The Causal Virtual Machine (CVM): Execution Semantics of CSL on the Causal Kernel

S. D. Bolduc
Causal Theory (CT-United Framework)

December 2025

Abstract

This paper introduces the **Causal Virtual Machine (CVM)**, the execution engine that transforms CSL (Causal Specification Language) programs into evolving causal geometries running on the Causal Kernel. Unlike classical virtual machines—which interpret bytecode, maintain stacks, address memory, manage threads, or simulate hardware—the CVM manipulates a causal manifold. Its units of computation are π-nodes, $\sqrt{2}$-edges, $\sqrt{3}$-surfaces, φ-flows, and $\ln 5$-decay fields. It enforces the four Patriarch Theorems as non-negotiable invariants and executes by continuous causal evolution rather than discrete instruction steps.

The CVM is the intermediary layer between the Causal Kernel (CT-hardware) and CSL (CT-language). It is not a simulator or interpreter; it is a causal geometric transformer. Its purpose is to ensure that any CSL program resolves into a stable, coherent causal configuration consistent with the laws of the CT manifold. This paper provides the full formal semantics, operational model, consistency checks, and geometric state transition rules of the CVM.

Contents

1 Introduction

The Causal Kernel defines the primitive structures of computation:

$$\pi\text{-nodes}, \quad \sqrt{2}\text{-edges}, \quad \sqrt{3}\text{-surfaces}, \quad \varphi\text{-flows}, \quad \ln 5\text{-decay}.$$

CSL defines the abstract causal shapes programmers can specify.

What is missing is:

> A machine that takes CSL shapes and realizes them as dynamic, evolving causal configurations inside the kernel.

This is the role of the **Causal Virtual Machine (CVM)**. It is not a VM in the classical sense. It does not:

- emulate an instruction set,
- maintain registers or program counters,
- run a stack machine,
- simulate hardware.

Instead, it performs:

> **geometric execution**.

A CSL program describes a configuration. The CVM evolves that configuration until it reaches stable causal equilibrium.

2 CVM Architecture Overview

The CVM consists of four layers:

1. **Parser Layer** (shape extraction)
2. **Causal Graph Constructor** (π, $\sqrt{2}$, $\sqrt{3}$ assembly)
3. **Flow Engine** (φ propagation)
4. **Stabilizer Engine** ($\ln 5$ dissipation + Patriarch enforcement)

The output is a stable causal geometry:

$$\mathcal{G} = (N, E, S, F, D)$$

where:

- N are π-nodes,
- E are $\sqrt{2}$-edges,
- S are $\sqrt{3}$-surfaces,
- F are φ-flows,
- D are $\ln 5$-dissipation fields.

3 CVM Parsing: From CSL Text to Causal Abstract Syntax

The parser identifies:

- node declarations,
- edge relations,
- surface enclosures,
- flow statements,
- decay instructions,
- supracausal operators.

A CSL program is parsed into a **Causal Abstract Syntax Graph (CASG)**:

$$\text{CASG} = (V, C)$$

where:

- V are abstract causal entities,
- C are constraints between them.

For example:

```
node A
node B
edge A <-> B
flow propagate from A to B
```

is parsed as:

$$V = \{A, B\}, \quad C = \{\text{relation}, \text{flow}\}.$$

4 Causal Graph Construction

Once parsed, the CVM constructs the executable causal graph:

$$\mathcal{G}_0 = (N_0, E_0, S_0, F_0, D_0).$$

This is not yet stable or valid. It is the raw geometry.

Examples:

$$N_0 = \{A, B, C\},\ E_0 = \{A \leftrightarrow B\},\ S_0 = \{S_1\}.$$

Surfaces are constructed as 2-dimensional curvature membranes around sets of nodes. φ-flows are vector fields seeded between node pairs.

5 Flow Engine: φ-Propagation

The flow engine evolves $\mathcal{G}_0$ through φ-iteration:

$$x_{n+1} = \varphi x_n + x_{n-1}.$$

Flows propagate across edges, deform surfaces, and create new relations.

The evolution continues until:

$$\|\mathcal{G}_{n+1} - \mathcal{G}_n\| < \epsilon.$$

6 Stabilizer Engine: $\ln 5$ Dissipation

Once φ propagation has created growth and change, $\ln 5$ dissipation removes:

- unused flows,
- incoherent curvature,
- conflicting edges,
- unstable nodes.

Decay evolution:

$$D_{n+1} = e^{-\ln 5} D_n = \frac{D_n}{5}.$$

This ensures the system returns to stability.

7 Patriarch Enforcement

The CVM enforces all constraints imposed by Q1–Q4.

Q1 Enforcement

Reject any configuration where flow cycles diverge.

Q2 Enforcement

Decompose edge clusters into orthogonal bases.

Q3 Enforcement

Check curvature–tension inequality.

Q4 Enforcement

Ensure entropy decreases monotonically.

8 Execution: Convergence to a Stable Causal Geometry

Final execution is:

$$\lim_{n\to\infty} \mathcal{G}_n = \mathcal{G}_\infty.$$

This limit is the **program result**.

There is no "output," only a final coherent shape.

9 CVM Safety Model

Because the CVM inherits CT invariants, it guarantees:

- no deadlocks,
- no race conditions,
- no memory corruption,
- no inconsistent states,
- no undefined behavior.

It is mathematically impossible for the CVM to produce an invalid result.

10 Examples

10.1 Self-Organizing Network

```
node A
node B
node C
edge A <-> B
edge B <-> C
flow stabilize from A to C
surface Domain encloses {A,B,C}
decay on Domain
```

Execution yields a triangle with a stable flow.

10.2 Two-Agent Influence

```
node X
node Y
edge X <-> Y
flow affect from X to Y
decay on {X,Y}
```

Execution yields a coherent influence relation.

11 Conclusion

The Causal Virtual Machine is the execution semantics of CSL. It transforms abstract causal shapes into stable geometric configurations consistent with the CT constants and Patriarch theorems. It is not an interpreter, a simulator, or a traditional VM: it is the dynamic realization of causal geometry.