Spherepop Calculus: Internalizing Probability, Concurrency, and Geometry

Flyxion

September 28, 2025

Abstract

Spherepop Calculus (SPC) is a novel computational formalism extending lambda calculus with geometric scope (`Sphere` and `Pop`), concurrent composition (`Merge`), probabilistic branching (`Choice`), and structural symmetries (`Rotate`). This paper explores SPC's design, emphasizing the `doomCoin p` construct as a canonical example of its probabilistic and tensorial semantics. We compare SPC with traditional and probabilistic lambda calculi, highlighting its ability to internalize probability, concurrency, and geometric structure. Implementations in Haskell and a Racket evaluator skeleton demonstrate practical realizations of SPC's concepts.

1 Introduction

Spherepop Calculus (SPC) reimagines lambda calculus by introducing geometric scoping through `Sphere` and `Pop`, parallel composition via `Merge`, probabilistic branching with `Choice`, and structural operations like `Rotate`. Unlike traditional lambda calculus, SPC natively supports concurrent and probabilistic computations within a type discipline extending the Calculus of Constructions, with semantics in a presheaf topos enriched with the distribution monad. This paper elucidates SPC's design, focusing on `doomCoin p` as a pedagogical archetype, and provides practical implementations in Haskell and Racket.

2 Core Constructs of Spherepop Calculus

SPC extends lambda calculus with these primitives:

- **Sphere/Pop**: geometric scoping of abstraction/application, where $\mathsf{Sphere}(x{:}A.\,t)$ denotes a function and $\mathsf{Pop}(f, u)$ applies it.
- **Merge**: parallel/nondeterministic composition, interpreted as a tensor product.
- **Choice**: probabilistic branching, returning either A (internal) or $\mathsf{Dist}(A)$.
- **Rotate**: cyclic rotation over homogeneous Boolean tensors.

2.1 Syntax and Typing

$$t, u \quad ::= \quad \mathsf{Var}(x) \mid \mathsf{Sphere}(x{:}A.\,t) \mid \mathsf{Pop}(t,u) \mid \mathsf{Merge}(t,u) \mid \mathsf{Choice}(p,t,u) \mid \mathsf{Rotate}(k,t) \mid \mathsf{LitUnit} \mid \mathsf{LitBool}\,b \mid \mathsf{LitNat}\,n \mid \mathsf{If}(b,t,u) \mid \mathsf{Add}(t,u)$$

$$\frac{\Gamma, x{:}A \vdash t : B}{\Gamma \vdash \mathsf{Sphere}(x{:}A.\,t) : A \to B} \qquad \frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{Pop}(f,u) : B}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash \mathsf{Merge}(t,u) : A \otimes B} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash u : A \quad p \in [0,1]}{\Gamma \vdash \mathsf{Choice}(p,t,u) : A}$$

$$\frac{\Gamma \vdash t : \mathsf{Bool}^{\otimes k},\ k \geq 1}{\Gamma \vdash \mathsf{Rotate}(k,t) : \mathsf{Bool}^{\otimes k}}$$

2.2 Operational Semantics

$$\mathsf{Pop}(\mathsf{Sphere}(x{:}A.\,t), v) \longrightarrow t[v/x] \qquad \mathsf{Choice}(p,t,u) \xRightarrow{p} t \qquad \mathsf{Choice}(p,t,u) \xRightarrow{1-p} u$$

$$\mathsf{Merge}(v,w) \longrightarrow v \otimes w \quad \mathsf{Rotate}\big(k, v_1 \otimes \cdots \otimes v_n\big) \longrightarrow v_{1+k \bmod n} \otimes \cdots \otimes v_{n+k \bmod n}$$

2.3 Categorical Semantics

Sphere/Pop are exponentials/evaluation morphisms, Merge is a tensor product, and Choice is convex combination in the distribution monad.

3 Canonical Example: `doomCoin p`

$$\mathsf{doomCoin}\,p \ \equiv\ \mathsf{Choice}\big(p, \mathsf{LitBool}\,\#t, \mathsf{LitBool}\,\#f\big).$$

3.1 Syntax and Typing

$$\frac{\Gamma \vdash \mathsf{LitBool}\,\#t : \mathsf{Bool} \quad \Gamma \vdash \mathsf{LitBool}\,\#f : \mathsf{Bool} \quad p \in [0,1]}{\Gamma \vdash \mathsf{doomCoin}\,p : \mathsf{Bool} \text{ or } \mathsf{Dist}(\mathsf{Bool})}$$

3.2 Operational Semantics

$$\mathsf{doomCoin}\,p \xRightarrow{p} \mathsf{LitBool}\,\#t \qquad \mathsf{doomCoin}\,p \xRightarrow{1-p} \mathsf{LitBool}\,\#f$$

For example,

$$\mathsf{Merge}\big(\mathsf{Choice}(0.2, \#t, \#f), \mathsf{Choice}(0.5, \#t, \#f)\big) \xRightarrow{0.2 \cdot 0.5} \#t \otimes \#t \quad \text{(and other paths).}$$

3.3 Denotational Semantics

$$[\![\mathsf{doomCoin}\, p]\!] = p \cdot \delta_{\#t} + (1-p) \cdot \delta_{\#f}.$$

For $\mathsf{Merge}(\mathsf{doomCoin}\, p_1, \ldots, \mathsf{doomCoin}\, p_n)$, the observable $\mathsf{anyDoom}$ yields:

$$\Pr\big[\mathsf{anyDoom}(\mathsf{Merge}(\mathsf{doomCoin}\, p_1, \ldots, \mathsf{doomCoin}\, p_n))\big] = 1 - \prod_{i=1}^{n}(1-p_i).$$

$$\begin{tikzcd}
\mathsf{doomCoin}\, p \arrow[r, "{[\![\cdot]\!]}"] \arrow[d, "\text{Choice}"'] & p \cdot \delta_{\#t} + (1-p) \cdot \delta_{\#f} \arrow[d, "\text{tensor}"] & \\
\mathsf{Dist}(\mathsf{Bool}) \arrow[r, "\mathsf{Merge}"'] & \mathsf{Dist}(\mathsf{Bool}^{\otimes n}) \arrow[r, "\mathsf{anyDoom}"] & \mathsf{Dist}(\mathsf{Bool})
\end{tikzcd}$$

Figure 1: Categorical flow: `doomCoin p` to `anyDoom` via tensor product.

3.4 Rationale

`doomCoin p` is canonical because it (i) makes p intrinsic, (ii) uses the Bernoulli base case, (iii) leverages tensorial independence, and (iv) is pedagogically clear.

4 Comparison with Lambda Calculi

- **Lambda Calculus**: external oracle (e.g., $\lambda r.$ `if` $r < p$ `then #t else #f`).
- **Probabilistic Lambda Calculus**: `choice`(p, e_1, e_2) but no geometric/tensorial structure.
- **SPC**: unifies Choice, Merge, and Sphere/Pop [1–3].

5 Operational Semantics (Appendix)

$$\frac{}{\mathsf{Pop}(\mathsf{Sphere}(x{:}A.\, t), v) \longrightarrow t[v/x]}\ \textsc{E-Beta} \qquad \frac{}{\mathsf{Choice}(p, t, u) \overset{p}{\Longrightarrow} t}\ \textsc{E-Choice-1} \qquad \frac{}{\mathsf{Choice}(p, t, u) \overset{1-p}{\Longrightarrow} u}\ \textsc{E-C}$$

6 Implementation in Haskell

(See `spherepop.hs`.) Type checking for Merge, Choice, Rotate; big-step evaluation via a distribution monad; anyDoom as an observable.

7 Racket Evaluator

(See `spherepop.rkt`.) Mirrors Haskell: typing/evaluation, distribution monad, and anyDoom.

References

[1] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

[2] Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22(3):328–350, 1981.

[3] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Information and Computation*, 100(1):1–40, 1992.