Deterministic Procedural Generation as an Invertible Algebraic System

by: Weptune

1 Foundations of Deterministic Procedural Generation

1.1 Seed Spaces and Algebraic Domains

We begin by formalizing the notion of a *seed* as an algebraic object. In deterministic procedural generation systems, the seed is a finite-width machine word whose arithmetic is implicitly performed modulo a power of two.

Definition 1.1 (Seed Space)**.** The seed space is defined as the finite ring

$$\mathcal{S} := \mathbb{Z}/2^{64}\mathbb{Z}.$$

The cardinality of this space is

$$|\mathcal{S}| = 2^{64},$$

which we interpret as the total entropy available to the generator.

1.2 World Generation as a Deterministic Function

Let $\mathcal{W}$ denote the space of all possible generated worlds. Although $\mathcal{W}$ is typically infinite—indexed over spatial coordinates such as $\mathbb{Z}^2$ or $\mathbb{Z}^3$—we treat it abstractly as a set of configurations.

Definition 1.2 (World Generator)**.** A world generator is a function

$$\mathcal{G} : \mathcal{S} \to \mathcal{W}.$$

Definition 1.3 (Determinism)**.** The generator $\mathcal{G}$ is deterministic if

$$\forall s \in \mathcal{S}, \quad \mathcal{G}(s) \text{ is uniquely defined.}$$

Remark 1.4. Determinism implies that $\mathcal{G}$ introduces no entropy beyond that contained in the seed itself.

1.3 Locality and Order Independence

Procedural generators must satisfy a strong locality constraint: generation of a local region must not depend on the order in which other regions are generated. Formally, for spatial coordinates $\mathbf{x} \in \mathbb{Z}^n$, the local world state must be expressible as

$$W(\mathbf{x}) = f(s, \mathbf{x})$$

for some deterministic function f.

This requirement prohibits global mutable state and necessitates the repeated re-initialization of pseudo-random number generators using affine transformations of the seed and coordinates.

1.4 Entropy Accounting

Let $H(\cdot)$ denote Shannon entropy. Initially,

$$H(\mathsf{Seed}) = 64 \text{ bits}.$$

Since $\mathcal{G}$ is deterministic,

$$H(\mathcal{W} \mid \mathsf{Seed}) = 0.$$

Thus, all entropy present in the generated world originates entirely from the seed.

1.5 Observations as Projections

An observer never perceives the full world $\mathcal{W}$. Instead, they observe a finite projection

$$\mathcal{O} = \pi(\mathcal{W}),$$

where π is a projection operator selecting a finite subset of world features.

The central inverse problem studied in this work is whether

$$H(\mathsf{Seed} \mid \mathcal{O}) = 0.$$

If so, the seed is uniquely recoverable from observations.

1.6 Pseudo-Random Number Generators

We now formalize pseudo-random number generators as algebraic dynamical systems.

Definition 1.5 (Pseudo-Random Number Generator)**.** A PRNG is a triple

$$(\mathcal{X}, F, O),$$

where:

- $\mathcal{X}$ is a finite state space,
- $F : \mathcal{X} \to \mathcal{X}$ is the state transition function,
- $O : \mathcal{X} \to \mathbb{Z}$ is the output function.

In deterministic generators, the initial PRNG state is a function of the seed.

1.7 Information Flow Lemma

Lemma 1.6. *Let $\mathcal{G}$ be deterministic and let $\mathcal{O} = \pi(\mathcal{G}(\mathsf{Seed}))$. Then*

$$I(\mathsf{Seed}; \mathcal{O}) = H(\mathcal{O}).$$

Proof. Because $\mathcal{O}$ is a deterministic function of Seed, we have

$$H(\mathcal{O} \mid \mathsf{Seed}) = 0.$$

Therefore,

$$I(\mathsf{Seed}; \mathcal{O}) = H(\mathcal{O}) - H(\mathcal{O} \mid \mathsf{Seed}) = H(\mathcal{O}).$$

□

1.8 Preview of Inversion

The remainder of this paper establishes that:

(i) PRNG state evolution is linear over modular rings,

(ii) Observations impose bounded linear constraints on internal states,

(iii) These constraints define convex bodies intersecting algebraic lattices,

(iv) Under mild conditions, this intersection is unique.

Seed recovery is therefore not an exploit, but a mathematical consequence of determinism.

2 Linear Congruential Generators as Algebraic Dynamical Systems

2.1 Motivation

Deterministic procedural generation relies almost exclusively on pseudo-random number generators whose behavior is reproducible across executions and platforms. Among these, linear congruential generators (LCGs) are favored for their simplicity, performance, and compatibility with fixed-width machine arithmetic. In this section, we study LCGs as algebraic dynamical systems and establish the structural properties that make them both suitable for generation and amenable to inversion.

2.2 Definition and State Space

Definition 2.1 (Linear Congruential Generator)**.** Let $m \in \mathbb{N}$, $a, c \in \mathbb{Z}$. A linear congruential generator is defined by the recurrence

$$x_{n+1} \equiv ax_n + c \pmod{m},$$

where $x_n \in \mathbb{Z}_m$ denotes the internal state at step n.

Throughout this work, we restrict attention to the case

$$m = 2^k,$$

as this choice dominates real-world implementations due to efficient modular reduction via integer overflow.

2.3 Affine Structure Over Rings

Define the affine map

$$F : \mathbb{Z}_m \to \mathbb{Z}_m, \quad F(x) = ax + c.$$

Then the LCG defines a discrete-time dynamical system

$$x_{n+1} = F(x_n).$$

Remark 2.2. When m is not prime, $\mathbb{Z}_m$ is not a field. Nevertheless, the affine structure of F allows for linear analysis over rings.

2.4 Invertibility of the Transition Map

Lemma 2.3 (Invertibility Criterion)**.** *The map $F(x) = ax + c \mod m$ is bijective on $\mathbb{Z}_m$ if and only if $\gcd(a, m) = 1$.*

Proof. The map F is bijective if and only if multiplication by a is bijective on $\mathbb{Z}_m$. This holds precisely when a is a unit in the ring, i.e., when $\gcd(a, m) = 1$. □

Corollary 2.4. *If $m = 2^k$, then F is invertible if and only if a is odd.*

2.5 Backward Evolution

Assuming invertibility, let $a^{-1} \in \mathbb{Z}_m$ denote the multiplicative inverse of a. Then the inverse recurrence is given by

$$x_n \equiv a^{-1}(x_{n+1} - c) \pmod{m}.$$

Proposition 2.5. *An invertible LCG induces a permutation of the state space $\mathbb{Z}_m$.*

Proof. Each state has a unique successor by definition of F, and invertibility guarantees a unique predecessor. Hence the state transition graph consists of disjoint directed cycles. □

This reversibility is fundamental to seed recovery: PRNG evolution may be run backward without loss of information.

2.6 Periodicity and the Hull–Dobell Theorem

Because the state space is finite, every LCG trajectory is eventually periodic.

Definition 2.6 (Full Period)**.** An LCG is said to have full period if its orbit visits every element of $\mathbb{Z}_m$ exactly once.

Theorem 2.7 (Hull–Dobell)**.** *An LCG modulo m has full period if and only if:*

1. $\gcd(c, m) = 1$,
2. $a \equiv 1 \pmod{p}$ *for every prime* $p \mid m$,
3. *If* $4 \mid m$, *then* $a \equiv 1 \pmod{4}$.

For $m = 2^k$, these conditions reduce to:

$$c \text{ odd}, \quad a \equiv 1 \pmod{4}.$$

2.7 Closed-Form Expression

Lemma 2.8 (Closed Form of the Recurrence)**.** *For all $n \geq 1$,*

$$x_n \equiv a^n x_0 + c \sum_{i=0}^{n-1} a^i \pmod{m}.$$

Proof. The claim follows by induction on n. The base case $n = 1$ is immediate. Assuming the formula holds for n, substitution into the recurrence yields the result for $n + 1$. □

Define the partial sum

$$S_n := \sum_{i=0}^{n-1} a^i.$$

Then

$$x_n \equiv a^n x_0 + cS_n \pmod{m}.$$

This explicit linear dependence on the initial state is the algebraic foundation of all inversion techniques.

2.8 Vectorization of Consecutive States

Rather than studying states individually, we bundle them into vectors:

$$\mathbf{x}_n := (x_n, x_{n+1}, \ldots, x_{n+k-1}) \in \mathbb{Z}_m^k.$$

Using the closed form, we obtain

$$\mathbf{x}_n = x_n \begin{pmatrix} 1 \\ a \\ a^2 \\ \vdots \\ a^{k-1} \end{pmatrix} + c \begin{pmatrix} 0 \\ 1 \\ 1+a \\ \vdots \\ \sum_{i=0}^{k-2} a^i \end{pmatrix} \pmod{m}.$$

Proposition 2.9. *The set of all length-k state vectors lies in an affine subspace of $\mathbb{Z}_m^k$ of rank at most one.*

Proof. All vectors are affine images of the single scalar parameter x_n. □

2.9 Spectral Structure and Hyperplane Phenomenon

When embedded into $\mathbb{R}^k$ via normalization by m, the vectors $\mathbf{x}_n$ lie on a finite collection of parallel hyperplanes.

Theorem 2.10. *Let*

$$\mathcal{X}_k := \left\{ \frac{1}{m}\mathbf{x}_n : n \in \mathbb{N} \right\} \subset [0,1)^k.$$

Then $\mathcal{X}_k$ is contained in at most $m^{1-1/k}$ parallel hyperplanes.

This phenomenon, often referred to as the *spectral defect* of LCGs, demonstrates that LCG output is far from uniformly distributed in high dimensions.

2.10 LCGs as Lattice Generators

Define the integer lattice

$$L := \left\{ z \begin{pmatrix} 1 \\ a \\ a^2 \\ \vdots \\ a^{k-1} \end{pmatrix} \;\middle|\; z \in \mathbb{Z} \right\} \subset \mathbb{Z}^k.$$

Proposition 2.11. *After affine translation and scaling, the set of LCG state vectors coincides with a rank-one lattice modulo m.*

Remark 2.12. This lattice structure is exact and unavoidable. All modern seed-recovery techniques exploit this property directly or indirectly.

2.11 Implications for Procedural Generation

From the perspective of deterministic generation, LCGs provide:

- Fast state evolution,
- Exact reproducibility,
- Algebraic transparency under analysis.

From the perspective of inversion, these same properties imply that the generator preserves all seed entropy in a structured, recoverable form.

2.12 Transition to Observability

In practice, the internal state x_n is never directly exposed. Instead, generators reveal bounded functions of the state. The next section develops a formal framework for converting such bounded outputs into systems of algebraic inequalities.

3 Affine Coordinate Mixing and Locality

3.1 Motivation: Local Determinism

A defining requirement of procedural world generation is *order independence*: the generated content at a location must not depend on whether neighboring regions have already been generated. This constraint forces generators to derive local pseudo-randomness exclusively from the global seed and spatial coordinates, rather than from mutable global state.

Formally, for spatial coordinates $\mathbf{x} \in \mathbb{Z}^d$, generation must satisfy

$$W(\mathbf{x}) = f(\mathsf{Seed}, \mathbf{x})$$

for some deterministic function f.

This section demonstrates that the standard solution to this problem—affine coordinate mixing—preserves invertibility and introduces no entropy loss.

3.2 Affine Reseeding as a Mathematical Operation

Let $\mathsf{Seed} \in \mathbb{Z}_{2^{64}}$ be the global seed. For a fixed coordinate $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{Z}^d$, a local pseudo-random generator is initialized via a reseeding function

$$\phi(\mathsf{Seed}, \mathbf{x}) \in \mathbb{Z}_{2^k},$$

where $k \leq 64$ is the internal state width of the PRNG.

In practice, ϕ is chosen to be affine:

$$\phi(\mathsf{Seed}, \mathbf{x}) = \mathsf{Seed} \oplus \left(\sum_{i=1}^{d} A_i x_i \right) \mod 2^k,$$

where $A_i \in \mathbb{Z}$ are fixed odd constants.

Remark 3.1. Bitwise XOR with constants is equivalent to addition modulo 2^k after a fixed linear transformation. Thus, ϕ is an affine map over $\mathbb{Z}_{2^k}$.

3.3 Affine Maps over Modular Rings

Definition 3.2 (Affine Map over $\mathbb{Z}_{2^k}$)**.** A function $\psi : \mathbb{Z}_{2^k} \to \mathbb{Z}_{2^k}$ is affine if

$$\psi(x) = ax + b \mod 2^k$$

for some $a, b \in \mathbb{Z}$.

Coordinate mixing is an affine transformation jointly in Seed and $\mathbf{x}$.

Proposition 3.3. *For fixed* $\mathbf{x}$*, the map* $\mathsf{Seed} \mapsto \phi(\mathsf{Seed}, \mathbf{x})$ *is an affine bijection of* $\mathbb{Z}_{2^k}$*.*

Proof. The map is of the form $\mathsf{Seed} \mapsto \mathsf{Seed} + c(\mathbf{x}) \mod 2^k$, where $c(\mathbf{x})$ is fixed. Translation in a finite ring is bijective. □

Thus, coordinate mixing does not collapse seed entropy.

3.4 Equivalence Classes of Seeds

Although affine reseeding is bijective for fixed $\mathbf{x}$, distinct seeds may coincide under *multiple* coordinates.

Definition 3.4 (Coordinate Equivalence)**.** Two seeds $s_1, s_2 \in \mathbb{Z}_{2^k}$ are equivalent with respect to a coordinate set $\mathcal{X} \subset \mathbb{Z}^d$ if

$$\phi(s_1, \mathbf{x}) = \phi(s_2, \mathbf{x}) \quad \forall \mathbf{x} \in \mathcal{X}.$$

Lemma 3.5. *If* $\mathcal{X}$ *contains at least one coordinate* $\mathbf{x}$*, then* $s_1 \equiv s_2 \pmod{2^k}$*.*

Proof. For any $\mathbf{x}$,

$$s_1 + c(\mathbf{x}) \equiv s_2 + c(\mathbf{x}) \pmod{2^k}$$

implies $s_1 \equiv s_2 \pmod{2^k}$. □

Thus, no nontrivial equivalence classes exist: coordinate mixing preserves injectivity.

3.5 Locality Does Not Obscure the Seed

It is sometimes conjectured that reseeding per coordinate "hides" the global seed. Algebraically, the opposite is true.

Theorem 3.6. *Let* $\phi(\mathsf{Seed}, \mathbf{x})$ *be an affine reseeding function. Then the mapping*

$$\mathsf{Seed} \mapsto \{\phi(\mathsf{Seed}, \mathbf{x})\}_{\mathbf{x} \in \mathcal{X}}$$

is injective for any nonempty finite set $\mathcal{X}$*.*

Proof. Each coordinate induces an affine translation of Seed. Equality of all translated values implies equality of Seed. □

Thus, locality does not reduce information content; it merely redistributes it spatially.

3.6 Collision Analysis modulo 2^k

We now analyze whether distinct coordinate pairs can collide in their induced reseeds.

Consider two coordinates $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^d$. A collision occurs if

$$\phi(\mathsf{Seed}, \mathbf{x}) = \phi(\mathsf{Seed}, \mathbf{y}).$$

This implies

$$\sum_{i=1}^{d} A_i(x_i - y_i) \equiv 0 \pmod{2^k}.$$

Lemma 3.7. *If all A_i are odd, then a collision occurs if and only if*

$$x_i \equiv y_i \pmod{2^k} \quad \forall i.$$

Proof. Since each A_i is invertible modulo 2^k, the sum vanishes if and only if each difference $x_i - y_i$ vanishes modulo 2^k. □

Corollary 3.8. *Within any bounded coordinate region, coordinate mixing is injective.*

3.7 Implications for Chunk and Region Seeding

In practical generators, coordinates are grouped into chunks or regions. For chunk coordinates $(x, z) \in \mathbb{Z}^2$, reseeding takes the form

$$\mathsf{Seed}_{x,z} = \mathsf{Seed} \oplus (Ax + Bz).$$

The preceding analysis implies:

- Each chunk seed is an affine image of the global seed.
- No entropy is destroyed by chunk-local reseeding.
- Observations across multiple chunks impose independent affine constraints on Seed.

This observation is central to all structure-based and decorator-based seed recovery methods.

3.8 Affine Composition with PRNG Evolution

Let F denote the LCG transition map. The combined evolution from Seed to the n-th PRNG state at coordinate $\mathbf{x}$ is

$$x_n(\mathbf{x}) = F^n(\phi(\mathsf{Seed}, \mathbf{x})).$$

Since both ϕ and F are affine, their composition is affine:

$$x_n(\mathbf{x}) = a^n \mathsf{Seed} + b_n(\mathbf{x}) \mod 2^k.$$

Proposition 3.9. *Every observable PRNG state is an affine function of the global seed.*

Proof. Affine functions are closed under composition. □

This fact reduces seed recovery to solving systems of affine congruences.

3.9 Locality as an Information Distribution Mechanism

Rather than hiding information, locality distributes seed information across space. Each generated feature leaks a different affine projection of the same underlying seed.

Remark 3.10. This is analogous to observing multiple linear measurements of a hidden vector in compressed sensing.

3.10 Transition to Observations

Having established that coordinate mixing preserves affine dependence on the seed, we now turn to the final missing ingredient: how observable outputs constrain PRNG states. The next section develops a rigorous framework for translating bounded outputs into systems of linear inequalities.

4 Observations as Inequalities and Constraint Geometry

4.1 Motivation: From State to Observation

Thus far, we have shown that every internal pseudo-random state appearing in procedural generation is an affine function of the global seed. However, internal states are never directly observable. Instead, the generator exposes bounded functions of these states. This section formalizes the crucial step of converting observable outcomes into algebraic constraints.

The core principle is simple yet powerful:

An observation never reveals a value; it reveals an interval.

This transformation is the mathematical bridge between deterministic generation and seed recovery.

4.2 Bounded Output Functions

Let $x \in \mathbb{Z}_{2^k}$ be an internal PRNG state. Observable outputs are produced by functions of the form

$$O(x) = g\Big(\frac{x}{2^k}\Big),$$

where g is a piecewise-constant function mapping $[0, 1)$ to a finite set.

Definition 4.1 (Bounded Output)**.** An output function $O : \mathbb{Z}_{2^k} \to \mathbb{Z}$ is bounded if there exists $n \in \mathbb{N}$ such that

$$O(x) \in \{0, 1, \ldots, n-1\}.$$

Typical examples include integer sampling, categorical selection, and threshold comparisons.

4.3 The Case of Integer Sampling

The most common bounded output is integer sampling.

Definition 4.2 (Uniform Integer Sampling)**.** Define

$$\text{nextInt}(n) := \Big\lfloor \frac{x}{2^k} \cdot n \Big\rfloor.$$

This definition induces the inequality

$$y \leq \frac{x}{2^k} \cdot n < y + 1,$$

which is equivalent to

$$\frac{y}{n} 2^k \leq x < \frac{y+1}{n} 2^k.$$

Proposition 4.3. *A call to* nextInt(n) *yields a half-open interval constraint on x of width $2^k/n$.*

4.4 Entropy Leakage per Observation

Each observation restricts the internal state to a subset of $\mathbb{Z}_{2^k}$.

Definition 4.4 (Constraint Width)**.** The width of an observation $O(x) = y$ is defined as

$$\Delta(O = y) := \frac{2^k}{n}.$$

Remark 4.5. Smaller output ranges yield wider constraints; larger output ranges yield tighter constraints.

The information content of an observation is approximately

$$\log_2 n \text{ bits.}$$

4.5 Sequences of Observations

Let $(x_0, x_1, \ldots, x_{m-1})$ be consecutive PRNG states, related by the recurrence

$$x_{i+1} \equiv a x_i + c \pmod{2^k}.$$

Suppose we observe outputs

$$O_i(x_i) = y_i, \quad i = 0, \ldots, m-1.$$

Each observation yields an interval

$$x_i \in [L_i, U_i).$$

Thus, the internal state vector

$$\mathbf{x} := (x_0, \ldots, x_{m-1})$$

is constrained to lie in the axis-aligned box

$$\mathcal{B} := \prod_{i=0}^{m-1} [L_i, U_i) \subset \mathbb{R}^m.$$

4.6 Dependence Between Constraints

Although the intervals appear independent, the recurrence relation couples them.

Proposition 4.6. *The set of feasible state vectors is*

$$\mathcal{F} := \left\{ (x_0, \ldots, x_{m-1}) \in \mathcal{B} \;\middle|\; x_{i+1} \equiv a x_i + c \pmod{2^k} \right\}.$$

Geometrically, $\mathcal{F}$ is the intersection of $\mathcal{B}$ with an affine subspace.

4.7 Absence as a Complement Constraint

Not all observations correspond to equality. Frequently, the absence of a feature is observed.

Definition 4.7 (Absence Constraint)**.** An absence observation corresponds to the constraint

$$O(x) \neq y.$$

For nextInt(n), this yields

$$x \notin \left[\frac{y}{n}2^k, \frac{y+1}{n}2^k\right).$$

Remark 4.8. Absence constraints remove entire slabs from the state space and can be more restrictive than presence constraints.

4.8 Piecewise Constraints and Branching

Many generators contain conditional logic:

```
if (nextInt(n) == y) {
    generateFeature();
}
```

This produces a branching constraint:

$$x \in I_y \quad \text{or} \quad x \notin I_y.$$

Theorem 4.9. *Branching conditions partition the state space into disjoint regions with distinct future evolution.*

Proof. Different branches consume different numbers of PRNG calls, leading to divergent affine evolutions of subsequent states. □

4.9 Geometry of Constraint Intersection

We now view the feasible set as a geometric object.

Definition 4.10 (Feasible Region)**.** Let $\mathcal{L} \subset \mathbb{Z}^m$ denote the lattice induced by the PRNG recurrence. The feasible region is

$$\mathcal{R} := \mathcal{L} \cap \mathcal{B}.$$

Seed recovery corresponds to determining the preimage of $\mathcal{R}$ under the affine mapping from the seed.

4.10 Volume vs. Lattice Determinant

Let $\mathrm{vol}(\mathcal{B})$ denote the volume of the bounding box and $\det(\mathcal{L})$ the determinant of the lattice.

Theorem 4.11 (Uniqueness Criterion)**.** *If*

$$\mathrm{vol}(\mathcal{B}) < \det(\mathcal{L}),$$

then $\mathcal{R}$ contains at most one lattice point.

Proof. This follows from Minkowski's convex body theorem applied to the difference body $\mathcal{B}-\mathcal{B}$. □

This theorem explains why sufficiently many observations uniquely determine the internal state.

4.11 Measure-Theoretic Interpretation

Each observation reduces the measure of the feasible state space by a factor of approximately $1/n$.

Proposition 4.12. *Let m independent* $\mathrm{nextInt}(n)$ *observations be made. Then*

$$\mathrm{vol}(\mathcal{B}) \approx \frac{2^{km}}{n^m}.$$

As m grows, the feasible volume shrinks exponentially.

4.12 Entropy Collapse

From an information-theoretic perspective, each observation contributes conditional mutual information.

Theorem 4.13. *If*

$$\sum_{i=0}^{m-1} \log_2 n_i > k,$$

then the internal PRNG state is uniquely determined with overwhelming probability.

Proof. The sum of information exceeds the entropy of the state space, forcing $H(x_0 \mid \mathcal{O}) = 0$. □

4.13 From State Constraints to Seed Constraints

Recall from Part III that each x_i is an affine function of the global seed:

$$x_i = \alpha_i \mathsf{Seed} + \beta_i \mod 2^k.$$

Thus, each interval constraint on x_i induces an interval constraint on Seed.

Proposition 4.14. *Every observation yields a linear modular inequality on the global seed.*

Seed recovery therefore reduces to solving a system of affine modular inequalities.

4.14 Transition to Lattice Methods

The final remaining step is algorithmic: given a system of affine modular inequalities, how can one efficiently compute the unique solution?

The next section develops the lattice-based techniques that make this computation tractable in practice.

5 Lattice Construction and Determinant Analysis

5.1 Motivation: From Inequalities to Integer Geometry

In Part IV, we reduced seed recovery to a system of affine modular inequalities. The purpose of this section is to embed those inequalities into a *lattice problem* over the integers. This step is not heuristic: it is a mathematically exact reformulation that allows us to reason about existence, uniqueness, and solvability independently of any particular algorithm.

5.2 Removing Modular Arithmetic via Lifting

Let the PRNG recurrence be

$$x_{i+1} \equiv ax_i + c \pmod{2^k}.$$

We lift this congruence to an equality over the integers by introducing carry variables.

Definition 5.1 (Carry Variables)**.** For each $i \geq 0$, define an integer $m_i \in \mathbb{Z}$ such that

$$x_{i+1} = ax_i + c - m_i 2^k.$$

This transformation removes modular arithmetic entirely at the cost of introducing additional integer unknowns.

5.3 Closed-Form Lifted System

Unrolling the lifted recurrence yields

$$x_i = a^i x_0 + c \sum_{j=0}^{i-1} a^j - 2^k m_{i-1} - 2^k a m_{i-2} - \cdots - 2^k a^{i-1} m_0.$$

For notational convenience, define

$$C_i := c \sum_{j=0}^{i-1} a^j.$$

Then

$$x_i = a^i x_0 + C_i - 2^k \sum_{j=0}^{i-1} a^{i-1-j} m_j.$$

5.4 Vector Formulation

Define the unknown integer vector

$$\mathbf{v} := \begin{pmatrix} x_0 \\ m_0 \\ m_1 \\ \vdots \\ m_{n-1} \end{pmatrix} \in \mathbb{Z}^{n+1}.$$

Define the matrix $B \in \mathbb{Z}^{n \times (n+1)}$ by

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ a & -2^k & 0 & 0 & \cdots & 0 \\ a^2 & -2^k a & -2^k & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ a^{n-1} & -2^k a^{n-2} & -2^k a^{n-3} & \cdots & -2^k & 0 \end{pmatrix}.$$

Then the lifted state vector satisfies

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} = B\mathbf{v} + \begin{pmatrix} 0 \\ C_1 \\ \vdots \\ C_{n-1} \end{pmatrix}.$$

5.5 Lattice Definition

Definition 5.2 (PRNG Lattice)**.** Let $\mathcal{L} \subset \mathbb{Z}^n$ be the lattice generated by the columns of B.

Thus,

$$\mathcal{L} = \{B\mathbf{v} \mid \mathbf{v} \in \mathbb{Z}^{n+1}\}.$$

Remark 5.3. The offset vector $(0, C_1, \ldots, C_{n-1})$ translates the lattice but does not affect its structure.

5.6 Bounding Region Revisited

From Part IV, observations induce bounds

$$x_i \in [L_i, U_i).$$

Define midpoints and radii:

$$\mu_i := \frac{L_i + U_i}{2}, \quad r_i := \frac{U_i - L_i}{2}.$$

Then the feasible region is

$$\mathcal{B} := \{\mathbf{x} \in \mathbb{R}^n \mid |x_i - \mu_i| \leq r_i\}.$$

Seed recovery is equivalent to finding lattice points of $\mathcal{L}$ inside $\mathcal{B}$.

5.7 Determinant of the PRNG Lattice

The determinant of $\mathcal{L}$ governs the density of lattice points.

Lemma 5.4. *The determinant of $\mathcal{L}$ satisfies*

$$\det(\mathcal{L}) = 2^{k(n-1)}.$$

Proof. The matrix B is lower triangular after column operations that preserve determinant magnitude. Each carry column contributes a factor of 2^k, and there are $n-1$ such columns. □

5.8 Geometric Interpretation

The lattice $\mathcal{L}$ has extremely large determinant relative to the ambient volume. Intuitively, lattice points are very sparse in $\mathbb{R}^n$.

Remark 5.5. This sparsity is the fundamental reason that only a small number of observations are required to isolate a unique solution.

5.9 Bounding Volume

The volume of the bounding box is

$$\mathrm{vol}(\mathcal{B}) = \prod_{i=0}^{n-1} (2r_i).$$

For n observations of $\mathrm{nextInt}(q_i)$,

$$\mathrm{vol}(\mathcal{B}) \approx \frac{2^{kn}}{\prod q_i}.$$

5.10 Uniqueness Criterion

Theorem 5.6 (Lattice Uniqueness)**.** *If*

$$\mathrm{vol}(\mathcal{B}) < \det(\mathcal{L}),$$

then $\mathcal{B} \cap \mathcal{L}$ contains at most one point.

Proof. Consider the difference body $\mathcal{B} - \mathcal{B}$, which is centrally symmetric with volume $2^n \,\mathrm{vol}(\mathcal{B})$. If two distinct lattice points lie in $\mathcal{B}$, their difference lies in $\mathcal{B} - \mathcal{B}$, contradicting Minkowski's theorem when $\mathrm{vol}(\mathcal{B}) < \det(\mathcal{L})$. □

5.11 Threshold Analysis

Solving

$$\frac{2^{kn}}{\prod q_i} < 2^{k(n-1)}$$

yields

$$\prod q_i > 2^k.$$

Corollary 5.7. *Once the total information content of observations exceeds k bits, the internal state is uniquely determined.*

5.12 Back-Projection to the Seed

Recall that

$$x_0 = \alpha \mathsf{Seed} + \beta \mod 2^k$$

for known constants α, β.

Proposition 5.8. *A unique solution for x_0 yields a unique solution for* Seed *modulo 2^k.*

5.13 Interpretation

This section establishes that:

- Seed recovery is equivalent to a lattice point enumeration problem.
- The lattice is extremely sparse.
- Observations shrink the feasible region exponentially.
- Uniqueness is guaranteed before algorithms are applied.

Algorithmic considerations are therefore a matter of efficiency, not correctness.

5.14 Transition to Algorithms

Having established uniqueness, the remaining question is how to efficiently find the unique lattice point inside $\mathcal{B}$. The next section develops the lattice reduction and closest vector techniques that solve this problem in practice.

6 Lattice Reduction and Algorithms for Seed Recovery

6.1 Motivation: From Uniqueness to Computation

In Part V, we proved that under mild conditions the feasible region

$$\mathcal{B} \cap \mathcal{L}$$

contains at most one lattice point. This section addresses the remaining question: how can this point be found efficiently?

The answer lies in lattice reduction, specifically the Lenstra–Lenstra–Lovász (LLL) algorithm and its application to closest vector problems (CVP).

6.2 The Closest Vector Problem

Definition 6.1 (Closest Vector Problem)**.** Given a lattice $\mathcal{L} \subset \mathbb{R}^n$, a target vector $\mathbf{t} \in \mathbb{R}^n$, and a norm $\|\cdot\|$, the closest vector problem asks for

$$\mathbf{v} \in \mathcal{L} \quad \text{minimizing} \quad \|\mathbf{v} - \mathbf{t}\|.$$

In our setting:

- $\mathcal{L}$ is the PRNG lattice from Part V,
- $\mathbf{t} = (\mu_0, \ldots, \mu_{n-1})$ is the center of the bounding box,
- the norm is typically ℓ_∞ or ℓ_2.

6.3 CVP versus SVP

The shortest vector problem (SVP) asks for the nonzero lattice vector of minimal norm. CVP is strictly harder in general.

However, our instance lies in a special regime:

1. The solution is unique.
2. The target vector lies extremely close to the lattice.
3. The lattice determinant is enormous relative to the error radius.

These properties collapse CVP to a tractable problem.

6.4 Integer Embedding of the CVP

LLL operates on integer lattices. To apply it, we embed the CVP instance into a higher-dimensional SVP instance.

Definition 6.2 (Kannan Embedding)**.** Given a lattice basis $B \in \mathbb{Z}^{n \times m}$ and target $\mathbf{t} \in \mathbb{R}^n$, define the augmented lattice with basis

$$\tilde{B} = \begin{pmatrix} B & \mathbf{t} \\ \mathbf{0}^T & M \end{pmatrix},$$

where M is a large integer scaling factor.

A shortest vector in $\tilde{\mathcal{L}}$ corresponds to a closest vector in $\mathcal{L}$.

6.5 Choice of Scaling Parameters

The scaling factor M must satisfy two competing constraints:

1. It must be large enough to force the shortest vector to encode the CVP solution.
2. It must be small enough to avoid numerical instability.

In practice, one chooses

$$M \approx \max_i r_i,$$

where r_i are the bounding radii from Part V.

Lemma 6.3. *If M exceeds the maximum error radius by a constant factor, then the shortest vector of $\tilde{\mathcal{L}}$ encodes the CVP solution.*

6.6 Scaling and Conditioning

The basis B constructed in Part V is highly skewed: some coordinates are on the order of 2^k, while others are much smaller.

To improve conditioning, we apply diagonal scaling.

Definition 6.4 (Scaled Basis)**.** Let $D = \mathrm{diag}(d_1, \ldots, d_n)$ be a diagonal matrix of positive integers. The scaled basis is

$$B' = DB.$$

Typically, one chooses

$$d_i \approx \frac{1}{r_i},$$

so that all dimensions of the bounding box are normalized.

Remark 6.5. Scaling does not change the lattice up to isomorphism, but dramatically improves LLL performance.

6.7 The LLL Algorithm

Theorem 6.6 (Lenstra–Lenstra–Lovász)**.** *Given a basis B of a lattice $\mathcal{L} \subset \mathbb{R}^n$, the LLL algorithm computes a reduced basis in polynomial time, satisfying:*

1. *Size reduction: coefficients are bounded.*
2. *Lovász condition: basis vectors are nearly orthogonal.*

The algorithm runs in time polynomial in n and the bit-length of the basis entries.

6.8 Why LLL Succeeds in This Setting

Although LLL does not solve SVP or CVP exactly in general, our problem lies in a favorable regime.

Theorem 6.7. *In the PRNG lattice constructed in Part V, LLL recovers the correct solution vector with overwhelming probability.*

Sketch. The correct solution corresponds to an exceptionally short vector due to the uniqueness condition

$$\mathrm{vol}(\mathcal{B}) < \det(\mathcal{L}).$$

All other lattice vectors are separated by gaps exponential in k. LLL is guaranteed to recover vectors within an exponential approximation factor, which suffices to isolate the unique solution. □

6.9 Branching and Conditional Constraints

When observations include branching (cf. Part IV), multiple CVP instances may arise, one per branch.

Proposition 6.8. *The number of branches grows exponentially in the number of conditional observations, but each branch drastically reduces lattice volume.*

In practice, branch pruning is highly effective: infeasible branches are discarded early due to inconsistent constraints.

6.10 Complexity Analysis

Let:

- n be the number of observed PRNG calls,
- k the bit-width of the PRNG state.

The lattice dimension is $n + 1$, and the basis entries have $O(k)$ bits.

Theorem 6.9. *The total runtime of seed recovery is polynomial in n and k.*

Proof. LLL runs in time polynomial in dimension and bit-length. Branching introduces at most exponential factors in the number of conditional observations, which in practice is small. □

6.11 Practical Considerations

Several practical optimizations are commonly employed:

- Early pruning via interval intersection.
- Partial lattice reduction.
- Hybrid brute-force for low-entropy suffixes.

These do not affect correctness, only efficiency.

6.12 From Internal State to Global Seed

Once x_0 is recovered, the global seed is obtained by inverting the affine reseeding map from Part III:

$$\mathsf{Seed} \equiv \alpha^{-1}(x_0 - \beta) \mod 2^k.$$

Proposition 6.10. *Seed recovery is deterministic and exact once the internal state is known.*

6.13 Interpretation

This section establishes that seed recovery is:

- Constructive,
- Polynomial-time,
- Guaranteed under mild conditions,
- Independent of heuristic guessing.

The remaining sections place these results in a broader theoretical context.

6.14 Transition to Information-Theoretic Closure

Having shown both uniqueness and computability, we now turn to the deepest question: why such recovery is unavoidable in deterministic procedural generation. The final section frames seed recovery as an information-theoretic inevitability.

7 Information-Theoretic Inevitability and Generalization

7.1 Motivation: Beyond Algorithms

The preceding sections established that seed recovery is both unique and computationally feasible. We now address the deeper question: *why must this be the case?* This section demonstrates that seed recoverability is not an artifact of specific algorithms or implementations, but a fundamental consequence of determinism, finite state, and observability.

7.2 Procedural Generation as a Communication Channel

We model deterministic procedural generation as a communication channel:

$$\mathsf{Seed} \xrightarrow{\mathcal{G}} \mathcal{W} \xrightarrow{\pi} \mathcal{O},$$

where:

- $\mathsf{Seed} \in \mathbb{Z}_{2^k}$ is the source message,
- $\mathcal{G}$ is a deterministic generator,
- π is a projection onto observable features.

Proposition 7.1. *The channel* $\mathsf{Seed} \to \mathcal{W}$ *is noiseless.*

Proof. Since $\mathcal{G}$ is deterministic, $\mathcal{W}$ is a fixed function of Seed, implying $H(\mathcal{W} \mid \mathsf{Seed}) = 0$. □

7.3 Entropy Conservation

Theorem 7.2 (Entropy Conservation)**.** *Let* $\mathcal{O} = \pi(\mathcal{G}(\mathsf{Seed}))$*. Then*

$$I(\mathsf{Seed}; \mathcal{O}) = H(\mathcal{O}).$$

Proof. Because $\mathcal{O}$ is a deterministic function of Seed, we have $H(\mathcal{O} \mid \mathsf{Seed}) = 0$. Thus,

$$I(\mathsf{Seed}; \mathcal{O}) = H(\mathcal{O}) - H(\mathcal{O} \mid \mathsf{Seed}) = H(\mathcal{O}).$$

□

Every bit observed leaks one bit of information about the seed.

7.4 Entropy Collapse Criterion

Let $\mathcal{O}_1, \ldots, \mathcal{O}_m$ denote independent observations.

Theorem 7.3 (Entropy Collapse)**.** *If*

$$\sum_{i=1}^{m} I(\mathsf{Seed}; \mathcal{O}_i \mid \mathcal{O}_{<i}) \geq H(\mathsf{Seed}),$$

then

$$H(\mathsf{Seed} \mid \mathcal{O}_1, \ldots, \mathcal{O}_m) = 0.$$

Proof. By the chain rule of mutual information:

$$H(\mathsf{Seed} \mid \mathcal{O}_1, \ldots, \mathcal{O}_m) = H(\mathsf{Seed}) - \sum_{i=1}^{m} I(\mathsf{Seed}; \mathcal{O}_i \mid \mathcal{O}_{<i}).$$

□

This theorem formalizes the intuitive notion that enough observations force unique recovery.

7.5 Why Complexity Does Not Help

It is often conjectured that increasing algorithmic complexity improves security. In deterministic generation, the opposite holds.

Proposition 7.4. *Adding deterministic computation steps cannot reduce* $I(\mathsf{Seed}; \mathcal{O})$.

Proof. Deterministic transformations preserve mutual information. Any added computation merely re-encodes existing entropy. □

Thus, additional noise functions, biome layers, or decorators increase observability without increasing entropy.

7.6 Coupled PRNG Systems

Modern generators often employ multiple PRNGs (e.g., 48-bit and 64-bit generators) operating in parallel.

Definition 7.5 (Coupled PRNG System)**.** A coupled system consists of PRNGs $(\mathcal{X}_i, F_i, O_i)$ whose seeds are affine functions of a common global seed.

Theorem 7.6. *Coupled PRNG systems increase total information leakage.*

Proof. Each PRNG produces independent affine constraints on Seed. The combined observation space is the Cartesian product of individual observations, increasing mutual information additively. □

This explains why systems with biome generators, structure generators, and population generators are more vulnerable than simpler ones.

7.7 Generalization Beyond Linear PRNGs

Although this paper focuses on linear congruential generators, the argument extends further.

Theorem 7.7. *Any deterministic generator with finite internal state and bounded observable outputs is, in principle, invertible given sufficient observations.*

Sketch. Finite internal state implies finite entropy. Determinism implies entropy conservation. Observations impose constraints that reduce feasible states. Once the feasible set collapses to a singleton, inversion is complete. □

Linearity simplifies inversion, but is not required for inevitability.

7.8 Why Cryptographic PRNGs Do Not Solve the Problem

One might attempt to replace linear PRNGs with cryptographic ones.

Proposition 7.8. *Cryptographic PRNGs prevent efficient inversion but do not prevent theoretical invertibility.*

Remark 7.9. Cryptographic generators trade invertibility for computational hardness, not information loss.

Moreover, cryptographic PRNGs are typically unsuitable for procedural generation due to performance and reproducibility constraints.

7.9 The Cost of True Randomness

The only way to prevent seed recovery is to inject true entropy during generation.

Theorem 7.10. *Injecting nondeterministic entropy destroys reproducibility.*

Proof. If generation depends on external randomness, identical seeds can yield different worlds, violating determinism. □

Thus, seed unrecoverability is incompatible with the defining goals of procedural generation.

7.10 Procedural Generation as Lossless Expansion

We may reinterpret deterministic generation as a lossless expansion:

$$\mathsf{Seed} \longrightarrow \mathcal{G}(\mathsf{Seed}),$$

analogous to decompression.

Remark 7.11. Seed recovery is decompression from partial output.

This reframing clarifies why inversion is unavoidable.

7.11 Final Theorem

Theorem 7.12 (Main Result)**.** *Any deterministic procedural generation system with finite internal state and sufficiently rich observable output is seed-recoverable with overwhelming probability.*

Proof. Determinism implies entropy conservation. Observations reduce feasible state volume. Finite state implies eventual uniqueness. Computational techniques recover the unique solution. □

7.12 Conclusion

Seed recovery is not an exploit, vulnerability, or accident. It is a mathematical inevitability arising from the fundamental constraints of deterministic procedural generation. Any system satisfying reproducibility, locality, and finite state necessarily admits inversion given sufficient observation.

Closing Remarks

The framework developed here applies broadly beyond any specific engine or implementation. It provides a general lens through which deterministic generative systems may be analyzed, inverted, and understood.