Chip-Firing Games & Graphical Riemann-Roch

A Machine-Assisted Proof Framework in Lean4

Dhyey Mavani
Advisor: Prof. Nathan Pflueger

Department of Mathematics
Amherst College

MATH Honors Thesis Defense
April 2025

Table of Contents

Table of Contents

Proof Assistants in Modern Mathematics

Key Proof Assistants

- Lean4 - Modern system and programming language
- Coq - Based on Calculus of Inductive Constructions
- Isabelle - Higher-order logic framework

Benefits of Machine-Assisted Proving in Lean4

Technical Advantages

- Systematic elimination of proof errors
- Modularity for breaking down complex proofs (Mathlib4)
- Independent verification of components

Collaborative & Educational Benefits

- Enables team-based mathematical research
- Educational tools like ”Natural Number Game”
- Popular among mathematicians (including Terence Tao)

Compilation Example

ChipFiringWithLean > Basic.lean > ...

```lean
import Mathlib.LinearAlgebra.Matrix.GeneralLinearGroup.Defs
import Mathlib.Algebra.BigOperators.Group.Finset

import Init.Core
import Init.NotationExtra

import Paperproof

set_option linter.unusedVariables false
set_option trace.split.failure true
set_option linter.unusedSectionVars false

open Multiset Finset

-- Assume V is a finite type with decidable equality
variable {V : Type} [DecidableEq V] [Fintype V]

-- Define a set of edges to be loopless only if it doesn't have loops
def isLoopless (edges : Multiset (V × V)) : Bool :=
  Multiset.card (edges.filter (λ e => (e.1 = e.2))) = 0

def isLoopless_prop (edges : Multiset (V × V)) : Prop :=
  ∀ v, (v, v) ∉ edges

lemma isLoopless_prop_bool_equiv (edges : Multiset (V × V)) :
    isLoopless_prop edges ↔ isLoopless edges = true := by
  unfold isLoopless_prop isLoopless
  constructor
  · intro h
    apply decide_eq_true
    rw [Multiset.card_eq_zero]
    simp only [Multiset.eq_zero_iff_forall_not_mem]
    intro e he
    have h_eq : e.1 = e.2 := by
      exact Multiset.mem_filter.mp he |>.2
    have he' : e ∈ edges := by
      exact Multiset.mem_filter.mp he |>.1
    cases e with
    | mk a b =>
      simp at h_eq
      have : (a, b) = (a, a) := by rw [h_eq]
      rw [this] at he'
      exact h a he'
```

```
▼ Basic.lean:43:4
▼ Tactic state
1 goal
▼ case mp.a
V : Type
inst✝¹ : DecidableEq V
inst✝ : Fintype V
edges : Multiset (V × V)
h : ∀ (v : V), (v, v) ∉ edges
e : V × V
he : e ∈ Multiset.filter (fun e ↦ e.1 = e.2) edges
h_eq : e.1 = e.2
he' : e ∈ edges
⊢ False
▶ All Messages (0)
Restart File
```

Lean4 vs CVC5: Different Paradigms, Different Powers

Lean4: Interactive Theorem Prover

- Based on **Type Theory**
- Emphasizes *constructive proofs* with verification via `type-checking`
- Supports formalizing pure math (e.g., `mathlib4`) and verified programming
- *Core paradigm:* **“Write proofs by hand, check by kernel”**

CVC5: SMT Solver

- Based on **First-Order Logic**
- Uses *automated decision procedures* for satisfiability
- Excellent for program verification
- *Core paradigm:* **“Decide if formula is satisfiable”**

Lean4 is a proof assistant; CVC5 is a solver.
Both powerful, but for fundamentally different tasks.

Table of Contents

What is the Dollar Game?

- Consider $G = (V, E)$, which is a *finite, connected, loopless, undirected* ***multigraph***
  - A set of unique vertices V = people; E = relationships
  - Each edge vw can appear multiple times in multiset of edges E
- Each vertex has an integer amount:
  - +ve = money, –ve = debt
- Person can ”fire” (lend) or ”borrow” $1 across each adjacent edge
- **Goal:** Redistribute wealth to make all values ≥ 0
- If such a sequence exists, the game is **winnable**

Example: A Simple Dollar Game

- Initial wealth distribution:
  - Alice: $2
  - Bob: –$3
  - Charlie: $4
  - Elise: –$1

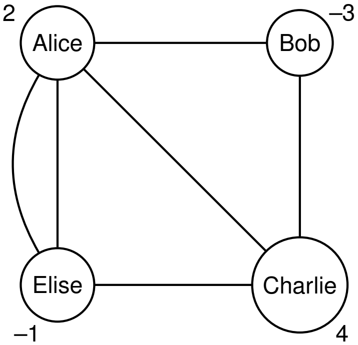


Figure: Situational wealth distribution & relationship setup.

Graphs: Formalizing Structure

- We define graphs as finite, undirected, loopless multigraphs.
- This is the illustration of Lean4 syntax for multigraph object:

```
-- Assume V is a finite type with decidable equality
variable {V : Type} [DecidableEq V] [Fintype V]

structure CFGraph (V : Type) [DecidableEq V] [Fintype V] :=
  (edges : Multiset (V × V))
  (loopless : isLoopless edges = true)
  (undirected: isUndirected edges = true)
```

Graphs: Loopless Property

```
-- Define a set of edges to be loopless only if it doesn't have
    loops
def isLoopless (edges : Multiset (V × V)) : Bool :=
  Multiset.card (edges.filter (λ e => (e.1 = e.2))) = 0

def isLoopless_prop (edges : Multiset (V × V)) : Prop :=
  ∀ v, (v, v) ∉ edges
```

Graphs: Undirected Property

```
-- Define a set of edges to be undirected only if it doesn't have
    both (v, w) and (w, v)
def isUndirected (edges : Multiset (V × V)) : Bool :=
  Multiset.card (edges.filter (λ e => (e.2, e.1) ∈ edges)) = 0

def isUndirected_prop (edges : Multiset (V × V)) : Prop :=
  ∀ v1 v2, (v1, v2) ∈ edges → (v2, v1) ∉ edges
```

Divisors: Formalizing Wealth

- A **divisor** $\mathrm{Div}(G) = \mathbb{Z}V = \{\sum_{v \in V} D(v)v : D(v) \in \mathbb{Z}\}$.
- The **degree** $\deg(D)$ of a divisor D is $\sum_{v \in V} D(v)$.
- For notational convenience, we refer to the number of edges incident on a vertex by **valence**.

```
def CFDiv (V : Type) := V → ℤ
def deg (D : CFDiv V) : ℤ := Σ v, D v

-- Degree (Valence) of a vertex as an integer
def vertex_degree (G : CFGraph V) (v : V) : ℤ :=
  ↑(Multiset.card (G.edges.filter (λ e => e.fst = v ∨ e.snd = v)))
```

Formalizing the Example in Lean4

```
inductive Person : Type
  | A | B | C | E
  deriving DecidableEq
instance : Fintype Person where
  elems := {Person.A, Person.B,
    Person.C, Person.E}
  complete := by {
    intro x
    cases x
    all_goals { simp }
  }
```

Key Elements

- `inductive Person` creates a custom finite set of options
- `DecidableEq` enables equality checking between persons
- `Fintype` instance provides completeness proof
- **Tactic** is a technical term for "strategy" (automatic proof assemblers like `simp`, `intro`, `cases`).

Formalizing the Example in Lean4 (continued...)

```
-- Loopless, undirected graph
def exampleEdges : Multiset (Person × Person) :=
  Multiset.ofList [
    (Person.A, Person.B), (Person.B, Person.C), (Person.C, Person.E)
  ]

theorem loopless_example_edges :
  isLoopless exampleEdges = true := by rfl

-- Graph with a loop
def edgesWithLoop : Multiset (Person × Person) :=
  Multiset.ofList [
    (Person.A, Person.B), (Person.A, Person.A), (Person.B, Person.C)
  ]
theorem loopless_test_edges_with_loop :
    isLoopless edgesWithLoop = false := by rfl
```

Formalizing the Example in Lean4 (continued...)

```
def example_graph :
    CFGraph Person := {
  edges := Multiset.ofList
    [
    (Person.A, Person.B),
    (Person.B, Person.C),
    (Person.A, Person.C),
    (Person.A, Person.E),
    (Person.A, Person.E),
    (Person.E, Person.C)
  ],
  loopless := by rfl,
  undirected := by rfl
}
```

```
def initial_wealth : CFDiv
    Person :=
  fun v => match v with
  | Person.A => 2
  | Person.B => -3
  | Person.C => 4
  | Person.E => -1
```

Key Insight

- Formalization and checking of vertex degrees, edge counts, and symmetry is **non-trivial.**

Firing Move: Lend from a Vertex

- A *firing move* at vertex v, $D \xrightarrow{v} D'$ is such that:

$$D' = D - \mathrm{val}(v) \cdot v + \sum_{vw \in E} w$$

- In Lean4:

```
def firing_move (G : CFGraph V) (D : CFDiv V) (v : V) : CFDiv V :=
  λ w => if w = v then D v - vertex_degree G v
        else D w + num_edges G v w
```

Set Firing

- **Set firing:** apply firing moves to each vertex in subset $S \subseteq V$
- Note that order of firing doesn't matter $\implies$ **abelian property**
- In Lean4:

```
def set_firing (G : CFGraph V) (D : CFDiv V) (S : Finset V) :
    CFDiv V :=
    λ w => D w + finset_sum S (firing_move)
```

Example: Sequence of Firing Moves

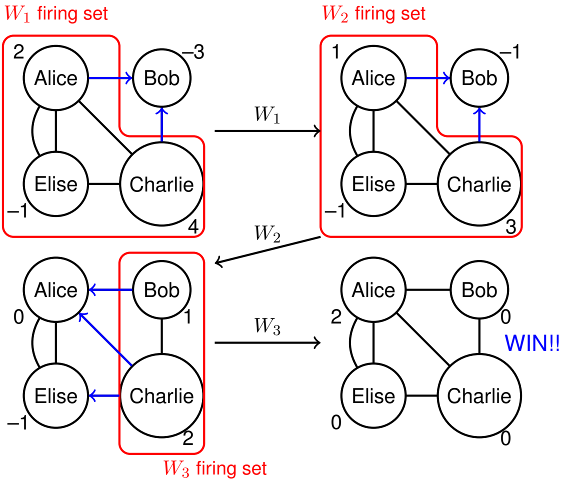


Figure: Application of set-firing moves leading to a win in the case of the divisor mentioned before

Example Walkthrough in Lean4

```
-- Test Charlie lending through an individual firing move
def after_charlie_lends := firing_move example_graph initial_wealth Person.C
theorem charlie_wealth_after_lending : after_charlie_lends Person.C = 1 := by rfl
theorem bob_wealth_after_charlie_lends : after_charlie_lends Person.B = -2 := by rfl

def W₁ : Finset Person := {Person.A, Person.E, Person.C} -- Test set firing W₁ = {A,E,C}
def after_W₁_firing := set_firing example_graph initial_wealth W₁
theorem alice_wealth_after_W₁ : after_W₁_firing Person.A = 1 := by rfl
theorem bob_wealth_after_W₁ : after_W₁_firing Person.B = -1 := by rfl
theorem charlie_wealth_after_W₁ : after_W₁_firing Person.C = 3 := by rfl
theorem elise_wealth_after_W₁ : after_W₁_firing Person.E = -1 := by rfl

def W₂ : Finset Person := W₁ -- Test set firing W₂ = {A,E,C}
def after_W₂_firing := set_firing example_graph after_W₁_firing W₂
theorem alice_wealth_after_W₂ : after_W₂_firing Person.A = 0 := by rfl
theorem bob_wealth_after_W₂ : after_W₂_firing Person.B = 1 := by rfl
theorem charlie_wealth_after_W₂ : after_W₂_firing Person.C = 2 := by rfl
theorem elise_wealth_after_W₂ : after_W₂_firing Person.E = -1 := by rfl

def W₃ : Finset Person := {Person.B, Person.C} -- Test set firing W₃ = {B,C}
def after_W₃_firing := set_firing example_graph after_W₂_firing W₃
theorem alice_wealth_after_W₃ : after_W₃_firing Person.A = 2 := by rfl
theorem bob_wealth_after_W₃ : after_W₃_firing Person.B = 0 := by rfl
theorem charlie_wealth_after_W₃ : after_W₃_firing Person.C = 0 := by rfl
theorem elise_wealth_after_W₃ : after_W₃_firing Person.E = 0 := by rfl

-- Test degree of divisors
theorem initial_wealth_degree : deg initial_wealth = 2 := by rfl
theorem after_W₁_degree : deg after_W₁_firing = 2 := by rfl
theorem after_W₂_degree : deg after_W₂_firing = 2 := by rfl
theorem after_W₃_degree : deg after_W₃_firing = 2 := by rfl
```

Linear Equivalence of Divisors

Key Concepts

- Linear equivalence between divisors $D \sim D'$ is defined to exist if we can obtain D' from D by a sequence of firing moves.
- Utilize group structure to capture all possible outcomes

```
instance : AddGroup (CFDiv V) := Pi.addGroup

def firing_vector (G : CFGraph V) (v : V) : CFDiv V :=
  λ w => if w = v then -vertex_degree G v else num_edges G v w

def principal_divisors (G : CFGraph V) :
    AddSubgroup (CFDiv V) :=
  AddSubgroup.closure (Set.range (firing_vector G))

-- Define linear equivalence of divisors
def linear_equiv (G : CFGraph V) (D D' : CFDiv V) : Prop :=
  D' - D ∈ principal_divisors G
```

Linear Equivalence is an Equivalence Relation

```
-- [Proven] Proposition: Linear equivalence is an
      equivalence relation on Div(G)
theorem linear_equiv_is_equivalence (G : CFGraph V) :
 Equivalence (linear_equiv G) := by
 apply Equivalence.mk
 -- Reflexivity
 · intro D
  unfold linear_equiv
  have h_zero : D - D = 0 := by simp
  rw [h_zero]
  exact AddSubgroup.zero_mem _

 -- Symmetry
 · intros D D' h
  unfold linear_equiv at *
  have h_symm : D - D' = -(D' - D) := by
    simp [sub_eq_add_neg, neg_sub]
  rw [h_symm]
  exact AddSubgroup.neg_mem _ h

 -- Transitivity
 · intros D D' D'' h1 h2
  unfold linear_equiv at *
  have h_trans : D'' - D = (D'' - D') + (D' - D) := by simp
  rw [h_trans]
  exact AddSubgroup.add_mem _ h2 h1
```

Key Tactics in Lean4

- `apply` - Sets proof structure
- `intro` - Brings variables into context
- `have` - Establish hypothesis
- `unfold` - Expands definitions
- `rw` - Rewrites expressions
- `simp` - Auto-simplification
- `exact` - Invokes precise match check

Objective: Effectiveness and Winnability

- A divisor D is **effective** if $D(v) \geq 0$ for all $v \in V$.
- **Objective** of the dollar game:
  - *Is a given divisor linearly equivalent to an effective divisor?*
- $\mathrm{Div}_+(G)$ is the set of effective divisors on G.
- D is **winnable** if $D \sim E$, where E is an effective divisor.

```
def effective (D : CFDiv V) : Prop := ∀ v : V, D v ≥ 0

def Div_plus (G : CFGraph V) : Set (CFDiv V) :=
  {D : CFDiv V | effective D}

def winnable (G : CFGraph V) (D : CFDiv V) : Prop :=
    ∃ D’ ∈ Div_plus G, linear_equiv G D D’
```

Laplacian Matrix & Firing Scripts

- **Firing scripts** encode multiple *firing moves* in one vector
- A *firing script* is a function $\sigma : V \to \mathbb{Z}$, which denotes the number of times each vertex v lends (fires) if $\sigma(v) > 0$.
- Laplacian matrix L maps firing scripts to divisor-transformations

```
def firing_script (V : Type) := V → ℤ

def laplacian_matrix (G : CFGraph V) : Matrix V V ℤ :=
  λ i j => if i = j then vertex_degree G i else - (num_edges G i j)

def apply_laplacian (G : CFGraph V) (σ : firing_script V) (D: CFDiv
    V) : CFDiv V :=
  fun v => (D v) - (laplacian_matrix G).mulVec σ v
```

Example (continued...): Laplacian

	V_{Alice}	V_{Bob}	$V_{Charlie}$	V_{Elise}
V_{Alice}	4	-1	-1	-2
V_{Bob}	-1	2	-1	0
$V_{Charlie}$	-1	-1	3	-1
V_{Elise}	-2	0	-1	3

$\vec{\sigma} = \begin{bmatrix} 0 \\ -1 \\ 1 \\ 0 \end{bmatrix}$. Thus,

$$D' = \begin{bmatrix} 2 \\ -3 \\ 4 \\ -1 \end{bmatrix} - \begin{bmatrix} 4 & -1 & -1 & -2 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -2 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 4 \\ -1 \end{bmatrix} - \begin{bmatrix} 0 \\ -3 \\ 4 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Duality: Lending vs Borrowing through Firing Scripts

Two Perspectives, One Result

Starting from the initial divisor: $D = \left[2, -3, 4, -1\right]^T$ both of the following lead to the same final configuration: $D' = \left[2, 0, 0, 0\right]^T$

Lending (Set-Firing):

- A, C, E fire (x 2)
- Then B, C fire

$$\sigma = \left[2, 1, 3, 2\right]^T$$

Borrowing (Negated Script):

- B borrows (x 2)
- Then A, E borrow

$$\sigma = \left[-1, -2, 0, -1\right]^T$$

Example (continued...): Laplacian in Lean4

```
-- Test Laplacian matrix values and symmetricity
def example_laplacian := laplacian_matrix example_graph
theorem laplacian_diagonal_A : example_laplacian Person.A Person.A = 4 := by rfl
theorem laplacian_diagonal_B : example_laplacian Person.B Person.B = 2 := by rfl
theorem laplacian_diagonal_C : example_laplacian Person.C Person.C = 3 := by rfl
theorem laplacian_diagonal_E : example_laplacian Person.E Person.E = 3 := by rfl
theorem laplacian_off_diagonal_AB : example_laplacian Person.A Person.B = -1 := by rfl
theorem laplacian_off_diagonal_AC : example_laplacian Person.A Person.C = -1 := by rfl
theorem laplacian_off_diagonal_AE : example_laplacian Person.A Person.E = -2 := by rfl
theorem laplacian_off_diagonal_BC : example_laplacian Person.B Person.C = -1 := by rfl
theorem laplacian_off_diagonal_BE : example_laplacian Person.B Person.E = 0 := by rfl
theorem laplacian_off_diagonal_CE : example_laplacian Person.C Person.E = -1 := by rfl

theorem check_example_laplacian_symmetry : Matrix.IsSymm example_laplacian := by {
 apply Matrix.IsSymm.ext
 intros i j
 cases i <;> cases j
 all_goals {
  rfl
 }
}

-- Test script firing through laplacians
def firing_script_example : firing_script Person := fun v => match v with
 | Person.A => 0
 | Person.B => -1
 | Person.C => 1
 | Person.E => 0
def res_div_post_lap_based_script_firing := apply_laplacian example_graph firing_script_example initial_wealth
theorem lap_based_script_firing_preserves_degree : deg res_div_post_lap_based_script_firing = 2 := by rfl
```

Table of Contents

Greedy Algorithm

Approach

- Choose debt vertices and make borrowing moves until either:
  - Everyone is debt-free (success!)
  - Every vertex has borrowed at least once (failure)
- Always terminates & Produces a unique firing script

Examples

- We implemented this in Python, producing the following output:

```
The game is winnable with the greedy algorithm.
Firing Script: {'A': -1, 'B': -2, 'C': 0, 'E': -1}
Resulting Divisor: {'A': 2, 'B': 0, 'C': 0, 'E': 0}
```

Preliminaries for Winnability

Debt Clustering (q-Reduced)

- Let $q \in V$ & $\widetilde{V} := V \setminus \{q\}$. A divisor D is called q**-reduced** if the vertex labeled as q volunteers to carry all the debt in such a way that no further "vaccum-pulling" of debt towards q is possible.
- A linear-equivalence class can be identified by a unique $D_q \implies$ (D is **winnable** $\iff D_q(q) \geq 0$).
- $D \xrightarrow{S \subseteq \widetilde{V}} D'$ is a **legal set-firing** if $D'(v) \geq 0$ for all $v \in S$
- **Property:** after $D \xrightarrow{S \subseteq \widetilde{V}} D'$, some dollars move towards q.

Dhar's Algorithm

The "Burning Process"

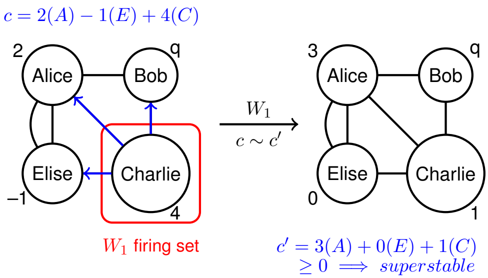
1. Choose a source vertex q
2. Start a "fire" at q
3. A vertex burns if it has fewer "firefighters" than burning edges
4. Continue until no new vertices burn [a]

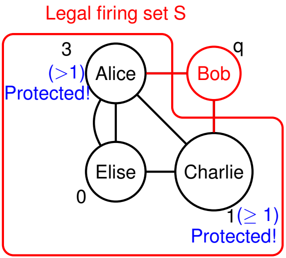
[a] Fun Note: *No need to worry; firefighters are rescued by an underground tunnel built by Amherst College.*

Outcomes

- Unburnt vertices = legal firing set
- To find q-reduced divisors (Don't need to go through $2^{|\widetilde{V}|} - 1$ subsets, termination guaranteed due to lexicographic ordering)
- At end, after full-burn, if $D_q(q) < 0$, then D is unwinnable!

Application to Our Example





Algorithm Terminates here & outputs the Legal firing set S

$c = 3(A) + 0(E) + 1(C) \geq 0$

Efficient Winnability Determination (EWD)

EWD Algorithm Outline

1. Choose a source vertex $q \in V$; let $\widetilde{V} := V \setminus \{q\}$.
2. Push all debt to q by firing from q and redistributing to $\widetilde{V}$.
3. Repeat Dhar's Algorithm:
   - While the returned set $S \neq \emptyset$, fire S.
4. Compute $D_q(q) := \deg(D) - \deg(c)$.
5. **Return TRUE** if $D_q(q) \geq 0$, else **FALSE**.

Examples

Our Python Implementation of EWD Algorithm:

```
The game is winnable using Dhar's algorithm.
Legal firing set: {'A', 'C', 'E'}
```

Our Contribution: Optimization on EWD Algorithm

Reverse-Distance Debt Concentration

- To optimize Step 2 of EWD algorithm, we **concentrate all debt at** q by systematically moving debt inward from the graph's periphery.
- **Key idea:** Borrow *furthest from* q first $\Rightarrow$ push debt toward q.

How It Works

1. Perform a **BFS** from q to compute the distance of each $v \in \widetilde{V}$.
2. Sort vertices in decreasing distance from q.
3. For each vertex v with $c(v) < 0$, perform a **borrowing operation** to "shift" debt closer to q.

Why This Works

- Once a vertex is cleared of debt, it stays non-negative since no future borrowing affects it.
- Fewer **Dhar iterations** needed than brute-force simulation.

Example Walkthrough for BFS-Optimized Dhar's

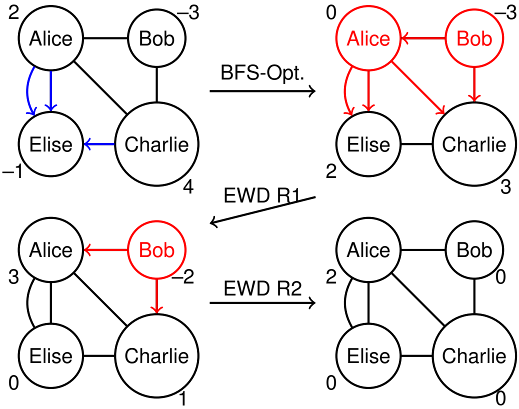


Figure: Modified EWD Algorithm Run: Debt is moved inward toward q (Bob) using BFS-based order $\{$E, $\{$A,C$\}\}$.

Table of Contents

Preliminaries for RRG: Orientations

- *indegree* = #edges directed towards v
- *outdegree* = #edges directed away from v
- $D(\mathcal{O}) = \sum_{v \in V} (\text{indeg}_{\mathcal{O}}(v) - 1) \cdot v$
- **canonical divisor** $K := D(\mathcal{O}) + D(\overline{\mathcal{O}})$, where $\overline{\mathcal{O}}$ is reverse orientation.
- K only depends on graph G since $K(v) = \deg_G(v) - 2$.
- The **genus** $g = |E| - |V| + 1$.

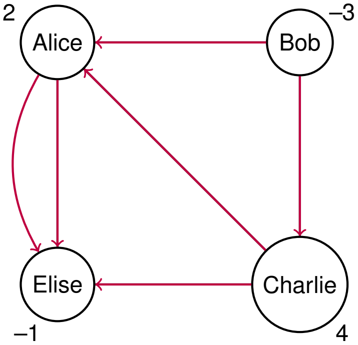


Figure: Orientation example with **source** Bob and **sink** $Elise$.

Dhar's Algorithm (Revisited)

Orientation-Based Perspective

- *Recall:* Dhar's algorithm determines superstability via burning.
- **New Viewpoint:** Every time a fire spreads from u to v, we **orient** the edge uv as $u \to v$, inducing an acyclic $\mathcal{O}$ with unique source q.

Modified Dhar's Algorithm (Sketch)

Given a non-negative configuration c and source q:

1. Initialize:
   - Burning set $B \leftarrow \{q\}$, legal firing set $S \leftarrow \widetilde{V}$, orientation $\mathcal{O} \leftarrow \emptyset$
2. While $B \neq V$:
   - For each $v \in S$, count edges E_v from v to B
   - If $|E_v| > c(v)$: Add v to B, remove from S; Orient all $e \in E_v$ toward v
3. If no new vertex burns, return $(S, \mathcal{O})$
4. If all burn: return $(\emptyset, \mathcal{O})$

Why Acyclic?

Problem: Non-Injectivity of the Orientation → Divisor Map

- Multiple orientations can lead to the **same divisor**.
- Example: O_a and O_b differ by a *cycle reversal*, but $\mathrm{D}(O_a) = \mathrm{D}(O_b)$.
- This makes the map from orientations to divisors **non-injective**.

Fix: Restrict to Acyclic Orientations

- In an acyclic orientation, **no directed cycle exists**.
- Any cycle reversal would introduce a cycle $\Rightarrow$ disallowed.
- Ensures the map $\mathcal{O} \mapsto D(\mathcal{O})$ is injective within the acyclic subset.

Takeaway

Acyclic orientations give a **well-defined representation** of divisors — a key step for bijections!

Rank: Measuring Winnability

Motivating Question

Are some games more winnable than others?
The **rank function** helps quantify this by asking: *How many chips can be removed from a divisor before it becomes unwinnable?*

Definition of Rank $r(D)$

Given a divisor $D \in \mathrm{Div}(G)$:

1. $r(D) = -1 \iff D$ is unwinnable
2. $r(D) \geq k \iff D - E$ winnable $\forall$ effective E s.t. $\deg(E) = k$
3. $r(D) \leq k \iff \exists E$ with $\deg(E) = k + 1$ s.t. $D - E$ is unwinnable

Computational Challenge

- Determining $r(D)$ is **NP-hard** (and runtime grows exponentially)

The Riemann-Roch Theorem for Graphs

Theorem (Baker–Norine, 2007)

Let D be a divisor on a loopless, undirected graph G. Then:

$$r(D) - r(K - D) = \deg(D) - g + 1$$

A Bridge to Algebraic Geometry

- **Graph Divisors:** Integer chip counts on vertices $\leftrightarrow$ Formal point sums on Riemann surfaces
- **Rank $r(D)$:** Max chips removable while staying winnable $\leftrightarrow$ Dimension of meromorphic function spaces (analytic with some discrete poles)
- **Genus $g = |E| - |V| + 1$:** Cycle complexity $\leftrightarrow$ Number of "handles" (or "holes") on a surface

Proof Strategy for Riemann-Roch

1. Characterize maximal unwinnable divisors using acyclic orientations
2. Show divisor $D(O)$ from orientation has degree $g-1$
3. Establish bijection between acyclic orientations and superstable configurations
4. Prove inequality: $\deg(D) - g + 1 \leq r(D) - r(K-D)$
5. Apply the same reasoning to $K-D$ to get opposite inequality
6. Conclude that equality must hold

Application to Determination of Winnability

Clifford's Theorem

If D is a divisor with $r(D) \geq 0$ and $r(K - D) \geq 0$, then $r(D) \leq \frac{\deg(D)}{2}$

Rank-Degree Relationship

- If $\deg(D) < 0$, then $r(D) = -1$
- If $0 \leq \deg(D) \leq 2g - 2$, then $r(D) \leq \frac{\deg(D)}{2}$
- If $\deg(D) > 2g - 2$, then $r(D) = \deg(D) - g$

Table of Contents

Current State of Formalization

Completed Components

- Core graph and divisor definitions
- Firing moves and linear equivalence
- Q-reduced divisors
- Configurations and orientations
- Main Riemann-Roch theorem
- Key corollaries (Clifford, rank characterization)

Examples

Modular Structure

- `Basic.lean`: Core structures
- `CFGraphExample.lean`
- `Config.lean`:
- `Orientation.lean`
- `Rank.lean`: Rank properties
- `Helpers.lean`: Propositional Helpers
- `RRGHelpers.lean`: Theorem helpers
- `RiemannRochForGraphs.lean`: Main theorem

Lean4 Implementation Highlights

Proof Techniques in Lean4

- Encoded key structures: divisors, configurations, and orientations.
- Used `rcases`, `linarith`, and modular lemma chaining.

Note

- Avoid (NP-Hard) case-heavy analysis or explicit rank computation.
- Instead rely on abstraction and structural reasoning.
- Introducing Axioms as placeholders for setting structure.

Challenges in Formalization (I)

From Intuition to Code

- Translating *high-level* math into *low-level* Lean4 constructs
- Classical math skips steps; Lean requires full construction
- *Example:* WLOG needs to be formalized with conditionals

Managing Complexity

- Proof split into modular lemmas: firing, rank, orientations, etc.
- Ensuring consistency in hypotheses (e.g., graph connectivity)
- Balancing generality vs. usability of lemmas

Challenges in Formalization (II)

Tactics & Proof Automation

- Lean4 tactics automate steps, not strategy
- Human guidance essential in complex inequalities
- We wrote custom tactics for recurring proof patterns

Termination & Computability

- Used non-executable, symbolic rank definitions
- Explicitly proved termination of algorithms like Dhar's using well-founded measures

Lean4 as a Developing Ecosystem

- Some standard graph theory not yet in Mathlib4
- Developed custom graph framework (`CFGraph`) and contributed reusable proofs

Example: Handshaking Lemma in Lean4

Handshaking Theorem

In any **loopless multigraph** G: $\sum_{v \in V} \text{val}(v) = 2 \cdot |E|$ That is, the total sum of vertex degrees equals twice the number of edges.

Lean4 Formal Statement

```
theorem helper_sum_vertex_degrees (G : CFGraph V) :
  Σ v, vertex_degree G v = 2 * ↑(Multiset.card G.edges)
```

Proof Sketch (Steps Lean Verifies)

- Expand `vertex_degree` as count of incident edges
- Use `Nat.cast_sum` to move cast outside
- Swap summation over vertices to summing over edges
- Each edge contributes exactly twice (to both endpoints)
- Final step: $2 \cdot$ number of edges

Handshaking Lemma in Lean4

```
theorem helper_sum_vertex_degrees (G : CFGraph V) :
  Σ v, vertex_degree G v = 2 * ↑(Multiset.card G.edges) := by
 -- Unfold vertex degree definition
 unfold vertex_degree
 calc
  -- Start with the definition of sum of vertex degrees
  Σ v, vertex_degree G v
  -- Express vertex degree as Nat cast of card filter
  = Σ v, ↑(Multiset.card (G.edges.filter (λ e => e.1 = v ∨ e.2 = v))) := by rfl
  -- Pull the Nat cast outside the sum over vertices
  _ = ↑(Σ v, Multiset.card (G.edges.filter (λ e => e.1 = v ∨ e.2 = v))) := by rw [Nat.cast_sum]
  -- Apply the sum swapping lemma (Nat version)
  _ = ↑(Multiset.sum (G.edges.map (λ e => (Finset.univ.filter (λ v => e.1 = v ∨ e.2 = v)).card))) := by
   rw [sum_filter_eq_map_inc_nat G]
  -- Apply the lemma relating sum of incidences to 2 * |E| (Nat version)
  _ = ↑(2 * (Multiset.card G.edges)) := by
   rw [map_inc_eq_map_two_nat G]
  -- Pull the constant 2 outside the Nat cast
  _ = 2 * ↑(Multiset.card G.edges) := by
   rw [Nat.cast_mul, Nat.cast_ofNat] -- Use Nat.cast_ofNat for Nat.cast 2
```

Helper: Zero Cardinality & Loopless Graphs

```
/-- [Proved] Helper lemma: Every divisor can be decomposed into a principal divisor and an effective divisor -/
lemma eq_nil_of_card_eq_zero {α : Type _} {m : Multiset α}
   (h : Multiset.card m = 0) : m = ∅ := by
  induction m using Multiset.induction_on with
  | empty => rfl
  | cons a s ih =>
   simp only [Multiset.card_cons] at h
   -- card s + 1 = 0 is impossible for natural numbers
   have : ¬(Multiset.card s + 1 = 0) := Nat.succ_ne_zero (Multiset.card s)
   contradiction

/-- [Proven] Helper lemma: In a loopless graph, each edge has distinct endpoints -/
lemma edge_endpoints_distinct (G : CFGraph V) (e : V × V) (he : e ∈ G.edges) :
   e.1 ≠ e.2 := by
  by_contra eq_endpoints
  have : isLoopless G.edges = true := G.loopless
  unfold isLoopless at this
  have zero_loops : Multiset.card (G.edges.filter (λ e' => e'.1 = e'.2)) = 0 := by
   simp only [decide_eq_true_eq] at this
   exact this
  have e_loop_mem : e ∈ Multiset.filter (λ e' => e'.1 = e'.2) G.edges := by
   simp [he, eq_endpoints]
  have positive : 0 < Multiset.card (G.edges.filter (λ e' => e'.1 = e'.2)) := by
   exact Multiset.card_pos_iff_exists_mem.mpr ⟨e, e_loop_mem⟩
  have : Multiset.filter (fun e' => e'.1 = e'.2) G.edges = ∅ := eq_nil_of_card_eq_zero zero_loops
  rw [this] at e_loop_mem
  cases e_loop_mem
```

Helper: Each Edge Has Exactly Two Incident Vertices

```
/-- [Proven] Helper lemma: Each edge is incident to exactly two vertices -/
lemma edge_incident_vertices_count (G : CFGraph V) (e : V × V) (he : e ∈ G.edges) :
  (Finset.univ.filter (λ v => e.1 = v ∨ e.2 = v)).card = 2 := by
 rw [Finset.card_eq_two]
 exists e.1
 exists e.2
 constructor
 · exact edge_endpoints_distinct G e he
 · ext v
  simp only [Finset.mem_filter, Finset.mem_univ, true_and,
       Finset.mem_insert, Finset.mem_singleton]
  -- The proof here can be simplified using Iff.intro and cases
  apply Iff.intro
  · intro h_mem_filter -- Goal: v ∈ {e.1, e.2}
   cases h_mem_filter with
   | inl h1 => exact Or.inl (Eq.symm h1)
   | inr h2 => exact Or.inr (Eq.symm h2)
  · intro h_mem_set -- Goal: e.1 = v ∨ e.2 = v
   cases h_mem_set with
   | inl h1 => exact Or.inl (Eq.symm h1)
   | inr h2 => exact Or.inr (Eq.symm h2)
```

Helper: Swapping Vertex & Edge Summations (Part 1)

```
/-- [Proven] Helper lemma: Swapping sum order for incidence checking (Nat version). -/
lemma sum_filter_eq_map_inc_nat (G : CFGraph V) :
 Σ v : V, Multiset.card (G.edges.filter (λ e => e.fst = v ∨ e.snd = v))
  = Multiset.sum (G.edges.map (λ e => (Finset.univ.filter (λ v => e.1 = v ∨ e.2 = v)).card)) := by
 -- Define P and g using Prop for clarity in the proof - Available throughout
 let P : V → V × V → Prop := fun v e => e.fst = v ∨ e.snd = v
 let g : V × V → ℕ := fun e => (Finset.univ.filter (P · e)).card

 -- Rewrite the goal using P and g for proof readability
 suffices goal_rewritten : Σ v : V, Multiset.card (G.edges.filter (P v)) = Multiset.sum (G.edges.map g) by
  exact goal_rewritten -- The goal is now exactly the statement `goal_rewritten`

 -- Prove the rewritten goal by induction on the multiset G.edges
 induction G.edges using Multiset.induction_on with
 -- Base case: s = ∅
 | empty =>
  simp only [Multiset.filter_zero, Multiset.card_zero, Finset.sum_const_zero,
        Multiset.map_zero, Multiset.sum_zero] -- Use _zero lemmas
 -- Inductive step: Assume holds for s, prove for a :: s
 | cons a s ih =>
  -- Rewrite RHS: sum(map(g, a::s)) = g a + sum(map(g, s))
  rw [Multiset.map_cons, Multiset.sum_cons]

  -- Rewrite LHS: Σ v, card(filter(P v, a::s))
  -- card(filter) -> countP
  simp_rw [← Multiset.countP_eq_card_filter]

  -- Use countP_cons _ a s inside the sum. Assumes it simplifies
  -- to the form Σ v, (countP (P v) s + ite (P v a) 1 0)
  simp only [Multiset.countP_cons]
```

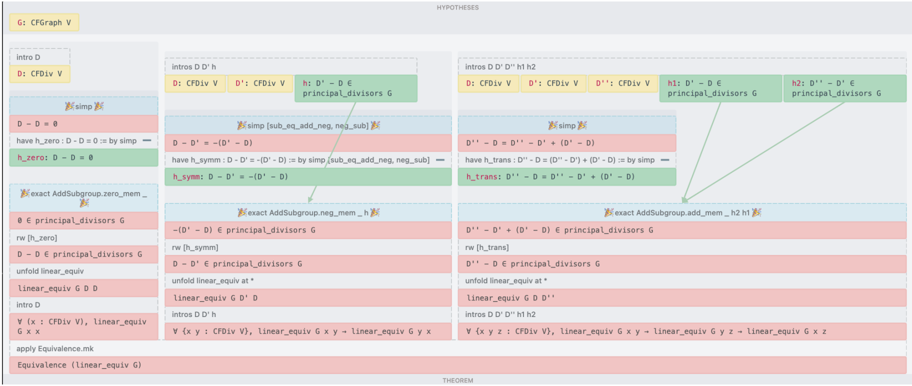
Helper: Swapping Vertex & Edge Summations (Part 2)

```
-- Distribute the sum
rw [Finset.sum_add_distrib]

-- Simplify the second sum (Σ v, ite (P v a) 1 0) to g a
have h_sum_ite_eq_card : Σ v : V, ite (P v a) 1 0 = g a := by
  -- Use Finset.card_filter: (s.filter p).card = Σ x ∈ s, if p x then 1 else 0
  rw [← Finset.card_filter]
  -- Should hold by definition of sum over Fintype and definition of g
rw [h_sum_ite_eq_card] -- Goal: Σ v, countP (P v) s + g a = g a + sum (map g s)

-- Rewrite the first sum's countP back to card(filter)
simp_rw [Multiset.countP_eq_card_filter] -- Goal: Σ v, card(filter (P v) s) + g a = g a + ...

-- Apply IH and finish
rw [add_comm] -- Goal: g a + Σ v, card(filter (P v) s) = g a + ...
rw [ih] -- Apply inductive hypothesis
```

Helper: Each Edge Contributes 2 to Degree Sum

```
/-- [Proven] Helper lemma: Summing mapped incidence counts equals summing constant 2 (Nat version). -/
lemma map_inc_eq_map_two_nat (G : CFGraph V) :
 Multiset.sum (G.edges.map (λ e => (Finset.univ.filter (λ v => e.1 = v ∨ e.2 = v)).card))
  = 2 * (Multiset.card G.edges) := by
 -- Define the function being mapped
 let f : V × V → ℕ := λ e => (Finset.univ.filter (λ v => e.1 = v ∨ e.2 = v)).card
 -- Define the constant function 2
 let g (_ : V × V) : ℕ := 2
 -- Show f equals g for all edges in G.edges
 have h_congr : ∀ e ∈ G.edges, f e = g e := by
  intro e he
  simp [f, g]
  exact edge_incident_vertices_count G e he
 -- Apply congruence to the map function itself first using map_congr with rfl
 rw [Multiset.map_congr rfl h_congr] -- Use map_congr with rfl
 -- Apply rewrites step-by-step
 rw [Multiset.map_const', Multiset.sum_replicate, Nat.nsmul_eq_mul, Nat.mul_comm]
```

Augmenting Tools



Visualization Tools

- PaperProof VSCode extension (easily integratable)
- Intuitive & Interactive visualization of Lean4 proofs
- Bridge between formal systems and intuitive understanding

Table of Contents

Reflections & Lessons Learned

What Formalization Taught Us

- Formalization enforces **clarity, precision**, and **modularity**.
- Every lemma corresponds to a concrete mathematical insight.
- Lean4 acts as a dialogue partner—pushing for explicit structure and sometimes revealing better proof paths.

Combinatorics Meets Computation

- Lean helped distinguish **canonical** ideas and **proof tricks**.
- Led to improved understanding of the theorem's anatomy and reusable strategies.

Future Directions

Mathematical Extensions

- Formalize **Brill–Noether theory** on graphs.
- Extend to **tropical curves** and **metric graphs**.
- Explore **chip-firing as network flows**, e.g., flow/cut algorithms.

Machine-Assisted Proving

- Exciting potential in **AI-assisted proving** (e.g., `TheoremLlama`, `MA-LoT`): turning proof sketches into formal tactics.
- Build graph-theoretic foundations further in **Mathlib4**

Vision: *Human Insight + Machine Precision Can Scale!*

Current State of Simulation Tools

- **Chip-Firing Tool** (Williams College, 2021)
  - Built as a final project for Math 334 (Graph Theory) under the guidance of *Prof. Ralph Morrison*.
  - Interactive graph drawing + chip-firing moves
  - Great educational resource for exploring the dollar game
- **Current Limitations:**
  - Non-standardized support for custom rule sets
  - Absence of Unit Testing and Code Access
  - Limited observability during the chip-firing process
- This inspired us to design a formal + programmable system for deeper experimentation to the benefit of researchers and educators.

`chipfiring`: Our Ongoing Python Work

Goals

- Build a unified simulation + algorithm toolkit for chip-firing games
- Bridge combinatorics, algebra, and computation in one library

Key Features

- Custom multigraph construction with labeled vertices
- Structural Support for Mathematical Objects
- Algorithmic Execution and Observability Support
- Extensively documented with type hints + PyPI + test suite

Documentation: `https://chipfiring.readthedocs.io`
Install: `pip install chipfiring`

Acknowledgements

- Grateful to my grandparents, parents, and sister for their unwavering support and belief in me.
- Deep thanks to **Prof. Nathan Pflueger** for exceptional mentorship, and to **Prof. Daniel Velleman** for Lean4 guidance.
- Appreciation to **Profs. Rasheed, Ching, Benedetto, Horton, Leise (late), Daniels, Kraisler, Elliott** for teaching, advising, encouragement, and cricket.
- Thankful for opportunities within and beyond Amherst College!
- Thanks to the **Lean4/Mathlib4 developers** for inspiring the formal methods in this thesis.
- Thanks in advance to the readers of my thesis work, especially **Prof. David Zureick-Brown** and **Prof. Joseph Palmer**!
- And to everyone who helped in small or big ways—thank you!

Thank you for being a wonderful audience!

Let's chat if you're curious about anything!

I'm all ears for questions, feedback, and collaborations.

A detailed list of references, and additional content details can be found in the thesis write-up.

Appendix: Validity of the Greedy Algorithm

Case 1: When a Solution Exists

- Suppose $D \sim D' \geq 0$ via some script σ (a firing sequence).
- Shift σ so that $\sigma \leq 0$ and some $\sigma(v) = 0$ (untouched vertices).
- The greedy strategy:
  - Borrow only from vertices with debt & Update $\sigma(u) \mapsto \sigma(u) + 1$.
  - Stop when $\sigma = 0$ (i.e., configuration matches D').
- Because σ increases by 1 each time and is bounded (term.)

Case 2: When No Solution Exists

- Any infinite borrowing sequence $\{D_i\}$ must repeat states.
- All $D_i(v)$ are bounded by $\max(D(v), \mathrm{val}(v) - 1)$.
- Since $\deg(D_i)$ stays fixed, the configuration space is finite.
- Hence: $\exists j < k$ such that $D_j = D_k$. σ satisfies $\mathrm{div}(\sigma) = 0$.
- By connectedness of G, ker(L) is only constant scripts: $\Rightarrow \sigma(v) = c$ for all $v \Rightarrow$ every vertex was borrowed from

Appendix: Preliminaries for Winnability

Debt Clustering (q-Reduced)

- Let $q \in V$. A divisor $D \in Div(G)$ is called q**-reduced** if the following conditions hold:
  1. $D(v) \geq 0$ for all $v \in V \setminus \{q\}$.
  2. For every nonempty subset $S \subseteq V \setminus \{q\}$, there exists a vertex $v \in S$ such that $D(v) < \mathrm{outdeg}_S(v)$, where $\mathrm{outdeg}_S(v)$ denotes the number of edges vw such that $w \notin S$.
- A linear-equivalence class can be identified by a unique D_q
- $\implies$ (D is **winnable** $\iff$ $D_q(q) \geq 0$).
- $D \xrightarrow{S \subseteq \widetilde{V}} D'$ is a **legal set-firing** if $D'(v) \geq 0$ for all $v \in S$

Appendix: Winnability Ordering

Motivation

Compare *"Winnability"* of two Divisors.

Ordering of Divisors

- For a spanning tree (T, q) of $G(V, E)$ rooted at a vertex q, let $(v_1 = q), v_2, \ldots, v_n$ be a tree ordering of the vertices, where:
  - T is a connected, cycle-free subgraph of G with V vertices and $n - 1$ edges,
  - if v_i lies on the unique path from q to v_j in T, then $i < j$.

  We say that $D' \prec D$ if either:

  1. $\deg(D') < \deg(D)$, or
  2. $\deg(D') = \deg(D)$ and i is smallest index s.t. $D'(v_i) > D(v_i)$.

- **Property:** after $D \xrightarrow{S \subseteq \widetilde{V}} D'$, some dollars move towards q, $D' \prec D$.

Appendix: More on Configurations

Configurations & Key Properties

- Fix a vertex $q \in V$ and define $\widetilde{V} := V \setminus \{q\}$.
- A **configuration** c is an element of Config$(G, q) = \mathbb{Z}\widetilde{V} \subseteq \mathrm{Div}(G)$.
  $\Rightarrow c$ omits tracking chips at q.
- **Non-negativity:** $c \geq 0$ if $c(v) \geq 0$ for all $v \in \widetilde{V}$.
- **Degree:** $\deg(c) = \sum_{v \in \widetilde{V}} c(v)$.
- **Linear equivalence:** $c \sim c'$ if c' can be reached from c via lending/borrowing operations.
- **Note:** $\deg(c)$ may differ from $\deg(c')$ (chip count at q is ignored).
- $c \xrightarrow{S \subseteq \widetilde{V}} c'$ is a **legal set-firing** if $c'(v) \geq 0$ for all $v \in S$.
- c is **superstable** $\iff$ $c \geq 0$ and no legal nonempty set-firing exists.

Appendix: Some Consequential Bridges

Acyclic Orientations $\leftrightarrow$ Maximal Superstables

Fix $q \in V$. Then, the map $\mathcal{O} \mapsto c(\mathcal{O})$ defines a **bijection** between:

- Acyclic orientations of G with q as the unique source, and
- **Maximal superstable configurations** $c \in \text{Config}(G, q)$
- **Maximal unwinnable q-reduced divisors**

This correspondence arises naturally from modified Dhar's.

Definitions That Connect the Dots

- **Maximal Superstable Configuration:** A superstable c such that for any $c' \geq c$, if c' is also superstable, then $c = c'$.
- **Maximal Unwinnable Divisor:** An unwinnable divisor D such that $D + v$ is winnable for every vertex $v \in V$.

Appendix: Indegree Determines Acyclic Orientation

Lemma (Indegree Determination Lemma)

*Let $\mathcal{O}, \mathcal{O}'$ be two **acyclic orientations** of a graph G. If for all $v \in V$:*

$$\text{indeg}_{\mathcal{O}}(v) = \text{indeg}_{\mathcal{O}'}(v) \implies \mathcal{O} = \mathcal{O}'$$

Proof Sketch

- In any acyclic orientation $\mathcal{O}$, there must exist at least one **source vertex** ($\text{indeg}(v) = 0$). Otherwise, the reverse orientation $\overline{\mathcal{O}}$ has no sink $\Rightarrow$ contains a cycle $\Rightarrow$ contradiction.
- Remove all source vertices V_1 and their incident edges to get a smaller acyclic orientation $\mathcal{O}_1$ on subgraph G_1.
- Repeat this process: remove sources layer by layer to get a vertex partition $(V_1, V_2, \ldots, V_k)$.
- This sequence is uniquely determined by the indegree function.
- Therefore, the indegree sequence determines the orientation.

Appendix: Acyclic O and Superstables Bijection

Lemma: Fix $q \in V$. Then the map: $\mathcal{O} \mapsto c(\mathcal{O})$ defines a **bijection** between:

- Acyclic orientations of G with q as the unique source, and
- Maximal superstable configurations in Config(G, q)

Proof Sketch

Injectivity:

- If $c(\mathcal{O}) = c(\mathcal{O}')$, then $\mathrm{indeg}_{\mathcal{O}} = \mathrm{indeg}_{\mathcal{O}'}$, which implies $\mathcal{O} = \mathcal{O}'$ because of indegree determination lemma.

Surjectivity:

- Let c be a maximal superstable configuration.
- Run modified Dhar's algorithm from q: terminates with $S = \emptyset$.
- Produces acyclic orientation $\mathcal{O}$ with q as the unique source.
- No other source or directed cycle (violation of superstability).

Appendix: Why is $\deg(c) \leq g$ for Superstables?

Key Idea

Every superstable configuration c is bounded in degree by a corresponding maximal one: $\deg(c) \leq \deg(c(\mathcal{O})) = g$

- Let $\mathcal{O}$ be an acyclic orientation with q as the unique source.
- Define: $c(\mathcal{O})(v) = \mathrm{indeg}_{\mathcal{O}}(v) - 1 \quad$ for $v \in V \setminus \{q\}$
- Then:

$$\deg(c(\mathcal{O})) = \sum_{v \neq q} (\mathrm{indeg}(v) - 1) = |E| - (|V| - 1) = g$$

- Since superstabilization reduces or maintains chip count:

$$\deg(c) \leq g$$

- Equality holds iff $c = c(\mathcal{O})$ maximal superstable.

Takeaway

Maximal superstables uniquely hit the genus g; others fall short.

Appendix: Maximal Superstables and Maximal Unwinnables

Let c be a superstable configuration and D a divisor. Then:

1. c is **maximal superstable** $\iff \deg(c) = g$
2. D is **maximal unwinnable** $\iff$ its q-reduced form is $c - q$, with c maximal superstable

Proof Sketch

- Every superstable c satisfies: $\deg(c) \leq g$
- Equality holds $\iff c = c(\mathcal{O})$ (maximal superstable)
- Now for (2) $\Rightarrow$, from $D = c + kq$, maximal unwinnable implies $k = -1 \Rightarrow D = c - q$
- $\Leftarrow$ If $D = c - q$ with maximal c, then:
  - D is unwinnable ($D(q) < 0$)
  - $D + v$ is winnable for all $v \in V$ by superstabilizing $c + v$ and tracking chip sent to q

Appendix: Acyclic Orientations & Maximal Unwinnable Divisors

Proposition: Bijection and Degree Bound

Let $q \in V$ be fixed. Then:

1. The map $\mathcal{O} \mapsto D(\mathcal{O}) := c(\mathcal{O}) - q$ defines a **bijection** between:
   - Acyclic orientations of G with q as unique source, and
   - Maximal unwinnable q-reduced divisors
2. If D is maximal unwinnable, then:
   $\deg(D) = g - 1 \Rightarrow \deg(D) \geq g \implies$ D is winnable.

- From prior results: Maximal unwinnables take the form $D = c - q$ where c is maximal superstable with $\deg(c) = g$. Hence, Proved.
- This provides a clean threshold for deciding winnability!

Takeaway

Acyclic orientations with source q uniquely correspond to maximal unwinnable divisors of degree $g - 1$.

Appendix: Why Define Orientations & Configurations?

Orientations: Encoding Graph Structure Algebraically

- Assign directions to edges $\Rightarrow$ define indegree-based divisors:

$$D(\mathcal{O}) := \sum_{v \in V} (\mathrm{indeg}_{\mathcal{O}}(v) - 1) \cdot v$$

- Restricting to **acyclic orientations** with unique source q ensures:
  - Well-defined, injective map to divisors
  - Canonical bridge to maximal superstables

Configurations: Localized Divisor Views

- Configuration $c \in \mathbb{Z}^{V \setminus \{q\}}$ omits chip count at q
- Enables formalization of:
  - **Superstability:** No legal set-firing in $V \setminus \{q\}$
  - **q-reduction:** Pushes all debt to q (central for winnability)

Appendix: Subadditivity of Rank

Corollary: Rank Inequality

For any divisors D, D' with $r(D), r(D') \geq 0$,

$$r(D + D') \geq r(D) + r(D')$$

Sketch of Proof

- Suppose $r(D) \geq k_1$ and $r(D') \geq k_2$
- Then for any $E_1, E_2 \geq 0$ with $\deg(E_1) = k_1$, $\deg(E_2) = k_2$:

$$D - E_1 \text{ and } D' - E_2 \text{ are winnable}$$

- So for $E'' = E_1 + E_2$ (with $\deg(E'') = k_1 + k_2$), we have:

$$(D + D') - E'' = (D - E_1) + (D' - E_2) \text{ is winnable}$$

- Therefore: $r(D + D') \geq k_1 + k_2 \Rightarrow r(D + D') \geq r(D) + r(D')$.

Appendix: More Details for Riemann-Roch

1. **Start with $r(D)$:** Use the definition to find an effective divisor E with $\deg(E) = r(D) + 1$ such that $D - E$ is unwinnable.
2. **Apply Dhar's Algorithm:** Find a q-reduced divisor equivalent to $D - E$, say $c + kq$ with $k < 0$.
3. **Link to Orientations:** Extend c to a maximal superstable c'.
   - Associate an acyclic orientation $\mathcal{O}$ such that $D(\mathcal{O}) = c' - q$.
4. **Define correction term H:**

$$H := (c' - c) - (k + 1)q \sim D(\mathcal{O}) - (D - E)$$

5. **Relate to the Canonical Divisor:** $K - H - D \sim D(\overline{\mathcal{O}}) - E$.
   - Since the RHS is unwinnable, deduce: $r(K - D) < \deg(H)$.
6. **Use degree bound:** Apply $\deg(D(\mathcal{O})) = g - 1$ and $\deg(E)$ from 1:

$$r(K-D) < g-1-\deg(D)+r(D)+1 \Rightarrow \deg(D)-g < r(D)-r(K-D)$$

7. **Apply symmetry:** Swap $D \leftrightarrow K - D$ for reverse inequality.
8. **Conclude equality:** $r(D) - r(K - D) = \deg(D) - g + 1$

Appendix: Canonical Duality of Maximal Unwinnables

Corollary: Duality via Canonical Divisor

A divisor D is **maximal unwinnable** if and only if $K - D$ is also maximal unwinnable.

Sketch of Proof

- If D is maximal unwinnable, then: $r(D) = -1$ and $\deg(D) = g - 1$.
- Use Riemann–Roch Theorem:

$$r(D) - r(K - D) = \deg(D) - g + 1 = 0 \Rightarrow r(K - D) = -1$$

- Compute degree:

$$\deg(K - D) = \deg(K) - \deg(D) = 2g - 2 - (g - 1) = g - 1$$

- Hence, $K - D$ is also maximal unwinnable.
- Reverse implication follows by symmetry: $D = K - (K - D)$

Appendix: Clifford's Theorem for Graphs

Clifford's Theorem (Graph-Theoretic Version)

If $D \in \mathrm{Div}(G)$ satisfies: $r(D) \geq 0 \quad$ and $\quad r(K - D) \geq 0$ then:

$$r(D) \leq \frac{1}{2}\deg(D)$$

Proof Sketch (Using Riemann–Roch)

- Use Riemann–Roch: $r(D) = r(K - D) + \deg(D) - g + 1$
- Use $D + (K - D) = K \Rightarrow r(K) \geq r(D) + r(K - D)$
- Substitute: $g - 1 = r(K) \geq r(D) + r(K - D)$
- Plug into earlier expression:

$$g - 1 \geq r(D) + r(D) - \deg(D) - 1 + g \Rightarrow r(D) \leq \frac{1}{2}\deg(D)$$

Appendix: Rank Determination by Degree

Corollary: Rank Behavior Based on Degree

Let $D \in \mathrm{Div}(G)$, then:

1. If $\deg(D) < 0$, then $r(D) = -1$
2. If $0 \leq \deg(D) \leq 2g - 2$, then $r(D) \leq \frac{1}{2}\deg(D)$
3. If $\deg(D) > 2g - 2$, then $r(D) = \deg(D) - g$

Proof Sketch Summary

- **(1)**: Immediate from the definition of rank.
- **(2)**:
  - If D is unwinnable, then $r(D) = -1 \leq \frac{1}{2}\deg(D)$
  - If $r(D) \geq 0$ and $r(K - D) = -1$: Riemann–Roch gives $r(D) = \deg(D) - g$, and since $g \geq \frac{1}{2}\deg(D) + 1$, we get the bound.
  - If $r(K - D) \geq 0$: Apply Clifford's Theorem directly.
- **(3)**: Since $\deg(D) > \deg(K)$, $K - D$ has negative degree $\Rightarrow r(K - D) = -1$. Apply RRG to get: $r(D) = \deg(D) - g$.