a $\mathfrak{g}$-module by defining $x.v = \phi(x)(v)$ for every $x \in \mathfrak{g}$, $v \in V$. Observe that for all $v, w \in V$, scalars $c \in \mathbb{F}$, and $x, y \in \mathfrak{g}$,

1. $x.(cv + w) = cx.v + x.w$ and
2. $[xy].v = x.y.v - y.x.v$.

Conversely, if an action of $\mathfrak{g}$ on V satisfies these conditions, the mapping $\phi : \mathfrak{g} \to gl(V)$ defined by $\phi(x)(v) := v$ is a representation of $\mathfrak{g}$.

When the target space of a representation of $\mathfrak{g}$ is a general linear matrix algebra, we may think of the homomorphism ϕ as encoding some of the structure of $\mathfrak{g}$ into a matrix algebra.

Definition 2.2.19. The *adjoint* of an element $x \in \mathfrak{g}$ is the function $\mathrm{ad}_x : \mathfrak{g} \to \mathfrak{g}$ that sends $y \mapsto [xy]$.

Definition 2.2.20. The *adjoint representation* of a Lie algebra $\mathfrak{g}$ is the function $\mathrm{ad} : \mathfrak{g} \to gl(\mathfrak{g})$ that sends $x \mapsto \mathrm{ad}_x$.

Proposition 2.18. For any Lie algebra $\mathfrak{g}$, ad is actually a representation.

Proof. We have to show that $[\mathrm{ad}\, x, \mathrm{ad}\, y] = \mathrm{ad}[xy] = \mathrm{ad}_{[xy]}$. For any $z \in \mathfrak{g}$,

$$\mathrm{ad}_{[xy]}\, z = [[xy]z] = -[z[xy]],$$

which by the Jacobi identity is equal to

$$\begin{aligned}
&= [x[yz]] + [y[zx]] \\
&= [x[yz] - [y[xz]] \\
&= [x, \operatorname{ad}_y z] - [y, \operatorname{ad}_x z] \\
&= \operatorname{ad}_x(\operatorname{ad}_y z) - \operatorname{ad}_y(\operatorname{ad}_x z) \\
&= (\operatorname{ad}_x \circ \operatorname{ad}_y - \operatorname{ad}_y \circ \operatorname{ad}_x)(z) \\
&= [\operatorname{ad} x, \operatorname{ad} y](z).
\end{aligned}$$

□

Representation theory gives us another way to think of, and a much simpler way to compute, the roots of a semisimple Lie algebra. For the remainder of this section, assume $\mathfrak{g}$ is semisimple over an algebraically closed field F of characteristic zero.[3]

Proposition 2.19. Let $\mathfrak{h}$ be a maximal toral subalgebra of $\mathfrak{g}$. The roots of $\mathfrak{g}$ with respect to $\mathfrak{h}$ are precisely the nonzero functionals $\phi : \mathfrak{h} \to F$ that annihilate $\det(\operatorname{ad} h - \phi(h)I)$, where $I \in gl(\mathfrak{g})$ denotes the identity transformation.

Proof. Suppose ϕ is a root. By definition, ϕ is not identically zero, and there is a nonzero $x \in \mathfrak{g}$ such that $[hx] - \phi(h)x = 0$. Since $[hx] - \phi(h)x = (\operatorname{ad} h - \phi(h)I)x$, and $x \neq 0$, we must have $\det(\operatorname{ad} h - \phi(h)I) = 0$.

Suppose $\phi \neq 0$ and $\det(\operatorname{ad} h - \phi(h)I) = 0$. Then, since the matrix $\operatorname{ad} h - \phi(h)I$ is singular, there is a nonzero $x \in \mathfrak{g}$ such that $(\operatorname{ad} h - \phi(h)I)x = 0$; which is to say, $[hx] - \phi(h)x = 0$. Hence, ϕ is a root. □

In the language of §2.2.9 below, this proposition says that roots are the "weights" of the adjoint representation, and thus the root spaces are the "weight spaces."

Example 2.3. We shall compute the roots of A_1 again (with respect to its diagonal subalgebra) using the above proposition.

[3]Taking $F = \mathbb{R}$ will suffice.

For any element of $\operatorname{diag} A_1$, say $h = \operatorname{diag}(d, -d)$, $\operatorname{ad} h$ is the linear map that sends

$$x = \begin{bmatrix} a & b \\ c & -a \end{bmatrix} \mapsto [hx] = \begin{bmatrix} 0 & -2db \\ 2dc & 0 \end{bmatrix}.$$

So $\operatorname{ad} h = \operatorname{diag}(0, -2d, d)$, and the nonzero functionals that annihilate the characteristic polynomial of $\operatorname{ad} h$ are exactly $\phi_1(h) = -2d$, $\phi_2(h) = 2d$.

Example 2.4. Using the same method, we shall compute the roots of A_2 with respect to its diagonal subalgebra.

An element of $\operatorname{diag} A_2$ looks like $h = \operatorname{diag}(h_1, h_2, -(h_1 + h_2))$. So $\operatorname{ad} h : \mathbb{R}^8 \to \mathbb{R}^8$ is the linear map that sends

$$x = \begin{bmatrix} a & b & c \\ d & e & f \\ g & i & -(a+e) \end{bmatrix} \mapsto [hx] = \begin{bmatrix} 0 & (h_1 - h_2)b & (2h_1 + h_2)c \\ (h_2 - h_1)d & 0 & (h_1 + 2h_2)f \\ (-2h_1 - h_2)g & (-h_1 - 2h_2)i & 0 \end{bmatrix}.$$

We can put this in matrix form:

$$\operatorname{ad} h = \operatorname{diag} \begin{bmatrix} 0 \\ h_1 - h_2 \\ 2h_1 + h_2 \\ h_2 - h_1 \\ 0 \\ h_1 + 2h_2 \\ -2h_1 - h_2 \\ -h_1 - 2h_2 \end{bmatrix}.$$

So the roots of A_2 are the linear functionals $\phi_i : \mathbb{R}^2 \to \mathbb{R}$ for which $\phi_i(h_1, h_2)$ annihilates this matrix's characteristic polynomial. In vector form, they are as follows:

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix}, \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} -2 \\ -1 \end{bmatrix}, \begin{bmatrix} -1 \\ -2 \end{bmatrix}.$$

Plotting these gives Figure 2.3, which is not a root system taken as-is. (Why?) There exists an invertible linear transformation, which we leave to the reader to work out, that maps this picture onto the root system A_2 given previously. In fact this is true more generally, as we will see.

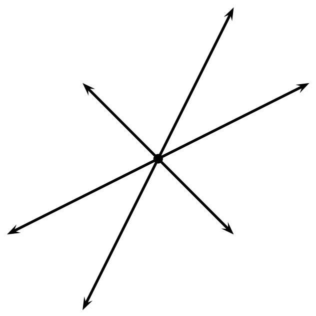


Figure 2.3: The not-quite-a-root-system given by the roots of A_2 with respect to $\operatorname{diag} A_2$.

Serre relations and Chevalley generators

The *Serre relations* provide a way to reconstruct a Lie algebra from its root system. In a sense, they encode the construction of a root system from a Lie algebra given previously into a presentation of that Lie algebra by generators and relations.

Theorem 2.8 (Serre)**.** Let Φ be a rank-n root system with base $\Delta = \{\phi_i\}_{i=1}^n$. Then the Lie algebra with generators $\{x_i, y_i, h_i\}_{i=1}^n$ subject to the relations

1. $[h_i, h_j] = 0$;
2. $[x_i, y_j] = \mathbf{1}_{i=j} h_i$;
3. $[h_i, x_j] = \langle \phi_j, \phi_i \rangle x_j$;
4. $[h_i, y_j] = -\langle \phi_j, \phi_i \rangle y_j$;
5. $(\operatorname{ad} x_i)^{1-\langle \phi_j, \phi_i \rangle}(x_j) = (\operatorname{ad} y_i)^{1-\langle \phi_j, \phi_i \rangle}(y_j) = 0$

is semisimple and has root system Φ.

Proof. See [Hum72] §18. □

Due to a theorem of Chevalley, every semisimple Lie algebra can be generated in this manner. Thus, the elements of the generating set $\{x_i, y_i, h_i\}_{i=1}^n$ are usually called the *Chevalley generators*.

*Combinatorial structure of semisimple Lie algebra representations**

This section may be skipped on a first reading. It bridges the preceding material with papers such as [Don03].

Let $\mathfrak{g}$ be semisimple over an algebraically closed field $\mathbb{F}$ of characteristic zero. Fix a Cartan subalgebra $\mathfrak{h}$ of $\mathfrak{g}$, and let Φ be the corresponding root system of rank $n = \dim \mathfrak{h}$. Let $\Delta = \{\alpha_i\}_{i=1}^n$ be a base for Φ, and let $\{x_i, y_i, h_i\}_{i=1}^n$ be the corresponding set of Chevalley generators of $\mathfrak{g}$.

In the n-dimensional Euclidean space of which Φ is a basis, let $\varepsilon = \{\varepsilon_i\}_{i=1}^n$ be the dual basis to Φ. Then if M is the associated Cartan matrix, we have $\varepsilon = M\Phi$.

Definition 2.2.21. Let Λ be the $\mathbb{Z}$-span of ε; that is, the set of all integral linear combinations of ε_i. Elements of Λ are called *weights*, and Λ itself is called the *weight lattice.* A weight $\lambda \in \Lambda$ is called *dominant* if it is a *positive* integral linear combination of ε_i.

Definition 2.2.22. Make a finite-dimensional vector space V over $\mathbb{F}$ into a $\mathfrak{g}$-module as stated in Definition 2.2.18. A subspace $W \leqslant V$ is called $\mathfrak{g}$-*stable* (sometimes $\mathfrak{g}$-*invariant* if $gW \subseteq W$ for any $g \in \mathfrak{g}$. If $v \in V$ is nonzero, $\mathfrak{g}.v$ denotes the smallest $\mathfrak{g}$-stable subspace containing v.

With the above setup, the following two theorems specify the structure of V.

Theorem 2.9. For a weight $\lambda \in \Lambda$, let the λ-*weight space* of V be given by

$$V_\lambda := \{v \in V : h_i.v = \langle \lambda, \alpha_i^\vee \rangle v\}.$$

Then $V = \bigoplus_{\lambda\in\Lambda} V_\lambda$. Furthermore, for any $v \in V_\lambda$, we have $x_i.v \in V_{\lambda+\alpha_i}$ and $y_i.v \in V_{\lambda-\alpha_i}$.

Proof. See [Hum72] §21. □

Let $\Pi(V) := \{\lambda \in \Lambda : \dim V_\lambda > 0\}$. We call the equality $V = \bigoplus_{\lambda\in\Pi(V)} V_\lambda$ the *weight space decomposition* of V. Any basis of V whose vectors lie in $\bigcup_{\lambda\in\Pi(V)}$ is called a *weight basis.*

Theorem 2.10. There are independent vectors $\{v_i\}_{i=1}^n$ with $v_i \in V_{\lambda_i}$ such that

1. v_i is "maximal," in the sense $x_i.v_j = 0$ for each $i, j \in [n]$ and
2. $V = \bigoplus_{i=1}^n \mathfrak{g}.v_i$.

If $V = \mathfrak{g}.v_i$, we say V is *irreducible.* In this case, if v_i' is any other maximal vector, $v_i' = cv_i$ for some scalar c, and if $W = \mathfrak{g}.w_i$ is another irreducible $\mathfrak{g}$-module, then $V \approx W$ are isomorphic $\mathfrak{g}$-modules.

Let $\mathfrak{g}(\lambda)$ be an indeterminate, irreducible $\mathfrak{g}$-module with maximal vector of dominant weight λ. The preceding theorem assures that we can write $V \approx \bigoplus_{i=1}^n \mathfrak{g}(\lambda_i)$. This decomposition completely determines the module, up to isomorphism.

Proof. See [Hum72] §21. □

We combinatorialize this information as follows.

Definition 2.2.23. Let $\{v_r\}_{r\in R}$ be a weight basis for a $\mathfrak{g}$-module V, indexed by the set R, as above. Let $p, q \in R$, let $i \in [n]$, and write

$$x_i.v_p = \sum_{r\in R} c_{r,p} v_r, \qquad y_i.v_q = \sum_{r\in R} d_{r,q} v_r.$$

The *supporting graph* for V is the edge-colored directed graph with vertex set R such that for any $p, q \in R$, $p \xrightarrow{i} s$ provided that in the above decomposition, $c_{q,p}$ and $d_{p,q}$ are not both zero. If we also associate the edge coefficients $c_{q,p}$ and $d_{p,q}$ with each edge, then we have a *representation diagram* for V.

For each $r \in R$, we set $\omega(r) = \lambda$ if $v_r \in V_\lambda$.

Theorem 2.11. Consider R as the Hasse diagram of a ranked poset. Then

1. If $r \xrightarrow{i} s$ in R, then $\omega(s) - \omega(r) = \alpha_i$.

2. $\langle \omega(r), \alpha_i^\vee \rangle = 2\rho_i(r) - l_i(r)$, where $\rho_i(r)$, $l_i(r)$ are respectively the rank of r within its i-component and the length of that i-component.
3. Define the weight-generating function of R as

$$WGF(R;\mathbf{z}) := \sum_{r\in R} \mathbf{z}^{\omega(r)} = \sum_{r\in R} \prod_{i=1}^{n} z_i^{\langle \omega(r), \alpha_i^\vee \rangle}.$$

   Then the $\mathfrak{g}$-module V is completely determined, up to isomorphism, by the weight-generating function of its supporting graph R.

Proof. See [Don03] and [Don18]. $\square$

CHAPTER 3

LATTICE MODELS OF LIE ALGEBRA REPRESENTATIONS

We now present the centerpiece of the work. Throughout, write $[n] = \mathbb{N} \cap [1, n]$.

3.1 The DCS relations

Definition 3.1.1. Let L be a diamond-colored distributive lattice with edges colored by $[n]$, and let $x \in L$. For each $i \in \mathbb{N}$, let $\top_i(x) \in L$ be the maximum element of the i-component in which x lies.

Define a weight function[1] $\omega(x) = [\omega_i(x)]_{i=1}^n$ by

$$\omega_i(x) = \rho_i(x) - (\rho_i(\top_i(x)) - \rho_i(x)) = 2\rho_i(x) - \rho_i(\top_i(x)).$$

Definition 3.1.2. Let Φ be a rank-n root system with base Δ, and let $\mathfrak{g}$ be the corresponding semisimple Lie algebra with Chevalley generators $\{x_i, y_i, h_i\}_{i=1}^n$. Let M be the Cartan matrix of $\mathfrak{g}$. Let L be a diamond-colored distributive lattice with edge colors corresponding to Δ.

Then L is said to be $\mathfrak{g}$-*structured* if for any edge $x \xrightarrow{i} y$ in L, $\omega(y) - \omega(x)$ gives the ith row of M.

Theorem 3.1 (Donnelly)**.** Let L be a $\mathfrak{g}$-structured DCDL. If for any edge $x \to y$ there exist $c_{yx}, d_{xy} \in \mathbb{C}$ such that

1. if $w, x, y, z \in L$ form a diamond with $w \to x \to z$, $w \to y \to z$, we have the *diamond relations* $c_{yw}d_{wx} = d_{xz}c_{zy}, c_{xw}d_{wy} = d_{yz}c_{zx}$,

[1]The quantity $(\rho_i(\top_i(x)) - \rho_i(x))$ is the distance of x "from the top" of its i-component, whereas $\rho_i(x)$ is the distance of x "from the bottom." Thus we sometimes call the former quantity the *depth* of x, whereupon the color-i weight of x is "rank minus depth."

2. and for any $x \in L$ and any $i \in [n]$, we have the *crossing relation*

$$\sum_{\substack{w \in L \\ w \xrightarrow{i} x}} c_{xw} d_{wx} - \sum_{\substack{z \in L \\ x \xrightarrow{i} z}} c_{zx} d_{xz} = \omega_i(x),$$

then $\mathfrak{g}$ is homomorphic to the Lie algebra $\mathfrak{l}$ generated by the $3n$ $|L|$-dimensional matrices $\{E_i, F_i, H_i\}_{i=1}^n$ (whose rows and columns are indexed by the elements of L) defined as follows:

1. $[E_i]_{yx} := c_{yx}$ if $x \xrightarrow{i} y$, or 0 otherwise;
2. $[F_i]_{xy} := d_{xy}$ if $x \xrightarrow{i} y$, or 0 otherwise;
3. $H_i := [E_i F_i]$.

We say that

1. the coefficients $\{c_{yx}, d_{xy}\}_{x,y \in L}$ are *DCS-satisfactory* for $\mathfrak{g}$, and that
2. *L realizes a representation of $\mathfrak{g}$.*

Additionally, if $\mathfrak{g}$ is simple, the representation so realized is faithful.

Proof. See Appendix B. □

Definition 3.1.3. If a DCDL L realizes a representation of $\mathfrak{g}$, and the associated DCS-satisfactory coefficients $\{c_{yx}, d_{xy}\}$ are uniquely determined, we say that the lattice L is *solitary.*

If instead the *edge products* $\pi_{xy} := c_{yx} d_{xy}$ are uniquely determined, then L is said to be *product solitary.*

If we work with edge products rather than edge coefficients, the DCS relations become the following. (The structure relations, which deal with vertex weights, remain unchanged.)

1. If $w, x, y, z \in L$ form a diamond with $w \to x \to z$, $w \to y \to z$, we have the *diamond relation* $\pi_{xz}\pi_{yz} = \pi_{wx}\pi_{wy}$.

2. For any $x \in L$ and any $i \in [n]$, we have the *crossing relation*

$$\sum_{\substack{w \in L \\ w \xrightarrow{i} x}} \pi_{wx} - \sum_{\substack{z \in L \\ x \xrightarrow{i} z}} \pi_{xz} = \omega_i(x).$$

Proposition 3.1. Suppose a lattice L is chain-factorizable, and suppose there exist a set of positive rational DCS-satisfactory edge products. If the edge products on one face of L are uniquely determined, then L is product solitary.

Proof. Let $L_{k,j}$ be a face of L, and suppose the edge coefficients on $L_{k,j}$ are uniquely determined.

1. Let $x \in L_{k,j}$. Since j is extremal, there is exactly one edge incident to x that does not belong to the face $L_{k,j}$. Call it $x \to x'$. Its edge product is uniquely determined by a crossing relation at x.
2. Suppose without loss of generality that j is minimal. Consider an edge $x' \to y'$ in $L_{k,j+1}$. Since L is chain-factorizable, $x' \to y'$ is part of a diamond $x \to x' \to y'$, $x \to y \to y'$. The edges $x \to x'$, $y \to y'$ are determined uniquely by (1), and the edge $x \to y$ is determined uniquely by assumption. Thus the edge $x' \to y'$ is determined uniquely by the diamond relation.

If $j + 1$ is nonmaximal, this argument can be repeated to show that the edge products in $L_{k,j+2}$ and the edge products of the edges connecting $L_{k,j+1}$ and $L_{k,j+2}$ are uniquely determined, and similarly for successive slices. Since L is chain-factorizable, this process determines every edge product in L. □

3.2 Example: chains

Definition 3.2.1. The *n-edge chain* is the lattice with vertex set $\{v_0, v_1, \ldots, v_n\}$ and a total ordering. Every edge is assigned the same color, so that the Hasse diagram of the lattice is

$$v_0 \xrightarrow{1} v_1 \xrightarrow{1} \ldots \xrightarrow{1} v_n.$$

Proposition 3.2. The n-edge chain realizes a representation of the Lie algebra A_1, whose Cartan matrix is diag(2).

Proof. That the n-edge chain is A_1-structured is immediate from $\omega_1(v_i) = 2i - n$. The reader may verify that choosing $c_{v_i,v_{i-1}} = n - i + 1$ and $d_{v_{i-1},v_i} = i$ gives a DCS-satisfactory set of coefficients. □

Example 3.1. Let's verify the DCS theorem for a 2-edge chain: $v_0 \xrightarrow{1} v_1 \xrightarrow{1} v_2$. We have the following information.

$\mathbf{i}$	$\omega_\mathbf{1}(\mathbf{v_i})$	$\mathbf{x} \xrightarrow{\mathbf{i}} \mathbf{y}$	$\mathbf{c_{yx}}$	$\mathbf{d_{xy}}$
0	-2	$v_0 \xrightarrow{1} v_1$	2	1
1	0	$v_1 \xrightarrow{1} v_2$	1	2
2	2			

Table 3.1: Vertex and edge data for a 2-edge chain.

We can then fill out the matrices

$$E = \begin{array}{c} \begin{array}{ccc} v_0 & v_1 & v_2 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \end{array}, \qquad F = \begin{array}{c} \begin{array}{ccc} v_0 & v_1 & v_2 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \end{array},$$

$$H = [EF] = \begin{array}{c} \begin{array}{ccc} v_0 & v_1 & v_2 \end{array} \\ \begin{bmatrix} -2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix} \end{array} \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \end{array}.$$

To show that a representation of A_1 is actually realized, we verify the Serre relations $[HE] = 2E$, $[HF] = -2F$. This step is left to the reader.

3.3 Example: Boolean lattices

Definition 3.3.1. The *Boolean lattice* $\mathscr{B}_n$ is defined as follows.

1. Vertices of $\mathscr{B}_n$ are subsets $x \subseteq [n]$.
2. $\mathscr{B}_n$ is ordered by containment; if $x, y \in \mathscr{B}_n$, $x \leqslant y$ if $x \subseteq y$.

3. If $x, y \in \mathscr{B}_n$, $\rho(x) = |x|$, and consequently $x \to y$ if and only if $|y\backslash x| = 1$.
4. Edges in $\mathscr{B}_n$ are colored by $[n]$; the edge $x \to y$ has color given by the element of the singleton set $y\backslash x$.
5. $\mathscr{B}_n$ embeds into $\mathbb{Z}^n$ as follows: if $x \in \mathscr{B}_n$, let $\phi(x)$ be the n-vector whose ith element is 1 if $i \in x$ and 0 otherwise.

Proposition 3.3. The join-irreducible elements of $\mathscr{B}_n$ are the singleton sets $\{i\}$ for each $i \in [n]$.

Proof. Left to reader. $\square$

Corollary 3.1. $j(\mathscr{B}_n)$ is a poset of n mutually incomparable elements.[2] The Hasse diagram of the lattice $\mathscr{B}_n$ is an n-hypercube.

Proposition 3.4. The Boolean lattice $\mathscr{B}_n$ realizes a representation of the Lie algebra $A_1^n = \bigoplus_{i=1}^n A_1$, whose Cartan matrix is $\operatorname{diag}(2, 2, \ldots, 2)$.

Proof. First we show that $\mathscr{B}_n$ is A_n^1-structured. Take $i, j \in [n]$ with $i \neq j$; it suffices to show that $M_{ij} = 0$.

Observe that every i-component of $\mathscr{B}_n$ is a one-edge chain. Thus consider an edge $x \xrightarrow{i} y$. Evidently $\omega_i(x) = -1$ and $\omega_i(y) = 1$. We need to show that $M_{ij} = \omega_j(y) - \omega_j(x) = 0$. This is apparent since $j \in x$ if and only if $j \in y$.

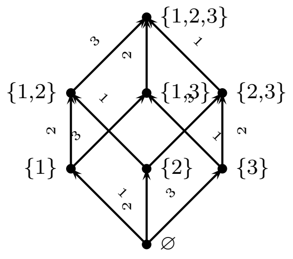
Since every i-component of $\mathscr{B}_n$ is a one-edge chain, taking $c_{yx} = d_{xy} = 1$ for every $x \to y$ gives a DCS-satisfactory set of coefficients. $\square$

3.4 Example: zero-padded subset lattices

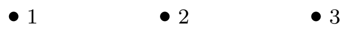
Definition 3.4.1. The *zero-padded subset lattice* $\mathscr{Z}_n$ (hereinafter *ZPS lattice*) is defined as follows.

1. Vertices of $\mathscr{Z}_n$ are subsets $x \subseteq [n]$.

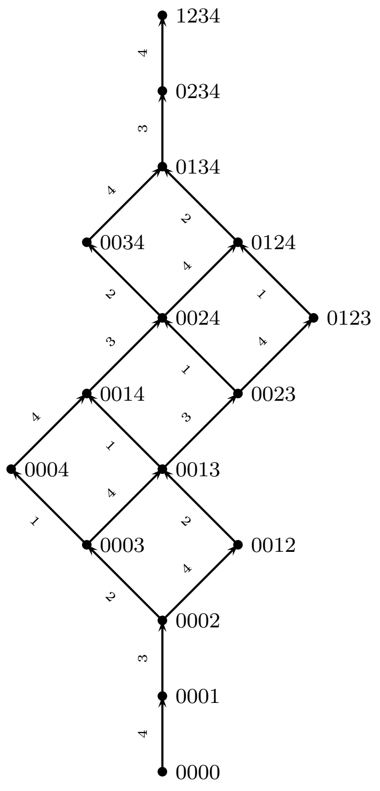
[2]Called an *antichain* of length n.



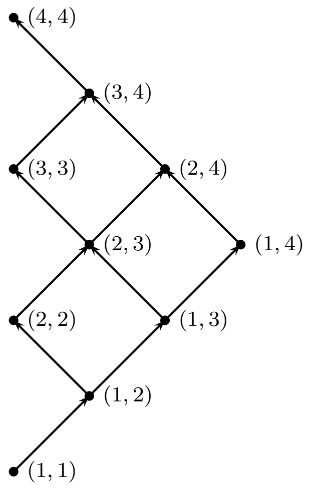
(a) The Boolean lattice $\mathscr{B}_3$.



(b) The antichain $j(\mathscr{B}_3)$.



(c) The zero-padded subset lattice $\mathscr{Z}_4$.



(d) The "angelfish" poset $j(\mathscr{Z}_4)$.

Figure 3.1: Two DCDLs and their posets of join-irreducibles.

2. If $x, y \in \mathscr{Z}_n$, enumerate the elements of (for instance) x by $x_1 \geqslant \cdots \geqslant x_n$, letting $x_j = 0$ if $j > |x|$. Then $x \leqslant y$ if $x_i \leqslant y_i$ for all $i \in [n]$.
3. If $x, y \in \mathscr{Z}_n$, $\rho(x) = \sum_{i=1}^n x_i$; we have $x \to y$ if and only if x, y differ in exactly one index $i \in [n]$, with $x_i + 1 = y_i$.
4. Edges in $\mathscr{Z}_n$ are colored by $[n]$. Keeping the above setup, the edge $x \to y$ has color $n - x_i$.
5. $\mathscr{Z}_n$ embeds directly into $\mathbb{Z}^n$ using the enumeration in (2) above.

The ZPS lattices were used in [Pro82] (in which $\mathscr{Z}_n$ is called $M(n)$) to prove a conjecture of Erdős concerning subset sums.

Proposition 3.5. The join-irreducible elements of $\mathscr{Z}_n$ are those whose nonzero elements are consecutive.

Proof. Follows from Corollary 2.1; left to reader. $\square$

Thus $\mathscr{Z}_n$ is the lattice of order ideals of the "angelfish" poset given by ordering the set $\{(i, j) : 1 \leqslant i \leqslant j \leqslant n\}$ componentwise.

Proposition 3.6. The ZPS lattice $\mathscr{Z}_n$ realizes a representation of the Lie algebra B_n, whose Cartan matrix is

$$M := \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & 0 & \\ & -1 & 2 & \ddots & & \\ & & \ddots & \ddots & -1 & \\ & 0 & & -1 & 2 & -2 \\ & & & & -1 & 2 \end{bmatrix}.$$

Proof. We first show that $\mathscr{Z}_n$ is B_n-structured.

1. As before, it is clear that if $x \xrightarrow{i} y$, then $\omega_i(y) - \omega_i(x) = 2 = M_{ii}$.
2. If $x \xrightarrow{n-1} y$, then x and y differ in exactly one coordinate, which increments from 1 (in x) to 2 (in y.) Consider the preceding coordinate as well; write $\square 1 \xrightarrow{n-1} \square 2$.

(If there is no preceding coordinate, then x is maximal and there is no edge $x \overset{n-1}{\to} y$.)

It is apparent that the square must be a 0. Thus, there must be a pattern of edges $00 \overset{n}{\to} 01 \overset{n-1}{\to} 02 \overset{n}{\to} 12$. Since every component of $\mathscr{Z}_n$ is a length-one chain, this implies that $\omega_n(x) = 1$, $\omega_n(y) = -1$, so $\omega_n(y) - \omega_n(x) = -2 = M_{n-1,n}$.

3. If $x \overset{n-2}{\to} y$, then we have $\square 2 \overset{n-2}{\to} \square 3$. (Note that there must be a preceding coordinate in this case.) The square is either 0 or 1.

   a) If the square is a 0, there is a pattern of edges $01 \overset{n-1}{\to} 02 \overset{n-2}{\to} 03$. Observe that there can be no color-$(n-1)$ edge incident to 03. So in this case $\omega_{n-1}(y) - \omega_{n-1}(x) = 0 - 1 = M_{n-2,n-1}$.

   b) If the square is a 1, there is a pattern of edges $12 \overset{n-2}{\to} 13 \overset{n-3}{\to} 23$. Observe that there can be no color-$(n-1)$ edge incident to 12. So in this case $\omega_{n-1}(y) - \omega_{n-1}(x) = -1 - 0 = M_{n-2,n-1}$.

This argument goes through for any entry $M_{n-k-1,n-k}$, $k = 1, \ldots, n-2$. A dual argument, considering succeeding coordinates, goes through for $M_{n-k+1,n-k}$, $k = 1, \ldots, n-1$. We leave this to the reader.

Since every k-component of $\mathscr{Z}_n$ is a length-one chain, taking $c_{yx} = d_{xy} = 1$ for every edge $x \to y$ satisfies the DCS relations. $\square$

3.5 Example: Fibonaccian lattices

Definition 3.5.1. The *Fibonaccian lattice* $L_A^{Fib}(n,k) = \mathscr{F}(n,k)$ is defined as follows.

1. Vertices of $\mathscr{F}(n,k)$ are k-tuples $x \in \mathbb{Z}^k$ such that

   a) $x_i \in \{n(i-1)+1, \ldots, ni\}$ for each $i \in [k]$;

   b) if $x_i = ni$, then $x_{i+1} \neq ni + 1$.[3]

2. $\mathscr{F}(n,k)$ is ordered componentwise: $x \leqslant y$ if $x_i \leqslant y_i$ for each $i \in [k]$.

[3]That is, x contains no consecutive integers.

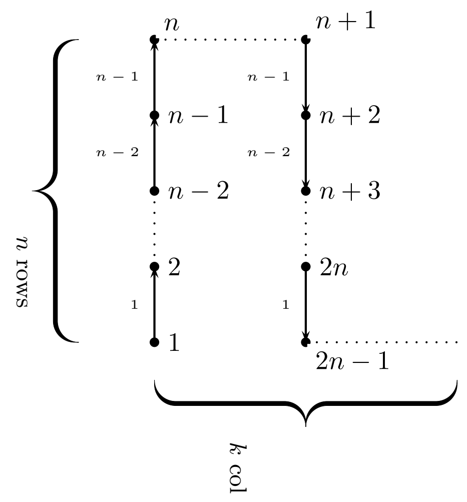


Figure 3.2: The snake rule for $\mathscr{F}(n,k)$.

3. If $x, y \in \mathscr{F}(n,k)$, $x \to y$ if and only if x and y differ in exactly one component $i \in [k]$, with $x_i + 1 = y_i$.
4. Edges in $\mathscr{F}(n,k)$ are colored by $[n-1]$ depending on the value of x_i, using the *snake rule* diagrammed in Figure 3.2.
5. $\mathscr{F}(n,k)$ embeds directly into $\mathbb{Z}^k$.

We now reproduce some results in [DDMN20].

Proposition 3.7. The vertex cardinality of $\mathscr{F}(n,k)$ is $(r_1^{n+1} - r_2^{n+1})/(r_1 - r_2)$, where $r_{1,2}$ are respectively the positive and negative roots of $x^2 - nx - 1$.

Proof. The elements of $\mathscr{F}(n,k+1)$ come about by appending an extra "space" at the end of an element of $\mathscr{F}(n,k)$. Clearly there are n possible choices for this space. However, appending $nk+1$ to those elements of $F(n,k)$ whose last element is nk, of which there are $|\mathscr{F}(n,k-1)|$, violates the consecutive-integer rule. Clearly every other choice is valid. Thus

$$|\mathscr{F}(n,k+1)| = n|\mathscr{F}(n,k)| - |\mathscr{F}(n,k-1)|.$$

The explicit formula then follows from a standard inductive argument[4], which we

[4]It can also be derived using generating functions.

leave to the reader. □

Corollary 3.2. $|\mathscr{F}(n,3)|$ is the Fibonacci number F_{2n+2}.

Theorem 3.2 (Donnelly-Dunkum)**.** For $n > 1$, $\mathscr{F}(n,k)$ realizes a representation of the Lie algebra A_{n-1}, whose Cartan matrix is

$$M := \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & \text{\Large 0} & \\ & -1 & 2 & \ddots & & \\ & & \ddots & \ddots & -1 & \\ & \text{\Large 0} & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}.$$

Proof. See [DDMN20]. □

Product solitarity of $\mathscr{F}(n,3)$

In this subsection, we consider the lattices $\mathscr{F}(n,3)$, depicted in Figure 3.3.

Proposition 3.8. $\mathscr{F}(3,3)$ is product solitary.

Proof. Let $\pi_{147,157} = \alpha$. Then we can deduce the contents of Table 3.2.

Having done this, the diamond relation $\pi_{148,149}\pi_{148,248} = \pi_{248,249}\pi_{149,249}$ becomes

$$\frac{8}{6-\alpha}\cdot(2-\alpha) = \frac{6-\alpha}{2+\alpha}\cdot 1$$
$$\iff 8(2-\alpha)(2+\alpha) = (6-\alpha)^2$$
$$\iff 8(4-\alpha^2) = \alpha^2 - 12\alpha + 36$$
$$\iff 0 = 9\alpha^2 - 12\alpha + 4 = (3\alpha-2)^2.$$

Thus $\alpha = 2/3$. The remaining coefficients are now uniquely determined as positive rational numbers, as the reader may verify. □

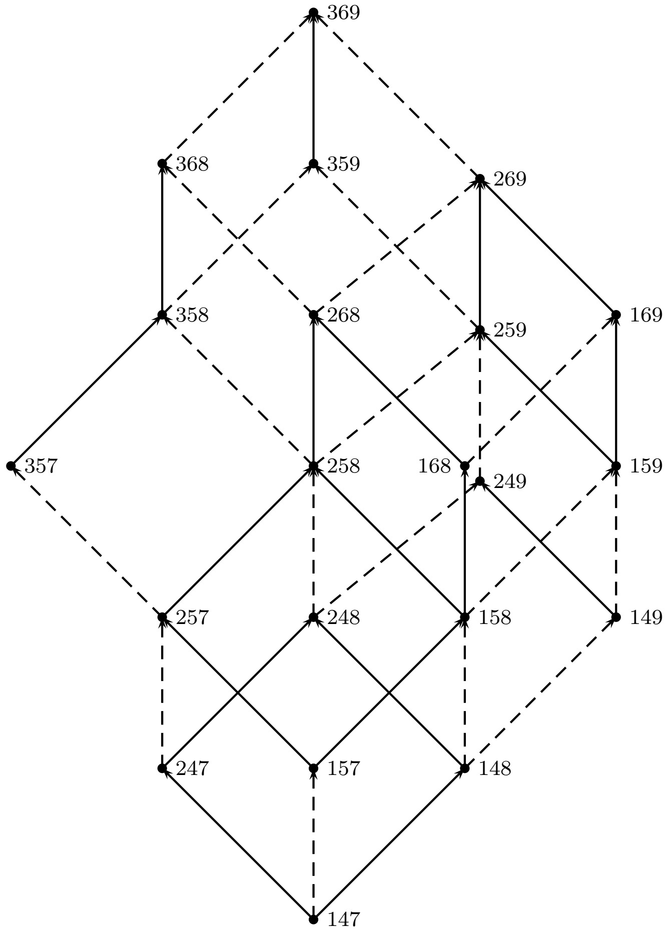
Theorem 3.3. $\mathscr{F}(4,3)$ and $\mathscr{F}(5,3)$ are product solitary.

$\mathbf{x} \to \mathbf{y}$	$\pi_{\mathbf{x},\mathbf{y}}$	**Justification**
$247 \xrightarrow{1} 248$	α	Color-1 crossing relation at 247
$147 \xrightarrow{1} 148$	$2-\alpha$	Color-1 crossing relation at 147
$148 \xrightarrow{1} 248$	$2-\alpha$	Color-1 crossing relation at 148
$147 \xrightarrow{2} 157$	1	Length-1 chain
$247 \xrightarrow{2} 257$	2	Length-2 chain
$157 \xrightarrow{1} 257$	$\alpha/2$	Diamond relation between $157, 257, 147, 247$
$157 \xrightarrow{1} 158$	$3-\alpha/2$	Color-1 crossing relation at 157
$257 \xrightarrow{1} 258$	$1+\alpha/2$	Color-1 crossing relation at 257
$158 \xrightarrow{1} 258$	$\dfrac{\alpha(6-\alpha)}{4+2\alpha}$	Diamond relation between $157, 257, 158, 258$
$148 \xrightarrow{2} 158$	$\dfrac{4-2\alpha}{6-\alpha}$	Diamond relation between $147, 148, 157, 158$
$148 \xrightarrow{2} 149$	$\dfrac{8}{6-\alpha}$	Color-2 crossing relation at 148
$158 \xrightarrow{1} 168$	$\dfrac{8}{2+\alpha}$	Color-1 crossing relation at 158
$258 \xrightarrow{1} 268$	$\dfrac{4\alpha}{2+\alpha}$	Color-1 crossing relation at 258
$168 \xrightarrow{1} 268$	$\dfrac{6-\alpha}{2+\alpha}$	Color-1 crossing relation at 268
$149 \xrightarrow{1} 249$	1	Length-1 chain
$248 \xrightarrow{2} 249$	$\dfrac{6-\alpha}{2+\alpha}$	Left to reader

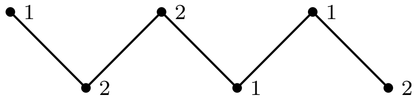
Table 3.2: Edge coefficient data for $\mathscr{F}(3,3)$.

Proof by computer. Figures 3.4 and 3.5 were generated by the author by using the software library SAGE to generate a Groebner basis for the nonlinear system of equations given by the diamond and crossing relations for each lattice. No other solutions exist. $\square$

Conjecture 3.1. $\mathscr{F}(n,3)$ is product solitary for any $n>1$.



(a) $\mathscr{F}(3,3)$. Solid edges are color 1. Dashed edges are color 2.



(b) $j(\mathscr{F}(3,3))$.

Figure 3.3: The Fibonaccian lattice $\mathscr{F}(3,3)$ and its poset of join-irreducibles.

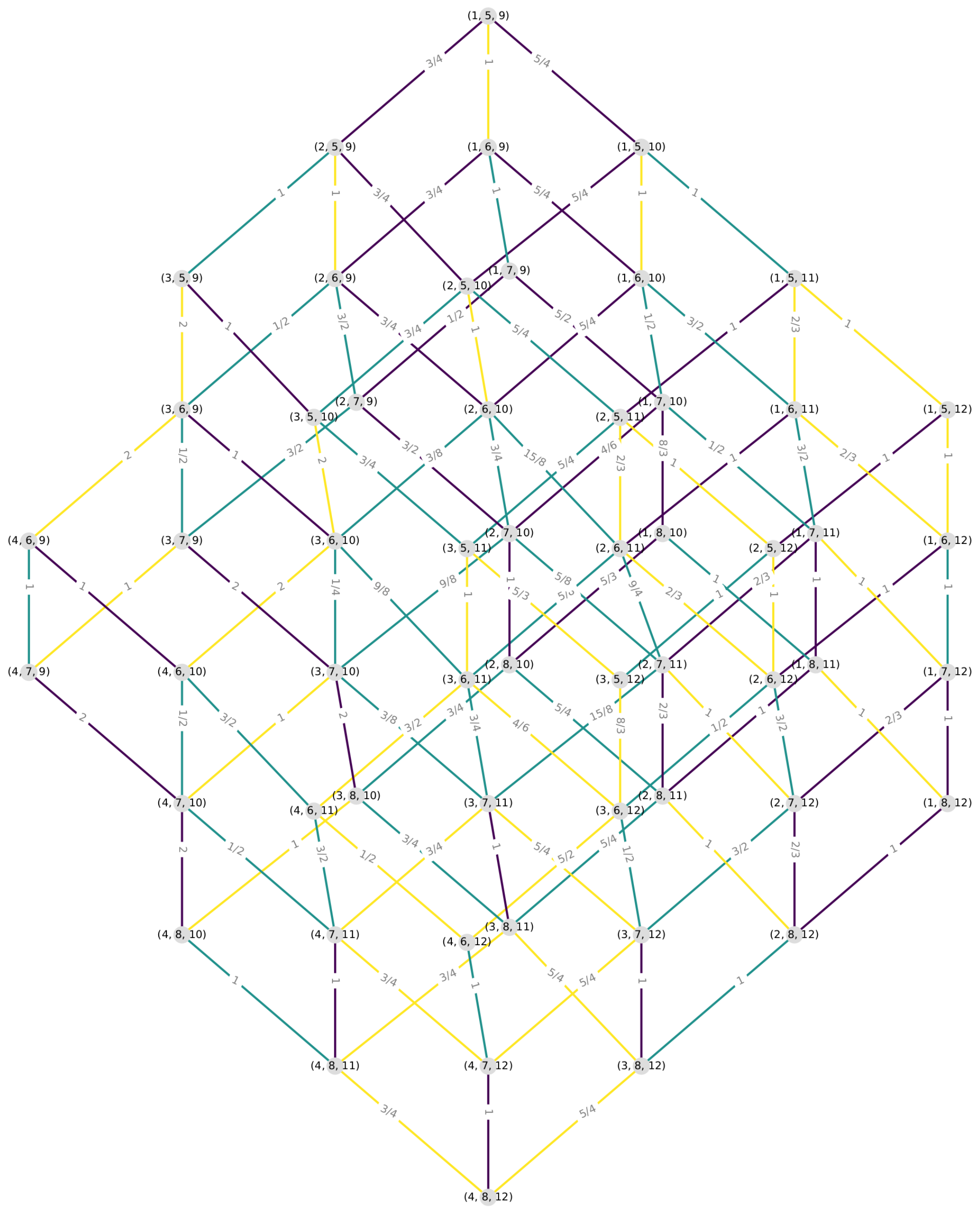


Figure 3.4: Edge coefficients for $\mathscr{F}(4,3)$.

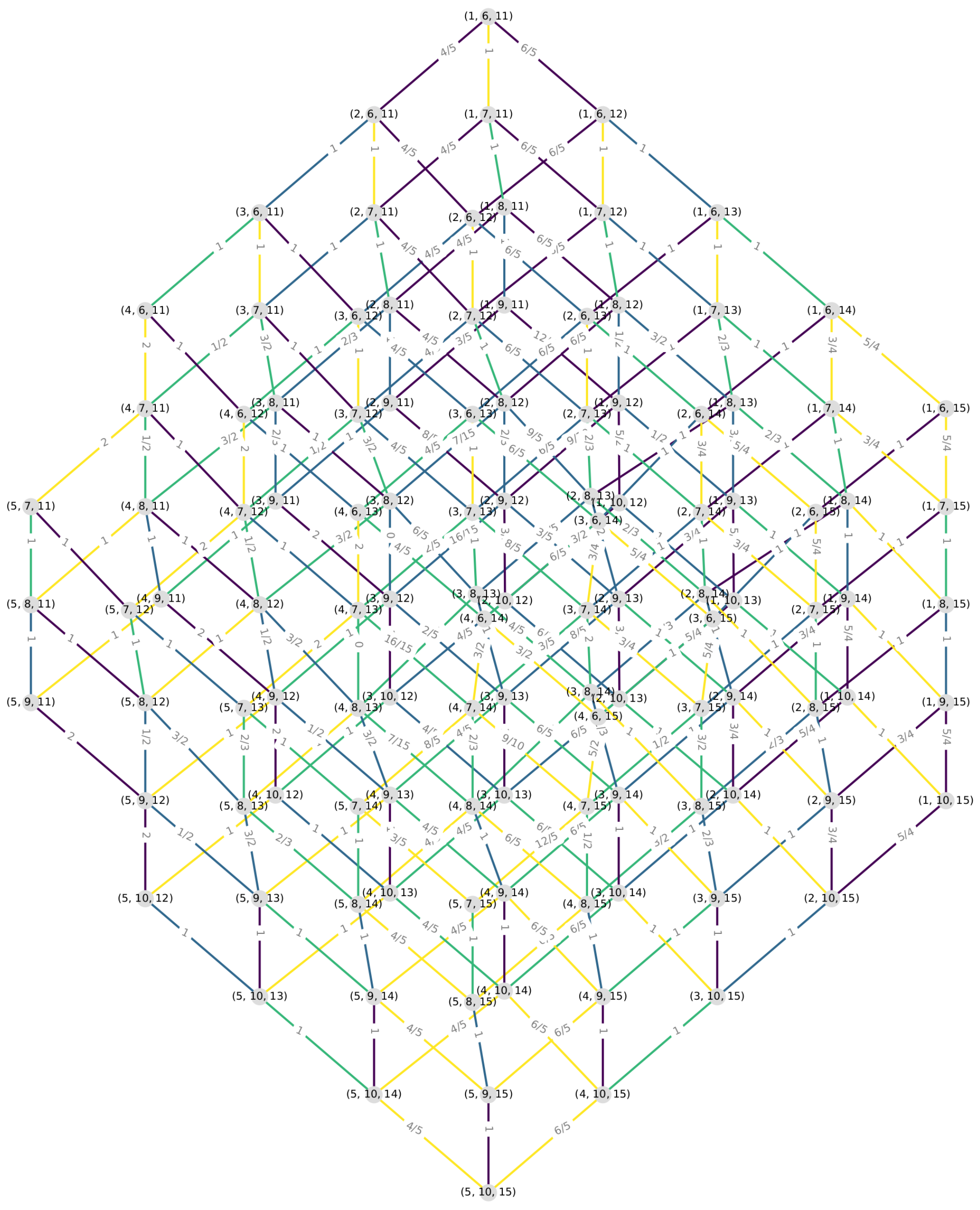


Figure 3.5: Edge coefficients for $\mathscr{F}(5,3)$.

CHAPTER 4

CATALANIAN LATTICES

This section introduces and proves the author's main results.

4.1 Gelfand-Tsetlin patterns

Definition 4.1.1. A *Gelfand-Tsetlin pattern* or just *GT pattern* of order n is an $n \times n$ lower triangular matrix with integer entries that satisfies $g_{i+1,j} \leqslant g_{ij} \leqslant g_{i+1,j+1}$ for any $1 \leqslant j \leqslant i \leqslant n$.[1]

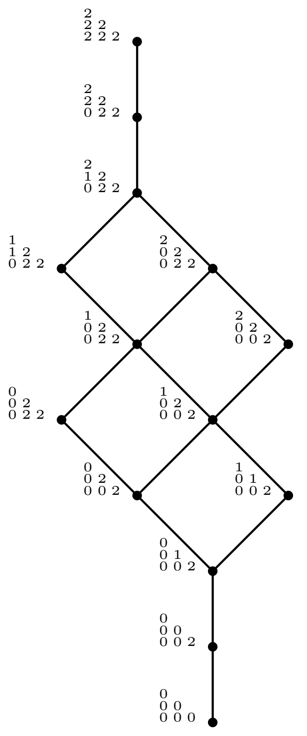
Visually, this says that entries in a GT pattern weakly decrease when moving northwest or down.

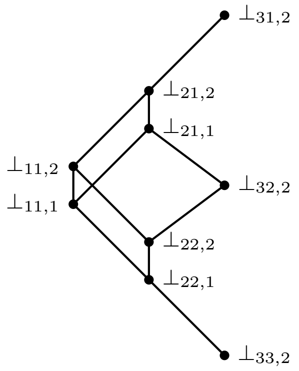
Definition 4.1.2. A GT pattern of order n is *symplectic* if g_{nj} is even for any j.

Definition 4.1.3. The *Catalanian lattice of order* n, k $L_C^{Cat}(n,k) = \mathscr{C}(n,k)$ is defined as follows.

1. Vertices of $\mathscr{C}(n,k)$ are symplectic GT patterns of order n and maximum entry $2k$.
2. $\mathscr{C}(n,k)$ is ordered componentwise.
3. If $x, y \in \mathscr{C}(n,k)$, $x \to y$ if and only if x and y differ in exactly one entry $(i,j) \in [n]^2$, with $x_{ij} + 1 = y_{ij}$ if $i < n$ or $x_{ij} + 2 = y_{ij}$ if $i = n$.
4. Edges of $\mathscr{C}(n,k)$ are colored by $[n]$ according to the value of i.
5. Upon concatenating rows (and dividing the final row by 2), $\mathscr{C}(n,k)$ embeds directly into $\mathbb{Z}^{\binom{n}{2}}$.

[1]This definition is equivalent to that given in [Sta01] §7.10. We represent GT patterns as matrices here for ease of computation and typesetting.


(a) $\mathscr{C}(3,1)$



(b) $j(\mathscr{C}(3,1))$

Figure 4.1: The Catalanian lattice $\mathscr{C}(3,1)$ and its poset of join-irreducibles.

4.2 The cardinality of $\mathscr{C}(n,k)$

Definition 4.2.1. The *Catalan sequence*[2] $\{c_n\}_{n=0}^{\infty}$ is given by $c_n = \dfrac{1}{n+1}\dbinom{2n}{n}$.

We shall have cause to represent the Catalan numbers in a slightly different form:

Proposition 4.1. $c_n = 4^n \displaystyle\prod_{k=1}^{n} \frac{k-1/2}{k+1}$.

Proof. Hereinafter, define the *double factorial* $n!! := \displaystyle\prod_{k=0}^{\lfloor (n-1)/2 \rfloor} (n-2k) = n\cdot(n-2)\cdot\ldots\cdot 1$. We have

$$c_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{n!(n+1)!}$$

[2]The Catalan sequence counts, among other things, the number of valid arrangements of n left and n right parentheses; that is, the number of permutations in which every left parenthesis has a matching right parenthesis.

$$\begin{aligned}
&= \frac{(2n)!!(2n-1)!!}{n!(n+1)!} \\
&= 2^n \frac{(2n-1)!!}{(n+1)!} \\
&= 4^n \prod_{k=1}^{n} \frac{k-1/2}{k+1}. \qquad \square
\end{aligned}$$

In [Don03], it was shown that $|\mathscr{C}(n,1)| = c_n$. This section generalizes that result.

Theorem 4.1 (Malone)**.** The vertex cardinality of the Catalanian lattice of order n, k is given by the $k \times k$ Hankel determinant

$$|\mathscr{C}(n,k)| = \det M = \begin{vmatrix} c_n & c_{n+1} & \cdots & c_{n+k-1} \\ c_{n+1} & c_{n+2} & \cdots & c_{n+k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n+k-1} & c_{n+k} & \cdots & c_{n+2k-2} \end{vmatrix}.$$

If we index this matrix by $i, j = 0, 1, \ldots, k-1$, then m_{ij}, i.e., the ijth entry of M, is c_{n+i+j}.

Before we prove this theorem, we reproduce, more legibly, a result from [GV89].

Lemma 4.1. $\det M = \prod_{j=0}^{k-1} \frac{(2n+2j)!(2j+1)!}{(n+j+1)!(k-1+n+j)!}$.

Proof. First, note that

$$\begin{aligned}
m_{ij} = c_{n+i+j} &= 4^{n+i+j} \prod_{l=1}^{n+i+j} \frac{l-1/2}{l+1} \\
&= 4^i 4^{n+j} \left(\prod_{l=1}^{n+j} \frac{l-1/2}{l+1} \right) \cdot \left(\prod_{l=n+j+1}^{n+j+i} \frac{l-1/2}{l+1} \right) \\
&= 4^i 4^{n+j} \left(\prod_{l=1}^{n+j} \frac{l-1/2}{l+1} \right) \cdot \left(\prod_{l=1}^{i} \frac{n+j+l-1/2}{n+j+l+1} \right) \\
&= 4^i c_{n+j} \prod_{l=1}^{i} \frac{n+j+l-1/2}{n+j+l+1}
\end{aligned}$$

Now consider the $n \times n$ matrix N whose ijth entry is

$$n_{ij} = \frac{c_{n+i+j}}{4^i c_{n+j}} = \prod_{l=1}^{i} \frac{n+j+l-1/2}{n+j+l+1}$$

$$= \frac{\Gamma(n+j+i-1/2)\Gamma(n+j+1)}{\Gamma(n+j-1/2)\Gamma(n+j+i+1)}$$

Applying Gauss' hypergeometric identity with $\gamma = n+j+1$, $\alpha = -i$, $\beta = 3/2$, we have

$$n_{ij} = \sum_{r=0}^{\infty} \prod_{l=0}^{r-1} \frac{(l-i)(l+3/2)}{(l+n+j+1)(l+1)}.$$

Now if $r-1-i \geqslant 0$, the product becomes zero. The maximum value of i is $k-1$, so the series terminates at or before the point when $r-1 = k-1$. Thus it suffices to cut the series off at $r = k-1$, and we have

$$\begin{aligned} n_{ij} &= \sum_{r=0}^{k-1} \prod_{l=0}^{r-1} \frac{(l-i)(l+3/2)}{(l+n+j+1)(l+1)} \\ &= \sum_{r=0}^{k-1} \left(\prod_{l=0}^{r-1} \frac{1}{l+n+j+1} \right) \left(\prod_{l=0}^{r-1} \frac{(l-i)(l+3/2)}{l+1} \right). \end{aligned}$$

Therefore, N can be written as a matrix product AB, with $n_{ij} = \sum_r a_{ir} b_{rj}$, where

$$\begin{aligned} a_{ij} &= \prod_{l=0}^{j-1} \frac{(l-i)(l+3/2)}{l+1}, \\ b_{ij} &= \prod_{l=0}^{j-1} \frac{1}{l+n+j+1}. \end{aligned}$$

In particular, we have $\det N = \det A \det B$. We tackle these determinants separately.

To compute $\det A$, note that $a_{ij} = 0$ if $j > i$; that is, A is lower triangular. Thus

$$\begin{aligned} \det A = \prod_{i=0}^{k-1} a_{ii} &= \prod_{i=0}^{k-1} \prod_{l=0}^{i-1} \frac{(l-i)(l+3/2)}{l+1} \\ &= \prod_{i=0}^{k-1} \left(\frac{(-1)^i i!}{i!} \prod_{l=0}^{i-1} l + 3/2 \right) \\ &= \prod_{i=0}^{k-1} (-1)^i \left(\prod_{l=0}^{i-1} l + 3/2 \right) \\ &= \prod_{i=0}^{k-1} (-1)^i 2^{-i} \prod_{i=0}^{k-1} 2l + 3 \\ &= \prod_{i=0}^{k-1} (-1)^i 2^{-i} \frac{(2i+1)!}{i! \cdot 2^i} \end{aligned}$$

$$= \prod_{i=0}^{k-1} (-1)^i 4^{-i} \frac{(2i+1)!}{i!}.$$

To compute $\det B$, note that the maximum value taken on by l is $k-2$, and write

$$b_{ij} = \frac{\prod_{l=j}^{k-2} l+n+i+1}{\prod_{l=0}^{k-2} l+n+i+1}.$$

If we factor out the denominator from each row j, then the matrix B looks like this:

$$B = \left(\prod_{j=0}^{k-1} \frac{1}{\prod_{l=0}^{k-2} l+n+j+1} \right) \begin{bmatrix} \prod_{l=0}^{k-2}(l+n+1) & \cdots & \prod_{l=0}^{k-2}(l+n+k-2+1) \\ \prod_{l=1}^{k-2}(l+n+1) & \cdots & \prod_{l=1}^{k-2}(l+n+k-2+1) \\ \vdots & \ddots & \vdots \\ \prod_{l=k-2}^{k-2}(l+n+1) & \cdots & \prod_{l=k-2}^{k-2}(l+n+k-2+1) \\ \prod_{l=k-1}^{k-2}(l+n+1) & \cdots & \prod_{l=k-1}^{k-2}(l+n+k-2+1) \end{bmatrix}.$$

Note that the bottom row is composed of empty products, i.e., 1s, and going up one row adds one additional term to the product. In addition, this product is unchanged across columns, except for the value of i. Therefore, subtracting an appropriate multiple of the bottom row from the penultimate row, we obtain $(k+n-1), (k+n), \ldots, (2k+n-3)$. Doing similarly with the antepenultimate row and the bottom two rows, we obtain $(k+n-1)^2, \ldots, (2k+n-3)^2$. This process can be repeated for every row without ever changing the determinant. So we conclude that B has the same determinant as

$$B' = \left(\prod_{j=0}^{k-1} \frac{1}{\prod_{l=0}^{k-2} l+n+j+1} \right) \begin{bmatrix} (k+n-1)^{k-1} & \cdots & (2k+n-3)^{k-1} \\ (k+n-1)^{k-2} & \cdots & (2k+n-3)^{k-2} \\ \vdots & \ddots & \vdots \\ (k+n-1) & \cdots & (2k+n-3) \\ 1 & \cdots & 1 \end{bmatrix}.$$

The Vandermonde matrix on the right, whose ijth entry is $(k+n-1+j)^{k-1-i}$, has determinant

$$\begin{aligned}
&\prod_{i=0}^{k-1}\prod_{j=i+1}^{k-1}[(k+n-1+i)-(k+n-1+j)]\\
=&\prod_{i=0}^{k-1}\prod_{j=i+1}^{k-1}(i-j)\\
=&\prod_{i=0}^{k-1}\prod_{j=1}^{k-1-i}(-j)\\
=&\prod_{i=0}^{k-1}(-1)^{k-1-i}(k-1-i)!\\
=&\prod_{i=0}^{k-1}(-1)^{i}i!.
\end{aligned}$$

So we conclude that

$$\begin{aligned}
\det B &= \left(\prod_{j=0}^{k-1}\frac{1}{\prod_{l=0}^{k-2} l+n+j+1}\right)\cdot\prod_{i=0}^{k-1}(-1)^i i!\\
&= \left(\prod_{j=0}^{k-1}\frac{(n+j)!}{(k-1+n+j)!}\right)\cdot\prod_{i=0}^{k-1}(-1)^i i!,
\end{aligned}$$

and thus that

$$\begin{aligned}
\det N &= \det A\det B\\
&= \left(\prod_{i=0}^{k-1}(-1)^i 4^{-i}\frac{(2i+1)!}{i!}\right)\cdot\left(\prod_{j=0}^{k-1}\frac{(n+j)!}{(k-1+n+j)!}\right)\cdot\prod_{i=0}^{k-1}(-1)^i i!\\
&= \left(\prod_{i=0}^{k-1}(-1)^i 4^{-i}(2i+1)!\right)\cdot\left(\prod_{j=0}^{k-1}\frac{(n+j)!}{(k-1+n+j)!}\right).
\end{aligned}$$

Now to obtain M from N, we must multiply each row by 4^i and each column by $c_{n+j}=\dfrac{(2n+2j)!}{(n+j+1)!(n+j)!}$. So we finally obtain

$$\begin{aligned}
\det M &= \left(\prod_{i=0}^{k-1}4^i\right)\cdot\left(\prod_{j=0}^{k-1}\frac{(2n+2j)!}{(n+j+1)!(n+j)!}\right)\cdot\det N\\
&= \left(\prod_{i=0}^{k-1}4^i\right)\cdot\left(\prod_{j=0}^{k-1}\frac{(2n+2j)!}{(n+j+1)!(n+j)!}\right)\cdot\left(\prod_{i=0}^{k-1}4^{-i}(2i+1)!\right)
\end{aligned}$$

$$\cdot \left(\prod_{j=0}^{k-1} \frac{(n+j)!}{(k-1+n+j)!}\right)$$

$$= \left(\prod_{j=0}^{k-1} \frac{(2n+2j)!}{(n+j+1)!(k-1+n+j)!}\right) \cdot \left(\prod_{i=0}^{k-1} (2i+1)!\right)$$

$$= \prod_{j=0}^{k-1} \frac{(2j+1)!(2n+2j)!}{(n+j+1)!(k-1+n+j)!}. \qquad \square$$

Lemma 4.2. $|L_C^{Cat}(n,k)| = \binom{n+k}{k} \prod_{i=1}^{n} \frac{i!(2k+2i-1)!}{(2i-1)!(2k+i)!}$.

Proof. [Don18], §8.10, §B.23 gives

$$|L_C^{Cat}(n,k)| = \binom{n+k}{k} \prod_{i=1}^{n} \prod_{j=0}^{n-i-1} \frac{2k+2n+1-2i-j}{2n+1-2i-j}$$

$$= \binom{n+k}{k} \prod_{i=1}^{n} \frac{(n-i+1)!(2k+2n-2i+1)!}{(2n-2i+1)!(2k+n-i+1)!}.$$

Making the substitution $i' = n-i+1$, we obtain

$$|L_C^{Cat}(n,k)| = \binom{n+k}{k} \prod_{i=1}^{n} \frac{i!(2k+2i-1)!}{(2i-1)!(2k+i)!}. \qquad \square$$

Therefore, the proof of the foregoing theorem is a matter of showing that this expression agrees with the one previously obtained for $\det M$.

Lemma 4.3. $\prod_{k=1}^{N} \frac{(2k-1)!}{(N+k)!} = \prod_{k=1}^{N} \frac{k!}{(2k)!}$.

Proof. Cross-multiply. Then both sides are equal to $\prod_{k=1}^{2N} k!$. $\square$

Theorem 4.2. For all positive integers n, k,

$$\prod_{i=1}^{k} \frac{(2n+2i)!(2i-1)!}{(n+i)!(n+i+k)!} = \binom{n+k}{n} \prod_{i=1}^{n} \frac{i!(2k+2i-1)!}{(2i-1)!(2k+i)!}.$$

Proof. We proceed by induction on the difference $k-n$ in either direction. There will be two separate inductive cases, each depending on the basis (B) as follows:

n/k	1	2	3	4	5
1	B	←	←	←	←
2	→	B	←	←	←
3	→	→	B	←	←
4	→	→	→	B	←
5	→	→	→	→	B

Basis: $k - n = 0$. So $k = n$ and we must show

$$\prod_{i=1}^{n} \frac{(2n+2i)!(2i-1)!}{(n+i)!(2n+i)!} = \binom{2n}{n} \prod_{i=1}^{n} \frac{i!(2n+2i-1)!}{(2i-1)!(2n+i)!}.$$

Applying Lemma 4.3 to the left-hand side, with $N = n$, $k = i$, we obtain

$$\begin{aligned}\prod_{i=1}^{n} \frac{(2n+2i)!(2i-1)!}{(n+i)!(2n+i)!} &= \prod_{i=1}^{n} \frac{i!(2n+2i)!}{(2i)!(2n+i)!} \\ &= \prod_{i=1}^{n} \left(\frac{i!(2n+2i-1)!}{(2i-1)!(2n+i)!} \cdot \frac{2n+2i}{2i} \right) \\ &= \prod_{i=1}^{n} \frac{i!(2n+2i-1)!}{(2i-1)!(2n+i)!} \cdot \prod_{i=1}^{n} \frac{n+i}{i} \\ &= \binom{2n}{n} \prod_{i=1}^{n} \frac{i!(2n+2i-1)!}{(2i-1)!(2n+i)!},\end{aligned}$$

which was to be shown.

Inductive case 1: $k - n > 0$. Assume inductively that the theorem holds for $k = n + r$, $r = 0, 1, \ldots$. Explicitly, assume

$$\prod_{i=1}^{n+r} \frac{(2n+2i)!(2i-1)!}{(n+i)!(2n+i+r)!} = \binom{2n+r}{n} \prod_{i=1}^{n} \frac{i!(2n+2r+2i-1)!}{(2i-1)!(2n+2r+i)!}.$$

We must show that the theorem holds for $k = n + r + 1$; explicitly, that

$$\prod_{i=1}^{n+r+1} \frac{(2n+2i)!(2i-1)!}{(n+i)!(2n+i+r+1)!} = \binom{2n+r+1}{n} \prod_{i=1}^{n} \frac{i!(2n+2r+2i+1)!}{(2i-1)!(2n+2r+i+2)!}.$$

By the inductive hypothesis, it suffices to show that the quotient of the $k = n+r+1$ case by the $k = n + r$ case on both sides is equal.

On the left, we have

$$\begin{aligned} &\frac{(4n+2r+2)!(2n+2r+1)!}{(2n+r+1)!} \cdot \frac{\prod_{i=1}^{n+r+1}(1/2n+i+r+1)!)}{\prod_{i=1}^{n+r}(2n+i+r+1)!} \\ = \; &\frac{(4n+2r+2)!(2n+2r+1)!}{(2n+r+1)!(3n+2r+2)!} \cdot \frac{\prod_{i=1}^{n+r}(2n+i+r)!}{\prod_{i=1}^{n+r}(2n+i+r+1)!} \\ = \; &\frac{(4n+2r+2)!(2n+2r+1)!}{(2n+r+1)!(3n+2r+2)!} \cdot \prod_{i=1}^{n+r} \frac{1}{2n+i+r+1} \\ = \; &\frac{(4n+2r+2)!(2n+2r+1)!}{(2n+r+1)!(3n+2r+2)!} \cdot \frac{(2n+r+1)!}{(3n+2r+1)!}\end{aligned}$$

$$= \frac{(4n+2r+2)!(2n+2r+1)!}{(3n+2r+1)!(3n+2r+2)!}.$$

On the right, we have

$$\frac{\binom{2n+r+1}{n}}{\binom{2n+r}{n}} \cdot \prod_{i=1}^{n}\prod_{i=1}^{n} \frac{(2n+2r+2i+1)!(2n+2r+i)!}{(2n+2r+2i-1)!(2n+2r+i+1)!}$$
$$= \frac{2n+r+1}{n+r+1} \cdot \prod_{i=1}^{n} \frac{(2n+2r+2i+1)(2n+2r+2i)}{(2n+2r+i+2)(2n+2r+i+1)}$$

Turning the terms depending on $2i$ into double factorials and the other terms into single factorials, we have

$$\begin{aligned}
&= \frac{2n+r+1}{n+r+1} \cdot \frac{(4n+2r+1)!!(4n+2r)!!}{(2n+2r+1)!!(2n+2r)!!} \cdot \frac{(2n+2r+2)!(2n+2r+1)!}{(3n+2r+2)!(3n+2r+1)!} \\
&= \frac{2n+r+1}{n+r+1} \cdot \frac{(4n+2r+1)!}{(2n+2r+1)!} \cdot \frac{(2n+2r+2)!(2n+2r+1)!}{(3n+2r+2)!(3n+2r+1)!} \\
&= \frac{2n+r+1}{n+r+1} \cdot \frac{(4n+2r+1)!(2n+2r+2)!}{(3n+2r+2)!(3n+2r+1)!} \\
&= \frac{4n+2r+2}{2n+2r+2} \cdot \frac{(4n+2r+1)!(2n+2r+2)!}{(3n+2r+2)!(3n+2r+1)!} \\
&= \frac{(4n+2r+2)!(2n+2r+1)!}{(3n+2r+2)!(3n+2r+1)!}.
\end{aligned}$$

This completes the first inductive case.

Inductive case 2: $k - n < 0$. This proceeds analogously to the previous inductive case, except the induction runs backwards.

Assume inductively that the theorem holds for $k = n - r$, $r = 0, 1, \ldots, n-2$. Explicitly, assume

$$\prod_{i=1}^{n-r} \frac{(2n+2i)!(2i-1)!}{(n+i)!(2n+i-r)!} = \binom{2n-r}{n} \prod_{i=1}^{n} \frac{i!(2n-2r+2i-1)!}{(2i-1)!(2n-2r+i)!}$$

We must show that the theorem holds for $k = n - r - 1$; explicitly,

$$\prod_{i=1}^{n-r-1} \frac{(2n+2i)!(2i-1)!}{(n+i)!(2n+i-r-1)!} = \binom{2n-r-1}{n} \prod_{i=1}^{n} \frac{i!(2n-2r+2i-3)!}{(2i-1)!(2n-2r+i-2)!}$$

As before, we show that the quotient of the $n - r$ case by the $n - r - 1$ case is the same on both sides.

On the left, we have:

$$\begin{aligned}
&\frac{(4n-2r)!(2n-2r-1)!}{(2n-r)!}\cdot\frac{\prod_{i=1}^{n-r}(1/(2n+i-r)!)}{\prod_{i=1}^{n-r-1}(1/(2n+i-r-1)!)}\\
=\;&\frac{(4n-2r)!(2n-2r-1)!}{(2n-r)!}\cdot\frac{\prod_{i=1}^{n-r-1}(2n+i-r-1)!}{\prod_{i=1}^{n-r}(2n+i-r)!}\\
=\;&\frac{(4n-2r)!(2n-2r-1)!}{(2n-r)!(3n-2r)!}\cdot\frac{\prod_{i=1}^{n-r-1}(2n+i-r-1)!}{\prod_{i=1}^{n-r-1}(2n+i-r)!}\\
=\;&\frac{(4n-2r)!(2n-2r-1)!}{(2n-r)!(3n-2r)!}\cdot\prod_{i=1}^{n-r-1}\frac{1}{2n+i-r}\\
=\;&\frac{(4n-2r)!(2n-2r-1)!}{(2n-r)!(3n-2r)!}\cdot\frac{(2n-r)!}{(3n-2r-1)!}\\
=\;&\frac{(4n-2r)!(2n-2r-1)!}{(3n-2r)!(3n-2r-1)!}.
\end{aligned}$$

And on the right, we have

$$\begin{aligned}
&\frac{\binom{2n-r}{n}}{\binom{2n-r-1}{n}}\cdot\prod_{i=1}^{n}\frac{(2n-2r+2i-1)!(2n-2r+i-2)!}{(2n-2r+2i-3)!(2n-2r+i)!}\\
=\;&\frac{2n-r}{n-r}\cdot\prod_{i=1}^{n}\frac{(2n-2r+2i-1)(2n-2r+2i-2)}{(2n-2r+i)(2n-2r+i-1)}\\
=\;&\frac{2n-r}{n-r}\cdot\frac{(4n-2r-1)!!(4n-2r-2)!!}{(2n-2r-1)!!(2n-2r-2)!!}\cdot\frac{(2n-2r)!(2n-2r-1)!}{(3n-2r)!(3n-2r-1)!}\\
=\;&\frac{2n-r}{n-r}\cdot\frac{(4n-2r-1)!}{(2n-2r-1)!}\cdot\frac{(2n-2r)!(2n-2r-1)!}{(3n-2r)!(3n-2r-1)!}\\
=\;&\frac{2n-r}{n-r}\cdot\frac{(4n-2r-1)!(2n-2r)!}{(3n-2r)!(3n-2r-1)!}\\
=\;&\frac{4n-2r}{2n-2r}\cdot\frac{(4n-2r-1)!(2n-2r)!}{(3n-2r)!(3n-2r-1)!}\\
=\;&\frac{(4n-2r)!(2n-2r-1)!}{(3n-2r)!(3n-2r-1)!}
\end{aligned}$$

This completes the proof. □

4.3 Product solitarity of $\mathscr{C}(n,k)$

In this section, we show the following result.

Theorem 4.3 (Dunkum-Donnelly-Malone)**.** If there exists a set of positive, rational, and DCS-satisfactory coefficients for the lattice $\mathscr{C}(n,k)$, then $\mathscr{C}(n,k)$ is product solitary.

Key to this result is a lemma appearing in [GT50].

Lemma 4.4 (Gelfand-Tsetlin 1950)**.** A color-k component of $\mathscr{C}(n,k)$, $k = 1,\dots,n-1$, realizes a representation of the Lie algebra A_k. Furthermore, the edge coefficients are uniquely determined as positive rationals.

Proof of Theorem 4.3. By Lemma 4.4, any component of color $1,\dots,n-1$ is product solitary.

We induct over color-n components. First we must assign an integer $s(C)$ to each color-n component. Observe that all vertices of a given color-n component differ only in the nth row. Therefore, if $p < n$, then $c_{p,q}$ is identical for every $c \in C$. The quantity we induct over will be the sum of these identical entries $s(C) := \sum_{p=1}^{n-1}\sum_{q=1}^{p} c_{p,q}$.

Basis, $s(C) = 0$. In this case, the nth row must be all zeros, with the exception of the bottom right entry, which must be one of $0, 2, \dots, 2k$. So C is a chain of length k, and its edge coefficients are uniquely determined positive rational numbers, as shown previously.

Induction. Assume inductively that for some nonnegative integer S, if C is a color-n component and $s(C) \leqslant S$, then C is product solitary.

Suppose we have a component C with $s(C) = S+1$. Let $\perp = \min C$. Let $r \in [n]$ be the index of the first column for which $\perp_{n,r} \neq x_{n,r}$ for some $x \in C$. Then $F = C_{(n,r),\perp_{n,r}}$ is a face of C; we claim its edge products are uniquely determined positive rationals.

List the identical elements of C by columns, left-to-right and bottom-to-top; i.e.,

$$c_{n-1,1}, c_{n-2,1}, \dots, c_{1,1}; \quad c_{n-1,2}, \dots, c_{2,2}; \quad \dots \quad c_{n-1,n-1}.$$

Since $s(C) > 0$, there is a nonzero entry in this list. Let $c_{p,q}$ be the *first* nonzero entry. For any $x \in F$, x is (p,q)-decrementable, since otherwise $c_{p,q}$ would equal one of $c_{p+1,q}$

or $c_{p-1,q-1}$, both of which precede it in the list. Therefore if $x \xrightarrow{n} y$ for $x, y \in F$, there must be edges $x' \xrightarrow{p} x$, $y' \xrightarrow{p} y$, and $x' \xrightarrow{n} y'$. Now we have the following.

1. $\pi_{x',y'}$ is uniquely determined as a positive rational by the inductive assumption, since $x' \xrightarrow{n} y'$ belongs to an n-component C' with $s(C') = S$.
2. $\pi_{x,x'}$ and $\pi_{y,y'}$ are uniquely determined as positive rationals by Lemma 4.4.
3. $\pi_{x,y}$ is uniquely determined as a positive rational by the diamond relation at $x' \to x \to y$, $x' \to y' \to y$.

Now, in view of Proposition 3.1, C is product solitary. This completes the induction.

□

We close this chapter with the natural counterpart to a uniqueness theorem.

Conjecture 4.1. There exists a set of positive rational DCS-satisfactory coefficients for $\mathscr{C}(n, k)$.

In theory, the inductive framework used in Theorem 4.3 should hold. However, it is unknown whether nonpositive numbers arise while carrying out the process described in the proof of Proposition 3.1. In other words, the following question is open.

Question. Let C be a chain-factorizable monochromatic lattice. Let F be a face of C, and suppose a set Π_F of positive rational edge products for F is known *a priori.* Under what conditions does Π_F propagate to a set of positive rational edge products for C?

CHAPTER 5

PATHS AND TABLEAUX

5.1 Motzkin paths

Definition 5.1.1. A *topside peakless Motzkin path* of length n is a sequence $\{p_i\}_{i=1}^n$ such that for each $1 \leqslant i \leqslant n$,

1. $p_i \in \{-1, 0, 1\}$;
2. $\sum_{k=1}^i p_k \geqslant 0$;
3. if $p_{i-1} = 1$, $p_i \neq -1$,

and for which $\sum_{i=1}^n p_i = 0$.

By calling $\{p_i\}_{i=1}^n$ a "path," we mean to identify it with the series of points $\{(i, \sum_{k=1}^i p_k)\}_{i=1}^n$. The above definition then says that the path goes from $(0, 0)$ to $(n, 0)$; that it remains on or above the x-axis (*topside*); and that no step $\nearrow$ is immediately followed by a step $\searrow$ (*peakless*).

There are four topside peakless Motzkin paths of length 4, depicted in Figure 5.1.

Proposition 5.1. Denote the number of topside peakless Motzkin paths of length n by P_n. Then we have the recurrence

$$P_n = \begin{cases} 1, & n = 0 \\ P_{n-1} + \sum_{i=3}^n P_{i-2}P_{n-i}, & n = 1, 2, \dots \end{cases}.$$

Proof. It is obvious that $P_0 = 1$. Consider an arbitrary path of length $n > 0$. Consider the first index i at which $\sum_{k=1}^i p_k = 0$, and let $P_n(i)$ be the number of length-n paths for which this occurs given some choice of i. Clearly i is well-defined for any path, so $P_n = \sum_{i=1}^n P_n(i)$.

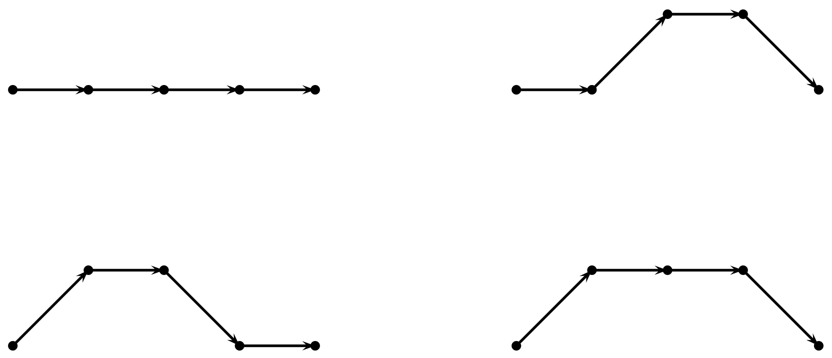


Figure 5.1: The four topside peakless Motzkin paths of length 4.

1. If $i = 1$, then $p_1 = 0$, and $\{p_k\}_{k=2}^n$ is a valid path of length $n-1$, so $P_n(1) = P_{n-1}$.
2. $P_n(2) = 0$. (Why?)
3. Otherwise, $3 \leqslant i \leqslant n$, and the subpaths $Q = \{p_k\}_{k=1}^i$ and $R = \{p_k\}_{k=i+1}^n$ are themselves two valid paths of lengths i and $n-i$ respectively. Since by hypothesis i was the *first* index at which $\sum_{k=1}^i p_k = 0$, $Q' = \{p_k\}_{k=2}^{i-1}$ is *also* a valid path of length $i-2$. Since the choices of Q' and R are completely free, we have $P_n(i) = P_{i-2}P_{n-i}$.

This completes the proof. □

The sequence P_n is A004148 in the OEIS [oei] and runs 1, 1, 1, 2, 4, 8, 17, 37, ...

5.2 Littlewood-Richardson tableaux

Definition 5.2.1. Let $P, Q \in \mathbb{Z}^n$ be weakly decreasing n-tuples of nonnegative integers with $p_i > q_i$ for each $i \in [n]$. A *Littlewood-Richardson tableau* of shape P/Q is an assignment of 1, 2, or 3 to each empty square in a diagram such as Figure 5.2 satisfying the following conditions:

1. each row is weakly increasing, read left to right;

2. each column is strongly increasing, read top to bottom;
3. when following the diagram by rows, *right to left,* top to bottom, it is always true that the count of 1s is at least the count of 2s is at least the count of 3s.

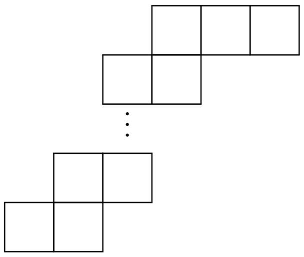


Figure 5.2: An example (unfilled) tableau. There are p_i total spaces on the ith row, q_i of which are unfilled.

We will mainly be concerned with tableaux of the shape $(n, n-1, n-3, \dots)/(n-2, n-4, \dots)$. (The dots indicate that the shape vector entries decrease to zero rather than becoming negative.) When $n = 4$, there are four valid LR tableaux of this shape, depicted in Figure 5.3.

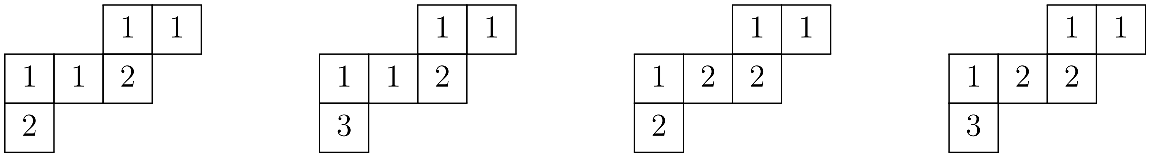


Figure 5.3: The four valid LR tableaux for $n = 4$.

5.3 Equinumeracy?

The alert reader will notice that the sequence that counts peakless Motzkin paths also apparently counts LR tableaux of the shape $(n, n-1, n-3)/(n-2, n-4, \dots)$.

Conjecture 5.1. For any $n \geqslant 0$, the number of LR tableaux of the given shape is given by P_n.

This is confirmed computationally (see Chapter 6) up to at least $n = 26$ (at which both sequences number 560954047.)

One possible attack on this conjecture is as follows. Define the *maximal prefix* of any LR tableau to be the largest n such that the first $1, 2, \ldots, n$ columns of the tableau (read from left to right) themselves form a valid tableau. Necessarily, if the maximal prefix is less than n, taking the first $n+1$ columns must violate one of the LR conditions.

The reader may verify that the tableaux in Figure 5.3 above, from left to right, have maximal prefixes 4, 0, 1, and 0. It is clear that exactly one tableau, the one given by taking the smallest allowable choice for each number, has maximal prefix n. Observe that exactly $2 = 4 - 2 = P_4 - P_3$ tableaux have maximal prefix 0; exactly $1 = P_3 - P_2$ tableau has maximal prefix 1; and $0 = P_2 - P_1 = P_1 - P_0$ tableaux have maximal prefix 2 or 3.

Conjecture 5.2. The number of n-column LR tableaux having maximal prefix $k = 0, 1, \ldots, n-1$ is given by $P_{n-k} - P_{n-k-1}$.

Since the maximal prefix is well-defined, this would immediately imply Conjecture 5.1. There is also computational evidence to support this pattern.

Since we have $P_{n-k} - P_{n-k-1} = \sum_{i=3}^{n-k} P_{i-2} P_{n-k-i}$, a characterization of a tableau having maximal prefix k as a concatenation of two freely chosen tableaux of the appropriate lengths, in the spirit of Proposition 5.1, would suffice.

CHAPTER 6

COMPUTATIONAL METHODS

This chapter outlines the Python code provided in Appendix A.

6.1 Overview of the `dc-lattices` library

In this section we describe the contents of each file listed in Appendix A.

1. `dc_lattice.py` contains the abstract initialization and population routines for generic DCDLs. We model a DCDL as a graph using the `networkx` library.
2. `incremental_lattice.py` overrides some methods in `dc_lattice.py` to handle generic *incremental* lattices (which, it should be noted, includes every example in this thesis.)
3. `boolean_lattice.py`, `zps_lattice.py`, and `fibonacci_lattice.py` implement the lattices described in Chapter 3.
4. `catalan_lattice.py` implements the lattices described in Chapter 4.
5. `gt_pattern.py` contains routines to generate all Gelfand-Tsetlin patterns of a prescribed size using a standard backtracking search.[1]
6. `lr_tableaux.py` contains routines to generate all Littlewood-Richardson tableaux of the shape described in Chapter 5, again using backtracking search.
7. `eppstein_layout.py` is described in the next section.

6.2 Drawing incremental lattices

Most of the figures in this document were generated by the `networkx` and `matplotlib` libraries, using in particular the force-directed graph drawing algorithm (see [KK89].)

[1] *Nota bene:* The constructor for `CatalanLattice` assumes the second argument is $2k$, not k. Calling, for instance, `CatalanLattice(3, 1)` will not work.

Implementation details appear in the file `dc_lattice.py` under the method `draw()`.

Eppstein's graph drawing algorithm

We also implement an algorithm due to [EFO08] that is tailored specifically to incremental lattices, which Eppstein calls *media*. Implementation details appear in the file `eppstein_layout.py`.

The idea is to linearly project a graph G onto $\mathbb{Z}^2$; that is, to produce vectors $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^n$ so that, given an embedding $\phi : G \to \mathbb{Z}^n$ of G, the vertex $v \in G$ is positioned at $\mathbf{x} \cdot \phi(v), \mathbf{y} \cdot \phi(v)$.

We would like to choose $\mathbf{x}$ and $\mathbf{y}$ so as to satisfy the following properties:

1. The entries of $\mathbf{x}$ and $\mathbf{y}$ should be distinct, so that the product structure of the graph is made apparent in the projection; that is, graph edges which correspond to increments in the same coordinate should be represented by the same vector.
2. Collinear edges should not overlap.
3. No edge should come within a fixed distance d of any vertex to which it is not incident.

Eppstein accomplishes this as follows. Fix an embedding $\phi : G \to \mathbb{Z}^n$ and let its components be denoted by $\phi_1, \ldots, \phi_n$. Note that the pullback $\phi_k^{-1}[j]$ is simply the k-slice through j. Now define

$$x_k = \begin{cases} 0, & k = 1 \\ \max\limits_{\substack{\phi_k^{-1}[j] \neq \varnothing \\ \phi_k^{-1}[j-1] \neq \varnothing}} \left(\min\limits_{v \in \phi_k^{-1}[j]} \sum_{i=1}^{k-1} x_i \phi_i(v) - \max\limits_{v \in \phi_k^{-1}[j-1]} \sum_{i=1}^{k-1} x_i \phi_i(v) \right) + d, & 1 < k \leqslant n; \end{cases}$$

$$y_k = \begin{cases} 0, & k = n \\ \max\limits_{\substack{\phi_k^{-1}[j] \neq \varnothing \\ \phi_k^{-1}[j-1] \neq \varnothing}} \left(\min\limits_{v \in \phi_k^{-1}[j]} \sum_{i=k+1}^{n} y_i \phi_i(v) - \max\limits_{v \in \phi_k^{-1}[j-1]} \sum_{i=k+1}^{n} y_i \phi_i(v) \right) + d, & 1 \leqslant k < n. \end{cases}$$

These values generally must be computed recursively. Intuitively, the quantity inside the parentheses is the amount by which the consecutive k-slices $\phi_k^{-1}[j-1]$, $\phi_k^{-1}[j]$ would overlap if G were projected with all of $x_k, x_{k+1}, \ldots, x_n$ (or $y_k, y_{k-1}, \ldots, y_1$) set to zero. Thus, setting X_k (or Y_k) to that value plus d guarantees d units of separation.

6.3 Certifying product solitarity

By *certifying the product solitarity* of a given DCDL L, we mean to answer the following question. Suppose a set of positive rational edge products exist for L. What is the largest subset of L with verifiably unique edge products?[2]

The algorithm we use, which is implemented in `dc_lattice.py` under the routines `solvable_from()` and `solvable_subgraph()`, is fairly crude. It relies on three facts which we have already shown:

1. The set of edge products for a chain are uniquely determined.
2. Once the edge products for one face of a monochromatic component are uniquely determined, the edge products for the entire component are.
3. Once the edge products on three edges of a diamond are determined, so is the fourth.

It proceeds as follows.

1. Initialize a queue Q of known-solvable edges, and a list S of "visited" solved edges.
2. Push any edges known *a priori* onto Q.
3. Determine all chains. Push their edges onto Q.
4. If Q is empty, halt and return S.
5. Pop an edge e from Q. Add it to S.

[2]Note that we need not actually *calculate* these edge products.

6. Check if the component e belongs to has a solvable face. If so, push all of that component's edges onto Q.
7. Check for any diamonds having three edges in S. If any are found, add those diamonds' fourth edges to Q.
8. Set $Q = Q \backslash S$.
9. Go to 4.

The foregoing facts assure that at any stage during the execution of this algorithm, the set S consists only of edges with uniquely determined products. It should also be clear that this algorithm halts, since step 8 ensures that any edge is handled at most once.

Note that this algorithm *does not* find a *maximal* subset on which edge products are uniquely determined. For instance, while $\mathscr{F}(3,3)$ is product solitary, this fact depends on the zero discriminant of a quadratic equation arising from a diamond relation, which is not one of the criteria used by the algorithm. So calling `solvable_from` on $\mathscr{F}(3,3)$ without any known edges fails. However, if the edge product $\pi_{147,157}$ is assumed known, the routine succeeds.

Some further improvements to this code should be fairly straightforward. For instance, the framework provided should make it simple to actually calculate the edge products. Doing so may provide empirical insight into the question asked at the end of Chapter 4. It should also be easy to produce interactive visualizations of the lattices generated, perhaps using an appropriate JavaScript library.

BIBLIOGRAPHY

[DDMN20] Robert G. Donnelly, Molly W. Dunkum, Sasha V. Malone, and Alexandra Nance. Symmetric Fibonaccian distributive lattices and representations of the special linear Lie algebras. In preparation, 2020.

[Don03] Robert G. Donnelly. Extremal properties of bases for representations of semisimple Lie algebras. *Journal of Algebraic Combinatorics*, 17, 2003.

[Don18] Robert G. Donnelly. Poset models for Weyl group analogs of symmetric functions and Schur functions, 2018.

[EFO08] David Eppstein, Jean-Claude Falmagne, and Sergei Ovchinnikov. *Media Theory: Interdisciplinary Applied Mathematics*, chapter 11. Springer-Verlag, 2008.

[EW06] Karin Erdmann and Mark J. Wildon. *Introduction to Lie algebras*. Springer-Verlag, 2006.

[GT50] Israel M. Gelfand and Michael L. Tsetlin. Finite-dimensional representations of the group of unimodular matrices. 71:825–8, 1950.

[GV89] Ira M. Gessel and Xavier G. Viennot. Determinants, paths and plane partitions. Unpublished preprint, 1989.

[Hum72] James E. Humphreys. *Introduction to Lie algebras and representation theory*. Springer-Verlag, 1972.

[KK89] Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31, 1989.

[oei] The Online Encyclopedia of Integer Sequences. http://oeis.org/A004148.

[Pro82] Robert A. Proctor. Solution of two difficult combinatorial problems with linear algebra. *American Mathematical Monthly*, 89(10), 1982.

[Sta97] Richard P. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge University Press, 1997.

[Sta01] Richard P. Stanley. *Enumerative Combinatorics*, volume 2. Cambridge University Press, 2001.

APPENDIX A

THE DC-LATTICES LIBRARY

This code is up to date as of November 16, 2020. A more recent revision may be available at gitlab.com/sverona/thesis.

Listing A.1: dc_lattice.py

```
#!/usr/bin/env python3

""" Computationally models generic diamond-crossing ("DC") lattices.
"""
from itertools import combinations

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import networkx as nx


class DCLattice(nx.Graph):
    def __init__(self):
        super().__init__()

        self.add_edges_from(self.generate_edges())

    def generate_vertices(self):
        """Should be overridden.
        """
        pass

    def potential_children(self, vertex):
        """Should be overridden.
        """
        pass

    def generate_edges(self):
        """Generate all edges in this lattice.
        """
        for vertex in self.generate_vertices():
            for potential_child in self.potential_children(
                vertex
            ):
                if self.is_valid_label(potential_child):
                    yield (
                        vertex,
                        potential_child,
                        {
                            "color": self.edge_color(
                                vertex, potential_child
                            )
                        },
                    )

    def edge_color(self, vertex1, vertex2):
        """Return the color the edge between `vertex1` and `vertex2`
           should have, or None
        if they are nonadjacent.

        Should be overridden.
        """
        pass

    def solvable_from(self, edge):
        sc = self.solvable_subgraph(edge)

        remaining_nodes = self.nodes() - sc.nodes()
        remaining_edges = self.edges() - sc.edges()
        solvable = not (remaining_nodes or remaining_edges)

        if not solvable:
            print("Remaining vertices:")
            for node in remaining_nodes:
                print("\t", node)
            print("Remaining edges:")
            for edge in remaining_edges:
                print("\t", edge)

        return solvable

    def solvable_subgraph(self, edge, return_order=False):
        total_edges = len(self.edges())
        solvable_edges = []

        Q = []

        unsolved_diamonds = list(self.diamonds())

        def push(Q, edge):
            edge = (min(edge), max(edge))

            if edge not in Q and edge not in solvable_edges:
                return [edge] + Q
            else:
                return Q

        if edge:
            Q = push(Q, edge)

        for chain in self.chains():
            for edge in chain.edges():
                # print("Pushing chain edge", edge)
                Q = push(Q, edge)

        while Q:
            """
            print("Q is:")
            for edge in Q:
                print(edge)
            print()

            print("solvable is:")
            for edge in solvable_edges:
                print(edge)
            print()
            """
            edge = Q.pop()
            # print("Popping", edge, len(solvable_edges), "/",
               total_edges, len(unsolved_diamonds))
            solvable_edges.append(edge)
            # Check if edge's component is solvable.
            component = self.component(
                min(edge), self.edge_color(*edge)
            )

            def component_is_solvable_now(component):
                for face in self.faces(component):
                    edges = face.edges()
                    edges = [
                        (min(edge), max(edge))
                        for edge in edges
                    ]

                    if not set(edges) - set(solvable_edges):
                        return True
                return False

            if component_is_solvable_now(component):
                for edge in component.edges():
                    # print("Pushing solvable component edge", edge)
                    Q = push(Q, edge)

            # Check for solvable diamonds.
            for diamond in unsolved_diamonds:
                top, left, right, bottom = diamond
                diamond_edges = set(
                    (
                        (top, left),
                        (top, right),
                        (left, bottom),
                        (right, bottom),
                    )
                )

                unsolved_edges = list(
                    diamond_edges - set(solvable_edges)
                )

                if len(unsolved_edges) == 1:
                    # print("Pushing lone diamond edge", edge)
                    Q = push(Q, unsolved_edges[0])
                    unsolved_diamonds.remove(diamond)

        if return_order:
            return solvable_edges

        return nx.Graph(solvable_edges)

    def colors(self):
        """Should be overridden.
        """
        pass

    def diamonds(self):
        for vertex in self:
            yield from self.diamonds_at_vertex(vertex)

    def diamonds_at_vertex(self, vertex):
        covers = [
            nbr for nbr in self[vertex] if nbr > vertex
        ]
        for left, right in combinations(covers, 2):
            yield (
                vertex,
                left,
                right,
                self.meet(left, right),
            )

    def meet(self, vertex, neighbor):
        """Should be overridden.
        """
        pass

    def join(self, vertex, neighbor):
        """Should be overridden.
        """
        pass

    def coordinates(self, vertex):
        return vertex

    def chains(self):
        """Return an iterator over all chains in this lattice.

        A color-component is a chain iff its vertex cardinality exceeds
            its
        edge cardinality by one.
        """

        chain_list = set()

        for vertex in self:
            for color in self.colors():
                component = self.component(vertex, color)
                if (
                    len(component)
                    == len(component.edges()) + 1
                ):
                    chain_list.add(component)

        return chain_list

    def breadth_first_search(self, node, key, graph=None):
        if not graph:
            graph = self

        found = set()
        found_edges = []
        frontier = set([node])

        while frontier:
            next_frontier = set()
            for fnode in frontier:
                found.add(fnode)

                for fnbr in graph[fnode]:
                    if fnbr not in found:
                        if key(fnode, fnbr):
                            if fnode < fnbr:
                                found_edges.append(
                                    (fnode, fnbr)
                                )
                            else:
                                found_edges.append(
                                    (fnbr, fnode)
                                )
                            next_frontier.add(fnbr)

            frontier = next_frontier

        found_edges = list(set(found_edges))
        # return nx.Graph(graph).subgraph(found)
        return nx.Graph(found_edges)

    def component(self, node, color):
        """Return the `color`-component in which `node` lies.
        """

        def search_key(fnode, fnbr):
            return self.edge_color(fnode, fnbr) == color

        return self.breadth_first_search(node, search_key)

    def coord(self, vertex1, vertex2):
        return min(
            i
            for i in range(len(vertex1))
            if vertex1[i] != vertex2[i]
        )

    def chain_factorization(self, component):
        """Return a chain factorization of `component`.
        """

        if not component:
            return [tuple()]

        chains = [
            [(min(component), nbr)]
            for nbr in component[min(component)]
        ]

        edge_added = True
        while edge_added:
            edge_added = False
            for chain in chains:
                tip = chain[-1][-1]

                for candidate_vtx in component[tip]:
                    if candidate_vtx > tip:
                        candidate_edge = (
                            tip,
                            candidate_vtx,
                        )
                        matching_edges = set(
                            tuple(edge)
                            for chain2 in chains
                            for edge in chain2
                            if self.coord(*edge)
                            == self.coord(*candidate_edge)
                            and chain2 != chain
                        )
                        if not matching_edges:
                            edge_added = True
                            chain.append(candidate_edge)

        return tuple(tuple(chain) for chain in chains)

    def chain_changes(self, component):
        cf = self.chain_factorization(component)

        changes = [
            set(self.coord(*e) for e in chain)
            for chain in cf
        ]

        return changes
```

```
def faces(self, component):
    if not component:
        return []

    top, bot = min(component), max(component)

    changes = self.chain_changes(component)

    for change in changes:

        def search_key(fnode, fnbr):
            return self.coord(fnode, fnbr) not in change

        yield self.breadth_first_search(
            top, search_key, graph=component
        )
        yield self.breadth_first_search(
            bot, search_key, graph=component
        )

def vertex_weight(self, node):
    """Return the weight of `node` in this lattice.
    """
    weight = []
    for color in self.colors():
        component = self.component(node, color)
        ranks = [self.rank(node) for node in component]
        top_rank, bottom_rank = min(ranks), max(ranks)

        rank = top_rank - self.rank(node)
        depth = self.rank(node) - bottom_rank

        this_weight = rank - depth
        weight.append(this_weight)

    return tuple(weight)

def node_label(self, node):
    return "".join(str(c) for c in node)

def draw(self, eppstein=False, coordinates=None, cmap="Blues"):
    if coordinates is None:
        coordinates = self.coordinates

    if eppstein:
        from eppstein_layout import eppstein_layout

        pos = eppstein_layout(
            self, coordinates=coordinates
        )
    else:
        pos = nx.kamada_kawai_layout(self)

    cmap = plt.cm.get_cmap(cmap, len(self.colors()))

    node_labels = {v: self.node_label(v) for v in self}

    edge_colors = {
        (v1, v2): c
        for v1, v2, c in self.edges(data="color")
    }

    nx.draw_networkx(
        self,
        pos,
        edge_color=edge_colors.values(),
        edge_cmap=cmap,
        vmin=min(self.colors()),
        vmax=max(self.colors()),
        node_size=0,
        labels=node_labels,
        font_size=8,
        font_family="serif",
    )

    sm = plt.cm.ScalarMappable(
        cmap=cmap,
        norm=plt.Normalize(
            vmin=min(self.colors()) - 1,
            vmax=max(self.colors()),
        ),
    )

    cbar = plt.colorbar(
        sm, ticks=[c - 0.5 for c in self.colors()]
    )

    cbar.ax.set_yticklabels(
        [str(x) for x in self.colors()]
    )

    plt.show()
```

Listing A.2: incremental_lattice.py

```
from dc_lattice import DCLattice


class IncrementalLattice(DCLattice):
    def __init__(self):
        super().__init__()

    def potential_children(self, vertex):
        for idx, value in enumerate(vertex):
            vertexcopy = [x for x in vertex]
            vertexcopy[idx] += 1

            yield tuple(vertexcopy)

    def meet(self, vertex1, vertex2):
        return tuple(max(x) for x in zip(vertex1, vertex2))

    def join(self, vertex1, vertex2):
        return tuple(min(x) for x in zip(vertex1, vertex2))

    def edge_color(self, vertex1, vertex2):
        for idx, value in enumerate(vertex1):
            if abs(vertex1[idx] - vertex2[idx]) == 1:
                return idx + 1
        return None

    def is_adjacent(self, vertex1, vertex2):
        for idx, value in enumerate(vertex1):
            if abs(vertex1[idx] - vertex2[idx]) == 1:
                return True
        return False
```

Listing A.3: boolean_lattice.py

```
#!/usr/bin/env python3

from itertools import product

from incremental_lattice import IncrementalLattice


class BooleanLattice(IncrementalLattice):
    def __init__(self, n):
        self.n = n

        super().__init__()

    def generate_vertices(self):
        yield from product([0, 1], repeat=self.n)

    def colors(self):
        return range(self.n)

    def is_valid_label(self, potential_child):
        return len(potential_child) == self.n and all(
            i in [0, 1] for i in potential_child
        )

    def edge_color(self, vertex1, vertex2):
        return min(
            i
            for i in range(len(vertex1))
            if vertex1[i] != vertex2[i]
        )
```

Listing A.4: zps_lattice.py

```
#!/usr/bin/env python3

from itertools import product

from incremental_lattice import IncrementalLattice


class ZPSLattice(IncrementalLattice):
    def __init__(self, n):
        self.n = n

        super().__init__()

    def generate_vertices(self):
        for bools in product([True, False], repeat=self.n):
            members = [
                x if b else 0
                for x, b in zip(range(1, self.n + 1), bools)
            ]

            members = tuple(sorted(members))
            yield members

    def colors(self):
        return range(1, self.n + 1)

    def is_valid_label(self, potential_child):
        for idx in range(self.n - 1):
            if potential_child[idx] not in range(
                0, self.n + 1
            ):
                return False
            if (
                0
                < potential_child[idx]
                == potential_child[idx + 1]
            ):
                return False
            if (
                potential_child[idx]
                > potential_child[idx + 1]
            ):
                return False

        if potential_child[-1] not in range(0, self.n + 1):
            return False
        return True

    def edge_color(self, vertex1, vertex2):
        diff_idx = min(
            i
            for i in range(len(vertex1))
            if vertex1[i] != vertex2[i]
        )

        min_coord = min(
            vertex1[diff_idx], vertex2[diff_idx]
        )

        return self.n - min_coord
```

Listing A.5: fibonacci_lattice.py

```
#!/usr/bin/env python3

"""Computationally models the Fibonaccian lattice $$L_A^{Fib}(n + 1, k).
   $$
Throughout, we refer to the parameters $$n + 1$$, $$k$$ as the _width_
   and _length_.
"""

from itertools import product, starmap

from incremental_lattice import IncrementalLattice


class FibonacciLattice(IncrementalLattice):
    def __init__(self, n, k):
        self.width = n
        self.length = k

        super().__init__()

    def generate_vertices(self):
        r"""Return all vertices in this lattice.
        A vertex of $$L_A^{Fib}(n + 1, k)$$ is a $$k$$-tuple $$v \in \
            mathbb Z^k$$ satisfying

          - $$(i - 1) < v_i / k \leq i $$;
          - $$v_{i + 1} - v_i > 1 $$ for each $$i = 1, dots, k - 1$$.
        """

        min_tab = [
            1 + self.width * i for i in range(self.length)
        ]
        offsets = product(
            range(self.width), repeat=self.length
        )

        for offset in offsets:
            tab = tuple(
                map(lambda c, o: c + o, min_tab, offset)
            )

            if self.is_valid_label(tab):
                yield tab

    def colors(self):
        return range(1, self.width)

    def edge_color(self, vertex1, vertex2):
        if not self.is_adjacent(vertex1, vertex2):
            return None

        diffs = [
            (comp1, comp2)
            for comp1, comp2 in zip(vertex1, vertex2)
            if comp1 != comp2
        ][0]

        lower_label = min(diffs)

        # The following variables refer to this zero-indexed grid.
        # 1   n-1 1   n-1 (k columns)
        # 2   n-2 2   n-2 ...
        # ...
        # n-1 1   n-1 1
        # The color is given by the element in column lower_label // n
        # and row lower_label % n.

        col = lower_label // self.width
        row = lower_label - self.width * col
        if col % 2 == 0:
            return row
        return self.width - row

    def is_valid_label(self, tableau):
        """Check if `tableau` is a valid vertex for this lattice.
        """

        for idx in range(1, len(tableau)):
            if tableau[idx] - tableau[idx - 1] == 1:
                return False

        for idx, coord in enumerate(tableau):
            if (
                not idx * self.width
                < coord
                <= (1 + idx) * self.width
            ):
                return False

        return True
```

Listing A.6: catalan_lattice.py

```
#!/usr/bin/env python3

import numpy as np
```

```
from dc_lattice import DCLattice
from gt_pattern import gt_patterns, is_gt


class CatalanLattice(DCLattice):
    def __init__(self, size, cap):
        self.size = size
        self.cap = cap

        super().__init__()

    def node_label(self, node):
        return "\n".join(
            "".join(str(i) for i in row) for row in node
        )

    def generate_vertices(self):
        for mat in gt_patterns(self.size, cap=self.cap):
            yield tuple(tuple(row) for row in mat)

    def is_valid_label(self, vertex):
        return is_gt(vertex, cap=self.cap)

    def potential_children(self, vertex):
        for x, row in enumerate(vertex):
            for y, entry in enumerate(row):
                if entry is not None:
                    if x == self.size - 1:
                        delta = 2
                    else:
                        delta = 1
                    yield tuple(
                        tuple(
                            c + delta
                            if (x == x2 and y == y2)
                            else c
                            for y2, c in enumerate(row2)
                        )
                        for x2, row2 in enumerate(vertex)
                    )

    def rank(self, vertex):
        return sum(sum(row) for row in vertex)

    def colors(self):
        return range(self.size)

    def is_adjacent(self, vertex1, vertex2):
        differences = 0
        for x in range(self.size):
            for y in range(x + 1):
                if vertex1[x][y] != vertex2[x][y]:
                    if x == self.size - 1:
                        delta = 2
                    else:
                        delta = 1

                    if (
                        abs(vertex1[x][y] - vertex2[x][y])
                        != delta
                    ):
                        return False
                    else:
                        differences += 1
        return differences == 1
```

```
def edge_color(self, vertex1, vertex2):
    if self.is_adjacent(vertex1, vertex2):
        for x in range(self.size):
            for y in range(x + 1):
                if vertex1[x][y] != vertex2[x][y]:
                    return x

def meet(self, vertex1, vertex2):
    return tuple(
        tuple(max(c1, c2) for c1, c2 in zip(row1, row2))
        for row1, row2 in zip(vertex1, vertex2)
    )

def join(self, vertex1, vertex2):
    return tuple(
        tuple(min(c1, c2) for c1, c2 in zip(row1, row2))
        for row1, row2 in zip(vertex1, vertex2)
    )

def edge_color(self, vertex1, vertex2):
    for row_idx, row in enumerate(vertex1):
        if row_idx == self.size - 1:
            delta = 2
        else:
            delta = 1

        for col_idx, entry in enumerate(row):
            if (
                abs(entry - vertex2[row_idx][col_idx])
                == delta
            ):
                return row_idx + 1
    return None

def is_adjacent(self, vertex1, vertex2):
    for row_idx, row in enumerate(vertex1):
        if row_idx == self.size - 1:
            delta = 2
        else:
            delta = 1

        for col_idx, entry in enumerate(row):
            if (
                abs(entry - vertex2[row_idx][col_idx])
                == delta
            ):
                return row_idx + 1
    return None

def coord(self, vertex1, vertex2):
    if self.is_adjacent(vertex1, vertex2):
        for x in range(self.size):
            for y in range(x + 1):
                if vertex1[x][y] != vertex2[x][y]:
                    return (x, y)

def coordinates(self, vertex):
    coords = list(list(row[:(idx + 1)]) for idx, row in enumerate(
        vertex))
    coords[-1] = list(coord // 2 for coord in coords[-1])
    return np.concatenate(coords)
```

Listing A.7: gt_pattern.py

```python
#!/usr/bin/env python3


def is_lower_triangular(mat):
    for row_idx, row in enumerate(mat):
        for col_idx, entry in enumerate(row):
            if (
                col_idx > row_idx
                and entry is not None
                and entry != 0
            ):
                return False
    return True


def is_positive_and_under_cap(mat, cap=2):
    for row_idx, row in enumerate(mat):
        for col_idx, entry in enumerate(row):
            if entry is not None and not (
                0 <= entry <= cap
            ):
                return False
    return True


def is_symplectic(mat):
    for col_idx, entry in enumerate(mat[-1]):
        if entry is not None and entry % 2 != 0:
            return False
    return True


def has_gt_property(mat):
    for row_idx, row in enumerate(mat):
        if row_idx == len(mat) - 1:
            continue
        for col_idx, entry in enumerate(row):
            if col_idx > row_idx:
                continue
            south_of_us = mat[row_idx + 1][col_idx]
            se_of_us = mat[row_idx + 1][col_idx + 1]
            if None in [entry, south_of_us, se_of_us]:
                continue
            if not (south_of_us <= entry <= se_of_us):
                return False
    return True


def is_gt(mat, cap=2):
    return (
        is_positive_and_under_cap(mat, cap=cap)
        and is_symplectic(mat)
        and is_lower_triangular(mat)
        and has_gt_property(mat)
    )


def is_complete(mat):
    for row in mat:
        if None in row:
            return False
    return True


def first_incrementable_entry(mat):
    for row_idx, row in enumerate(mat):
        for col_idx, entry in enumerate(row):
            if entry is None:
                return [row_idx, col_idx]
    return None


def backtrack(candidate, cap=2):
    if not is_gt(candidate, cap=cap):
        return
    elif is_complete(candidate):
        yield candidate

    first_increment = first_incrementable_entry(candidate)

    if first_increment is None:
        return
    else:
        (
            first_increment_row,
            first_increment_col,
        ) = first_increment
        for val in range(cap + 1):
            copy_of_candidate = [
                [x for x in row] for row in candidate
            ]
            copy_of_candidate[first_increment_row][
                first_increment_col
            ] = val
            yield from backtrack(copy_of_candidate, cap=cap)


def gt_patterns(size, cap=2):
    initial = [
        [0 if col > row else None for col in range(size)]
        for row in range(size)
    ]

    yield from backtrack(initial, cap=cap)
```

Listing A.8: lr_tableaux.py

```
def columns_are_strongly_increasing(tab):
    for col in tab:
        if None in col:
            continue
        if len(col) == 2 and not (col[0] < col[1]):
            return False
    return True


def rows_are_weakly_decreasing(tab):
    if len(tab) < 2:
        return True

    if len(tab) % 2 == 0:
        num_rows_to_check = len(tab) // 2
    else:
        num_rows_to_check = len(tab) // 2 + 1

    for row_idx in range(num_rows_to_check):
        if row_idx == 0:
            row = [tab[0][0], tab[1][0]]
        elif row_idx * 2 + 1 == len(tab):
            row = [tab[-2][1], tab[-1][0]]
        else:
            row = [
                tab[row_idx * 2 - 1][1],
                tab[row_idx * 2][0],
                tab[row_idx * 2 + 1][0],
            ]

        if None in row:
            continue

        for n in range(len(row) - 1):
            if not (row[n] >= row[n + 1]):
                return False
    return True


def is_lattice_word(tab):
    frequencies = [0, 0, 0]
    for col in tab:
        for entry in col:
            if entry is not None:
                frequencies[entry - 1] += 1

            if not (
                frequencies[0]
                >= frequencies[1]
                >= frequencies[2]
            ):
                return False
    return True


def is_lr(tab):
    return (
        columns_are_strongly_increasing(tab)
        and rows_are_weakly_decreasing(tab)
        and is_lattice_word(tab)
    )


def first_incrementable_entry(tab):
    for col_idx, col in enumerate(tab):
        if None in col:
            return tuple([col_idx, col.index(None)])
    return None


def is_complete(tab):
    for col_idx, col in enumerate(tab):
        if None in col:
            return False
    return True


def backtrack(candidate):
    if not is_lr(candidate):
        return
    elif is_complete(candidate):
        yield candidate

    first_increment = first_incrementable_entry(candidate)
    if first_increment is None:
        return
    else:
        (
            first_increment_row,
            first_increment_col,
        ) = first_increment

        for val in [1, 2, 3]:
            copy_of_candidate = [
                [x for x in row] for row in candidate
            ]
            copy_of_candidate[first_increment_row][
                first_increment_col
            ] = val
            yield from backtrack(copy_of_candidate)


def lr_tableaux(size):
    columns = [1 + x % 2 for x in range(size)]
    initial = [
        [None for _ in range(col)] for col in columns
    ]
    yield from backtrack(initial)
```

Listing A.9: eppstein_layout.py

```
import numpy as np
import networkx as nx


def dot_product(a, b, range_=None):
    if not range_:
        range_ = range(len(a))
    return sum(a[k] * b[k] for k in range_)


def eppstein_layout(G, coordinates=np.concatenate):
    X, Y = eppstein_projection(G, coordinates)

    return {
        v: tuple(
            [
                dot_product(X, coordinates(v)),
                dot_product(Y, coordinates(v)),
            ]
        )
        for v in G
    }


def eppstein_projection(G, coordinates=np.concatenate):
    def possible_entries(G, i):
        return [coordinates(v)[i] for v in G]

    def slice(G, i, j):
        for v in G:
            if coordinates(v)[i] == j:
                yield v

    for v in G:
        d = len(coordinates(v))
        break
    X = [None for _ in range(d)]
    Y = [None for _ in range(d)]

    X[0] = 0
    for i in range(1, d):
        current_max = None
        for j in sorted(possible_entries(G, i)):
            this_slice = list(slice(G, i, j))
            last_slice = list(slice(G, i, j - 1))
            if len(this_slice) == 0 or len(last_slice) == 0:
                continue
            candidate_max_min = min(
                dot_product(coordinates(v), X, range(i))
                for v in this_slice
            )
            candidate_max_max = max(
                dot_product(coordinates(v), X, range(i))
                for v in last_slice
            )
            candidate_max = (
                candidate_max_max - candidate_max_min + 1
            )
            if (
                current_max is None
            ) or candidate_max > current_max:
                current_max = candidate_max
        X[i] = current_max

    Y[-1] = 0
    y_range = list(range(d - 1))[::-1]
    for i in y_range:
        current_max = None
        for j in sorted(possible_entries(G, i)):
            this_slice = list(slice(G, i, j))
            last_slice = list(slice(G, i, j - 1))
            if len(this_slice) == 0 or len(last_slice) == 0:
                continue
            candidate_max_min = min(
                dot_product(
                    coordinates(v), X, range(i + 1, d)
                )
                for v in this_slice
            )
            candidate_max_max = max(
                dot_product(
                    coordinates(v), X, range(i + 1, d)
                )
                for v in last_slice
            )
            candidate_max = (
                candidate_max_max - candidate_max_min + 1
            )
            if (
                current_max is None
            ) or candidate_max > current_max:
                current_max = candidate_max
        Y[i] = current_max

    return (X, Y)
```

APPENDIX B

PROOF OF THE DCS THEOREM

We reproduce the DCS theorem here for convenience:

Theorem B.1 (Donnelly)**.** Let L be a $\mathfrak{g}$-structured DCDL. If for any edge $x \to y$ there exist $c_{yx}, d_{xy} \in \mathbb{Q}_+$ such that

1. if $w, x, y, z \in L$ form a diamond with $w \to x \to z$, $w \to y \to z$, we have the *diamond relations* $c_{yw}d_{wx} = d_{xz}c_{zy}, c_{xw}d_{wy} = d_{yz}c_{zx}$,
2. and for any $x \in L$ and any $i \in [n]$, we have the *crossing relation*

$$\sum_{\substack{w \in L \\ w \xrightarrow{i} x}} c_{xw}d_{wx} - \sum_{\substack{z \in L \\ x \xrightarrow{i} z}} c_{zx}d_{xz} = \omega_i(x),$$

then $\mathfrak{g}$ is homomorphic to the Lie algebra $\mathfrak{l}$ generated by the $3n$ $|L|$-dimensional matrices $\{E_i, F_i, H_i\}_{i=1}^n$ (whose rows and columns are indexed by the elements of L) defined as follows:

1. $[E_i]_{yx} := c_{yx}$ if $x \xrightarrow{i} y$, or 0 otherwise;
2. $[F_i]_{xy} := d_{xy}$ if $x \xrightarrow{i} y$, or 0 otherwise;
3. $H_i := [E_i F_i]$.

We say that L *realizes a representation of* $\mathfrak{g}$. Additionally, if $\mathfrak{g}$ is simple, the representation so realized is faithful.

We will need the following lemma about the structure of H_i.

Lemma B.1. Consider E_i, F_i, H_i as elements of $GL(|L|, \mathbb{R})$, and let $B = \{v_x\}_{x \in L}$ be a standard basis. Then relative to B, $H_i = \mathrm{diag}\{\omega_i(x)\}_{x \in L}$.

Proof of lemma. First, we compute

$$\begin{aligned}
(E_iF_i)_{xy} &= \sum_{w\in L}(E_i)_{xw}(F_i)_{wy} \\
&= \sum_{w\in L}(\mathbf{1}_{w\xrightarrow{i}x}c_{xw})(\mathbf{1}_{w\xrightarrow{i}y}d_{wy}) \\
&= \sum_{w\in L}\mathbf{1}_{x\xleftarrow{i}w\xrightarrow{i}y}c_{xw}d_{wy} \\
&= \sum_{\substack{w\in L\\ x\xleftarrow{i}w\xrightarrow{i}y}} c_{xw}d_{wy}, \\
(F_iE_i)_{xy} &= \sum_{z\in L}(F_i)_{xz}(E_i)_{zy} \\
&= \sum_{z\in L}(\mathbf{1}_{x\xrightarrow{i}z}d_{xz})(\mathbf{1}_{y\xrightarrow{i}z}c_{zy}) \\
&= \sum_{z\in L}\mathbf{1}_{x\xrightarrow{i}z\xleftarrow{i}y}d_{xz}c_{zy} \\
&= \sum_{\substack{z\in L\\ x\xrightarrow{i}z\xleftarrow{i}y}} d_{xz}c_{zy}.
\end{aligned}$$

Now consider an entry in H_i, say $(H_i)_{xy}$. If $x \neq y$, then we have

$$(H_i)_{xy} = \sum_{\substack{w\in L\\ x\xleftarrow{i}w\xrightarrow{i}y}} c_{xw}d_{wy} - \sum_{\substack{z\in L\\ x\xrightarrow{i}z\xleftarrow{i}y}} d_{xz}c_{zy}.$$

Since L is distributive and diamond-colored, for every $w \in L$ such that $x \xleftarrow{i} w \xrightarrow{i} y$ there is exactly one $z \in L$ with $x \xrightarrow{i} z \xleftarrow{i} y$. Then w, x, y, z form a diamond of color-i edges, and we have

$$(H_i)_{xy} = \sum_{\substack{\text{diamonds}\\ w,x,y,z}} c_{xw}d_{wy} - d_{xz}c_{zy} = 0$$

by the diamond relations. If $x = y$, then we have

$$(H_i)_{xx} = \sum_{\substack{w\in L\\ w\xrightarrow{i}x}} c_{xw}d_{wx} - \sum_{\substack{z\in L\\ x\xrightarrow{i}z}} d_{xz}c_{zx} = \omega_i(x)$$

by the crossing relation. □

Proof of DCS theorem. First we need to show that E_i, F_i, H_i satisfy the Serre relations.

1. $[H_i H_j] = 0$ for any i, j.

   Since the H_i are diagonal matrices, they commute.

2. $[E_i F_j] = 0$ for $i \neq j$.

   Consider an entry

   $$([E_i F_j])_{xy} = \sum_{\substack{w \in L \\ x \xleftarrow{i} w \xrightarrow{j} y}} c_{xw} d_{wy} - \sum_{\substack{z \in L \\ x \xrightarrow{j} z \xleftarrow{i} y}} d_{xz} c_{zy}.$$

   Note that if $x = y$ the sums are empty. As before, for every $w \in L$ such that $x \xleftarrow{i} w \xrightarrow{j} y$ there is exactly one $z \in L$ such that $x \xrightarrow{j} z \xleftarrow{i} y$, and we have

   $$([E_i F_j])_{xy} = \sum_{\substack{\text{diamonds} \\ w,x,y,z}} c_{xw} d_{wy} - d_{xz} c_{zy} = 0$$

   by diamond relations.

3. $[H_i E_j] = M_{ji} E_j$ for any i, j.

   As before, we have

   $$([H_i E_j])_{xy} = \sum_{z \in L} (H_i)_{xz} (E_j)_{zy} - \sum_{z \in L} (E_j)_{xz} (H_i)_{zy} \quad = ((H_i)_{xx} - (H_i)_{yy}) (E_j)_{xy}$$

   Thus $([H_i E_j])_{xy} = (\omega_i(x) - \omega_i(y)) \cdot (E_j)_{xy}$. Now if it is not the case that $y \xrightarrow{j} x$, then $(E_j)_{xy} = 0 = ([H_i E_j])_{xy}$. So assume $y \xrightarrow{i} x$; then, since L is $\mathfrak{g}$-structured, $\omega_i(x) - \omega_i(y) = M_{ji}$, which was to be shown.

4. $[H_i F_j] = -M_{ji} F_j$ for any i, j.

   Entirely analogous to the previous item.

5. $(\operatorname{ad} E_i)^{1-M_{ji}}(E_j) = (\operatorname{ad} F_i)^{1-M_{ji}}(F_j) = 0$ for $i \neq j$.

A completely rigorous proof that these two relations hold is outside our scope; we direct the interested reader to [Don03], Proposition 3.4, and [Hum72], Proposition 18.1. Here we content ourselves with intuition.

Consider $(\operatorname{ad} E_i)E_j = [E_iE_j]$. An entry of this matrix looks like

$$([E_iE_j])_{xy} = \sum_{z\in L}(E_i)_{xz}(E_j)_{zy} - (E_j)_{xz}(E_i)_{zy}.$$

Necessary for this entry to be nonzero is that there exist a vertex z with either $y \xrightarrow{i} z \xrightarrow{j} x$ or $y \xrightarrow{j} z \xrightarrow{i} x$. Now consider $(\operatorname{ad} E_i)^2E_j = [E_i[E_iE_j]]$. Similarly, we have

$$([E_i[E_iE_j]])_{xy} = \sum_{z\in L}(E_i)_{xz}([E_iE_j])_{zy} - ([E_iE_j])_{xz}(E_i)_{zy},$$

and *this* entry is nonzero only if there is a chain $y \to w \to z \to x$ or $y \to z \to w \to x$ (with the appropriate edge colors.) Successive adjoints require longer chains, but the lattice is finite. Thus, the matrix eventually becomes zero. (The preceding references prove that the given exponent works.)

Now, by Serre's theorem, $\{E_i, F_i, H_i\}_{i=1}^n$ generate a semisimple Lie algebra $\mathfrak{l}$. Since $\mathfrak{l}$ satisfies the same (Serre) relations as $\mathfrak{g}$, the function $\phi : \mathfrak{g} \to \mathfrak{l}$ that sends the Chevalley generators of $\mathfrak{g}$ $\{x_i, y_i, h_i\} \mapsto \{E_i, F_i, H_i\}$ is a homomorphism.[1] If $\mathfrak{g}$ is simple, then it remains to show that ϕ is injective.[2] This follows because $E_i, F_i \neq 0$, so $\ker\phi \neq \mathfrak{g}$. Since $\ker\phi$ is an ideal, we must have $\ker\phi = 0$, so ϕ is injective. □

[1]That is to say, a representation of $\mathfrak{g}$.

[2]Or *faithful*, in the language of representation theory.