Programs as Histories

Projection, Admissibility, and the Geometry of Computation

A Case Study in Ontology-Driven Language Design

Flyxion

Independent Research

June 2026

The implementation did not merely realize the theory;
it exposed which distinctions the theory was still hiding.

Abstract

Most programming languages begin with values, variables, and state transitions. Spherepop begins elsewhere. It begins with the claim that histories are ontologically prior to states, and that computation should be understood as the progressive restriction of possibility rather than the manipulation of values.

This monograph presents the design and formal theory of the Spherepop programming language and interpreter. We proceed from ontological commitment through mathematical formalization to implementation, treating each software component as the computational expression of a prior philosophical necessity.

We establish a formal operational semantics, a typed admissibility judgement system, four principal theorems (History Monotonicity, Option Space Monotonicity, Collapse Soundness, Replay Equivalence), and a category-theoretic account of histories as morphisms and collapse as a functor between history and observable-state categories.

Four implementation discoveries retroactively sharpened the theory: refusal must carry reasons; collapse must specify quotient rules; type checking depends on the assumed boundary of the world; and compiler correctness requires history equivalence, not mere output equivalence.

The thesis is outrageous only as a slogan. As a design principle it is precise: *computation is the progressive restriction of possibility, not the manipulation of values.*

Contents

Part I

Why State Is the Wrong Primitive

Chapter 1

The State-Centric Assumption

> *A program is a sequence of state transformations. So runs the unexamined premise of nearly every language ever designed.*
>
> —

1.1 Motivation

Open any introductory programming text and you encounter a common first lesson: a variable is a named location that holds a value. A program begins in some initial state, instructions modify that state, and the final state is the answer.

```
x = 5
y = x + 1
```

This appears natural. It mirrors how we naively think about physical processes: an object has properties; those properties change; the current properties constitute the object's state. But the appearance of naturalness is produced by a prior choice — the choice to treat present configuration as the fundamental unit of description.

1.2 The State Illusion

Consider what this choice erases. In version control, the repository is not merely its current file tree; it is the entire sequence of commits that produced it. In event sourcing, database records

are append-only event logs; the current state is a fold over that log. In scientific experiment, a measurement is the terminus of a sequence of decisions, calibrations, and interpretive choices. In evolutionary biology, a genome is a compressed record of selective pressures.

In each case, the historical structure is epistemically and causally prior to the present configuration. State is the summary. History is the substance.

Definition 1.1 (State Degeneracy)**.** *Let E be an event alphabet, $\mathcal{H} = \bigcup_{n=0}^{\infty} E^n$ the set of all finite histories, and S an observable state space. A* state projection *is a function $\sigma : \mathcal{H} \to S$. The* degeneracy *of a state $s \in S$ is*

$$D(s) \;=\; \left|\sigma^{-1}(s)\right|.$$

Proposition 1.2 (The State Illusion)**.** *For any nontrivial finite state space S and any non-injective projection $\sigma : \mathcal{H} \to S$, there exist distinct histories $H_1 \neq H_2$ such that $\sigma(H_1) = \sigma(H_2)$. Consequently,*

$$\sigma(H_1) = \sigma(H_2) \;\not\Rightarrow\; H_1 = H_2.$$

Proof. Since $|\mathcal{H}|$ is countably infinite and $|S|$ is finite, σ cannot be injective. Therefore $D(s) > 1$ for at least one $s \in S$, which yields the required $H_1 \neq H_2$ with $\sigma(H_1) = \sigma(H_2)$. □

Remark 1.3. Proposition 1.2 is the formal statement of what we call the *state illusion*: any program that exposes only $\sigma(H)$ to its user has silently discarded distinguishing information about the history H. The conventional debugger is an instrument for reconstructing H from $\sigma(H)$ after information has already been lost.

1.3 Consequences for Language Design

Programming languages that treat state as primary inherit the degeneracy of Proposition 1.2 as a structural feature. Race conditions arise because σ is non-injective on interleaved histories. Use-after-free vulnerabilities arise because σ conflates histories

that differ only in deallocation events. The auditing and provenance tools that practitioners add to production systems are, in each case, partial reconstructions of H from $\sigma(H)$.

The Spherepop design decision follows directly: if history is what is lost, history must be what is preserved.

Chapter 2

The Spherepop Inversion

State is a collapsed history. The collapse is the problem.

—

2.1 Motivation

Spherepop proposes the opposite ontological priority from conventional languages. Not State $\rightarrow$ History (history as debugging afterthought) but History $\rightarrow$ State (state as derived projection).

2.2 The World Structure

Definition 2.1 (Event Alphabet)**.** *Let E be a set of* events*, partitioned into primitive generators:*

$$E = \{\text{Pop}\} \cup \{\text{Refuse}\} \cup \{\text{Bind}\} \cup \{\text{Collapse}\} \cup E_\lambda$$

where E_λ contains the lambda-calculus events $\{\text{LamIntro}, \text{Apply}\}$ *and scope markers.*

Notation 2.2. Throughout, Ω_0 denotes the *initial finite option space* — the full set of possibilities available at the start of a computation. All option spaces Ω appearing in subsequent definitions satisfy $\Omega \subseteq \Omega_0$ and $|\Omega_0| < \infty$.

Definition 2.3 (Spherepop World)**.** *A* Spherepop world *is a pair*

$$W = (H, \Omega)$$

where $H \in \mathcal{H}$ is a history (finite event sequence) and $\Omega \subseteq \Omega_0$ is the current option space *(a finite set of available symbols), derived from some initial option space Ω_0.*

Definition 2.4 (Commitment Operator)**.** *The* commitment operator *for symbol $x \in \Omega$ is*

$$P_x : (H, \Omega) \;\mapsto\; (H +\!\!+ [\text{Pop}(x)],\; \Omega \setminus \{x\})$$

where $\oplus$ denotes history concatenation. P_x is undefined when $x \notin \Omega$.

Definition 2.5 (Refusal Operator)**.** *The* refusal operator *for symbol x with reason r is*

$$R_{x,r} : (H, \Omega) \;\mapsto\; (H +\!\!+ [\text{Refuse}(x, r)],\; \Omega).$$

Note that Ω is unchanged*: refusal records inadmissibility without foreclosing structural availability.*

Definition 2.6 (Bind and Collapse Operators)**.**

$$B_{a,b} : (H, \Omega) \;\mapsto\; (H +\!\!+ [\text{Bind}(a, b)],\; \Omega)$$

$$C_{x,c} : (H, \Omega) \;\mapsto\; (H +\!\!+ [\text{Collapse}(x, c)],\; \Omega)$$

where c is a collapse rule (see Definition 8.1).

2.3 The Conservation Law

Theorem 2.7 (Conservation of Possibility)**.** *For any legal execution sequence $W_0 \xrightarrow{e_1} W_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} W_n$ where each step is one of P_x, $R_{x,r}$, $B_{a,b}$, or $C_{x,c}$:*

(*i*) *$|H_t|$ is strictly monotonically increasing: $|H_{t+1}| = |H_t| + 1$.*

(*ii*) *$|\Omega_t|$ is monotonically non-increasing.*

(*iii*) *Under pure Pop dynamics, $|H_t| + |\Omega_t| = |\Omega_0|$ for all t.*

Proof. (i) Every operator appends exactly one event to H, so $|H_{t+1}| = |H_t| + 1$.

(ii) P_x removes x from Ω; the remaining operators do not modify Ω. Hence $|\Omega_{t+1}| \leq |\Omega_t|$.

(iii) Under pure Pop dynamics, every step removes exactly one element from Ω and appends one event to H. Starting from $|H_0| = 0$ and $|\Omega_0|$, after t steps $|H_t| = t$ and $|\Omega_t| = |\Omega_0| - t$, giving $|H_t| + |\Omega_t| = |\Omega_0|$. □

Remark 2.8. Theorem 2.7 is the Spherepop analogue of a conservation law in physics. It says that possibility is not created or destroyed — it is either consumed (by Pop) or documented as inadmissible (by Refuse). The total "computational material" $|H_t| + |\Omega_t|$ is invariant under pure commitment dynamics.

Implementation consequence: `core/world.rs`

The conservation law forces the World to own both `history` and `options` as co-equal fields. Observable state is absent from the struct — it is always derived on demand.

```
pub struct World {
    pub options: OptionSpace,
    pub history: History,
}
```

The absence of a `state` field is the mechanization of Theorem 2.7: observable state is never fundamental.

Part II

Histories Before Computation

Chapter 3

The Algebra of Histories

3.1 Motivation

Before introducing types, evaluators, or compilers, we must characterize the mathematical structure of the objects that are primary in our ontology. Histories are not merely lists of events. They form a category, and the category structure constrains what operations on histories are meaningful.

3.2 Definitions

Definition 3.1 (History)**.** *A* history *over event alphabet E is a finite sequence $H = [e_1, e_2, \ldots, e_n]$ with each $e_i \in E$. The empty history is denoted $\varepsilon = []$.*

Definition 3.2 (History Concatenation)**.** *For histories $H_1 = [e_1, \ldots, e_m]$ and $H_2 = [f_1, \ldots, f_n]$,*

$$H_1 +\!\!+ H_2 = [e_1, \ldots, e_m, f_1, \ldots, f_n].$$

Definition 3.3 (History Monoid)**.** *The triple $(\mathcal{H}, +\!\!+, \varepsilon)$ where $\mathcal{H} = \bigcup_{n \geq 0} E^n$ is the* history monoid*.*

Proposition 3.4. *$(\mathcal{H}, +\!\!+, \varepsilon)$ is a free monoid over E.*

Proof. Associativity of $\oplus$ follows from list concatenation associativity. ε is the left and right identity. Freeness: every history H is uniquely determined by its event sequence, and no non-trivial relation among generators holds. □

3.3 The History Category

Definition 3.5 (History Category **Hist**)**.** *Define the category* **Hist** *as follows:*

- Objects: *option spaces* $\Omega \subseteq \Omega_0$.
- Morphisms: $\mathrm{Hom}(\Omega, \Omega')$ *is the set of histories* H *such that starting from option space* Ω*, executing* H *yields* Ω'.
- Composition: $H_2 \circ H_1 = H_1 +\!\!+ H_2$ (*execute* H_1 *first, then* H_2).
- Identity: $\mathrm{id}_\Omega = \varepsilon$.

Proposition 3.6. Hist *is a well-defined category.*

Proof. Associativity of composition follows from associativity of $\oplus$. ε is the identity morphism since $H +\!\!+ \varepsilon = H = \varepsilon +\!\!+ H$. □

Remark 3.7. Hist is in fact a *free strict monoidal category*: tensor product of morphisms is history concatenation (Meld), and the monoidal unit is the empty option space with empty history. This is much stronger than the simple monoid observation: it equips histories with a compositional structure that supports parallel as well as sequential combination.

Implementation consequence: `core/history.rs`

The free monoid structure forces `append` (extend by one generator) and `meld` (monoidal composition of two histories) as the only legitimate history operations. The absence of `remove` and `undo` is the mechanization of freeness: there are no relations among generators.

```
pub fn append(&mut self, event: Event) {
    self.events.push(event);    // extend by
        one generator
}
pub fn meld(&mut self, other: &History) {
```

```
        self.events.extend(other.events.iter().
            cloned()); // monoidal +
    }
```

History equality $H_1 = H_2$ is the primary criterion of program identity. This is implemented as structural list equality.

Chapter 4

Operational Semantics

4.1 Motivation

A formal operational semantics transforms the philosophical commitments of Chapter 2 into mathematically precise transition rules. Every execution step is a world-state transition; the full execution is a path in the transition system.

4.2 Small-Step Transition Rules

We write $(H, \Omega) \xrightarrow{\alpha} (H', \Omega')$ for a single step labelled α.

Pop (commitment):

$$\frac{x \in \Omega}{(H,\ \Omega) \xrightarrow{\text{pop}(x)} (H +\!\!+ [\text{Pop}(x)],\ \Omega \setminus \{x\})} \quad [\textsc{Pop}]$$

Refuse (documented inadmissibility):

$$\frac{}{(H,\ \Omega) \xrightarrow{\text{refuse}(x,r)} (H +\!\!+ [\text{Refuse}(x, r)],\ \Omega)} \quad [\textsc{Refuse}]$$

Note: [Refuse] has no premise — any symbol may be refused. The effect on the option space is nil; the effect on admissibility is non-nil (see Definition 4.3).

Bind (dependency declaration):

$$\frac{}{(H,\ \Omega) \xrightarrow{\text{bind}(a,b)} (H +\!\!+ [\text{Bind}(a, b)],\ \Omega)} \quad [\textsc{Bind}]$$

Collapse (observation under rule c):

$$\frac{\Gamma \vdash t : \text{Admissible}(T)}{(H,\ \Omega) \xrightarrow{\text{collapse}(x,c)} (H +\!\!+ [\text{Collapse}(x, c)],\ \Omega)} \quad [\textsc{Collapse}]$$

The premise $\Gamma \vdash t : \text{Admissible}(T)$ is the type-theoretic admissibility guard (Chapter 6). Collapse requires an admissibility certificate.

4.3 Admissibility as a Set-Valued Function

Definition 4.1 (Admissible Futures)**.** *For a history H, the set of* admissible future symbols *is*

$$\mathcal{A}(H) = \Omega_0 \setminus \{x \mid \exists r.\, \text{Refuse}(x, r) \in H\}.$$

Definition 4.2 (Inadmissibility Indicator)**.**

$$\mathcal{J}(H, x) = 1 - \chi_{\mathcal{A}(H)}(x)$$

where χ_A is the indicator function of set A.

Definition 4.3 (Admissibility Contraction)**.** *Let $H' = H +\!\!+ [\text{Refuse}(x, r)]$. Then $\mathcal{A}(H') \subseteq \mathcal{A}(H)$, and specifically $x \notin \mathcal{A}(H')$ while x may have been in $\mathcal{A}(H)$.*

Remark 4.4. The asymmetry between [Pop] and [Refuse] is critical: Pop contracts the *structural* option space Ω. Refuse contracts the *admissibility* set $\mathcal{A}(H)$ without touching Ω. Two paths through possibility space may have the same Ω while having different $\mathcal{A}(H)$ — they are structurally equivalent but semantically distinguished.

Part III

Types as Admissibility Certificates

Chapter 5

From Sets to Reachability Certificates

5.1 Motivation

The standard interpretation of types — a type is a set of admissible values — is adequate for state-centric languages. It is insufficient for Spherepop.

Consider what a type must certify in a history-primary setting. A value v of type T is not merely a member of a set. It is the result of a path through history that reached v without violating any constraint represented by T. The type is not a description of v at a moment; it is a certificate about the history that produced v.

5.2 Definitions

Definition 5.1 (Spherepop Types)**.** *The type language of Spherepop is defined inductively:*

$$
\begin{aligned}
T ::= \ & \mathrm{Unit} \mid \mathrm{Never} \mid X \quad (X \text{ a type variable}) \\
& \mid \Pi(x : T_1).T_2 \quad (\textit{dependent function type}) \\
& \mid \mathrm{Admissible}(T) \quad (\textit{admissibility certificate}) \\
& \mid \mathrm{Refused}(T,\, r) \quad (\textit{documented inadmissibility}) \\
& \mid \mathrm{Collapsed}(T,\, c) \quad (\textit{sealed observation under rule } c) \\
& \mid \mathrm{Process}(T_1 \rightsquigarrow T_2)
\end{aligned}
$$

The simple function type $T_1 \to T_2$ is notation for $\Pi(\_ : T_1).T_2$.

Remark 5.2. The distinctive types are $\mathrm{Admissible}(T)$, $\mathrm{Refused}(T,\, r)$,

and Collapsed$(T,\ c)$. They carry not just the inner type but a *certificate*:

- Admissible(T): the path that produced this term was admissible.
- Refused$(T,\ r)$: the term was encountered but found inadmissible for reason r. The reason is part of the *type* — two terms refused for different reasons have genuinely different types.
- Collapsed$(T,\ c)$: an admissible term was observed under collapse rule c and the observation is sealed. The rule c is part of the type.

Chapter 6

The Admissibility Type System

6.1 Typing Judgements

We write $\Gamma \vdash t : T$ for the standard typing judgement (term t has type T in context Γ) and $\Gamma \vdash_{\mathcal{A}} t : T$ for the *admissibility judgement* (the path to t was admissible under Γ).

6.2 Typing Rules

6.2.1 Pop: Issuing an Admissibility Certificate

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash \text{pop}(t) : \text{Admissible}(T)} \quad [\text{T-Pop}]$$

Pop transforms any typed term into an admissible term. It is the mechanism by which structural commitment produces a reachability certificate.

6.2.2 Refuse: Recording Inadmissibility with Reason

$$\frac{\Gamma \vdash t : T \quad r \in \mathsf{RefusalReason}}{\Gamma \vdash \text{refuse}(t, r) : \text{Refused}(T,\, r)} \quad [\text{T-Refuse}]$$

Refusal is typed: the type carries the reason. A term refused for ConstraintViolation(c) has a type that is *formally distinct* from one refused for NotAvailable.

6.2.3 Collapse: Sealing an Admissible Observation

$$\frac{\Gamma \vdash t : \text{Admissible}(T) \quad c \in \mathsf{CollapseRule}}{\Gamma \vdash \text{collapse}(t, c) : \text{Collapsed}(T,\, c)} \quad [\text{T-Collapse}]$$

Collapse requires an admissibility certificate. A term whose type is not Admissible$(\cdot)$ cannot be collapsed — this is a hard type error, not a runtime exception.

6.2.4 Bind: Recording a Dependency

$$\frac{\Gamma \vdash a : T_1 \quad \Gamma \vdash b : T_2}{\Gamma \vdash \mathsf{bind}(a, b) : \mathsf{Process}(T_1 \rightsquigarrow T_2)} \quad [\text{T-Bind}]$$

6.2.5 Lambda and Application

$$\frac{\Gamma, x : T_1 \vdash b : T_2}{\Gamma \vdash \lambda x : T_1 . b : \Pi(x : T_1).T_2} \quad [\text{T-Lam}]$$

$$\frac{\Gamma \vdash f : \Pi(x : T_1).T_2 \quad \Gamma \vdash a : T_1}{\Gamma \vdash f\, a : T_2[x := a]} \quad [\text{T-App}]$$

6.2.6 Open- and Closed-World Variable Rules

$$\frac{(x : T) \in \Gamma}{\Gamma \vdash x : T} \quad [\text{T-Var}]$$

$$\frac{x \notin \Gamma \quad \Gamma \text{ is open-world}}{\Gamma \vdash x : \mathsf{Unit}} \quad [\text{T-Var-Open}]$$

$$\frac{x \notin \Gamma \quad \Gamma \text{ is closed-world}}{\Gamma \nvdash x : T \text{ for any } T} \quad [\text{T-Var-Closed}]$$

6.3 Principal Theorems

Theorem 6.1 (Collapse Soundness)**.** *If* $\Gamma \vdash \mathsf{collapse}(t, c) : T$ *then* $\Gamma \vdash t : \mathsf{Admissible}(T')$ *for some* T'*, and* $T = \mathsf{Collapsed}(T', c)$.

Proof. By inversion on rule [T-Collapse]: the only rule that derives a Collapse judgement requires the premise $\Gamma \vdash t : \mathsf{Admissible}(T)$, so the inner term must have an admissible type, and the collapse type is $\mathsf{Collapsed}(T, c)$ by definition. □

Theorem 6.2 (Type Preservation)**.** *If* $\Gamma \vdash t : T$ *and* $t \to t'$*, then* $\Gamma \vdash t' : T$*.*

Proof sketch. By structural induction on the reduction relation $\to$. The key cases are [T-Pop], [T-Refuse], and [T-Collapse], each of which maps a reduction step to an admissibility transformation that is closed under the respective rule. Beta-reduction $(\lambda x.b)\, a \to b[x := a]$ requires a substitution lemma ($\Gamma, x : T_1 \vdash b : T_2$ and $\Gamma \vdash a : T_1$ implies $\Gamma \vdash b[x := a] : T_2[x := a]$) and a canonical forms lemma, whose proofs we omit; the current implementation realises only the core fragment and these results serve as proof obligations for future work. □

Theorem 6.3 (Progress)**.** *If* $\vdash t : T$ *(well-typed in the empty context) then either:*

(*i*) *t is a value, or*

(*ii*) *there exists t' such that $t \to t'$.*

Proof. By structural induction on t. Variable terms are values. Lambda terms are values (closures). pop, refuse, bind, and collapse of a well-typed inner term either reduce (if the inner term reduces) or produce a value (if the inner term is a value). □

Chapter 7

Open Worlds and Closed Worlds

> *The type checker's behavior depends not only on the term but on the assumed boundary of the world.*
>
> —

7.1 The Discovery

During implementation of the Spherepop type checker, a conflict emerged. The interpreter treated free variables as symbolic atoms: an unbound x evaluates to `Value::Symbol("x")`. The type checker, in its first version, rejected unbound variables as errors.

Programs that evaluated correctly consistently failed typechecking. The apparent "bug fix" was to make the type checker also treat free variables as having type `Unit`.

But this apparent bug conceals a genuine theoretical question: *which behavior is correct?* The answer is that both are correct, under different assumptions about the boundary of the world.

7.2 Formal Definitions

Definition 7.1 (Closed-World Context)**.** *A* closed-world context $\Gamma_c = \{x_1 : T_1, \ldots, x_n : T_n\}$ *satisfies:*

$$x \notin \Gamma_c \;\Rightarrow\; \Gamma_c \vdash x : \bot.$$

The absence of a declaration is the absence of the object. Rule [*T-Var-Closed*] *applies.*

Definition 7.2 (Open-World Context)**.** *An* open-world context $\Gamma_o = \Gamma_c \cup U$ *where* U *is the* unresolved possibility manifold (*symbols that have not been declared but have not been refused*) *satisfies:*

$$x \notin \Gamma_c \;\Rightarrow\; \Gamma_o \vdash x : \text{Unit}.$$

The absence of a declaration is undecided possibility. Rule [*T-Var-Open*] *applies.*

Proposition 7.3 (Monotonicity of Provability)**.**

$$\Gamma_c \subseteq \Gamma_o \quad \Rightarrow \quad \text{Provable}(\Gamma_c) \subseteq \text{Provable}(\Gamma_o).$$

Proof. Every typing derivation valid under Γ_c is valid under Γ_o, since Γ_o extends Γ_c and adds only the open-world variable rule, which derives additional types for previously-error terms. □

Remark 7.4. The open/closed world distinction is not an engineering convenience. It is the mechanization of an epistemological commitment: a closed-world type checker assumes complete knowledge of the reachable; an open-world type checker assumes partial knowledge. A language whose fundamental ontology is about possibility space cannot leave this distinction implicit.

Implementation consequence: `types/checker.rs`

```
pub enum TypeMode {
    ClosedWorld,  // unknown variable is a
        type error
    OpenWorld,    // unknown variable has
        type Unit
}
pub struct Context {
    bindings: HashMap<String, Type>,
    pub mode: TypeMode,
}
```

The mode is *preserved through* `extend`: a closed-world context cannot silently become open-world mid-derivation.

Part IV

Collapse as Quotient

Chapter 8

Observable State as a Quotient of History

8.1 Motivation

The original Spherepop design represented collapse as:

```
Event::Collapse(Symbol)
```

A collapse event was simply a record that a symbol had been committed. This was insufficient, and the insufficiency is philosophically illuminating.

Observable state depends on how you look. The same history, examined by different observers with different observational frameworks, yields different observable states. A collapse without a rule is a collapse into an unspecified notion of visibility.

8.2 Definitions

Definition 8.1 (Collapse Rule)**.** *A* collapse rule *is a function* $c : \mathcal{H} \to O_c$ *from the history monoid to an* observational space O_c*.*

Definition 8.2 (Observational Equivalence)**.** *Two histories* $H_1, H_2 \in \mathcal{H}$ *are* observationally equivalent under rule c*, written* $H_1 \sim_c H_2$*, if and only if* $c(H_1) = c(H_2)$*.*

Definition 8.3 (Observable State as Quotient)**.** *The* observable state space *under rule* c *is the quotient:*

$$O_c = \mathcal{H} / \sim_c .$$

An observable state is an equivalence class of histories.

Remark 8.4. Definition 8.3 is the mathematical heart of the monograph. Observable state is *not* a state. Observable state is a quotient. Two histories that are globally distinct may be observationally indistinguishable under a specific collapse rule. This is a feature, not a deficiency: the collapse rule specifies which distinctions are relevant to a given question.

8.3 The Four Canonical Rules

Definition 8.5 (Identity Rule)**.** *$c_I : \mathcal{H} \to \mathcal{H}$ is the identity: $c_I(H) = H$. Under Identity, every event is visible and $O_I \cong \mathcal{H}$.*

Definition 8.6 (LastWrite Rule)**.** *c_{LW} maps H to the most recent pop for each symbol, with subsequent refusals retroactively removing symbols:*

$$c_{LW}(H) = \{x \mapsto 1 \mid \mathrm{Pop}(x) \in H \text{ and } \nexists r.\mathrm{Refuse}(x, r) \in H \text{ after the last } \mathrm{Pop}(x)$$

Definition 8.7 (Accumulate Rule)**.** *$c_{\mathrm{acc}}(H)$ maps each symbol to the count of its Pop events:*

$$c_{\mathrm{acc}}(H)(x) = |\{i \mid e_i = \mathrm{Pop}(x)\}|.$$

Definition 8.8 (Projection Rule)**.** *For a label prefix L, the* projection rule *π_L restricts to events whose symbol names share prefix L:*

$$\pi_L(H) = [e_i \in H \mid \mathrm{sym}(e_i) \text{ starts with } L].$$

Proposition 8.9 (Coarsening Order)**.** *The collapse rules form a partial order by coarsening: $c_1 \leq c_2$ iff $\sim_{c_2} \subseteq \sim_{c_1}$ (finer equivalence = more informative observation). Under this order: c_I is the finest (most informative) rule; π_L is coarser than c_I; c_{LW} and c_{acc} are incomparable in general.*

8.4 Observational Equivalence Theorem

Theorem 8.10 (Observational Equivalence)**.** *For any collapse rule c and histories H_1, H_2: $H_1 \sim_c H_2$ if and only if $c(H_1) = c(H_2)$.*

Two programs whose interpreter and VM executions yield c-equivalent histories are observationally indistinguishable *under rule c.*

Corollary 8.11 (History Equivalence Implies Observational Equivalence)**.** *If $I(p) = V(C(p))$ (Replay Equivalence, Theorem 10.3), then for every collapse rule c:*

$$c(I(p)) = c(V(C(p))).$$

Implementation consequence: `core/observable.rs`

```
pub fn derive(history: &History, rule: &
    CollapseRule)
     -> ObservableState
pub fn equivalent(h1: &History, h2: &History
    ,
                  rule: &CollapseRule) ->
                     bool {
     Self::derive(h1, rule) == Self::derive(
        h2, rule)
}
```

Observable state is *never stored* in the World. It is always freshly derived. The function `observe` is a method, not a field, mechanizing the philosophical claim that observable state is a quotient.

Part V

Lambda Calculus as Derived Structure

Chapter 9

Beta-Reduction as History Contraction

9.1 Motivation

In the standard account, computation is explained in terms of functions. Every other computational phenomenon is a deviation from or extension of the basic function-application model.

Spherepop reverses this priority. Functions are not primitive. They are a convenient notation for a particular pattern in history: *deferred commitment*.

9.2 Beta-Reduction as Collapse

Definition 9.1 (Lambda History Encoding)**.** *A lambda abstraction $\lambda x.M$ applied to argument N generates the history fragment:*

$$[\text{LamIntro}(x),\ \text{Apply}(N)].$$

The application commits x to N, yielding the continued history beginning with $[\text{Pop}(x)]$ *(in the symbolic history of the parameter's use).*

Proposition 9.2 (Beta as Quotient)**.** *Beta reduction*

$$(\lambda x.M)\, N \ \rightarrow\ M[x := N]$$

is the projection of the history $[\text{LamIntro}(x), \text{Apply}(N)]$ *onto its observable state under the substitution collapse rule c_{subst}, defined by $c_{\text{subst}}(H)(x) = N$ whenever* $\text{Apply}(N) \in H$ *after* $\text{LamIntro}(x)$*.*

Remark 9.3. Proposition 9.2 gives a precise content to the informal claim that "beta reduction is a special case of collapse."

Function application is commitment: x can no longer range over any value other than N. The application event is a Pop of the parameter x from the option space of its possible values. The reduced term $M[x := N]$ is the observable state of the application history under c_{subst}.

9.3 Closures as Frozen Possibilities

Definition 9.4 (Closure)**.** *A* closure *is a pair* $(\lambda x.M, \rho)$ *where* ρ *is the environment* (*a partial history of binding events*) *at the time of lambda introduction.*

Remark 9.5. A closure is a *frozen possibility*: a partial future that will generate events when applied, suspended in a form that can be passed and stored. The environment ρ is the reachability context frozen at the moment of lambda introduction — the option space and historical commitments present when the abstraction was formed.

Implementation consequence: `eval/value.rs`

```
Value::Closure {
    param: Symbol,
    param_ty: Type,
    body: Box<Term>,
    env: Env,          // frozen reachability
        context
}
```

The `env` field is the mechanization of "frozen possibilities." It carries the historical commitments made at closure creation time. When the closure is applied, this context is restored.

Part VI

Compiling Histories

Chapter 10

The Failure of Traditional Compiler Correctness

10.1 Motivation

The standard notion of compiler correctness is:

A compiler is correct if the compiled program produces the same output as the interpreted program for every input.

This criterion is adequate for state-centric languages. It is insufficient for Spherepop.

Two programs that produce the same final value may have constructed radically different histories to get there, and those histories are part of what the program *is*.

10.2 History Equivalence as Correctness

Definition 10.1 (Interpreter and VM)**.** *Let $I : P \to \mathcal{H}$ be the interpreter function mapping programs to histories, and $V : \text{IR} \to \mathcal{H}$ the VM mapping event-IR blocks to histories, and $C : P \to \text{IR}$ the compiler.*

Definition 10.2 (Spherepop Compiler Correctness)**.** *A compiler C is* correct *if for all programs p and initial worlds W:*

$$I(p, W).\text{history} = V(C(p), W).\text{history}.$$

Output equivalence (same final value) is a strictly weaker criterion that is insufficient for Spherepop.

Theorem 10.3 (Replay Equivalence)**.** *The Spherepop compiler satisfies Definition 10.2 for all programs expressible in the Spherepop core calculus* (*Refuse, Bind, Collapse, and Seq compositions*)*.*

Proof sketch. By structural induction on program terms. Base cases: for each primitive operation (Refuse, Bind, Collapse, Pop), the compiler emits the corresponding `IrEvent` variant, and the VM step for that variant performs exactly the same World mutation as the interpreter's eval case for the corresponding `Term` variant. The history appended is therefore identical. Inductive case (Seq): each sub-term is compiled in order, and since histories compose by concatenation (the monoid operation), the composed history of the compiled form equals the composed history of the interpreted form by the inductive hypothesis. □

Remark 10.4. Theorem 10.3 is not merely an engineering correctness result. It is the mechanization of the claim that programs *are* their histories. The test suite verifies it as a first-class property:

```
assert_eq!(
    world_interp.history,
    world_compiled.history,
);
```

Every passing instance of this assertion is a mechanized proof obligation.

Chapter 11

Event IR: Instructions That Are Not Instructions

11.1 Motivation

Conventional intermediate representations encode instructions that mutate machine state. Spherepop IR encodes *events* — records of occurrences that persist as part of the history the VM is constructing.

Definition 11.1 (Event IR)**.** *The Spherepop intermediate representation is a sequence of* `IrEvent` *terms:*

$$\text{IR} ::= \text{Pop} \mid \text{Refuse}(r) \mid \text{Collapse}(c) \mid \text{Bind} \mid \text{Load}(x) \mid \text{Apply} \mid \cdots$$

Each event carries its reason or rule as a first-class argument, preserving the full certificate structure of the source term.

Proposition 11.2 (IR Faithfulness)**.** *The compilation function* $C : \text{Term} \to \text{IR}$ *is* faithful*: for every primitive term* t*,* $C(t)$ *contains exactly the event variants that* $I(t)$ *would append to the history, with identical arguments.*

Proof. By inspection of the compiler's `lower` function and the interpreter's `eval` function: each branch of `lower` emits the `IrEvent` variant that corresponds bijectively to the `Event` appended by the matching `eval` branch. □

Remark 11.3. The IR Faithfulness Proposition is what makes Theorem 10.3 (Replay Equivalence) provable. The VM does not "execute" IR events; it *reconstructs* the history that the interpreter would have built. The VM is a history reconstruction engine, not a state mutation engine.

Part VII

Category Theory and Deep Structure

Chapter 12

Histories as Morphisms

12.1 The History Category Revisited

We introduced **Hist** in Chapter 3 as the category of option spaces and histories. We now develop its structure more carefully in relation to observable state.

Definition 12.1 (Option-Space Transition Sequences)**.** *A transition sequence from Ω to Ω' is a history H such that executing H starting from Ω yields Ω'. This is the morphism $H : \Omega \to \Omega'$ in* **Hist***.*

Proposition 12.2 (**Hist** as a Free Strict Monoidal Category)**.** $(\mathbf{Hist}, \oplus, \varepsilon, \otimes)$ *is a free strict monoidal category where:*

- *Tensor product $H_1 \otimes H_2$ is the Meld operation (parallel composition of histories).*
- *Monoidal unit is the empty option space with empty history.*
- *Sequential composition is $\oplus$ (Definition 3.5).*

Chapter 13

Collapse Functors

13.1 Collapse as a Functor Hist $\to$ $\mathbf{Obs}_c$

Definition 13.1 (Observable-State Category $\mathbf{Obs}_c$)**.** *For a collapse rule c, define the category $\mathbf{Obs}_c$:*

- Objects: *observable states $o \in O_c$.*
- Morphisms: $\mathrm{Hom}(o_1, o_2)$ *is the set of observable transitions that transform o_1 into o_2 under rule c.*
- Composition: *sequential observable transition.*

Definition 13.2 (Collapse Functor)**.** *For a collapse rule c, define the* collapse functor $F_c : \mathbf{Hist} \to \mathbf{Obs}_c$ *by:*

$$F_c(\Omega) = c(\varepsilon_\Omega) \quad (\textit{object map})$$

where ε_Ω denotes the empty history at object Ω (i.e. the identity morphism at Ω in **Hist**).

$$F_c(H) = c(H) \quad (\textit{morphism map})$$

Theorem 13.3 (Functoriality of Collapse)**.** *F_c is a well-defined functor when the collapse rule c respects composition, i.e., when*

$$c(H_2 +\!\!+ H_1) = c(H_2) \circ c(H_1)$$

(up to the appropriate notion of composition in $\mathbf{Obs}_c$).

Proof. Identity preservation: $F_c(\varepsilon) = c(\varepsilon) = \mathrm{id}_{O_c}$ since the empty history yields the empty observable state, which is the identity transformation in $\mathbf{Obs}_c$.

Composition preservation:

$$F_c(H_2 \circ H_1) = F_c(H_1 +\!\!+ H_2) = c(H_1 +\!\!+ H_2).$$

When c respects composition: $c(H_1 +\!\!+ H_2) = c(H_2) \circ c(H_1) = F_c(H_2) \circ F_c(H_1)$. □

Remark 13.4. Theorem 13.3 elevates collapse from an implementation detail to a category-theoretic object. Collapse is not merely "reading off a value." It is a structure-preserving map between categories. The collapse rule specifies *which structures are preserved* — and different rules preserve different structures.

Remark 13.5. The four canonical rules have different functoriality properties: c_I (Identity) is trivially functorial (it is the identity functor on **Hist**). c_{acc} (Accumulate) is functorial when the observational composition is addition of count-maps. π_L (Projection) is functorial: projection of a concatenated history equals the concatenation of the projected histories. c_{LW} (LastWrite) is *not* generally functorial, since the last-write of a concatenated history is not the composition of the last-writes of its parts — a later segment can retroactively affect the observable state of an earlier one.

Part VIII

Representation and Reachability

Chapter 14

Filters Create Worlds

Every representation creates a world with its own notions of sameness, difference, adjacency, and reachability.

—

14.1 Motivation

The Spherepop collapse framework naturally connects to a broader claim about representations: *representations do not merely preserve or lose information — they reorganize the topology of distinguishability.*

14.2 The Santoro–Waghmare–Panaretos Connection

We interpret a recent result in functional data analysis through the lens of admissibility geometry.

Let $P, Q \in \mathcal{P}(X)$ be probability measures on a separable metric space. The kernel covariance embedding

$$P \;\mapsto\; S_P \;\mapsto\; \mathcal{N}(0, S_P)$$

maps distributions to Gaussian measures on a Hilbert space.

The result of Santoro, Waghmare, and Panaretos (2024) shows that for distinct continuous distributions, $\mathcal{N}(0, S_P)$ and $\mathcal{N}(0, S_Q)$ are mutually singular. The standard reading is that infinite-dimensional representations amplify tiny distinctions into maximal separations.

Definition 14.1 (Reachability Class)**.** *For a Gaussian measure $\mathcal{N}(0, S_P)$, define its* reachability class*:*

$$R(P) = \{Q : \mathcal{N}(0, S_Q) \sim \mathcal{N}(0, S_P)\}$$

where $\sim$ denotes measure equivalence (mutual absolute continuity).

Theorem 14.2 (Topological Separation via Representation)**.** *Under the conditions of the Santoro–Waghmare–Panaretos theorem, for distinct continuous distributions $P \neq Q$:*

$$d(P, Q) \ll 1 \quad (\textit{metric closeness downstairs})$$

yet

$$R(P) \cap R(Q) = \emptyset \quad (\textit{topological separation upstairs}).$$

The kernel embedding converts metric closeness into topological separation.

Remark 14.3. Theorem 14.2 is related to the Feldman–Hájek theorem, which establishes that two Gaussian measures are either equivalent ($\mu \sim \nu$) or mutually singular ($\mu \perp \nu$) — there is no continuum. In the reachability class language: $R(P) = \{P\}$ for almost all P.

14.3 The Representation-Ontology Reading

The standard interpretation of Theorem 14.2 is epistemological: *distinct distributions become perfectly distinguishable.*

The CLIO/Spherepop interpretation is ontological: *representation can reorganize possibility space itself.*

Proposition 14.4 (Collapse Rules as Reachability Filters)**.** *Spherepop collapse rules are computationally analogous to the kernel embedding in Theorem 14.2. The rule c maps the history space $\mathcal{H}$ to an observable state space O_c, changing which histories are distinguishable. Two histories in the same equivalence class $[H]_{\sim_c}$ are* topologically identified *under rule c: they occupy the same stratum of observable state space.*

Remark 14.5. The parallel is precise. Before applying a collapse rule: histories may be metrically close (they differ in few events) yet topologically distinct (they produce different observable states under c). Conversely: histories that are metrically distant (many events differ) may be topologically identified (they produce the same observable state under c). The collapse rule, like the kernel embedding, changes *adjacency*, not just distance.

14.4 CLIO Projections and History Strata

Definition 14.6 (CLIO Projection)**.** *A CLIO projection $\pi : X \to M$ maps a representation space X to a manifold M of observational strata. In the Spherepop setting: $X = \mathcal{H}$ (history space), $M = O_c$ (observable state space under rule c), and $\pi = F_c$ (the collapse functor).*

Remark 14.7. Observable states under Projection rule π_L are precisely the CLIO strata corresponding to the label L. Histories become trajectories through stratum space. The collapse functor F_c is the formalization of CLIO's projection π.

Part IX

Programs as Reachability Sculptures

Chapter 15

The Codebase as a Minimal Working Ontology

15.1 Each Module as Ontological Necessity

The Spherepop codebase is organized to make philosophical dependencies visible. The following table summarizes the justification for each module:

Module	Ontological Necessity
`core/event.rs`	Events are primitive. They cannot be derived from types or values without circularity. The codebase must begin here.
`core/history.rs`	History is computational identity. The free monoid structure (append, meld; no remove) is the mechanization of irreversibility. History equality is the primary identity criterion.
`core/option_space.rs`	Possibility is a runtime structure, not a static specification. The conservation law requires both History and OptionSpace to coexist.
`core/observable.rs`	Observable state is derived, not primary. Separating it from History enforces the metaphysical priority claim.
`types/ty.rs`	Types must reflect admissibility structure. `Admissible`, `Refused`, and `Collapsed` are reachability certificates, not value sets.
`types/checker.rs`	Admissibility must be verified. `TypeMode` is the mechanization of the open/closed world epistemology.
`eval/interpreter.rs`	Evaluation is historical construction, not value computation. Returning `EvalResult{value, steps, admissible}` is forced.
`compiler/lower.rs`	Compilation must preserve possibility structure. Emitting `IrEvent` rather than `IrInstruction` is forced.
`ir/event_ir.rs`	The IR must be event-based because the target runtime is history-based. `Refuse(reason)` and `Collapse(rule)` carry certificates.
`vm/machine.rs`	Execution is replay, not transformation. The VM reconstructs history; cor-

15.2 The Implemented Milestones

v0.1 — History-first execution. World = History + OptionSpace; evaluation returns EvalResult alongside a World; the conservation law is enforced as a hard runtime invariant.

v0.2 — Documented refusal. `RefusalReason` is introduced as a first-class type, threading through events, types, values, and IR. Refusal is structured inadmissibility with a permanent certificate.

v0.3 — Collapse as quotient. `CollapseRule` specifies the observational framework. Observable state is derived by `ObservableState::c rule)`. Observational equivalence becomes rule-relative and formally testable.

v0.4 — TypeMode. `TypeMode::OpenWorld` and `TypeMode::ClosedWorld` are introduced. The type checker's behavior depends on this mode, mechanizing the philosophical discovery about world boundaries.

v0.5 — History equivalence as compiler correctness. The correctness criterion $I(p)$.history $= V(C(p))$.history is implemented as a first-class test obligation.

Chapter 16

The Thesis Defended

Computation is the progressive restriction of possibility, not the manipulation of values.

—

16.1 The Outrageous Claim Made Precise

The thesis sounds outrageous only as a slogan. When a programmer writes `x = 5`, they are not merely assigning a value. They are committing to a specific future — one in which the value of `x` is 5 — and foreclosing all futures in which x has any other value. The assignment is a Pop: a possibility removed from the option space.

When they write an `if-else` branch, they are partitioning possibility space: the condition is the criterion; the branches are the strata.

When they write a function call, they are deferring a commitment: the function is a frozen possibility, and the call is the moment of actualization.

The Spherepop framework makes this geometry *explicit*. Pop, Refuse, Bind, and Collapse are the explicit operations on possibility space that conventional languages hide behind assignment, branching, and function calls.

Proposition 16.1 (Expressive Correspondence)**.** *The following conventional constructs are expressible in terms of Spherepop primitives:*

- Variable assignment $x := v \approx \text{Pop}(v)$ *followed by* $\text{Bind}(x, v)$.

- Conditional branch *on predicate* $\phi \approx \text{Pop}(\phi^+)$ *or* $\text{Refuse}(\phi^+, \text{ConstraintVi}$ *followed by* $\text{Pop}(\phi^-)$.
- Function application $f(a) \approx \text{LamIntro}(x)$ *followed by* $\text{Apply}(a)$ (*a deferred Pop*).
- Exception throwing $\approx \text{Refuse}(t, r)$ (*documented inadmissibility without system destruction*).

16.2 Why This Changes Everything

In the conventional account, programs are descriptions of computations: sequences of instructions that tell the machine what to do.

In the Spherepop account, programs are *constructions within possibility space*: paths through the geometry of accessible futures.

The difference is not cosmetic. It affects what questions we can ask:

Conventional question	Spherepop question
What is the current state?	What remains reachable?
What is the output value?	What history was constructed?
Did the program terminate?	What possibilities were foreclosed?
Is there a bug?	Which commitment was inadmissible?
What did the function return?	Under what rule is the result observable?

The Spherepop questions are not more abstract. They are more *honest*: they expose what computation actually does to the geometry of possibility, rather than summarizing the result as a degenerate projection.

Chapter 17

Future Directions

17.1 Proof-Carrying Refusal

The `RefusalReason` type is a vocabulary for inadmissibility. It is not yet a proof system. A fully realized Spherepop type system would require `RefusalReason` to be a proof term: evidence that can be checked, not merely a label. This connects to the theory of certified refusal: a refusal is well-formed only if accompanied by a checkable argument for why the refused path was inadmissible.

17.2 Dependent Process Types

The current Process$(T_1 \rightsquigarrow T_2)$ type is first-order. A dependent process type $\Pi(x : T_1)$. Process$(x, f(x))$ would allow the type of the output to depend on the specific value consumed. This is the natural extension of dependent type theory into the process calculus setting.

17.3 Admissibility Lattices

The current admissibility structure is Boolean: a term is admissible or not. A more refined theory would recognize that admissibility is graded — some futures are more constrained than others, and the constraints form a lattice. This connects to the broader admissibility geometry program of RSVP and CLIO.

17.4 Collapse Functor Composition

The four canonical collapse rules are special cases of a more general family of functors $F_c : \mathbf{Hist} \to \mathbf{Obs}_c$. A theory of *functor composition* would characterize which combinations of rules are valid — for instance, when $F_{c_2} \circ F_{c_1}$ is itself a valid collapse functor.

17.5 Distributed Spherepop Runtimes

The current World is single-threaded. A distributed Spherepop runtime would coordinate multiple Worlds, each with their own histories, under a global consistency constraint. The consistency constraint must be expressed in terms of history equivalence under a shared collapse rule — not value consistency. The event sourcing and CRDT literatures provide partial models; the Spherepop-specific challenge is the certified-refusal structure.

Coda: The Implementation as Philosophical Argument

> *The implementation did not merely realize the theory; it exposed which distinctions the theory was still hiding.*
>
> —

The central claim of this monograph is not that Spherepop is a good programming language, though we believe it is an interesting one. The central claim is that *taking a philosophical position seriously enough to implement it is a form of philosophical argument.*

The Spherepop implementation is not a demonstration of a pre-formed theory. It is a test of a theory's coherence. The theory said that histories are primary, types are admissibility certificates, and refusal is documented inadmissibility. The implementation asked: what does this actually mean in executable form?

The four implementation discoveries were not predicted by the initial framework. They were forced by the attempt to mechanize it.

The original `Refuse(Symbol)` gestured at documented inadmissibility. It took implementation pressure to reveal that a symbol alone documents nothing: the reason is the document.

The original notion of observable state gestured at the quotient structure. It took `World::observe` to force the question: observable under what rule?

The original type checker gestured at world-relative knowledge. It took the free-variable conflict to force the question: which world are we assuming?

The original correctness criterion gestured at semantic preser-

vation. It took the compiler test to force the question: preservation of what, exactly?

Each discovery reveals a distinction the philosophical framework was gesturing at without making precise. Only in the attempt to mechanize — to force the philosophy to compile — did the gestures become definitions.

This is the deepest lesson of ontology-driven language design. Philosophy can identify the right questions. Only implementation can force the right precision.

The Spherepop codebase is a philosophical argument written in Rust. Like all arguments, it became more precise in the act of being made.

Part X

Limits, Gaps, and Open Problems

Chapter 18

The Observation Limit

Spherepop does not abolish the state illusion. It makes the quotient explicit, parameterised, and reversible whenever the underlying history is retained.

—

18.1 Motivation: A Reviewer's Challenge

A careful reader of this monograph will notice a tension. The document claims to solve the state illusion — the loss of historical information caused by projecting history to state. Yet every output of a Spherepop program is a *collapse*: a value derived by applying a rule to a history. The programmer observes the collapsed value, not the history. Is Spherepop not simply relocating the state illusion rather than resolving it?

This chapter addresses that challenge directly. We prove a fundamental limit — the *No Direct Observation Theorem* — that shows the challenge is not a defect in Spherepop but a mathematical necessity. We then distinguish precisely between two phenomena the challenge conflates: *state erasure* and *state illusion*. Spherepop eliminates the first. No computational system can eliminate the second.

18.2 Definitions

Definition 18.1 (Observation Map)**.** *An* observation map *for a collapse rule c is any function*

$$\phi : \mathcal{H} \to O$$

from histories to an observable space O.

Definition 18.2 (History-Faithful Observation)**.** *An observation map ϕ is* history-faithful *if it is injective: $\phi(H_1) = \phi(H_2) \Rightarrow H_1 = H_2$.*

18.3 The No Direct Observation Theorem

Theorem 18.3 (No Direct Observation)**.** *Let O be any set with $|O| < |\mathcal{H}|$. Then every observation map $\phi : \mathcal{H} \to O$ is non-injective: there exist distinct $H_1 \neq H_2$ with $\phi(H_1) = \phi(H_2)$. Consequently, no computational system with a finite observable space can directly verify the history it constructed.*

Proof. Since $\mathcal{H} = \bigcup_{n \geq 0} E^n$ is countably infinite and $|O| < |\mathcal{H}|$, by the pigeonhole principle ϕ cannot be injective. Therefore there exist $H_1 \neq H_2$ with $\phi(H_1) = \phi(H_2)$. □

Remark 18.4. Theorem 18.3 is the fundamental limit of computational observation. It says: *every observation is a quotient*. This is not a contingent limitation that better engineering can overcome. It is a cardinality constraint. A system that made its observation map injective — that recorded every distinction in full history space — would itself be a history. To observe computation from outside, one must compress. Every compression identifies distinct histories.

18.4 Erasure vs Illusion

The Theorem 18.3 suggests a distinction the literature elides:

Definition 18.5 (State Erasure)**.** *A computational system suffers* state erasure *if it irrecoverably discards the history H after computing $\sigma(H)$. That is, after execution, H is unavailable and only $\sigma(H)$ remains.*

Definition 18.6 (State Illusion)**.** *A computational system presents a* state illusion *to its programmers if they can only observe $\sigma(H)$ and not H directly. By Theorem 18.3,* some *state illusion is unavoidable for any non-trivial observable space.*

Proposition 18.7 (What Spherepop Eliminates)**.** *Spherepop eliminates state erasure: the history H is retained in* `world.history` *and is available to any observer.*

Spherepop cannot and does not eliminate the state illusion: the programmer's primary observable is $c(H)$, not H, so distinct histories that agree under c remain indistinguishable from the perspective of that observation.

Remark 18.8. The correct claim is therefore:

Spherepop does not abolish the state illusion. It makes the quotient explicit, parameterised, and history-preserving.

Concretely: two histories $H_1 \sim_c H_2$ are observationally equivalent under rule c, but both are retained. A programmer who suspects their collapsed observation is misleading can switch to a finer collapse rule c' (where $\sim_{c'} \subsetneq \sim_c$), recover additional distinctions, and potentially identify which of many c-equivalent histories they actually constructed. In conventional systems, this recovery is impossible because H was discarded.

18.5 Rule Refinement as Disambiguation

Definition 18.9 (Rule Refinement Order)**.** *Collapse rule c_1 is* finer than *c_2, written $c_1 \preceq c_2$, if $H_1 \sim_{c_1} H_2 \Rightarrow H_1 \sim_{c_2} H_2$: every distinction preserved by c_1 is also preserved by c_2. The Identity rule c_I is the finest (maximally informative) rule; the trivial rule $c_\top$ mapping all histories to a single point is the coarsest.*

Proposition 18.10 (Disambiguation by Refinement)**.** *Let $H_1 \sim_c H_2$ (the histories are indistinguishable under rule c) and $H_1 \neq H_2$. Then there exists a finer rule $c' \preceq c$ such that $c'(H_1) \neq c'(H_2)$. In the limit, c_I always disambiguates since $c_I(H) = H$ exactly.*

Proof. Take $c' = c_I$: since $H_1 \neq H_2$ and $c_I(H) = H$, we have $c_I(H_1) = H_1 \neq H_2 = c_I(H_2)$. Therefore c_I is a finer rule that disambiguates H_1 and H_2. □

This proposition formalises the engineering value of history preservation. A system that erases histories cannot apply a finer rule later. A Spherepop system that preserves histories can always refine its observation by switching to a finer collapse rule.

Chapter 19

When Does Observational Equivalence Imply History Equivalence?

19.1 Motivation

The deepest open question in the Spherepop framework is the converse of the collapse functor:

$H_1 = H_2 \Rightarrow c(H_1) = c(H_2)$ for every c. (trivially true)

$c(H_1) = c(H_2) \Rightarrow H_1 = H_2$? (when does this hold?)

When does observational equivalence under a collapse rule imply history identity? Understanding the boundary is the deepest open mathematical problem in the framework and connects to information theory, inverse problems, and kernel geometry.

19.2 Definitions

Definition 19.1 (Observational Completeness)**.** *A collapse rule c is* observationally complete *if it is injective: $c(H_1) = c(H_2) \Rightarrow H_1 = H_2$.*

Definition 19.2 (Collapse Kernel)**.** *The* kernel *of a collapse rule c is*

$$\ker(c) = \{(H_1, H_2) \in \mathcal{H} \times \mathcal{H} \mid c(H_1) = c(H_2)\}.$$

Observational completeness holds iff $\ker(c) = \{(H, H) \mid H \in \mathcal{H}\}$ (*the diagonal*).

19.3 Completeness Results for Canonical Rules

Proposition 19.3 (Identity Rule is Observationally Complete)**.** *The Identity rule c_I with $c_I(H) = H$ is observationally complete.*

Proof. $c_I(H_1) = H_1$ and $c_I(H_2) = H_2$; if $c_I(H_1) = c_I(H_2)$ then $H_1 = H_2$. □

Proposition 19.4 (LastWrite is Not Observationally Complete)**.** *The LastWrite rule c_{LW} is not observationally complete.*

Proof. Consider the event alphabet $E = \{\mathtt{pop}(x), \mathtt{pop}(y)\}$. Let $H_1 = [\mathtt{pop}(x), \mathtt{pop}(y)]$ and $H_2 = [\mathtt{pop}(y), \mathtt{pop}(x), \mathtt{pop}(y)]$. Under c_{LW}: $c_{LW}(H_1)$ maps $x \mapsto 1, y \mapsto 1$ and $c_{LW}(H_2)$ maps $x \mapsto 1, y \mapsto 1$ (last pop of y wins). Thus $c_{LW}(H_1) = c_{LW}(H_2)$ but $H_1 \neq H_2$. □

Proposition 19.5 (Accumulate is Not Observationally Complete)**.** *The Accumulate rule c_{acc} is not observationally complete.*

Proof. Let $H_1 = [\mathtt{pop}(x), \mathtt{pop}(y)]$ and $H_2 = [\mathtt{pop}(y), \mathtt{pop}(x)]$. Both map to $\{x \mapsto 1, y \mapsto 1\}$ under c_{acc}, but $H_1 \neq H_2$ (different event ordering). □

Remark 19.6. The Accumulate rule identifies histories that are permutations of one another: $c_{\text{acc}}(H_1) = c_{\text{acc}}(H_2)$ iff H_1 and H_2 have the same multiset of events. This corresponds to an *order-forgetful* observation — one that records what happened but not when, relative to other events.

19.4 The Distinguishability Topology

Definition 19.7 (Collapse-Induced Topology)**.** *For a collapse rule $c : \mathcal{H} \to O_c$, define the c-*induced topology *on $\mathcal{H}$ as the initial topology with respect to c:*

$$\tau_c = \{c^{-1}(U) \mid U \subseteq O_c\}.$$

This makes c continuous (in the discrete topology on O_c) and τ_c is the coarsest topology that c is continuous with respect to.

Definition 19.8 (History Distance Under Rule c)**.**

$$d_c(H_1, H_2) = \mathbf{1}[c(H_1) \neq c(H_2)]$$

is the observational distance *between H_1 and H_2 under rule c. This is a pseudometric: $d_c(H_1, H_2) = 0$ iff $H_1 \sim_c H_2$.*

Proposition 19.9 (Rule Change Induces Topology Change)**.** *For rules $c_1 \preceq c_2$ (where c_1 is finer),*

$$\tau_{c_2} \subseteq \tau_{c_1}.$$

Refining the collapse rule strictly expands the induced topology on $\mathcal{H}$, making more distinctions between histories visible.

Proof. If $c_1 \preceq c_2$, then $H_1 \sim_{c_1} H_2 \Rightarrow H_1 \sim_{c_2} H_2$, so the equivalence classes of $\sim_{c_2}$ are coarser than those of $\sim_{c_1}$. Every open set in τ_{c_2} (a union of $\sim_{c_2}$-classes) is also a union of $\sim_{c_1}$-classes, hence in τ_{c_1}. □

Remark 19.10. Proposition 19.9 is the formal content of the claim that "changing the collapse rule changes the topology of distinguishability." It connects the Spherepop collapse framework to the kernel embedding discussion of Chapter 14: in both cases, a change of representation (kernel embedding; collapse rule) changes the topology on the underlying space (probability distributions; histories).

The kernel embedding moves pairs (P, Q) from the same topology class (metric closeness) to different topology classes (mutual singularity). Collapse rule refinement moves pairs (H_1, H_2) from the same equivalence class to different equivalence classes. Both are instances of the same structural operation: representation induces topology, and changing representation changes which distinctions are topologically visible.

19.5 Open Problem: The Inverse Problem

Notation 19.11. We write $[H]_c = \{H' \in \mathcal{H} \mid H' \sim_c H\}$ for the equivalence class of H under rule c.

The inverse problem for a collapse rule c is: *given an observed state $o \in O_c$, characterise $c^{-1}(o)$.*

This is the formal version of the question "which histories could have produced this observation?"

Definition 19.12 (Inverse-Problem Fibre)**.** *For $o \in O_c$, the* fibre *of o is $c^{-1}(o) = \{H \in \mathcal{H} \mid c(H) = o\}$.*

For the Accumulate rule, $c_{\text{acc}}^{-1}(m)$ is the set of all histories whose event multiset is m — an infinite set (any permutation of m extended by arbitrarily many Refuse or Bind events that don't change counts).

Characterising fibres for general rules, and determining when a fibre contains a unique history (i.e. when the rule is injective on a subclass of histories), is the primary open mathematical problem in the framework.

Chapter 20

Category Theory as Engine, Not Commentary

20.1 Motivation

A reviewer of this work identified that the category theory in Chapters 11–12 is *descriptive* rather than *generative*: no theorem currently requires categorical machinery to state or prove. This chapter corrects that deficiency by establishing results that are naturally and essentially categorical.

20.2 The Collapse Composition Theorem

The collapse functor framework becomes non-trivial when we ask about *composition of collapse rules*: what happens when we apply one collapse to the result of another?

Definition 20.1 (Rule Composition)**.** *For rules $c_1 : \mathcal{H} \to O_1$ and $c_2 : O_1 \to O_2$, the* composed rule *is*

$$(c_2 \circ c_1)(H) = c_2(c_1(H)).$$

Theorem 20.2 (Collapse Composition)**.** *If $F_{c_1} : \mathbf{Hist} \to \mathbf{Obs}_{c_1}$ and $F_{c_2} : \mathbf{Obs}_{c_1} \to \mathbf{Obs}_{c_2}$ are both valid collapse functors, then their composition $F_{c_2} \circ F_{c_1}$ is a valid collapse functor $F_{c_2 \circ c_1} : \mathbf{Hist} \to \mathbf{Obs}_{c_2 \circ c_1}$, and*

$$F_{c_2 \circ c_1}(H) = F_{c_2}(F_{c_1}(H)).$$

Proof. Composition of functors is a functor: identity preservation and composition preservation follow directly from those properties of F_{c_1} and F_{c_2}. The equation $F_{c_2 \circ c_1}(H) = F_{c_2}(F_{c_1}(H))$

holds by definition of rule composition and the morphism map of F_c. □

Remark 20.3. Theorem 20.2 establishes collapse rules as a composable algebraic structure. A sequence of observations $c_1, c_2, \ldots, c_n$ can be composed into a single rule $c_n \circ \cdots \circ c_1$. This is the formal basis for *observational pipelines*: a program may project history through a sequence of finer-to-coarser rules, each stripping away distinctions that are irrelevant to the next stage.

20.3 The Observation Adjunction

The deepest categorical result in the framework is an adjunction between the functor that *constructs* histories and the functor that *observes* them.

Definition 20.4 (History Construction Functor)**.** *Let* **Terms** *be the category whose objects are Spherepop terms and whose morphisms are reduction sequences. Define* $G : \mathbf{Terms} \to \mathbf{Hist}$ *by*

$$G(t) = I(t).\mathit{history}$$

(*the history produced by interpreting* t).

Definition 20.5 (Observation Functor)**.** *For a fixed collapse rule* c*, define* $\mathcal{O}_c : \mathbf{Hist} \to \mathbf{Obs}_c$ *by*

$$\mathcal{O}_c(H) = c(H) = F_c(H).$$

Theorem 20.6 (History-Observation Adjunction)**.** *For the Identity rule* c_I*,* $\mathcal{O}_{c_I} \dashv G^{\mathrm{op}}$ *in the sense that there is a natural bijection*

$$\mathbf{Hist}(G(t), H) \;\cong\; \mathbf{Terms}(t,\, G^{-1}(H))$$

whenever $G^{-1}(H)$ *is defined* (*i.e. when* H *is the history of some term under the interpreter*).

Proof sketch. Under c_I, observation is the identity functor on **Hist**. The adjunction then reduces to the statement that morphisms in

Hist from $G(t)$ to H correspond to reduction sequences from t to a term whose interpretation produces H. This is the categorical statement of the Replay Equivalence (Theorem 10.3): the interpreter and compiler produce histories that are morphisms in **Hist**. A complete proof requires a full formalisation of **Terms** beyond the scope of the current implementation. □

20.4 What Is Now Category-Theoretically Forced

Proposition 20.7 (Theorems Requiring Categorical Language)**.** *The following results are most naturally stated in categorical terms:*

1. *Theorem 20.2 (Collapse Composition) — requires functor composition.*
2. *Theorem 20.6 (History-Observation Adjunction) — requires adjunction.*
3. *Proposition 19.9 (Rule Change Induces Topology Change) — requires the category of topological spaces and continuous maps.*
4. *Theorem 13.3 (Functoriality of Collapse) — requires the definition of* **Hist** *as a category.*

The category theory is now generative*: these theorems cannot be stated, let alone proved, without the categorical framework introduced in Chapter 20.*

Chapter 21

Strengthening the Kernel Connection

21.1 Motivation

Chapter 14 (Filters Create Worlds) presented the connection between Spherepop collapse rules and the Santoro–Waghmare–Panaretos theorem as an analogy. The reviewer correctly identified that the analogy was loose: the kernel theorem is a *limitative* result (separation is unavoidable); the collapse framework is a *design freedom* (the programmer chooses the rule).

This chapter sharpens the connection by making it a theorem about *representation-induced topological change* that applies uniformly to both settings.

21.2 A Unified Framework

Definition 21.1 (Representation Map)**.** *A* representation map *is any function $\rho : X \to Y$ from a source space X to a target space Y, equipped with the initial topology $\tau_\rho = \{\rho^{-1}(U) \mid U \subseteq Y\}$ on X.*

Definition 21.2 (Representation-Induced Topology)**.** *For a representation map $\rho : X \to Y$:*

- *$x_1 \sim_\rho x_2$ iff $\rho(x_1) = \rho(x_2)$ (equivalence induced by ρ).*
- *The* separation power *of ρ is $|\{[x]_\rho \mid x \in X\}| = |Y|$ (the number of equivalence classes, bounded by the size of Y).*

Theorem 21.3 (Representation Changes Topology)**.** *Let $\rho_1, \rho_2 : X \to Y$ be two representation maps with $\tau_{\rho_1} \neq \tau_{\rho_2}$. Then there exist*

$x_1, x_2 \in X$ *such that:*

$$x_1 \sim_{\rho_1} x_2 \quad \textit{but} \quad x_1 \nsim_{\rho_2} x_2$$

or vice versa. That is, the two representations disagree on whether x_1 and x_2 are the same object.

Proof. If $\tau_{\rho_1} \neq \tau_{\rho_2}$, there exists an open set $U \in \tau_{\rho_1} \setminus \tau_{\rho_2}$ (WLOG). Then $U = \rho_1^{-1}(V)$ for some $V \subseteq Y$, but $U \neq \rho_2^{-1}(W)$ for any W. Therefore there exist x_1, x_2 with $\rho_1(x_1) = \rho_1(x_2)$ (both in the same fibre of ρ_1) but $\rho_2(x_1) \neq \rho_2(x_2)$. □

Remark 21.4. Theorem 21.3 is the general form of the observation that "representations reorganise topology." It applies to:

- **Kernel embeddings**: $\rho_1 = \mathrm{id}$ (identity on distributions), $\rho_2 = \mathcal{N}(0, S_\cdot)$ (Gaussian embedding). By Theorem 14.2, τ_{ρ_1} and τ_{ρ_2} differ: ρ_2 separates distributions that ρ_1 places in the same metric neighbourhood.

- **Collapse rules**: $\rho_1 = c_{\text{acc}}$, $\rho_2 = c_I$. By Proposition 19.9, $\tau_{c_I} \supsetneq \tau_{c_{\text{acc}}}$: the identity rule separates histories that the accumulate rule identifies.

Both are instances of Theorem 21.3. The connection is now a theorem, not an analogy.

Remark 21.5. The difference between the kernel case and the collapse case is now also precise:

- In the kernel case, the topological change is *forced*: the Gaussian embedding is the unique representation with the mutual-singularity property (by Feldman–Hájek). The programmer does not choose whether the separation occurs.

- In the collapse case, the topological change is *chosen*: the programmer selects the collapse rule c, thereby choosing which distinctions are topologically visible.

One is a mathematical limit; the other is a design parameter. Both are instances of Theorem 21.3, applied in different regimes of freedom.

Part XI

Deeper Mathematical Structure

Chapter 22

Derivations: Structures That Emerge Inevitably

A framework matures not when it accumulates more propositions but when it shows that many seemingly separate propositions are consequences of the same underlying object.

—

22.1 Motivation

The preceding chapters have established definitions and verified properties one by one. This chapter takes a different posture: it asks which results can be *derived* from a small number of primitives, so that the reader sees not merely that something is true but that it *had* to be true given the ontological commitments of the framework.

We present six derivations, each revealing a theorem as the inevitable consequence of a prior commitment.

22.2 Derivation 1: The Generalised Possibility Functional

Theorem 2.6 (Conservation of Possibility) established $|H_t|+|\Omega_t| = |\Omega_0|$ under pure Pop dynamics. That result treats all events equally. The following derivation reveals the original theorem as a special case of a weighted conservation measure.

Definition 22.1 (Event Weight Function)**.** *Define* $w : E \to \mathbb{N}$ *by*

$$w(\text{Pop}(x)) = 1, \quad w(\text{Refuse}(x, r)) = 0, \quad w(\text{Bind}(a, b)) = 0, \quad w(\text{Collapse}(x,$$

Definition 22.2 (Generalised Possibility Functional)**.** *For a world* (H, Ω)*, define*

$$\Pi(H, \Omega) = |\Omega| + \sum_{e \in H} w(e).$$

Theorem 22.3 (Conservation of the Possibility Functional)**.** *For any legal execution sequence* $(H_0, \Omega_0) \to (H_1, \Omega_1) \to \cdots \to (H_n, \Omega_n)$*,*

$$\Pi(H_t, \Omega_t) = |\Omega_0|$$

for all $t \geq 0$*.*

Proof. At $t = 0$: $\Pi(H_0, \Omega_0) = |\Omega_0| + 0 = |\Omega_0|$. At each step, exactly one of four operators is applied:

- **Pop**: $|\Omega|$ decreases by 1; $w(\text{Pop}) = 1$ is added. Net change: $(-1) + 1 = 0$.
- **Refuse**: $|\Omega|$ unchanged; $w(\text{Refuse}) = 0$ is added. Net change: 0.
- **Bind**: $|\Omega|$ unchanged; $w(\text{Bind}) = 0$ is added. Net change: 0.
- **Collapse**: $|\Omega|$ unchanged; $w(\text{Collapse}) = 0$ is added. Net change: 0.

In every case $\Pi(H_{t+1}, \Omega_{t+1}) = \Pi(H_t, \Omega_t)$. By induction, $\Pi(H_t, \Omega_t) = \Pi(H_0, \Omega_0) = |\Omega_0|$. □

Remark 22.4. Theorem 2.6 follows immediately: setting all weights to 1 and restricting to Pop-only executions recovers $|H_t| + |\Omega_t| = |\Omega_0|$. But Theorem 22.3 is more general: the functional Π is conserved even when Refuse, Bind, and Collapse are intermixed. The weight function w makes explicit what counts as "consuming possibility" (Pop, weight 1) versus "documenting structure without consuming it" (Refuse, Bind, Collapse, weight 0).

Corollary 22.5 (Irreversibility of Execution)**.** *For any non-empty execution sequence, $|H_{t+1}| > |H_t|$. Therefore no legal execution can return to a previous world state (H_t, Ω_t).*

Proof. Every operator appends exactly one event to H (Definition 2.3), so $|H_{t+1}| = |H_t| + 1 > |H_t|$. Since H grows strictly and executions are deterministic, $(H_{t+1}, \Omega_{t+1}) \neq (H_t, \Omega_t)$. □

Remark 22.6. Corollary 22.5 establishes that Spherepop worlds form a *directed acyclic execution graph*: world states are nodes, operators are edges, and there are no cycles. This gives a rigorous foundation for the informal claim that Spherepop histories are intrinsically irreversible.

22.3 Derivation 2: Universal Property of Collapse

The quotient construction $O_c = \mathcal{H}/\sim_c$ is not merely a convenient definition. It has a *universal property* that explains why collapse is the canonical projection — not one representation among many but the coarsest representation preserving exactly the distinctions specified by rule c.

Theorem 22.7 (Universal Property of Collapse)**.** *Let $q_c : \mathcal{H} \to O_c$ be the quotient map for rule c. If $f : \mathcal{H} \to X$ is any function satisfying*

$$H_1 \sim_c H_2 \implies f(H_1) = f(H_2)$$

(i.e. f is constant on c-equivalence classes), then there exists a unique $\bar{f} : O_c \to X$ such that $f = \bar{f} \circ q_c$.

Proof. By the universal property of quotient sets. Define $\bar{f}([H]_c) = f(H)$. This is well-defined because f is constant on equivalence classes. Uniqueness: any $\bar{f}$ satisfying $f = \bar{f} \circ q_c$ must map $[H]_c$ to $f(H)$, so $\bar{f}$ is uniquely determined. □

Remark 22.8. Theorem 22.7 is philosophically important. It says that every observation f that "respects" a collapse rule c factors uniquely through O_c. Observable state is therefore not merely

one representation among many. It is the *universal* representation among all representations that identify the same histories as c does. Any coarser observation necessarily passes through O_c.

22.4 Derivation 3: Unique History Factorisation

Theorem 22.9 (Unique History Factorisation)**.** *Every non-empty history $H = [e_1, e_2, \ldots, e_n]$ admits a unique factorisation into primitive generators:*

$$H = e_1 \mathbin{+\!\!+} e_2 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} e_n.$$

No other factorisation into generators exists.

Proof. Since $\mathcal{H}$ is the free monoid over E, every element has a unique normal form as a sequence of generators. History concatenation $\oplus$ is the monoid operation, and the generators are singletons $[e]$. The unique factorisation is the sequence of singleton generators composing to H; freeness of the monoid guarantees there are no non-trivial relations among generators. □

Remark 22.10. Theorem 22.9 provides the mathematical basis for replay equivalence. If two execution paths produce the same history, they must have produced the same sequence of primitive events, in the same order. There is no ambiguity about which commitments generated a trajectory. This is the uniqueness that makes $I(p)$.history $= V(C(p))$.history a meaningful and decidable correctness criterion.

22.5 Derivation 4: Compiler Full Faithfulness

Theorem 11.1 (Replay Equivalence) established one direction: $I(p)$.history $= V(C(p))$.history for all p. We now strengthen this to a biconditional.

Theorem 22.11 (Compiler Full Faithfulness)**.** *Let $\mathcal{C} : \mathbf{P} \to \mathbf{Hist}$ map programs to their interpreter-produced histories. The compiled*

semantics $\mathcal{V} = V \circ C : \mathbf{P} \to \mathbf{Hist}$ *satisfies:*

$$I(p_1).\mathit{history} = I(p_2).\mathit{history} \iff V(C(p_1)).\mathit{history} = V(C(p_2)).\mathit{history}.$$

That is, the compiler is fully faithful*: it preserves and reflects history identity.*

Proof. ($\Rightarrow$) Suppose $I(p_1).H = I(p_2).H$. By Theorem 11.1, $V(C(p_i)).H = I(p_i).H$ for $i = 1, 2$. Therefore $V(C(p_1)).H = I(p_1).H = I(p_2).H = V(C(p_2)).H$.

($\Leftarrow$) Suppose $V(C(p_1)).H = V(C(p_2)).H$. By Theorem 11.1 (applied to each p_i), $I(p_i).H = V(C(p_i)).H$. Therefore $I(p_1).H = V(C(p_1)).H = V(C(p_2)).H = I(p_2).H$. □

Remark 22.12. Full faithfulness means compilation introduces and loses no historical distinctions. Two programs that are history-equivalent under interpretation remain history-equivalent under compilation, and vice versa. In categorical language: C is a faithful and full functor from **P** to **Hist** (when restricted to histories).

22.6 Derivation 5: Observation-Recovery Tradeoff

Theorem 22.13 (Observation–Recovery Tradeoff)**.** *Let $c : \mathcal{H} \to O_c$ be a collapse rule. Exact history recovery from observation is possible if and only if c is injective. Equivalently:*

$$\forall o \in O_c,\ |c^{-1}(o)| = 1.$$

Proof. Immediate from the definition of injectivity and the definition of $c^{-1}(o)$ as a fibre. c is injective iff every fibre has exactly one element iff every observation uniquely determines the history. □

Corollary 22.14 (Tradeoff Sharpness)**.** *Every collapse rule is either:*

(i) *injective (= Identity), in which case observation is history-faithful but the observable space is as large as history space; or*

(*ii*) *non-injective* (= *any useful compression*), *in which case some distinct histories are identified and exact recovery fails.*

Every useful rule sacrifices recoverability. Every recoverable rule approaches identity. Observation and compression form a fundamental tradeoff.

Remark 22.15. Corollary 22.14 gives a rigorous, constructive foundation for the erasure-vs-illusion distinction of Chapter 18. State erasure is a *choice* to discard the history after applying c, leaving only $c(H)$. State illusion is the *necessary* consequence of applying any non-injective c. A system that retains H after computing $c(H)$ eliminates erasure while accepting the unavoidable illusion.

22.7 Derivation 6: Observational Entropy and Fibre Geometry

Definition 22.16 (Observational Entropy)**.** *For a collapse rule c and observable state $o \in O_c$, define the* observational entropy *of o as*

$$S_c(o) = \log |c^{-1}(o)|$$

(*taking* $\log 0 = -\infty$ *by convention, though fibres of well-defined rules are non-empty*).

Theorem 22.17 (Entropy Monotonicity Under Rule Refinement)**.** *If $c_1 \preceq c_2$ (c_1 finer than c_2, i.e. c_1 preserves strictly more distinctions), then for every $o \in O_{c_1}$:*

$$S_{c_1}(o) \leq S_{c_2}(c_2(c_1^{-1}(o))).$$

Equivalently, finer rules have smaller or equal fibre sizes.

Proof. Since $c_1 \preceq c_2$, every $\sim_{c_1}$-equivalence class is contained in a $\sim_{c_2}$-equivalence class. Therefore $|c_1^{-1}(o)| \leq |c_2^{-1}(c_2(H))|$ for any $H \in c_1^{-1}(o)$. Taking logarithms: $S_{c_1}(o) \leq S_{c_2}(c_2(H))$. □

Remark 22.18. Definition 22.16 makes the connection to CLIO explicit. The CLIO framework defines representational entropy as $S_\pi(m) = \log \mathrm{Vol}(\pi^{-1}(m))$ for a projection $\pi : X \to M$. Definition 22.16 is exactly this quantity in the discrete Spherepop setting: $\pi = c$, $X = \mathcal{H}$, $M = O_c$, $\mathrm{Vol} = |\cdot|$.

Thus the Spherepop collapse framework is a discrete computational instance of the CLIO projection framework. Observational entropy, representational ambiguity, and fibre size are the same concept at different levels of abstraction.

Proposition 22.19 (Fibre Partition of History Space)**.** *For any collapse rule c:*

$$\mathcal{H} = \bigsqcup_{o \in O_c} c^{-1}(o).$$

The fibres $\{c^{-1}(o)\}_{o \in O_c}$ form a partition of history space into disjoint, non-empty subsets.

Proof. By definition of quotient: every $H \in \mathcal{H}$ belongs to exactly one equivalence class $[H]_c = c^{-1}(c(H))$. □

Remark 22.20. Proposition 22.19 reframes the entire framework geometrically. Observable states are *strata* of history space. Histories are *trajectories* through that space. Fibres are *hidden manifolds* of indistinguishable constructions.

At this resolution, the connection to Hidden Manifolds, CLIO, admissibility geometry, and the kernel-embedding discussion of Chapters 14 and 21 is mathematically explicit. The Spherepop framework is a theory of the fibre structure induced by projections from computation space into observation space.

Chapter 23

Category-Theoretic Foundations of History and Observation

23.1 Purpose

The main text develops the category **Hist** and the observable-state categories $\mathbf{Obs}_c$. This appendix collects categorical constructions that clarify the structural role of histories, observations, and replay equivalence. The goal is to show that the Spherepop framework occupies a well-defined position within categorical semantics.

23.2 Histories as a Free Category

Let G be the directed graph whose vertices are option spaces $\Omega \subseteq \Omega_0$ and whose edges are primitive event transitions.

Definition 23.1 (Free History Category)**.** *The category* **Hist** *is the* free category generated by G*. Objects are option spaces. Morphisms are finite paths in* G*. Composition is path concatenation. The identity morphism at* Ω *is the empty path* ε_Ω*.*

Proposition 23.2 (Universal Property of **Hist**)**.** *Every functor* $F : G \to \mathcal{C}$ *from the underlying graph* G *to any category* $\mathcal{C}$ *extends uniquely to a functor* $\tilde{F} : \mathbf{Hist} \to \mathcal{C}$*.*

Proof. By the universal property of free categories: a path $[e_1, \dots, e_n]$ maps to $F(e_n) \circ \cdots \circ F(e_1)$. This is unique because path composition determines the functor on all morphisms. □

Remark 23.3. Proposition 23.2 explains why histories serve as

primitive objects of the framework. Any semantics defined on primitive events extends uniquely to complete histories. The collapse functor F_c, the interpreter I, and the VM semantics $V \circ C$ are all instances of this universal extension.

23.3 Replay Equivalence as Faithfulness

Definition 23.4 (Semantic Faithfulness)**.** *A compiler is* semantically faithful *when*

$$I(p_1).\mathit{history} = I(p_2).\mathit{history} \implies V(C(p_1)).\mathit{history} = V(C(p_2)).\mathit{history}.$$

Proposition 23.5 (Replay Implies Faithfulness)**.** *Theorem 11.1* (*Replay Equivalence*) *implies semantic faithfulness.*

Proof. By Theorem 22.11 (Full Faithfulness), replay equivalence gives both directions of the biconditional, which is stronger than mere faithfulness. □

23.4 Collapse as a Reflector

Definition 23.6 (Reflective Observation)**.** *A collapse rule c is* reflective *if there exists a right adjoint $R_c : \mathbf{Obs}_c \to \mathbf{Hist}$ such that $F_c \dashv R_c$. The right adjoint chooses a* canonical representative history *for each observable state.*

Remark 23.7. When R_c exists, observation is not merely lossy compression. It is a reflective localisation: $\mathbf{Obs}_c$ is a reflective subcategory of **Hist**, and every observable state corresponds to a canonical history. For the Identity rule, $F_{c_I} \dashv \mathrm{id}_{\mathbf{Hist}}$ trivially. For LastWrite, no right adjoint exists in general because the choice of canonical representative among c_{LW}-equivalent histories is not natural.

23.5 The Observational Lattice

Proposition 23.8 (Collapse Rules Form a Lattice)**.** *The set of collapse rules ordered by refinement $c_1 \preceq c_2$ (finer $\preceq$ coarser) forms a complete lattice.*

Proof. The meet of $\{c_i\}$ is the rule whose equivalence relation is the intersection $\bigcap \sim_{c_i}$ (the finest rule at least as fine as all). The join is the equivalence relation generated by $\bigcup \sim_{c_i}$. Completeness follows from lattice operations on equivalence relations. □

Remark 23.9. The lattice of collapse rules organises all possible notions of visibility. c_I (Identity) is the minimal (finest) element. The trivial rule $c_\top$ (mapping all histories to one point) is the maximal (coarsest) element. The four canonical rules — Identity, LastWrite, Accumulate, Projection — are elements of this lattice, incomparable in general.

Chapter 24

Sheaf-Theoretic Semantics of Observation

24.1 Motivation

A recurring theme of this monograph is that observable states are *local views* of larger histories. Different collapse rules expose different fragments of the same underlying computation. The mathematical language invented for relating local observations to global structure is *sheaf theory*. This appendix sketches a sheaf-theoretic interpretation of Spherepop histories, connecting the collapse framework to the CLIO program.

24.2 Observational Sites

Definition 24.1 (Observational Region)**.** *For a collapse rule c and observable state $o \in O_c$, the* observational region *corresponding to o is the fibre $c^{-1}(o) \subseteq \mathcal{H}$.*

Definition 24.2 (Observational Site)**.** *Fix a collection of collapse rules $\{c_i\}$ and corresponding observable states $\{o_i\}$. The* observational site *$\mathcal{O}$ is the category whose objects are observational regions $\{c_i^{-1}(o_i)\}$ and whose morphisms are inclusions:*

$$c_j^{-1}(o_j) \hookrightarrow c_i^{-1}(o_i) \quad \textit{when} \quad c_j^{-1}(o_j) \subseteq c_i^{-1}(o_i).$$

Remark 24.3. The observational site encodes the refinement order among collapse rules. A finer rule creates a smaller, more informative observational region. A coarser rule creates a larger, less discriminating region.

24.3 The History Presheaf

Definition 24.4 (History Presheaf $\mathcal{H}$)**.** *Define the presheaf $\mathcal{H}$ on $\mathcal{O}$ by:*

$$\mathcal{H}(U) = \{H \in \mathcal{H} \mid H \in U\}$$

for each observational region U, with restriction maps given by set inclusion.

Proposition 24.5 ($\mathcal{H}$ is a Sheaf)**.** *$\mathcal{H}$ satisfies the sheaf condition: compatible local histories glue uniquely to global histories. Specifically, if $\{U_i\}$ covers a region U and $\{H_i \in \mathcal{H}(U_i)\}$ is a compatible family (agreeing on overlaps), then there exists a unique $H \in \mathcal{H}(U)$ restricting to each H_i.*

Proof. Histories are sequences of events. Compatibility means the sequences agree on their shared prefixes. Two compatible event sequences agreeing on all overlaps determine a unique concatenation, giving the unique glueing. □

Remark 24.6. Proposition 24.5 formalises a common engineering intuition: multiple partial logs can reconstruct a single global execution history. If the logs are compatible (they agree where they overlap), they uniquely determine the full history. This is the sheaf-theoretic content of replay equivalence.

24.4 Observable States as Sections

A collapse rule $c : \mathcal{H} \to O_c$ induces a sheaf of observable states.

Definition 24.7 (Observable Sheaf $\mathcal{O}_c$)**.** *Define $\mathcal{O}_c(U) = \{c(H) \mid H \in U\}$, the set of observable states reachable within region U.*

Remark 24.8. Different observers (using different collapse rules) correspond to different sheaves on the same underlying history space. The same history looks different to an observer using c_{LW} versus one using c_{acc}. The sheaf structure makes this perspectivalism mathematically precise.

24.5 State Illusion as Failure of Unique Lifting

Proposition 24.9 (State Illusion as Non-Injectivity of Sections)**.** *A collapse rule c exhibits state illusion (in the sense of Definition 18.5) if and only if the observable sheaf $\mathcal{O}_c$ fails to determine a unique global section of $\mathcal{H}$. That is, there exist distinct $H_1, H_2 \in \mathcal{H}(\mathcal{H})$ with the same section in $\mathcal{O}_c(\mathcal{H})$.*

Proof. By definition: state illusion holds iff c is non-injective iff there exist $H_1 \neq H_2$ with $c(H_1) = c(H_2)$ iff the global section of $\mathcal{O}_c$ does not uniquely lift to a global section of $\mathcal{H}$. □

Remark 24.10. State illusion is a failure of *unique lifting*: many global histories project to the same observable section. Theorem 18.3 (No Direct Observation) is therefore the statement that for any non-trivial c, unique lifting fails for at least one section. The erasure/illusion distinction translates directly: state *erasure* is discarding $\mathcal{H}$ after computing $\mathcal{O}_c$ (no sheaf structure retained); state *illusion* is the inherent non-injectivity of c (sheaf structure retained but lifting is non-unique).

24.6 Refinement as Sheaf Morphism

Proposition 24.11 (Refinement Induces Sheaf Morphism)**.** *If $c_1 \preceq c_2$ (finer c_1), there is a morphism of sheaves*

$$\phi : \mathcal{O}_{c_1} \to \mathcal{O}_{c_2}$$

mapping more-informative observations to less-informative ones.

Proof. Since $c_1 \preceq c_2$, there exists a map $\pi : O_{c_1} \to O_{c_2}$ such that $c_2 = \pi \circ c_1$. This induces $\phi(U)(s) = \pi(s)$ for sections s of $\mathcal{O}_{c_1}$. Naturality with respect to restriction maps follows from the commutativity of quotient maps. □

Remark 24.12. The sheaf morphism ϕ maps a finer observation to a coarser one. Refining in the opposite direction (going from $\mathcal{O}_{c_2}$ to $\mathcal{O}_{c_1}$) requires a right adjoint (Section B.3), which may not

exist. This asymmetry is exactly the state illusion: you can always coarsen an observation (lose information), but you cannot always refine one (recover information) — unless the full history $\mathcal{H}$ is available.

24.7 Connection to CLIO

Remark 24.13 (CLIO as Sheaf Theory on Representation Space)**.** The CLIO framework (Constraint-Limited Information Ontology) models representations as projections $\pi : X \to M$ from a representation space to a manifold of observational strata.

In sheaf-theoretic terms:

- $X = \mathcal{H}$ (history space, or more generally a semantic space).
- $M = O_c$ (observable state space, or more generally a manifold).
- $\pi = c$ (the collapse functor / projection map).
- Fibres $c^{-1}(o) = \pi^{-1}(m)$ are the hidden manifolds of indistinguishable constructions.
- Representational entropy $S_c(o) = \log |c^{-1}(o)|$ is the CLIO quantity $S_\pi(m) = \log \mathrm{Vol}(\pi^{-1}(m))$ in the discrete setting.

The entire Spherepop framework is thus a discrete, computational instantiation of the CLIO projection program. Histories are the objects; collapse rules are the projections; observable states are the strata; fibres are the hidden manifolds.

The sheaf structure ties the local (what one observer sees under one rule) to the global (the full history that all rules project from). Sheaf cohomology would measure the obstruction to lifting local observations to global histories — which is precisely the observational entropy $S_c(o)$.

Chapter 25

BNF Grammar for the Spherepop Core Language

25.1 Purpose

This appendix gives a compact Backus–Naur Form grammar for the core Spherepop surface language implemented by the current crate. The grammar specifies the minimal executable fragment used by the interpreter, compiler, VM, REPL, and test suite.

This grammar intentionally separates parsing from admissibility: the parser recognises possible terms, while the type checker decides which of those terms are admissible under a chosen world assumption.

25.2 Lexical Categories

$$\langle ident \rangle ::= \langle letter \rangle \{ \langle letter \rangle \mid \langle digit \rangle \mid \_ \}^*$$
$$\langle symbol \rangle ::= \langle ident \rangle$$

Reserved words: `pop`, `refuse`, `bind`, `collapse`, `seq`, `let`, `in`, `fn`, `Unit`, `Never`, `Type`.

25.3 Terms

$$\langle term \rangle ::= \langle var \rangle \mid \langle lambda \rangle \mid \langle app \rangle \mid \langle let \rangle \mid \langle pop \rangle \mid \langle refuse \rangle \mid \langle bind \rangle \mid \langle collapse \rangle$$

$$\langle var \rangle ::= \langle symbol \rangle$$

$$\langle lambda \rangle ::= (\lambda \mid \texttt{fn})\ \langle symbol \rangle\ [: \langle type \rangle]\ (\rightarrow \mid .)\ \langle term \rangle$$

$$\langle app \rangle ::= \langle term \rangle\ \langle term \rangle$$

$$\langle let \rangle ::= \texttt{let}\ \langle symbol \rangle = \langle term \rangle\ \texttt{in}\ \langle term \rangle$$

$$\langle pop \rangle ::= \texttt{pop}\ \langle term \rangle$$

$$\langle refuse \rangle ::= \texttt{refuse}\ \langle term \rangle\ [\langle reason \rangle]$$

$$\langle bind \rangle ::= \texttt{bind}\ \langle term \rangle\ \langle term \rangle$$

$$\langle collapse \rangle ::= \texttt{collapse}\ \langle term \rangle\ [\langle rule \rangle]$$

$$\langle seq \rangle ::= \texttt{seq}\ [\ \langle term \rangle \{; \langle term \rangle\}^*\ ]$$

25.4 Refusal Reasons

$\langle reason \rangle ::= \texttt{violation}(\langle symbol \rangle) \mid \texttt{explicit}(\langle symbol \rangle) \mid \varepsilon$ (defaults to Expl

Runtime correspondence:

$$\texttt{violation}(c) \rightsquigarrow \texttt{ConstraintViolation}(c)$$
$$\texttt{explicit}(s) \rightsquigarrow \texttt{Explicit}(s)$$

25.5 Collapse Rules

$\langle rule \rangle ::= \texttt{id} \mid \texttt{last\_write} \mid \texttt{accumulate} \mid \texttt{proj}(\langle symbol \rangle) \mid \langle ident \rangle \mid \varepsilon$ (defau

25.6 Types

$\langle type \rangle ::= \texttt{Unit} \mid \texttt{Never} \mid \texttt{Type} \mid \langle type\_var \rangle \mid \texttt{Admissible}(\langle type \rangle) \mid \langle type \rangle \rightarrow \langle ty$

The richer types (Refused, Collapsed, Process, Π-types) are *synthesised* by the type checker (Chapter 5) rather than present in the surface syntax. The surface grammar therefore describes the input language; the typed language is the output of elaboration. A reader cannot infer the typed language from the surface grammar alone, and a full specification of the elaborated typed language is a proof obligation for future work (see Chapter 17).

25.7 Operational Summary

Evaluating a program produces (V, H, Ω, a) where: V is the value, H the accumulated history, Ω the remaining option space, and $a \in \{\top, \bot\}$ the admissibility flag.

Surface form	Type produced	Event appended	Ω chan
`pop x`	Admissible(Unit)	Pop(x)	$\Omega \setminus \{x$
`refuse x violation(c)`	Refused(Unit, r)	Refuse(x, r)	none
`bind a b`	Process$(T_a \rightsquigarrow T_b)$	Bind(a, b)	none
`collapse (pop x) id`	Collapsed(Unit, c_I)	Collapse(x, c_I)	none

The type checker enforces the precondition for collapse: the inner term must have type Admissible(T) (see [T-Collapse]).

Appendix A

Mechanised Witnesses: Rust Module Map and Selected Tests

A.1 Motivation

The formal theorems of this monograph are not purely abstract. Each has a mechanical correlate in the Spherepop codebase: either a module whose existence is forced by the theorem, a type whose structure reflects a definition, or a test whose passing constitutes evidence for the theorem.

This appendix presents the module map and selected tests as *mechanised witnesses* for the principal theoretical results.

A.2 Module-to-Theorem Map

Module	Theorem / Definition	Role
`core/event.rs`	Def. 2.1 (Event Alphabet)	Defines the ge
`core/history.rs`	Prop. 3.2 (Free Monoid)	Mechanises ($\mathcal{I}$
`core/option_space.rs`	Thm. 2.6 (Conservation)	Enforces $\lvert\Omega_t\rvert$
`core/observable.rs`	Def. 8.3 (Observable Quotient)	Implements $\mathcal{F}$
`types/checker.rs`	Thm. 6.1 (Collapse Soundness)	Enforces [T-C
`types/checker.rs`	Def. 7.1 / 7.2 (TypeMode)	Mechanises op
`eval/interpreter.rs`	Thm. 6.2 (Type Preservation)	Evaluation res
`compiler/lower.rs`	Prop. 12.1 (IR Faithfulness)	Bijective event
`vm/machine.rs`	Thm. 11.1 (Replay Equivalence)	History identi

A.3 Selected Tests as Proof Obligations

Remark A.1 (Evidential Status)**.** The tests below are *mechanised evidence*, not formal proofs. Each passing test witnesses the relevant theorem for the specific program terms used in that test; it does not constitute a proof that the theorem holds for all programs in the language. A complete formal proof would require either a mechanised proof assistant (Coq, Lean, Agda) formalisation of the type theory and operational semantics, or a proof by structural induction covering all syntactic forms. Both are proof obligations for future work.

Each test below is annotated with the theorem it mechanically witnesses.

Witness for Theorem 2.6 (Conservation of Possibility)

```
#[test]
fn world_history_grows_option_space_shrinks() {
    let mut w = World::new(OptionSpace::new([
        Symbol::new("a"), Symbol::new("b"),
    ]));
    let initial = w.options.len();
    w.pop(Symbol::new("a")).unwrap();
    // |H| increased by 1; |Omega| decreased by
        1
    assert_eq!(w.history.len(), 1);
    assert_eq!(w.options.len(), initial - 1);
    // Refuse does NOT shrink option space
    w.refuse(Symbol::new("b"),
             RefusalReason::NotAvailable);
    assert_eq!(w.options.len(), initial - 1);
       // unchanged
    assert_eq!(w.history.len(), 2);
       // history grew
}
```

This test witnesses: the asymmetry between Pop (which de-

creases $|\Omega|$) and Refuse (which does not), formalised in Theorem 2.6(ii) and Definition 4.2 (Admissibility Contraction).

Witness for Theorem 6.1 (Collapse Soundness)

```
#[test]
fn collapse_requires_admissible_certificate() {
    // refuse(x) -> Refused, not Admissible
    // collapse(refuse(x)) must fail
    let term = Term::collapse(
        Term::refuse(Term::var("x"),
                     RefusalReason::
                        NotAvailable),
        CollapseRule::Identity,
    );
    assert!(eval_closed(&term).is_err(),
        "collapse of refused term must be a
           type error");
}
```

This test witnesses: Theorem 6.1, which requires the premise $\Gamma \vdash t : \text{Admissible}(T)$ for [T-Collapse]. A refused term has type $\text{Refused}(T, r)$, not $\text{Admissible}(T)$, so collapse is rejected.

Witness for Theorem 11.1 (Replay Equivalence)

```
fn assert_same_history(term: &Term) {
    // Interpret
    let (_, world_interp) = eval_closed(term).
       unwrap();
    // Compile and run via VM
    let block = compile(term);
    let mut machine = Machine::empty();
    machine.run(&block).unwrap();
    // THE CORRECTNESS CRITERION
    assert_eq!(
        world_interp.history,
```

```
        machine.world.history,
        "I(p).history != V(C(p)).history"
    );
}

#[test]
fn replay_equivalence_refuse_with_reason() {
    assert_same_history(&Term::refuse(
        Term::var("x"),
        RefusalReason::ConstraintViolation(
            Symbol::new("c")),
    ));
}

#[test]
fn replay_equivalence_seq_of_binds_and_refuses
   () {
    assert_same_history(&Term::seq(vec![
        Term::bind(Term::var("src"), Term::var(
           "dst")),
        Term::refuse(Term::var("src"),
                     RefusalReason::
                        AlreadyRefused(
                         Symbol::new("src"))),
    ]));
}
```

These tests witness: Theorem 11.1. $I(p).\text{history} = V(C(p)).\text{history}$ for these program terms. Each passing assertion is a mechanised proof obligation for the correctness criterion.

Witness for Proposition 18.10 (Rule Refinement)

```
#[test]
fn
   finer_rule_disambiguates_equivalent_histories
   () {
    // Two histories equivalent under
```

```
        Accumulate
    // but distinct under Identity
    let mut w1 = World::new(OptionSpace::new([
        Symbol::new("x"), Symbol::new("y"),
    ]));
    w1.pop(Symbol::new("x")).unwrap();
    w1.pop(Symbol::new("y")).unwrap();

    let mut w2 = World::new(OptionSpace::new([
        Symbol::new("x"), Symbol::new("y"),
    ]));
    w2.pop(Symbol::new("y")).unwrap();
    w2.pop(Symbol::new("x")).unwrap();

    // Under Accumulate: both map to {x:1, y:1}
        -- equivalent
    let obs1_acc = w1.observe(&CollapseRule::
        Accumulate);
    let obs2_acc = w2.observe(&CollapseRule::
        Accumulate);
    assert_eq!(obs1_acc.entries, obs2_acc.
        entries,
        "accumulate should not distinguish pop
            order");

    // Under Identity: histories differ in
        event order
    assert_ne!(w1.history, w2.history,
        "identity rule distinguishes pop order"
            );
}
```

This test witnesses: Proposition 18.10. The Accumulate rule places H_1 and H_2 in the same equivalence class. The Identity rule (finer than Accumulate) separates them. Switching to the finer rule recovers the distinction.

Witness for Theorem 18.3 (No Direct Observation)

```
#[test]
fn
    distinct_histories_may_agree_under_any_coarse_rule
    () {
     let mut w1 = World::empty();
     w1.refuse(Symbol::new("x"),
               RefusalReason::NotAvailable);
     w1.refuse(Symbol::new("y"),
               RefusalReason::NotAvailable);

     let mut w2 = World::empty();
     w2.refuse(Symbol::new("y"),
               RefusalReason::NotAvailable);
     w2.refuse(Symbol::new("x"),
               RefusalReason::NotAvailable);

     // Histories are distinct (different order)
     assert_ne!(w1.history, w2.history);

     // Under Accumulate (order-insensitive),
        they agree
     let acc1 = w1.observe(&CollapseRule::
        Accumulate);
     let acc2 = w2.observe(&CollapseRule::
        Accumulate);
     // (Both have refuse_count 2, but the
        entries
     //  map doesn't record refuses in
        accumulate)
     assert_eq!(w1.history.refuse_count(),
                w2.history.refuse_count());
}
```

This test witnesses: Theorem 18.3. The two histories are distinct but observationally equivalent under any order-insensitive rule. This is an instance of the cardinality argument: any $|O| < |\mathcal{H}|$

observation map must identify some distinct histories.

A.4 Summary Table

Theorem	Test name
Thm. 2.6 (Conservation)	`world_history_grows_option_space_sh`
Thm. 6.1 (Collapse Soundness)	`collapse_requires_admissible_certif`
Thm. 11.1 (Replay Equiv.)	`replay_equivalence_*`
Prop. 18.10 (Refinement)	`finer_rule_disambiguates`
Thm. 18.3 (No Obs.)	`distinct_histories_agree_coarse`

Part XII

Toward an Admissibility Field Semantics

Appendix B

The CLIO Framework

A representation is not a container of information. It is an operator that reorganises which distinctions survive projection.

—

B.1 Background and Purpose

The Constraint-Limited Information Ontology (CLIO) is a representational framework developed to study how projections restructure the geometry of a semantic space [**?**]. Its central claim is that representations do not merely compress data — they determine which distinctions count as differences.

CLIO appears throughout this monograph as the ambient theoretical context for Spherepop's collapse framework. This chapter gives a self-contained summary of the CLIO concepts we use, so that readers who do not have prior exposure to CLIO can follow the later connections.

B.2 Core Definitions

Definition B.1 (CLIO Projection)**.** *A* CLIO projection *is a surjective map*

$$\pi : X \to M$$

from a high-dimensional representation space X to a lower-dimensional manifold M of observational strata.

Definition B.2 (Fibre)**.** *The* fibre *above a stratum $m \in M$ is*

$$\pi^{-1}(m) = \{x \in X \mid \pi(x) = m\}.$$

Definition B.3 (Representational Entropy)**.** *The* representational entropy *of stratum m under projection π is*

$$S_\pi(m) = \log \mathrm{Vol}(\pi^{-1}(m))$$

where Vol *is the appropriate measure on X (cardinality in finite settings; Lebesgue or Riemannian volume in continuous ones).*

Definition B.4 (Admissibility Manifold)**.** *Given a projection π, the* admissibility manifold *$\mathcal{A}(\pi)$ is the subset of M consisting of strata m whose fibres are non-empty and satisfy the constraints of the observational framework. $\mathcal{A}(\pi)$ is the space of* reachable observations *under π.*

B.3 The CLIO–Spherepop Correspondence

The Spherepop collapse framework is a discrete, computational instance of CLIO:

CLIO concept	Spherepop realisation
Representation space X	History space $\mathcal{H}$
Projection $\pi : X \to M$	Collapse rule $c : \mathcal{H} \to O_c$
Stratum $m \in M$	Observable state $o \in O_c$
Fibre $\pi^{-1}(m)$	Fibre $c^{-1}(o)$
Representational entropy $S_\pi(m)$	Observational entropy $S_c(o) = \log \lvert c^{-1}($
Admissibility manifold $\mathcal{A}(\pi)$	Admissible option space $\mathcal{A}(H)$

Remark B.5. The identification is exact in the discrete setting (replacing volume with cardinality). The key insight is that collapse rules are not ad-hoc implementation choices — they are instances of the CLIO projection operation. Choosing a collapse rule is choosing a geometry of observational distinction.

Appendix C

Programs as Operators on Admissibility Fields

Programs are not histories. Programs are what histories do to the future.

—

C.1 The Forward Shift

Every version of Spherepop through v3 pointed *backward*: histories record what happened; states are projections of what happened; types certify that what happened was admissible. The arrow was retrospective.

This chapter introduces a forward-pointing structure. The central claim is:

A history is fundamental not because it records the past but because it constrains the future.

The causal arrow is History $\rightarrow$ Future Geometry, not History $\rightarrow$ Past Record.

C.2 The Future Reachability Map

Definition C.1 (Reachability Geometry)**.** *For a world* (H, Ω)*, the* reachability geometry *is the pair*

$$\mathcal{R}(H, \Omega) = (\mathcal{A}(H),\ \Omega)$$

where $\mathcal{A}(H) \subseteq \Omega_0$ is the admissibility set (Definition 4.1) and $\Omega \subseteq \Omega_0$ is the structural option space. $\mathcal{R}(H, \Omega)$ describes what can still happen *from the current world state.*

Definition C.2 (Future Reachability Space)**.** *Let $\mathcal{F}$ denote the set of all reachability geometries:*

$$\mathcal{F} = \{(\mathcal{A}, \Omega) \mid \mathcal{A} \subseteq \Omega_0,\ \Omega \subseteq \mathcal{A}\}.$$

Definition C.3 (The Forward Projection)**.** *Define the* forward projection

$$\rho : \mathcal{H} \times 2^{\Omega_0} \to \mathcal{F}$$

by

$$\rho(H, \Omega) = \mathcal{R}(H, \Omega) = (\mathcal{A}(H), \Omega).$$

Remark C.4. ρ is the forward analogue of the collapse rule c. Where $c : \mathcal{H} \to O_c$ maps histories to observable (past) states, $\rho : \mathcal{H} \times 2^{\Omega_0} \to \mathcal{F}$ maps world states to reachable (future) geometries.

The full semantic picture is therefore:

$$\text{World } (H, \Omega) \xrightarrow{\rho} \mathcal{F} \xrightarrow{c} O_c.$$

Observable state is the *second* projection — a projection of the future reachability geometry, not a direct projection of history.

C.3 Histories as Deformations

Definition C.5 (Deformation Operator)**.** *Each primitive event $e \in E$ induces a* deformation $\delta_e : \mathcal{F} \to \mathcal{F}$*:*

$$\begin{aligned}
\delta_{\text{Pop}(x)}(\mathcal{A}, \Omega) &= (\mathcal{A}, \Omega \setminus \{x\}) \\
\delta_{\text{Refuse}(x,r)}(\mathcal{A}, \Omega) &= (\mathcal{A} \setminus \{x\}, \Omega) \\
\delta_{\text{Bind}(a,b)}(\mathcal{A}, \Omega) &= (\mathcal{A}, \Omega) \\
\delta_{\text{Collapse}(x,c)}(\mathcal{A}, \Omega) &= (\mathcal{A}, \Omega)
\end{aligned}$$

Proposition C.6 (Histories as Composed Deformations)**.** *For any history* $H = [e_1, \ldots, e_n]$*,*

$$\rho(H, \Omega_0) = (\delta_{e_n} \circ \cdots \circ \delta_{e_1})(\Omega_0, \Omega_0).$$

Every history acts as a composed deformation operator on the initial reachability geometry.

Proof. By induction on $|H|$. Base case: $H = \varepsilon$, $\rho(\varepsilon, \Omega_0) = (\Omega_0, \Omega_0)$. Inductive step: appending event e applies δ_e to the current geometry. By Definition C.5, each δ_e transforms $(\mathcal{A}, \Omega)$ exactly as the operational semantics transforms $\mathcal{A}(H)$ and Ω (Definitions 4.1 and 2.3–2.6). □

Remark C.7. Proposition C.6 is the forward-pointing reformulation of the history-first ontology. A program is not a record of what happened; it is a composition of deformation operators that progressively reshapes the reachability geometry.

Pop deforms the structural option space (forecloses a structural possibility). Refuse deforms the admissibility set (forecloses a semantic possibility). Bind and Collapse leave the reachability geometry unchanged (they record structure without foreclosing futures).

C.4 Admissibility Fields

Definition C.8 (Admissibility Field)**.** *An* admissibility field *is a function*

$$\phi : \mathcal{H} \to [0, 1]$$

assigning to each history a degree of admissibility *for what follows.* $\phi(H) = 1$ *means every future is admissible from* H*;* $\phi(H) = 0$ *means no futures are admissible.*

Remark C.9. The Boolean admissibility of earlier chapters is the special case where ϕ takes values in $\{0, 1\}$. A graded admissibility field assigns intermediate values to histories that have partially foreclosed future options. This connects Spherepop to the

RSVP (Relativistic Scalar-Vector Plenum) framework, in which dynamics are governed by a field of varying admissibility over a computational or physical trajectory space.

Definition C.10 (Field Deformation by a History)**.** *Each event e acts on admissibility fields by pullback:*

$$(e^*\phi)(H) = \phi(H +\!\!+ [e]).$$

A history $H = [e_1, \dots, e_n]$ deforms the field by $(e_1 \cdots e_n)^*\phi = e_n^* \circ \cdots \circ e_1^*\phi$.

Theorem C.11 (Monotone Field Deformation)**.** *For any admissibility field ϕ and any Pop event* $\mathrm{Pop}(x)$*:*

$$(\mathrm{Pop}(x)^*\phi)(H) \leq \phi(H) \quad \textit{for all } H.$$

Pop events are non-admissibility-increasing deformations.

Proof. Pop removes x from the option space (Definition 2.4), strictly reducing the set of available futures. Formally: the set of admissible continuations of $H +\!\!+ [\mathrm{Pop}(x)]$ is a subset of the admissible continuations of H (since x is now unavailable). Any field measuring admissibility of continuations is non-increasing under Pop. □

Remark C.12. Theorem C.11 makes precise the intuition that *programs are reachability sculptures*: they carve possibility space by applying sequences of non-increasing deformations to the admissibility field. The sculpture is the final reachability geometry $\rho(H, \Omega)$; the tool strokes are the deformation operators δ_e.

C.5 The Two-Projection Architecture

The full semantic picture of Spherepop v4 is a two-stage projection:

Definition C.13 (Two-Projection Architecture)**.**

$$\underset{\text{World}}{(H,\Omega)} \xrightarrow{\rho} \underset{\text{Future Geometry}}{\mathcal{F}} \xrightarrow{\pi_c} \underset{\text{Observable State}}{O_c}$$

where:

- $\rho(H,\Omega) = (\mathcal{A}(H),\Omega)$ *is the* forward projection (*history to future geometry*)*;*
- $\pi_c : \mathcal{F} \to O_c$ *is the* observational projection (*future geometry to collapsed state*)*.*

Remark C.14. In v1–v3 of this monograph, observable state was described as a direct projection of history: $c : \mathcal{H} \to O_c$. Definition C.13 reveals this as a composite: $c = \pi_c \circ \rho|_{\mathcal{H}}$. Observable state is a projection of a projection.

This is not a semantic change; it is a structural clarification. What changes is what counts as *fundamental*: not histories, not observable states, but the reachability geometry $\mathcal{F}$ that mediates between them.

Theorem C.15 (Fibre Factorisation via Two-Projection)**.** *For any observable state $o \in O_c$:*

$$c^{-1}(o) = \rho^{-1}(\pi_c^{-1}(o)).$$

The fibre of an observable state under the composite is the preimage of the fibre of o under π_c pulled back through ρ.

Proof. Standard composition of preimages: $c^{-1}(o) = (\pi_c \circ \rho)^{-1}(o) = \rho^{-1}(\pi_c^{-1}(o))$. □

Remark C.16. Theorem C.15 reveals that the observational entropy $S_c(o) = \log|c^{-1}(o)|$ has two layers of ambiguity:

- *Geometric ambiguity*: how many future geometries $f \in \mathcal{F}$ project to o under π_c (i.e. $|\pi_c^{-1}(o)|$).
- *Historical ambiguity*: for each such geometry f, how many histories produce it under ρ (i.e. $|\rho^{-1}(f)|$ for $f \in \pi_c^{-1}(o)$).

Total entropy: $S_c(o) = \sum_{f \in \pi_c^{-1}(o)} \log|\rho^{-1}(f)|$.

C.6 Slogan Progression

Version	Slogan	Funda
v1	Programs are histories	$H \in \mathcal{H}$
v2	Programs are constructions in possibility space	(H, Ω)
v3	Programs are reachability sculptures	$\mathcal{H}/\sim_c$
v4	Programs are operators that deform admissibility fields	$\rho : (H,$

Remark C.17. Each version does not replace the previous but subsumes it. Histories remain fundamental — but now as inputs to the forward projection ρ, not as endpoints of a narrative. Observable states remain real — but now as secondary projections of a richer intermediate geometry. The admissibility field $\phi : \mathcal{H} \to [0, 1]$ connects Spherepop to RSVP dynamics, where physical or computational fields vary continuously over a trajectory space.

Appendix D

Toward a Unified Projection Ontology

D.1 The Common Structure

The Spherepop framework, CLIO projections, kernel embeddings, admissibility fields, and sheaf-theoretic observation are all instances of a single mathematical pattern:

$$\pi : X \to Y$$

with:

- a *total space* X carrying the full information,
- a *base space* Y carrying the observable information,
- *fibres* $\pi^{-1}(y)$ carrying the hidden information.

The entire program — from the original state-illusion critique through the conservation law, collapse quotients, admissibility types, compiler correctness, observational entropy, sheaf conditions, and admissibility fields — is the study of how computation relates to this structure.

Definition D.1 (Projection Ontology)**.** *A* projection ontology *for a computational system is a triple* (X, Y, π) *with:*

- X*: the* total computational space (*in Spherepop:* $\mathcal{H} \times 2^{\Omega_0}$*, world states*)*;*
- Y*: the* observable space (*in Spherepop:* O_c*, collapsed states, or* $\mathcal{F}$*, future geometries*)*;*

- $\pi : X \to Y$: *the* projection (*collapse rule c, or forward projection* ρ).

D.2 All Major Theorems as Projection Theorems

Proposition D.2 (Projection Unification)**.** *All major theorems of this monograph are instances of projection geometry:*

Theorem	*Projection* π	*Content*
State Illusion (*Prop. 1.2*)	$\sigma : \mathcal{H} \to S$	π *non-injective* $\Rightarrow$ *informa*
Conservation (*Thm. 2.6*)	$\rho : \mathcal{H} \to \mathcal{F}$	$\lvert\rho^{-1}(f)\rvert$ *bounded by* $\lvert\Omega_0\rvert$
No Direct Obs. (*Thm. 18.3*)	$\phi : \mathcal{H} \to O$	$\lvert O\rvert < \lvert\mathcal{H}\rvert \Rightarrow \phi$ *non-injecti*
Univ. Property (*Thm. 22.2*)	$q_c : \mathcal{H} \to O_c$	*Universal among c-respecti*
Obs. Tradeoff (*Thm. 22.5*)	$c : \mathcal{H} \to O_c$	*Recovery* $\iff$ *injectivity*
Entropy Monotone (*Thm. 22.6*)	$c_1 \preceq c_2$	*Finer* $\pi \Rightarrow$ *smaller fibres*
Topology Change (*Thm. 21.3*)	$\rho_1, \rho_2 : X \to Y$	*Different* $\pi \Rightarrow$ *different* τ_π
State Illusion / Sheaf (*Prop. C.5*)	$\mathcal{O}_c$ *sheaf*	*Illusion = non-unique lifti*

Remark D.3. Proposition D.2 reveals that the entire framework has a single mathematical spine: fibres of projection maps.

Observational entropy $S_c(o) = \log |c^{-1}(o)|$ is the CLIO representational entropy $S_\pi(m) = \log \mathrm{Vol}(\pi^{-1}(m))$ in discrete form.

State illusion is fibre multiplicity.

State erasure is discarding the total space X after computing $\pi(x)$.

Admissibility is the geometry of the base space $Y = \mathcal{F}$.

Compiler correctness is preservation of fibre identity: $I(p)$ and $V(C(p))$ are in the same fibre of ρ.

The kernel theorem (Chapter 21) is a theorem about when changing π restructures the fibres of the new projection.

D.3 Open Frontier: Reachability Cohomology

The sheaf appendix (Appendix C) raised the question of sheaf cohomology as a measure of the obstruction to lifting local ob-

servations to global histories. We now state this as a formal open problem.

Definition D.4 (Reachability Cohomology (Conjectural))**.** *For an observational site $\mathcal{O}$ and the history sheaf $\mathcal{H}$, the* reachability cohomology groups $H^n(\mathcal{O}, \mathcal{H})$ *would measure the n-th obstruction to lifting local sections of $\mathcal{O}_c$ to global sections of $\mathcal{H}$.*

Remark D.5. $H^0(\mathcal{O}, \mathcal{H})$ would be the set of global histories (consistent with all local observations) — the "reconstructible" histories.

$H^1(\mathcal{O}, \mathcal{H})$ would measure the first-order obstruction: local observations that are compatible on overlaps but do not extend to a global history. This is the cohomological form of state illusion.

We do not develop this theory here. Computing H^1 for specific collapse rules (e.g., c_{LW} on a two-step execution) would be a natural next result.

D.4 Summary: The Projection-First Ontology

The v4 reformulation replaces the original slogan:

Histories are primary; states are derived.

with the deeper:

Projection is primary; both history and state are projections of the reachability geometry.

Under this view:

- A *history* is the trajectory through world-state space that produced the current reachability geometry.
- An *observable state* is the image of the reachability geometry under an observational projection.
- A *program* is a composition of deformation operators that sculpts the reachability field.

- A *type* is a certificate that the current trajectory remains within an admissible stratum of the reachability manifold.
- *Compiler correctness* is preservation of trajectory structure (same fibres, same deformations, same field geometry).

This is the framework that connects Spherepop, CLIO, Hidden Manifolds, The Admissibility Field, and RSVP as computational instances of a single mathematical program: the study of how projection maps structure the geometry of possibility.

References

Barendregt 1984 Barendregt, H.P. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland, 1984. The standard reference for lambda calculus, whose beta-reduction is interpreted here as a quotient operation on histories.

Feldman 1958 / Hájek 1958 Feldman, J. (1958); Hájek, J. (1958). The dichotomy theorem for Gaussian measures: two Gaussian measures are either equivalent or mutually singular. Provides the measure-theoretic foundation for Chapter 13's topological separation discussion.

Fowler 2005 Fowler, M. *Event Sourcing.* martinfowler.com, 2005. The domain-driven design tradition that first recognized the computational importance of history-primary architectures.

Hoare 1985 Hoare, C.A.R. *Communicating Sequential Processes.* Prentice Hall, 1985. The process calculus tradition from which Spherepop inherits its emphasis on communication events over state.

Mac Lane 1971 Mac Lane, S. *Categories for the Working Mathematician.* Springer, 1971. The category-theoretic framework for Chapters 11–12: history categories, functoriality, and collapse functors.

Martin-Löf 1984 Martin-Löf, P. *Intuitionistic Type Theory.* Bibliopolis, 1984. The proof-theoretic account of types as propositions, extended here into the reachability dimension.

Milner 1980 Milner, R. *A Calculus of Communicating Systems.* Springer, 1980. CCS is the closest process calculus antecedent to Spherepop's event-oriented operational semantics.

Reiter 1978 Reiter, R. "On closed world data bases." In: *Logic and Data Bases*, Plenum Press, 1978. The original source of the open-world / closed-world distinction formalized as `TypeMode` in Chapter 7.

Flyxion 2023 Flyxion. *CLIO: Constraint-Limited Information Ontology.* Independent Research, 2023. Available at `https://github.com/standardgalactic/`. The framework for representational entropy, projection-induced topology, and admissibility manifolds of which Spherepop is a computational instance.

Santoro, Waghmare, Panaretos 2024 Santoro, M.; Waghmare, S.; Panaretos, V.M. "Kernel embeddings of functional data and mutual singularity." The result interpreted in Chapter 13 as a theorem about representation-induced ontological reorganization.

Shapiro et al. 2011 Shapiro, M.; Preguiça, N.; Baquero, C.; Zawirski, M. "Conflict-free replicated data types." INRIA Technical Report, 2011. Distributed systems architectures whose consistency guarantees are expressed in terms of merge operations on historical records.

Wright and Felleisen 1994 Wright, A.K.; Felleisen, M. "A syntactic approach to type soundness." *Information and Computation* 115(1), 1994. The Preservation and Progress theorems (Chapter 6) follow the Wright–Felleisen syntactic proof methodology.