Spherepop Calculus

Flyxion

September 20, 2025

Abstract

The *Spherepop Calculus* (SPC) is a novel functional language and type-theoretic framework that unifies abstraction, concurrency, and probabilistic choice within a single geometric model of computation. SPC extends the λ-calculus by reinterpreting abstraction and application as `Sphere` and `Pop`, visualizing scope as nested spheres rather than syntactic parentheses. It introduces `Merge` as a categorical parallel operator with tensorial semantics, and `Choice` as a primitive for probabilistic branching, expressible either internally or via a distribution monad. The type system extends the Calculus of Constructions with dependent types, while the denotational semantics is given in a presheaf topos enriched with the Giry distribution monad. We establish meta-theoretic properties of SPC, including preservation, progress, and adequacy of the probabilistic semantics, together with a formal *Independent Channels Lemma* that quantifies aggregated risk across merged probabilistic branches. Historical analysis positions SPC at the intersection of λ-calculus, π-calculus, probabilistic λ-calculi, and dependent type theory, while translations demonstrate that SPC strictly subsumes these calculi up to natural fragments. The result is a language that serves both as a foundation for probabilistic and concurrent computation and as a categorical framework for reasoning about structured scope, uncertainty, and interaction.

Contents

1 Introduction

The study of computation has been shaped by a small number of canonical calculi, each capturing different aspects of programming and reasoning. The λ-calculus, introduced by Church in the 1930s, provides a minimal yet universal account of higher-order functional abstraction. Process calculi such as Milner's π-calculus emphasize concurrency and interaction, while extensions of λ-calculus with probabilistic primitives provide a foundation for reasoning under uncertainty. Dependent type theories, beginning with Martin-Löf, further embed computation into a constructive foundation for mathematics, realized in proof assistants such as Coq.

The *Spherepop Calculus* (SPC) proposes a synthesis of these traditions, introducing a geometric reinterpretation of abstraction and application as `Sphere` and `Pop`. Rather than treating scope as a merely syntactic device (parentheses), SPC represents computation as the structured opening and closing of *spheres*, which can be nested, merged, or collapsed. This geometric intuition makes scope explicit, supports natural visualization, and provides a foundation for extending the calculus with parallelism and probability.

SPC departs from the classical λ-calculus in three key ways:

1. **Merge.** A primitive operator for nondeterministic or parallel composition, interpreted categorically as a tensor product in a monoidal category.

2. **Choice.** A probabilistic branching operator, admitting two semantics: an internal version, where choice returns a value of type A, and a distribution-monad version, where choice yields an element of $\mathsf{Dist}(A)$.

3. **Dependent extension.** SPC generalizes the Calculus of Constructions by allowing Π- and Σ-types in the presence of Merge and Choice, ensuring compositionality with categorical semantics in a presheaf topos.

The contributions of this paper are fourfold:

- We define the syntax, typing rules, and operational semantics of SPC, highlighting its core primitives (`Sphere`, `Pop`, `Merge`, `Choice`).

- We give a denotational semantics in the presheaf topos $[\mathsf{Sphere}^{op}, \mathsf{Set}]$, enriched with the Giry distribution monad, and prove adequacy with respect to the operational semantics.

- We establish meta-theoretic properties, including preservation, progress, and a formal *Independent Channels Lemma* that quantifies disjunctive risk aggregation across merged probabilistic branches.

- We situate SPC within the historical development of computation, compare it with λ-calculus, π-calculus, and probabilistic λ-calculi, and provide translations showing that SPC strictly subsumes these calculi up to natural fragments.

Taken together, SPC offers a unified framework for functional abstraction, concurrency, and probabilistic reasoning, supported by dependent types and categorical semantics. It is simultaneously a programming language, a logical foundation, and a conceptual model of computation where scope is geometric, parallelism is tensorial, and probability is compositional.

1.1 Motivation and Overview

We establish the foundational properties of the Spherepop Calculus (SPC) through three main technical contributions.

Type Safety. We prove *Preservation* (well-typed terms remain well-typed under reduction) and *Progress* (a closed well-typed term is either a value or can take a reduction step). These results extend the standard type-safety arguments for λ-calculus to a setting enriched with dependent types, Merge, and Choice.

Adequacy of Probabilistic Semantics. We show that the denotational semantics in the presheaf topos, enriched with the distribution monad, is adequate with respect to the operational semantics. Specifically, the expectation of any observable function under the operational distribution coincides with its expectation under the denotational distribution. This adequacy ensures that probabilistic reasoning in SPC is compositional.

Expressivity. We define translations from the simply-typed λ-calculus, probabilistic λ-calculus, and a nondeterministic parallel fragment of π-calculus into SPC. These translations preserve typing and operational behavior, and demonstrate that SPC strictly subsumes each source calculus. As a corollary, SPC provides a unifying framework for higher-order functions, probabilistic choice, and parallel composition.

Independent Channels Lemma. Finally, we formalize the aggregation of independent probabilistic branches under Merge. We prove that the probability of disjunctive failure across n independent channels is given by $1 - \prod_{i=1}^{n}(1 - p_i)$, aligning the operational and denotational accounts. This lemma illustrates how SPC captures structured reasoning about risk in concurrent probabilistic systems.

The contributions of this paper are as follows:

- **Geometric scope model.** We introduce *Sphere* and *Pop* as geometric abstractions of λ-abstraction and application, making scope explicit through spherical nesting.
- **Parallel and probabilistic primitives.** We extend the calculus with *Merge* (categorical tensor for parallel composition) and *Choice* (probabilistic branching), with both internal and distribution-monad interpretations.
- **Dependent type system.** We integrate Π- and Σ-types into SPC, yielding a *probabilistic, monoidal calculus of constructions.*
- **Denotational semantics.** We give a semantics in the presheaf topos $[\mathsf{Sphere}^{op}, \mathsf{Set}]$ enriched with the Giry distribution monad, and prove *adequacy* with respect to operational semantics.
- **Meta-theory.** We establish preservation, progress, and a formal *Independent Channels Lemma* quantifying disjunctive risk aggregation under Merge.
- **Expressivity.** We provide translations from λ-calculus, probabilistic λ-calculus, and a parallel fragment of π-calculus into SPC, proving SPC strictly subsumes these calculi.

1.2 Structure of the Paper

The remainder of the paper is organized as follows. Section 4 introduces the syntax, types, and typing rules of SPC. Section 5 defines the operational semantics, including β-reduction, Merge, and Choice. Section 6 presents the denotational semantics in a presheaf topos enriched with the distribution monad. Section 7 establishes the meta-theoretic results: preservation, progress, adequacy, and the Independent Channels Lemma. Section 8 situates SPC in historical context, tracing antecedents from λ-calculus to probabilistic programming. Section 9 compares SPC to related calculi and formalizes its expressivity. Section 10 gives constructive translations from λ-calculus, probabilistic λ-calculus, and a fragment of π-calculus into SPC. Finally, the appendices provide auxiliary lemmas, substitution proofs, and worked examples illustrating the calculus in action.

2 Related Work

The Spherepop Calculus (SPC) builds on several long-standing traditions in logic, type theory, and programming language semantics, while introducing new geometric primitives for abstraction, concurrency, and probabilistic choice.

Foundations in λ-calculus and type theory. The λ-calculus [5, 3] is the canonical foundation of functional abstraction, and type-theoretic refinements such as Martin-Löf's intuitionistic type theory [17] and the Calculus of Constructions [6] enable expressive proof systems with dependent types. SPC inherits this tradition but reinterprets scope geometrically through `Sphere` and `Pop`, extending it with native parallel and probabilistic constructs. The broad development of type systems is summarized by Pierce [23], which SPC builds upon by fusing typing with concurrency and uncertainty.

Categorical semantics. The Curry–Howard–Lambek correspondence [14], Mac Lane's coherence results [16], and Lawvere's elementary topos theory [15] frame the categorical interpretation of logical systems. Awodey [2] provides a modern reference that situates these ideas within contemporary category theory. SPC draws on this lineage by giving its denotational semantics in a presheaf topos enriched with a distribution monad, with `Merge` modeled as a tensor. Street's formal theory of monads [24] and Moggi's computational monads [21] provide the conceptual background for SPC's treatment of probabilistic and parallel computation as compositional effects.

Linear and resource-sensitive calculi. Girard's linear logic [8] introduced explicit control of resources in proof theory and computation. While SPC is not linear, its geometric nesting of scopes and its tensorial `Merge` operator echo concerns of resource management and structural invariants familiar from the linear tradition.

Probabilistic semantics and programming. The semantics of probabilistic programs has a long history [13, 9], recently extended to dependent types and constructive foundations [1]. Jacobs and Mandemaker's coalgebraic approach [12] and Fritz's synthetic treatment of Markov kernels [7] highlight how categorical structures support probability theory. SPC contributes to this tradition by internalizing probabilistic branching as a primitive operator (`Choice`), interpretable both operationally and via monads. Contemporary probabilistic programming languages such as Church, Anglican, and WebPPL [10, 25, 11] show the practical importance of such models; SPC offers a categorical and type-theoretic foundation that could undergird future implementations.

Concurrency and process calculi. Milner's CCS [18] and π-calculus [19, 20] provided the foundational frameworks for concurrency and mobility of processes. SPC's `Merge` resembles nondeterministic parallel composition but is grounded in categorical semantics rather than name-passing. This places SPC closer to categorical models of concurrency, while retaining the operational intuition of interaction between independent channels.

Implementations and proof assistants. The Calculus of Inductive Constructions is realized in Coq [4], while Agda [22] demonstrates dependently typed programming in a more lightweight style. SPC shares the constructive spirit of these systems but extends their expressivity with concurrency and probabilistic constructs at the core. This suggests that SPC could serve as a foundation for future proof assistants or domain-specific probabilistic programming tools with certified semantics.

Summary. SPC thus situates itself at the intersection of four lines of research: λ-calculus and dependent types, categorical logic and monads, probabilistic semantics, and concurrency theory. Its novelty lies in synthesizing these strands into a single, geometrically motivated calculus where scope, parallelism, and probability are treated as first-class compositional notions.

3 Overview of the Language

The **Spherepop Calculus (SPC)** is a higher-order, dependent-typed functional language designed to model computation through **spherical scoping primitives**. Its distinctive features are:

3.0.1 1. Core Primitives

- **Sphere** – abstraction operator, analogous to λ, but visualized as opening a scope ("entering a bubble").
- **Pop** – application operator, analogous to β-reduction, representing the act of collapsing a sphere with an argument.
- **Merge** – parallel or disjunctive composition, modeling nondeterministic or concurrent branching.
- **Choice** – probabilistic branching operator, either interpreted internally (as a weighted selector) or via the distribution monad.
- **Nest** – syntactic sugar for hierarchical application, enforcing structured scoping.

Together, these primitives generalize λ-calculus while encoding higher-order reasoning about scope, concurrency, and probability.

3.0.2 2. Type Discipline

SPC extends dependent type theory:

- **Dependent function types** (Π-types) capture parameterized spheres.
- **Dependent sum types** (Σ-types) capture merged data.
- **Merge** preserves type alignment, enforcing that both branches produce the same type.
- **Choice** comes in two flavors:
  - *Internal*: returns an element of type A.
  - *Monadic*: returns a distribution over A, aligning with Giry's distribution monad.

Thus, SPC can be seen as a **probabilistic Calculus of Constructions** variant.

3.0.3 3. Operational Semantics

- **β-reduction**: $\mathrm{Pop}(\mathrm{Sphere}(x : A.\, t), u) \to t[u/x]$.
- **Merge**: associative and commutative; normalizes via structural equivalence.
- **Choice**: reduces by stochastic sampling (t w.p. p, u w.p. $1 - p$).

The calculus is confluent for the deterministic fragment and admits stochastic confluence properties for probabilistic terms.

3.0.4 4. Denotational Semantics

SPC is interpreted in a **presheaf topos** $[\mathsf{Sphere}^{op}, \mathsf{Set}]$:

- **Terms**: morphisms in the topos.
- **Sphere/Pop**: exponential and evaluation morphisms.
- **Merge**: monoidal product.
- **Choice**: convex combination in the distribution monad, yielding true probabilistic semantics.

Adequacy theorems ensure that operational sampling aligns with denotational expectation.

3.0.5 5. Meta-Theory

SPC satisfies standard safety results:

- **Preservation** – reduction preserves types.
- **Progress** – well-typed closed terms reduce or are values.
- **Adequacy** – denotational semantics agrees with operational probabilities.
- **Independent Channels Lemma** – formalizes disjunctive risk aggregation:

$$\Pr[\text{doom in merged channels}] = 1 - \prod_i (1 - p_i).$$

3.0.6 6. Conceptual Role

Unlike traditional λ-calculus:

- SPC emphasizes **scoping as geometry** (spheres rather than parentheses).
- Merge and Choice bring **parallelism and probability** into the core calculus.
- The categorical semantics makes SPC not just a programming language, but a **theory of structured, probabilistic reasoning**, equally at home in logic, semantics, and physics.

4 Syntax and Typing

4.1 Grammar of Terms

$$\begin{array}{rcll}
t, u & ::= & x & \text{variable} \\
 & \mid & a & \text{atom / constant} \\
 & \mid & \mathrm{Sphere}(x{:}A.\, t) & \text{abstraction} \\
 & \mid & \mathrm{Pop}(t, u) & \text{application} \\
 & \mid & \mathrm{Merge}(t, u) & \text{parallel / disjunction} \\
 & \mid & \mathrm{Nest}(t, u) & \text{syntactic sugar for Pop} \\
 & \mid & \mathrm{Choice}(p, t, u) & \text{probabilistic choice}
\end{array}$$

4.2 Types and Contexts

Contexts: $\Gamma ::= \cdot \mid \Gamma, x{:}A$.

4.3 Typing Rules

$$\frac{x{:}A \in \Gamma}{\Gamma \vdash x : A}\ \textsc{Var} \qquad \frac{}{\Gamma \vdash a : A}\ \textsc{Atom} \qquad \frac{\Gamma \vdash A : \mathsf{Type}_i \qquad \Gamma, x{:}A \vdash B : \mathsf{Type}_j}{\Gamma \vdash \Pi x{:}A.\, B : \mathsf{Type}_{\max(i,j)}}$$

$$\frac{\Gamma, x{:}A \vdash t : B}{\Gamma \vdash \mathrm{Sphere}(x{:}A.\, t) : \Pi x{:}A.\, B} \qquad \frac{\Gamma \vdash f : \Pi x{:}A.\, B \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathrm{Pop}(f, u) : B[u/x]} \qquad \frac{\Gamma \vdash A : \mathsf{Type}_i \qquad \Gamma, x{:}A \vdash B : \mathsf{Type}_j}{\Gamma \vdash \Sigma x{:}A.\, B : \mathsf{Type}_{\max(i,j)}}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B[a/x]}{\Gamma \vdash (a, b) : \Sigma x{:}A.\, B} \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathrm{Merge}(t, u) : A}\ \textsc{Merge}$$

$$\frac{\Gamma \vdash p : \mathsf{Prob} \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathrm{Choice}(p, t, u) : A}\ \textsc{Choice}$$

4.4 Dependent Types and Extensions

The Spherepop Calculus extends the simply-typed core with full *dependent types*, generalizing the Calculus of Constructions. This enables the type of a term to vary with the values of its arguments, and allows expressive encodings of logical quantification, data structures, and program specifications.

Dependent Functions (Π-types). A dependent function type $\Pi x{:}A.\, B(x)$ represents a family of results $B(x)$ parameterized by an argument $x : A$. In SPC this is introduced by the abstraction form $\mathsf{Sphere}(x{:}A.\, t)$ and eliminated by application $\mathsf{Pop}(f, u)$. Typing rules are:

$$\frac{\Gamma \vdash A : \mathsf{Type}_i \qquad \Gamma, x : A \vdash B(x) : \mathsf{Type}_j}{\Gamma \vdash \Pi x : A.\, B(x) : \mathsf{Type}_{\max(i,j)}}\ \Pi\text{-}\textsc{Form}$$

$$\frac{\Gamma, x : A \vdash t : B(x)}{\Gamma \vdash \mathsf{Sphere}(x{:}A.\, t) : \Pi x : A.\, B(x)}\ \Pi\text{-}\textsc{Intro}$$

$$\frac{\Gamma \vdash f : \Pi x : A.\, B(x) \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{Pop}(f, u) : B(u)}\ \Pi\text{-}\textsc{Elim}$$

Dependent Pairs (Σ-types). A dependent sum $\Sigma x{:}A.\, B(x)$ represents pairs (a, b) where $a : A$ and $b : B(a)$. SPC introduces such terms directly and allows pattern decomposition:

$$\frac{\Gamma \vdash A : \mathsf{Type}_i \qquad \Gamma, x : A \vdash B(x) : \mathsf{Type}_j}{\Gamma \vdash \Sigma x : A.\, B(x) : \mathsf{Type}_{\max(i,j)}}\ \Sigma\text{-}\textsc{Form}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B(a)}{\Gamma \vdash (a, b) : \Sigma x : A.\, B(x)}\ \Sigma\text{-}\textsc{Intro}$$

Elimination is definable via dependent projection functions, aligning with the standard rules of dependent type theory.

Interaction with Merge. The tensorial operator Merge requires both arguments to have the same type. In the dependent setting, this condition is refined: if $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$, then $\Gamma \vdash M \otimes N : A$. This ensures that parallel branches preserve a uniform dependent type, while still allowing the type A itself to depend on contextual variables.

Interaction with Choice. The probabilistic operator $\mathsf{Choice}(p, M, N)$ is well-typed whenever both branches agree on type:

$$\Gamma \vdash M : A, \quad \Gamma \vdash N : A \quad \Rightarrow \quad \Gamma \vdash \mathsf{Choice}(p, M, N) : A.$$

Alternatively, in the distributional extension, we admit:

$$\Gamma \vdash \mathsf{Choice}(p, M, N) : \mathsf{Dist}(A).$$

This makes explicit the transition from internal probabilistic branching to external distribution-valued semantics.

Universes. As in the Calculus of Constructions, SPC assumes a cumulative hierarchy of type universes Type_i indexed by natural numbers i. Dependent types can be formed at any level, and type constructors preserve universe stratification.

Expressive Power. With Π- and Σ-types, SPC can encode first-order and higher-order logic, inductive families, and dependent specifications of probabilistic processes. In particular, the combination of dependent sums with Choice permits *probabilistic data structures* whose shape depends on sampled random values, while dependent functions over Merge enable *concurrent families of proofs and programs.*

In summary, the dependent extension equips SPC with the full expressivity of modern type theories, while ensuring that Merge and Choice remain type-safe and compositional within this enriched framework.

Lemma 4.1 (Merge of Dependent Pairs yields a Joint Distribution)**.** *Let A be a base type and let* $\mathsf{Vec} : \mathbb{N} \to \mathsf{Type}$ *be a family. Suppose*

$$\Gamma \vdash P_1 : \Sigma n : \mathbb{N}.\, \mathsf{Vec}(n) \qquad \textit{and} \qquad \Gamma \vdash P_2 : \Sigma m : \mathbb{N}.\, \mathsf{Vec}(m).$$

Then:

(Typing) $\Gamma \vdash \mathsf{Merge}(P_1, P_2) \ : \ \big(\Sigma n : \mathbb{N}.\, \mathsf{Vec}(n)\big) \otimes \big(\Sigma m : \mathbb{N}.\, \mathsf{Vec}(m)\big).$

(Operational) *If $P_1 \Rightarrow (n, v)$ and $P_2 \Rightarrow (m, w)$ by (possibly probabilistic) reduction, then*

$$\mathsf{Merge}(P_1, P_2) \ \Rightarrow \ (n, v) \parallel (m, w),$$

modulo the associativity/commutativity congruence of Merge*.*

(Denotational) *In the presheaf topos with distribution monad* Dist*, if $P_1 \in \mathsf{Dist}\big(\Sigma n.\, \mathsf{Vec}(n)\big)$ and $P_2 \in \mathsf{Dist}\big(\Sigma m.\, \mathsf{Vec}(m)\big)$ are independent, then*

$$\mathsf{Merge}(P_1, P_2) \ = \ P_1 \otimes P_2 \ \in \ \mathsf{Dist}\Big(\big(\Sigma n.\, \mathsf{Vec}(n)\big) \times \big(\Sigma m.\, \mathsf{Vec}(m)\big)\Big),$$

where $\otimes$ is the product (independent) measure and the codomain identifies the tensor with the cartesian product object in the underlying topos.

Sketch. Typing. By the Merge typing rule (tensor introduction), if both branches share well-formed types, then $\Gamma \vdash \mathsf{Merge}(P_1, P_2)$ has the tensor of those types.

Operational. Reduction of Merge proceeds by reducing either branch; if both normalize to (n, v) and (m, w), the merged term normalizes to their parallel composition, up to A/C congruence.

Denotational. By the semantics of Merge as tensor and of probability via Dist, independent branches denote product measures. Therefore the denotation of the merge is the independent product of the branch distributions, as stated. □

Corollary 4.2 (Pushforward of Merged Dependent Pairs)**.** *Let P_1 and P_2 be as in Lemma 4.1, and let*

$$\Phi : \big(\Sigma n : \mathbb{N}.\, \mathsf{Vec}(n)\big) \times \big(\Sigma m : \mathbb{N}.\, \mathsf{Vec}(m)\big) \ \to \ B$$

be a dependent function into some type B. Then the SPC term

$$\Phi\big(\mathsf{Merge}(P_1, P_2)\big)$$

is well-typed with type B, and its denotational semantics is the pushforward measure of the product distribution:

$$\Phi(\mathsf{Merge}(P_1, P_2)) \ = \ \Phi_{\big(\, P_1 \otimes P_2 \,\big)}.$$

Sketch. Typing. By Lemma 4.1, $\mathsf{Merge}(P_1, P_2)$ has type $(\Sigma n.\, \mathsf{Vec}(n)) \otimes (\Sigma m.\, \mathsf{Vec}(m))$, which is isomorphic to the product $(\Sigma n.\, \mathsf{Vec}(n)) \times (\Sigma m.\, \mathsf{Vec}(m))$. Thus applying Φ yields a term of type B.

Operational. Whenever P_1 reduces to (n, v) and P_2 to (m, w), the merged term reduces to $(n, v) \| (m, w)$, and applying Φ reduces further to $\Phi((n, v), (m, w))$.

Denotational. Since $\mathsf{Merge}(P_1, P_2)$ is the product distribution, and Φ is interpreted as a measurable map, functoriality of the distribution monad implies that its denotation is the pushforward Φ. □

Worked Example: Dependent Probabilistic Function. Consider a type $\mathbb{N}$ of natural numbers, and let $\mathsf{Vec}(n)$ denote the type of vectors of length n over some base type A. We show how SPC can express a probabilistic program that first samples a natural number n and then returns a vector of that length.

Step 1: Sample a length. Define a probabilistic choice over natural numbers:

$$L \;=\; \mathsf{Choice}(p,\,0,\,1) : \mathbb{N}.$$

Here, L evaluates to 0 with probability p and to 1 with probability $1-p$.

Step 2: Dependent family of vectors. We form a dependent function type:

$$F : \Pi n : \mathbb{N}.\, \mathsf{Vec}(n).$$

Given n, F produces a vector of length n.

Step 3: Construct dependent pair. We bind the probabilistic result of L with a vector of matching length:

$$P \;=\; (\,L,\; \mathsf{Sphere}(n{:}\mathbb{N}.\, \mathsf{Vec}(n))\,) : \Sigma n : \mathbb{N}.\, \mathsf{Vec}(n).$$

Step 4: Probabilistic extension. If we evaluate P operationally, we obtain:

$$P \;\to\; \begin{cases} (0,\; \mathsf{Vec}(0)) & \text{with probability } p, \\ (1,\; \mathsf{Vec}(1)) & \text{with probability } 1-p. \end{cases}$$

Denotational semantics. In the presheaf topos enriched with the distribution monad, the denotation is:

$$P \;=\; p \cdot \delta_{(0,\, \mathsf{Vec}(0))} \;+\; (1-p) \cdot \delta_{(1,\, \mathsf{Vec}(1))}.$$

Interpretation. This term demonstrates how Σ-types interact with Choice: the type of the second component (the vector) depends on the sampled value of the first component (the length). SPC thus supports *probabilistic dependent structures*, where the sampled data determines the type of subsequent computation.

Worked Example: Dependent Pairs under Merge. Suppose we model two independent random processes that each return a natural number n together with a dependent vector of length n.

Step 1: Two independent probabilistic samplers. Let

$$L_1 = \mathsf{Choice}(p,\,0,\,1) : \mathbb{N}, \qquad L_2 = \mathsf{Choice}(q,\,1,\,2) : \mathbb{N}.$$

Here, L_1 yields 0 or 1, while L_2 yields 1 or 2.

Step 2: Dependent pairs. We construct dependent pairs where the second component is a vector of the chosen length:

$$P_1 = (\,L_1,\, \mathsf{Vec}(L_1)\,) : \Sigma n : \mathbb{N}.\, \mathsf{Vec}(n),$$
$$P_2 = (\,L_2,\, \mathsf{Vec}(L_2)\,) : \Sigma m : \mathbb{N}.\, \mathsf{Vec}(m).$$

Step 3: Merge composition. We combine these two processes using `Merge`:

$$M = \mathsf{Merge}(P_1, P_2).$$

Step 4: Operational behavior. Operationally, M branches into the four possible outcomes:

$$M \;\to\; \begin{cases} (0, \mathsf{Vec}(0)) \;\;||\;\; (1, \mathsf{Vec}(1)) & \text{with prob. } p \cdot q, \\ (0, \mathsf{Vec}(0)) \;\;||\;\; (2, \mathsf{Vec}(2)) & \text{with prob. } p \cdot (1-q), \\ (1, \mathsf{Vec}(1)) \;\;||\;\; (1, \mathsf{Vec}(1)) & \text{with prob. } (1-p) \cdot q, \\ (1, \mathsf{Vec}(1)) \;\;||\;\; (2, \mathsf{Vec}(2)) & \text{with prob. } (1-p) \cdot (1-q). \end{cases}$$

Step 5: Denotational interpretation. In the presheaf topos with distribution monad, Merge corresponds to the product measure of two independent distributions. Thus

$$M = P_1 \otimes P_2,$$

which is the convex combination above, represented as a joint distribution over dependent pairs.

Interpretation. This example shows how SPC can reason about *dependent random structures in parallel.* Merge enforces independence at the semantic level (tensor product of distributions), while Σ-types ensure that each component is well-typed with respect to the dependent relation between the chosen number and the vector constructed from it. The combination thus yields a joint dependent distribution.

Worked Example: Concatenation as a Pushforward. Let $\mathsf{Vec} : \mathbb{N} \to \mathsf{Type}$ and let

$$P_1 : \Sigma n : \mathbb{N}.\, \mathsf{Vec}(n) \qquad \text{and} \qquad P_2 : \Sigma m : \mathbb{N}.\, \mathsf{Vec}(m)$$

be the dependent probabilistic pairs from the previous example. Define the dependent post-processing map

$$\Phi \;:\; \big(\Sigma n : \mathbb{N}.\, \mathsf{Vec}(n)\big) \times \big(\Sigma m : \mathbb{N}.\, \mathsf{Vec}(m)\big) \;\longrightarrow\; \Sigma k : \mathbb{N}.\, \mathsf{Vec}(k)$$

by

$$\Phi\big((n, v), (m, w)\big) \;\triangleq\; \big(n{+}m,\; \mathsf{concat}(v, w)\big),$$

where $\mathsf{concat} : \mathsf{Vec}(n) \times \mathsf{Vec}(m) \to \mathsf{Vec}(n{+}m)$ is length-index respecting.

Typing. For any (n, v) and (m, w) with $v : \mathsf{Vec}(n)$ and $w : \mathsf{Vec}(m)$, $\mathsf{concat}(v, w) : \mathsf{Vec}(n{+}m)$; hence $\Phi((n, v), (m, w)) : \Sigma k.\, \mathsf{Vec}(k)$ with $k = n{+}m$. By Cor. 4.2, $\Phi(\mathsf{Merge}(P_1, P_2))$ is well-typed.

Operational behavior. If $P_1 \Rightarrow (n, v)$ and $P_2 \Rightarrow (m, w)$, then

$$\mathsf{Merge}(P_1, P_2) \;\Rightarrow\; (n, v) \parallel (m, w) \quad \text{and} \quad \Phi\big(\mathsf{Merge}(P_1, P_2)\big) \;\Rightarrow\; \big(n{+}m,\; \mathsf{concat}(v, w)\big).$$

Denotational semantics (general form). Write $\mu = P_1 \in \mathsf{Dist}(\Sigma n.\, \mathsf{Vec}(n))$ and $\nu = P_2 \in \mathsf{Dist}(\Sigma m.\, \mathsf{Vec}(m))$, assumed independent. Then

$$\Phi(\mathsf{Merge}(P_1, P_2)) \;=\; \Phi(\, \mu \otimes \nu \,) \;\in\; \mathsf{Dist}\big(\Sigma k.\, \mathsf{Vec}(k)\big),$$

i.e. the pushforward of the product measure along $((n, v), (m, w)) \mapsto (n{+}m, \mathsf{concat}(v, w))$.

Binary-choice instance. Instantiate P_1, P_2 with

$$L_1 = \mathsf{Choice}(p, 0, 1), \quad P_1 = (L_1, \mathsf{Vec}(L_1)), \qquad L_2 = \mathsf{Choice}(q, 1, 2), \quad P_2 = (L_2, \mathsf{Vec}(L_2)).$$

Then the four outcomes and their probabilities are:

Outcome of $\Phi(\mathsf{Merge}(P_1, P_2))$	Probability
$\big(0{+}1,\; \mathsf{concat}(\mathsf{Vec}(0), \mathsf{Vec}(1))\big)$	$p\,q$
$\big(0{+}2,\; \mathsf{concat}(\mathsf{Vec}(0), \mathsf{Vec}(2))\big)$	$p\,(1-q)$
$\big(1{+}1,\; \mathsf{concat}(\mathsf{Vec}(1), \mathsf{Vec}(1))\big)$	$(1-p)\,q$
$\big(1{+}2,\; \mathsf{concat}(\mathsf{Vec}(1), \mathsf{Vec}(2))\big)$	$(1-p)\,(1-q)$

Equivalently,

$$\Phi(\mathsf{Merge}(P_1, P_2)) = pq\,\delta_{(1,\_)} + p(1-q)\,\delta_{(2,\_)} + (1-p)q\,\delta_{(2,\_)} + (1-p)(1-q)\,\delta_{(3,\_)},$$

where each Dirac mass is at the corresponding concatenated vector of the shown length (underscores emphasize value dependence on v, w).

Takeaway. This instance illustrates Cor. 4.2: the **type-level shape** (the length index) evolves by the deterministic map $k = n{+}m$ while the **distribution** over shapes and values is exactly the pushforward of the joint (independent) distribution under concatenation. The construction is general: any dependent post-processing Φ (e.g. zipping, padding, folding) yields a well-typed SPC term whose semantics is the corresponding pushforward measure.

5 Operational Semantics

5.1 β-Reduction and Pop

$$\begin{aligned}
\mathrm{Pop}(\mathrm{Sphere}(x{:}A.\,t), u) &\to t[u/x] \\
\mathrm{Nest}(t, u) &\to \mathrm{Pop}(t, u) \\
\mathrm{Choice}(p, t, u) &\to \begin{cases} t & \text{with prob. } p \\ u & \text{with prob. } 1-p \end{cases}
\end{aligned}$$

5.2 Merge as Parallel Composition

$$\mathrm{Merge}(t, u) \equiv \mathrm{Merge}(u, t), \quad \mathrm{Merge}(t, \mathrm{Merge}(u, v)) \equiv \mathrm{Merge}(\mathrm{Merge}(t, u), v)$$

5.3 Choice (Internal and Monadic Variants)

Option B (Distribution Monad).

$$\frac{\Gamma \vdash A : \mathsf{Type}_i}{\Gamma \vdash \mathsf{Dist}(A) : \mathsf{Type}_i}\ \textsc{Dist-Form}$$

$$\frac{\Gamma \vdash p : \mathsf{Prob} \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathrm{Choice}(p, t, u) : \mathsf{Dist}(A)}\ \textsc{Choice-Dist}$$

5.4 Equational Properties of Merge

The `Merge` operator in SPC is intended to model parallel or nondeterministic composition. Its equational theory ensures that the operational, type-theoretic, and denotational perspectives coincide.

Commutativity and Associativity. At the syntactic level, `Merge` is treated modulo the laws of a commutative monoid (with unit Skip, if introduced):

$$\mathrm{Merge}(t, u) \ \equiv\ \mathrm{Merge}(u, t), \qquad \mathrm{Merge}(t, \mathrm{Merge}(u, v)) \ \equiv\ \mathrm{Merge}(\mathrm{Merge}(t, u), v).$$

These identifications allow arbitrary `Merge`-trees to be flattened into multisets of branches.

Idempotence (optional law). If `Merge` is interpreted as nondeterministic disjunction, one may further impose

$$\mathrm{Merge}(t, t) \ \equiv\ t,$$

yielding a commutative idempotent monoid. However, when `Merge` is interpreted tensorially in a symmetric monoidal category, this law does not hold in general. We therefore treat idempotence as an admissible equational extension in the nondeterministic reading, but not in the tensorial semantics.

Congruence. `Merge` is a congruence with respect to SPC reduction:

$$t \to t' \qquad \Longrightarrow \qquad \mathrm{Merge}(t, u) \to \mathrm{Merge}(t', u),$$

and symmetrically for the right argument. This ensures that parallel branches reduce independently.

Distribution over Choice. Operationally, `Merge` distributes over `Choice` in the sense that

$$\mathrm{Merge}(\mathrm{Choice}(p,t,u),v) \quad\Rightarrow\quad \mathrm{Choice}\big(p,\mathrm{Merge}(t,v),\mathrm{Merge}(u,v)\big).$$

This law ensures compositional alignment between parallel and probabilistic constructs. In the distribution-monad semantics, it corresponds to bilinearity of convex combinations.

5.5 Equational Properties of Merge

The `Merge` operator behaves as a commutative and associative combiner of terms, subject to the following equations:

$$\begin{aligned}
\mathrm{Merge}(t,u) &\equiv \mathrm{Merge}(u,t) && \text{(commutativity)}\\
\mathrm{Merge}(t,\mathrm{Merge}(u,v)) &\equiv \mathrm{Merge}(\mathrm{Merge}(t,u),v) && \text{(associativity)}\\
\mathrm{Merge}(t,t) &\equiv t && \text{(idempotence, optional)}\\
\mathrm{Merge}(t,\mathbf{0}) &\equiv t && \text{(neutral element, if defined)}
\end{aligned}$$

Here **0** denotes an optional identity element for `Merge`, representing the absence of a branch. Together, these laws allow any nested or reordered use of `Merge` to be flattened into a canonical form.

Summary. Thus, the equational properties of `Merge` situate SPC in the general theory of symmetric monoidal categories. Depending on interpretation, `Merge` may behave as:

- a commutative monoid (nondeterministic or set-theoretic reading),
- or a symmetric monoidal tensor (categorical reading).

Both views agree on commutativity, associativity, and congruence, while diverging on idempotence. This flexibility allows SPC to uniformly encode both algebraic nondeterminism and categorical concurrency.

6 Denotational Semantics

6.1 Presheaf Topos Semantics

The presheaf category $[\mathsf{Sphere}^{op}, \mathsf{Set}]$ forms a topos, providing:

- Subobject classifier (truth sphere).
- Finite limits and colimits.
- Exponentials.
- Internal intuitionistic higher-order logic: propositions = subspheres; proofs = morphisms preserving truth.

Distribution monad on $\mathcal{E}$. For each object X of $\mathcal{E}$, let $\mathsf{Dist}(X)$ denote the object of *finitely supported subprobability measures* on X (presheafwise), with unit and bind:

$$\eta_X : X \to \mathsf{Dist}(X) \qquad \mu^{\sharp}_{X,Y} : \mathsf{Dist}(X) \times (X \to \mathsf{Dist}(Y)) \to \mathsf{Dist}(Y)$$

satisfying the monad laws (left/right unit and associativity). In syntax we write $\mathsf{return} \equiv \eta$ and $\mathsf{bind} \equiv \mu^{\sharp}$.

Core clauses.

$$\begin{aligned}\text{Sphere}(x{:}A.\,t) &: \ \Gamma \to A \Rightarrow B \quad (\text{exponential in } \mathcal{E})\\ \text{Pop}(t,u) &= \ \mathsf{ev} \circ \langle t,u\rangle \quad : \Gamma \to B\\ \text{Merge}(t,u) &= \ \langle t,u\rangle \quad : \Gamma \to A \times A\end{aligned}$$

Probabilities and Choice (Option B). We interpret Prob as the presheaf of reals in $[0,1]$ (constant presheaf suffices). Given $p : \Gamma \to [0,1]$ and $t, u : \Gamma \to A$, the denotation of Choice is the *convex mixture*:

$$\text{Choice}(p,t,u) \ = \ \big(\lambda\gamma.\, p(\gamma)\cdot\delta_{t(\gamma)} \ + \ (1-p(\gamma))\cdot\delta_{u(\gamma)}\big)$$

6.2 Sphere/Pop as Exponentials

In categorical semantics, λ-abstraction and application correspond to the exponential adjunction $C^A \dashv - \times A$. The Sphere operator in SPC generalizes abstraction by constructing a morphism $\mathsf{Sphere}(f) : A \to B$ as an object inside the exponential B^A. Operationally, entering a Sphere is equivalent to opening a scope, while Pop corresponds to instantiating this scope with an argument. Formally, given a context $\Gamma \vdash f : A \to B$, we define:

$$\Gamma \vdash \mathsf{Sphere}(f) : B^A \qquad \Gamma \vdash \mathsf{Pop}(\mathsf{Sphere}(f), a) : B.$$

The β- and η-laws of SPC reproduce the expected adjunction laws of the exponential. In this sense, Sphere/Pop are not merely syntactic operators but geometric witnesses of the exponential structure in the underlying cartesian closed category.

6.3 Merge as Tensor

While ordinary λ-calculus models computation in a cartesian closed category (CCC), SPC extends the structure to a symmetric monoidal category, where Merge is interpreted as the tensor $\otimes$. Two computations M and N may be combined via

$$\Gamma \vdash M : A \quad \Delta \vdash N : B \quad \Longrightarrow \quad \Gamma, \Delta \vdash M \otimes N : A \otimes B.$$

This generalizes parallel composition, enabling the expression of independent channels of computation. The associativity and symmetry laws of the monoidal tensor ensure that Merge is well-behaved up to canonical isomorphism:

$$(M \otimes N) \otimes P \cong M \otimes (N \otimes P), \qquad M \otimes N \cong N \otimes M.$$

Thus Merge internalizes concurrency and compositional independence in a categorical framework, echoing both tensor products in linear logic and parallel composition in process calculi.

6.4 Choice as Convex Mixture

To model probabilistic branching, SPC introduces Choice. At the operational level, this corresponds to selecting among terms with given probabilities. Categorically, it is modeled as a convex combination of morphisms. For two computations $M, N : A$ and $p \in [0,1]$, we define:

$$\Gamma \vdash \mathsf{Choice}(p, M, N) : A$$

with denotational interpretation

$$\mathsf{Choice}(p, M, N) = p \cdot M + (1-p) \cdot N.$$

More generally, Choice extends to finite distributions $\{(p_i, M_i)\}_i$ with $\sum_i p_i = 1$. This interpretation equips SPC with the structure of a convex algebra, enabling probabilistic reasoning to be internalized into the type system.

6.5 Distribution Monad Structure

The probabilistic semantics of SPC is governed by the distribution monad $\mathcal{D}$ [9]. Objects A are mapped to probability measures $\mathcal{D}(A)$, and morphisms $f : A \to \mathcal{D}(B)$ are Markov kernels. The monadic unit $\eta : A \to \mathcal{D}(A)$ injects a pure value as a Dirac measure, while the Kleisli extension $f^* : \mathcal{D}(A) \to \mathcal{D}(B)$ defines the semantics of probabilistic bind. In SPC, the interpretation of `Choice` is given by convex linear combinations in $\mathcal{D}$, and the `Merge` operator lifts to independent product measures when applied to independent channels. Thus the distribution monad provides the categorical backbone for SPC's probabilistic semantics, ensuring that randomness is compositional and well-behaved with respect to scope and parallelism.

7 Meta-Theory

We establish several key meta-theoretic results for SPC. Proofs follow standard techniques from type theory and probabilistic semantics, adapted to the Sphere/Pop and Merge/Choice constructs.

Preservation. If $\Gamma \vdash M : A$ and $M \to M'$, then $\Gamma \vdash M' : A$. This is proved by induction on typing derivations, with separate cases for `Sphere/Pop`, `Merge`, and `Choice` reductions.

Progress. If $\emptyset \vdash M : A$, then either M is a value, or there exists M' such that $M \to M'$. Probabilistic branching requires extending the progress theorem to guarantee that at least one reduct exists with nonzero probability.

Adequacy. Operational and denotational semantics agree: if $\emptyset \vdash M : A$, then the distribution generated by reduction steps of M coincides with its denotation in the distribution monad. This ensures that probabilistic choice is faithfully represented.

Independent Channels Lemma. If $M : A$ and $N : B$ are independent computations, then

$$M \otimes N = M \otimes N,$$

where the right-hand side denotes the product measure in $\mathcal{D}(A \times B)$. This lemma formalizes the compositionality of probability in SPC, ensuring that risks aggregate correctly across independent branches.

Consistency and normalization. Since SPC extends the Calculus of Constructions conservatively with monadic probability and tensorial parallelism, strong normalization for the pure fragment implies consistency of the extended system. We conjecture, but do not yet prove, that SPC enjoys probabilistic normalization almost surely, as in other probabilistic λ-calculi.

7.1 Preservation and Substitution

Theorem 7.1 (Preservation). *If $\Gamma \vdash M : A$ and $M \to M'$, then $\Gamma \vdash M' : A$.*

Sketch. By induction on the typing derivation of M.

Case (Pop). If $M = \mathsf{Pop}(\mathsf{Sphere}(x{:}A.\, t), u)$, then by Π-Intro we have $\Gamma, x : A \vdash t : B$ and by typing of application we know $\Gamma \vdash u : A$. Substitution gives $\Gamma \vdash t[u/x] : B[u/x]$, which is exactly the reduct M'.

Case (Merge). If $M = M_1 \otimes M_2$, reduction preserves the tensor typing $\Gamma \vdash M_1 : A,\ \Delta \vdash M_2 : B \ \Rightarrow\ \Gamma, \Delta \vdash M_1 \otimes M_2 : A \otimes B$. Any step taken by M_1 or M_2 yields a well-typed pair, by induction.

Case (Choice). If $M = \mathsf{Choice}(p, M_1, M_2)$, then both branches satisfy $\Gamma \vdash M_1 : A$ and $\Gamma \vdash M_2 : A$. The reduct is either M_1 or M_2, hence $\Gamma \vdash M' : A$. (Alternatively, under a distributional typing discipline, $\Gamma \vdash M : \mathcal{D}(A)$, and reduction samples from this distribution. In either case typing is preserved.)

All other cases follow structurally. $\square$

7.2 Progress and Normal Forms

Theorem 7.2 (Progress)**.** *If $\cdot \vdash M : A$, then either M is a value, or there exists M' such that $M \to M'$.*

Sketch. By induction on typing.

Values. Atoms and Sphere abstractions are values. Tensors $V_1 \otimes V_2$ are values whenever V_1, V_2 are values.

Pop. If $M = \mathsf{Pop}(\mathsf{Sphere}(x{:}A.\, t), u)$ with u a value, then M reduces by β-reduction to $t[u/x]$.

Choice. If $M = \mathsf{Choice}(p, M_1, M_2)$, then by definition it can reduce to one of M_1 or M_2 (or, in the distributional interpretation, to a probabilistic mixture). Thus progress holds.

Merge. If $M = M_1 \otimes M_2$ and at least one of M_1, M_2 reduces, then M reduces modulo the congruence laws of associativity and commutativity. If both are values, then M is a value.

Thus every well-typed closed term either is a value or reduces. □

7.3 Adequacy of Probabilistic Semantics

Theorem 7.3 (Adequacy of Denotational Semantics)**.** *For any closed term t of type A:*

- *If $t \to^* v$ where v is a value, then $t = \delta_v$ (soundness).*
- *If t involves probabilistic Choice, then for any measurable $h : A \to \mathbb{R}$,*

$$\mathbb{E}_{operational}[h] \;=\; \mathbb{E}_t[h].$$

Sketch. For the deterministic fragment, by induction on evaluation contexts and the soundness of β-reduction. For Choice, adequacy follows from the definition of Choice(p, t, u) as the convex mixture $p\delta_t + (1 - p)\delta_u$, together with the monad laws for sequencing. □

7.4 Independent Channels Lemma

Lemma 7.4 (Independent Channels)**.** *Let $R_i =$ Choice(p_i, D, S) with independent probability parameters $p_i \in [0, 1]$. Then*

$$\Pr[\mathsf{FoldOr}(\mathrm{Merge}(R_1, \ldots, R_n)) = D] = 1 - \prod_{i=1}^{n}(1 - p_i).$$

Sketch. By semantic interpretation in the distribution monad: $R_i = p_i\delta_D + (1-p_i)\delta_S$. Independence is given by the product structure of Dist. Folding with disjunction yields the stated probability. □

[column sep=small] $\Gamma \times A[r, "\langle f, \pi_2\rangle"][d, swap, "\text{oper.}"](A \Rightarrow B) \times A[r, "\mathsf{ev}"]B\Gamma \times A[rr, swap, "\mathrm{Pop}(\mathrm{Sphere}(x{:}A.\, t), u)"]B$

[column sep=small] t $\times u[rr, " \cong "][d, swap, "\text{oper. flatten}"]t \otimes u[d, "\text{tensor}"]\mathrm{Merge}(t, u)[rr, equal]\mathrm{Merge}(t, u)$

[column sep=small] $\Gamma[r, "(p, t, u)"][d, swap, "\text{sampling}"][0, 1] \times A^2[r, "\mathsf{mix}"]\mathsf{Dist}(A)\Gamma[rr, swap, "\mathrm{Choice}(p, t, u)"]\mathsf{Dist}(A)$

[column sep=small] $\prod_i \mathsf{Dist}(O)[r, " \otimes "][d, swap, "\text{oper. Merge}"]\mathsf{Dist}(O^n)[r, "\mathsf{anyDoom}"]\mathsf{Dist}(O)\mathsf{Dist}(O)[rr, equal]\mathsf{Dist}(O)$

8 Historical Antecedents

8.1 Lambda Calculus and Type Theory

The λ-calculus, introduced by Church in the 1930s [5], established the principle of treating functions as first-class terms and substitution as the primary mechanism of computation. Its simply-typed and polymorphic extensions formed the basis of modern type systems. Martin-Löf's dependent type theory in the 1970s [17] deepened this foundation by permitting types to depend on terms, thus aligning syntax, proof, and semantics in a single formalism. These traditions provided the logical backbone on which the Spherepop Calculus inherits its Sphere (abstraction) and Pop (application) constructs.

8.2 Categorical and Topos-Theoretic Foundations

Category theory in the 1960s and 70s (Mac Lane [16], Lambek and Scott [14], Lawvere [15]) reinterpreted logic and type theory within the algebraic framework of functors, natural transformations, and monoidal products. Lawvere's elementary toposes demonstrated how categorical universes can serve as foundations for constructive logic, while Lambek's insights connected λ-calculus and intuitionistic logic to cartesian closed categories. Street's higher-categorical work [24] extended these correspondences, providing tools for reasoning about substitution, coherence, and higher-order structure. SPC draws directly on these traditions by interpreting Merge as tensor and by embedding its semantics into presheaf toposes.

8.3 Probabilistic Semantics and Monads

The rise of probabilistic program semantics in the 1980s (Kozen [13]) and the categorical probability monad (Giry [9]) marked a turning point: probability could be treated not as an external measure but as a compositional effect. The monadic treatment unified branching, randomness, and expectation with the categorical structure of computation. In the 2000s–2010s, categorical probability theory and probabilistic programming languages (Goodman and Stuhlmüller [10], Gordon et al. [11], van de Meent et al. [25]) refined these approaches, incorporating them into type theory and proof assistants. SPC inherits this strand in its treatment of Choice as convex mixture, grounded in the distribution monad.

8.4 Concurrency and Merge Operators

In parallel, the theory of concurrency advanced through Milner's CCS and π-calculus [18, 19], formalizing nondeterministic composition and process interaction. Category theory again provided the unifying abstraction, treating concurrency via symmetric monoidal categories and tensorial composition. SPC borrows this lineage for its Merge operator, capturing both the algebraic laws of nondeterminism (commutativity, associativity, optional idempotence) and the structural laws of tensor products in monoidal categories. This dual inheritance allows SPC to represent both nondeterministic branching and true concurrency within a uniform syntax.

8.5 Timeline of Antecedents

- **1930s–40s: Lambda Calculus.** Church introduces the untyped and simply-typed λ-calculus [5], laying the foundation for functional abstraction and substitution.
- **1960s: Category Theory Foundations.** Mac Lane formalizes monoidal categories and coherence theorems [16]. Lambek articulates the Curry–Howard–Lambek correspondence [14].
- **1970s: Dependent Type Theory and Toposes.** Martin-Löf develops intuitionistic dependent type theory [17]. Lawvere introduces elementary toposes [15], embedding logic in category theory.

- **1980s: Higher Categories and Probabilistic Semantics.** Street formalizes 2-categories and coherence structures [24]. Coquand and Huet define the Calculus of Constructions [6]. Kozen introduces semantics of probabilistic programs [13]. Giry develops the probability distribution monad [9].
- **1990s: Calculus of Inductive Constructions.** Coq and related systems integrate inductive types with dependent type theory [4].
- **2000s–2010s: Probabilistic Type Theory and Programming.** Growth of probabilistic programming languages [10, 11, 25]; categorical probability theory connects monads, measure theory, and logic [12, 7].
- **2020s: Toward SPC.** Probabilistic dependent type theory (e.g. Adams et al. [1]). Categorical unification of probabilistic semantics, concurrency, and constructive logic inspires the Spherepop Calculus.

9 Positioning of SPC

9.1 Comparison with λ-calculus

Like the λ-calculus, SPC is founded on abstraction and application, via `Sphere` and `Pop`. The β-reduction rule is preserved verbatim, guaranteeing that SPC inherits the expressivity of higher-order functional computation. Where SPC diverges is in the explicit visualization of scope as a *geometric sphere*, and in the enrichment with `Merge` and `Choice`, which lie outside the traditional λ syntax.

9.2 Comparison with π-calculus

Milner's π-calculus introduced communication and concurrency via channels. SPC's `Merge` operator is reminiscent of nondeterministic choice and parallel composition in process calculi, but with a type-directed and categorical interpretation: `Merge` is modeled as a tensor in a monoidal category, rather than as an interleaving of processes. SPC thus aligns with structural concurrency but at a higher level of abstraction.

9.3 Comparison with Probabilistic λ-calculus

Kozen's semantics of probabilistic programs and subsequent probabilistic λ-calculi introduced random choice as a primitive. SPC's `Choice` operator situates itself within this tradition, but it offers two levels of semantics: an *internal* version, where Choice directly returns a value of type A, and a *monadic* version, where Choice produces a distribution in $\mathsf{Dist}(A)$. The latter connects SPC directly to Giry's distribution monad and modern probabilistic programming semantics, ensuring compositionality.

9.4 Comparison with CoC/CIC

SPC extends the Calculus of Constructions with probabilistic and parallel operators. Its type system supports Π- and Σ-types, but enriches the language with monoidal and stochastic constructs, which are not native to CoC or CIC. In this sense, SPC is a *probabilistic, monoidal calculus of constructions.*

9.5 Summary and Observation

SPC can thus be positioned as:

- A higher-order functional calculus (like λ-calculus),
- With categorical concurrency (akin to π-calculus, but tensorial),
- And integrated probabilistic semantics (via Choice and distribution monads),

- Embedded within dependent type theory (as in CoC).

This synthesis marks SPC as a novel framework: not merely a probabilistic extension of λ-calculus, nor a categorical gloss on process calculi, but a unification of *functional abstraction, concurrency, and probability* within one geometrically motivated language.

Discussion. The comparison highlights how SPC synthesizes elements from prior traditions while introducing new primitives. From λ-calculus it inherits higher-order abstraction and β-reduction; from π-calculus it adapts the idea of parallel composition, but reinterprets it categorically as `Merge`. Probabilistic choice is integrated more deeply than in probabilistic λ-calculi, via both internal and distribution-monad semantics. Dependent types and presheaf semantics provide a constructive foundation absent in the other calculi. Finally, SPC's visualization of scope as spheres establishes a geometric, intuitive model of computation that distinguishes it from purely syntactic approaches.

Observation 9.1 (Positioning of SPC)**.** *The Spherepop Calculus inherits the expressive power of λ-calculus through `Sphere/Pop`, while extending it with `Merge` as a categorical parallel operator and `Choice` as a probabilistic primitive. Unlike π-calculus, SPC does not rely on channel-based interleaving, but instead models concurrency via tensorial semantics in a monoidal category. Compared to probabilistic λ-calculi, SPC admits both an internal and a distribution-monad interpretation of choice, ensuring compositional semantics. Finally, by embedding all constructs into dependent type theory and presheaf topos semantics, SPC achieves a synthesis of functional abstraction, concurrency, and probability within a single constructive framework.*

Feature	λ-calc.	π-calc.	Prob. λ	**SPC**
Core abstraction	$\lambda x.\, t$	Processes/channels	$\lambda x.\, t$	Sphere$(x{:}A.\, t)$
Application	$t\, u$	Channel comm.	$t\, u$	Pop(t, u)
Scope model	Parentheses	Channel scope	Parentheses	Spheres (bubble scope)
Concurrency	Not built-in	Parallel comp.	Not built-in	Yes: Merge(t, u)
Probabilistic choice	Not built-in	Not built-in	flip(p), rand. prims	Choice(p, t, u)
Dependent types	No	No	Rare ext.	Yes (prob. CoC)
Categorical semantics	CCC (cart. closed cat.)	Proc. cats.	Monads for prob.	Presheaf topos + Dist. monad
Evaluation	β-red.	Proc. interaction	β-red. + sampling	β-red. + merge + samp.
Scope visualization	Not emphasized	Channel diagrams	Not emphasized	Spheres, nesting

Table 1: Comparison of SPC with classical computational calculi.

9.6 Expressivity Proposition

Proposition 9.2 (Expressivity of SPC)**.** *The Spherepop Calculus strictly subsumes:*

1. *the simply-typed λ-calculus (via `Sphere/Pop`),*
2. *probabilistic λ-calculus (via `Choice` with internal or monadic semantics),*
3. *and a fragment of π-calculus corresponding to nondeterministic parallel composition (via `Merge`).*

Sketch. (1) Every λ-term can be translated homomorphically:

$$\lambda x.\, t \mapsto \text{Sphere}(x{:}A.\, t), \qquad t\, u \mapsto \text{Pop}(t, u).$$

Subject reduction and β-equivalence are preserved by the operational semantics.

(2) For probabilistic λ-calculus, constructs such as $\mathbf{flip}(p); t; u$ translate directly to $\mathrm{Choice}(p, t, u)$ in SPC. In the monadic variant, the distribution semantics of SPC subsumes the denotational semantics of Kozen and Giry.

(3) The parallel composition fragment of π-calculus, restricted to independent nondeterministic branches, is captured by $\mathrm{Merge}(t, u)$, with the tensorial semantics in SPC corresponding to categorical models of process interleaving. While SPC does not encode full name-passing mobility, it subsumes this nondeterministic fragment.

Strictness follows because SPC adds dependent types and presheaf topos semantics, neither present in λ, probabilistic λ, nor π. Thus, SPC properly extends each calculus. $\square$

10 Translations into SPC

We present compositional translations from three source calculi into the Spherepop Calculus (SPC): simply-typed λ-calculus, probabilistic λ-calculus, and a nondeterministic parallel fragment of π-calculus. Each translation is type-directed, preserves typing, and enjoys operational correspondence with SPC reduction.

10.1 Notation

We write $\mathcal{T}_\lambda$, $\mathcal{T}_{\mathrm{prob}\,\lambda}$, $\mathcal{T}_\pi$ for the translations. A source typing context Γ is mapped pointwise to the SPC context with the same variable|type pairs. Types are mapped homomorphically to SPC types (base types unchanged; function types map to Π-types).

10.2 Translation $\mathcal{T}_\lambda$ (Simply-typed λ)

Source.

$$\begin{aligned} \text{Types:}\quad & \tau ::= \alpha \mid \tau \to \tau \\ \text{Terms:}\quad & e ::= x \mid \lambda x{:}\tau.\, e \mid e\, e \end{aligned}$$

Type translation.

$$\alpha = \alpha, \qquad \tau_1 \to \tau_2 = \Pi x{:}\tau_1.\ \tau_2 \ \ (\text{with } x \notin \mathrm{FV}).$$

Term translation (compositional).

$$\begin{aligned} \mathcal{T}_\lambda(x) &= x \\ \mathcal{T}_\lambda(\lambda x{:}\tau.\, e) &= \mathrm{Sphere}(x{:}\tau.\ \mathcal{T}_\lambda(e)) \\ \mathcal{T}_\lambda(e_1\, e_2) &= \mathrm{Pop}(\mathcal{T}_\lambda(e_1),\ \mathcal{T}_\lambda(e_2)) \end{aligned}$$

Preservation. If $\Gamma \vdash e : \tau$ then $\Gamma \vdash \mathcal{T}_\lambda(e) : \tau$.

Operational correspondence. If $e \to_\beta e'$ then $\mathcal{T}_\lambda(e) \to \mathcal{T}_\lambda(e')$ in SPC (one β-step).

Worked example.

$$(\lambda x{:}\alpha.\, x)\ a \quad \mapsto \quad \mathrm{Pop}(\mathrm{Sphere}(x{:}\alpha.\, x),\ a) \ \to\ a.$$

10.3 Translation $\mathcal{T}_{\mathrm{prob}\,\lambda}$ (Probabilistic λ)

Source (internal choice). Extend e with $\mathbf{choice}(p, e_1, e_2)$ ($p \in [0, 1]$). Typing: if $\Gamma \vdash e_i : \tau$ and $\Gamma \vdash p : \mathsf{Prob}$ then $\Gamma \vdash \mathbf{choice}(p, e_1, e_2) : \tau$.

Type translation. As in λ; additionally $\mathsf{Prob} = \mathsf{Prob}$.

Term translation.

$$\mathcal{T}_{\text{prob}\,\lambda}(\text{pure } e) = \mathcal{T}_\lambda(e)$$
$$\mathcal{T}_{\text{prob}\,\lambda}(\mathbf{choice}(p, e_1, e_2)) = \text{Choice}(p,\ \mathcal{T}_\lambda(e_1),\ \mathcal{T}_\lambda(e_2)).$$

Preservation. If $\Gamma \vdash e : \tau$ in the probabilistic λ-calculus (internal choice), then $\Gamma \vdash \mathcal{T}_{\text{prob}\,\lambda}(e) : \tau$ in SPC (Choice–A rule).

Operational correspondence. A single probabilistic step $\mathbf{choice}(p, e_1, e_2) \Rightarrow e_1$ w.p. p and $\Rightarrow e_2$ w.p. $1-p$ corresponds to $\text{Choice}(p, \mathcal{T}_\lambda(e_1), \mathcal{T}_\lambda(e_2))$ stepping to the corresponding branch with the same probabilities.

Worked example.

$$\mathbf{choice}\big(\tfrac{1}{2},\ (\lambda x{:}\alpha.\,x)\ a,\ (\lambda x{:}\alpha.\,a)\ x\big) \mapsto \text{Choice}\big(\tfrac{1}{2},\ \text{Pop}(\text{Sphere}(x{:}\alpha.\,x), a),\ \text{Pop}(\text{Sphere}(x{:}\alpha.\,a), x)\big)\,.$$

Monadic variant (Option B). If the source calculus types **choice** as $\mathsf{Dist}(\tau)$, use:

$$\mathcal{T}^{\mathsf{mon}}_{\text{prob}\,\lambda}(\mathbf{choice}(p, e_1, e_2)) = \text{Choice}(p,\ \mathcal{T}_\lambda(e_1),\ \mathcal{T}_\lambda(e_2)) : \mathsf{Dist}(\tau),$$

and compose with $\mathsf{return}/\mathsf{bind}$ in SPC as required. Preservation and adequacy follow from the distribution-monad semantics.

10.4 Translation $\mathcal{T}_\pi$ (Nondeterministic π-fragment)

Source fragment. Processes $P, Q ::= 0 \mid P \mid Q \mid (\nu a)P \mid a(x).P \mid \overline{a}\langle b\rangle.P$, with *nondeterministic parallel* reduction where independent branches may proceed. We translate the *branching/parallel* structure and independent, non-communicating subprocesses. (Name passing and synchronization are not encoded in this fragment.)

Type and term carriers. Assume a ground outcome type O with $\mathsf{Safe}, \mathsf{Doom} : O$. Let each process P denote an SPC term $t_P : O$ (the observable outcome). Communication-free parallelism is mapped to `Merge`; restriction (νa) is ignored in this fragment or reflected in types as an abstract scope (no effect on O).

Translation.

$$\begin{aligned}
\mathcal{T}_\pi(0) &= \mathsf{Safe} : O \\
\mathcal{T}_\pi(P \mid Q) &= \text{Merge}(\mathcal{T}_\pi(P),\ \mathcal{T}_\pi(Q)) : O \\
\mathcal{T}_\pi((\nu a)P) &= \mathcal{T}_\pi(P) : O \quad \text{(in the non-communicating fragment)} \\
\mathcal{T}_\pi(\overline{a}\langle b\rangle.P) &= \mathcal{T}_\pi(P) \\
\mathcal{T}_\pi(a(x).P) &= \mathcal{T}_\pi(P)
\end{aligned}$$

Intuitively, we erase name-passing but preserve *parallel branching structure.* When the observable is a disjunctive property (e.g. “any branch dooms?”), SPC’s Merge correctly aggregates branches.

Preservation (fragment). If P and Q are well-formed processes in the fragment, then $\mathcal{T}_\pi(P \mid Q)$ is well-typed in SPC and has the same outcome type O.

Operational correspondence (fragment). If $P \mid Q \Rightarrow P' \mid Q$ by reducing an independent branch, then $\text{Merge}(\mathcal{T}_\pi(P), \mathcal{T}_\pi(Q)) \to \text{Merge}(\mathcal{T}_\pi(P'), \mathcal{T}_\pi(Q))$ modulo Merge congruence. Similarly for the right branch.

Worked examples.

$$(0 \mid 0) \;\mapsto\; \mathrm{Merge}(\mathsf{Safe}, \mathsf{Safe})$$
$$(0 \mid (\nu a)0) \;\mapsto\; \mathrm{Merge}(\mathsf{Safe}, \mathsf{Safe})$$
$$P \mid Q \mid R \;\mapsto\; \mathrm{Merge}(\mathcal{T}_\pi(P), \mathrm{Merge}(\mathcal{T}_\pi(Q), \mathcal{T}_\pi(R))) \equiv \mathrm{Merge}(\mathcal{T}_\pi(P), \mathcal{T}_\pi(Q), \mathcal{T}_\pi(R)) \text{ (flattened).}$$

10.5 Compositionality and Merge/Choice Interaction

Flattening. SPC treats Merge as associative/commutative; thus the image of parallel composition flattens canonically: $\mathcal{T}_\pi(P_1 \mid \cdots \mid P_n) = \mathrm{Merge}(\mathcal{T}_\pi(P_1), \ldots, \mathcal{T}_\pi(P_n))$.

Probabilistic branches. If a process or program contains probabilistic behavior (e.g. random guards), translate local randomness to Choice and compose with Merge. Under independence, the doom-aggregation law holds:

$$\Pr[\text{doom}] \;=\; 1 - \prod_{i=1}^{n}(1 - p_i) \quad \text{for } \mathrm{Merge}\big(\mathrm{Choice}(p_i, \mathsf{Doom}, \mathsf{Safe})\big)_{i=1}^{n}.$$

10.6 Summary of Translation Properties

- (**Typing**) Each translation preserves typing derivations into SPC.
- (**Operational**) Source one-step reductions map to SPC steps (or equalities modulo Merge congruence) with matching probabilities for choice.
- (**Adequacy**) For the monadic version, denotations commute with translation: $\mathcal{T}_{\mathrm{prob}\,\lambda}(e) = \mathcal{T}^{\mathcal{E}}(e)$, where $\mathcal{T}^{\mathcal{E}}$ is the induced functor on denotations.

A Appendices

A.1 Context and Substitution Lemmas

We recall the compositional translation $\mathcal{T}_\lambda$:

$$\mathcal{T}_\lambda(x) = x, \qquad \mathcal{T}_\lambda(\lambda x{:}\tau.\, e) = \mathrm{Sphere}(x{:}\tau.\, \mathcal{T}_\lambda(e)), \qquad \mathcal{T}_\lambda(e_1\, e_2) = \mathrm{Pop}(\mathcal{T}_\lambda(e_1),\, \mathcal{T}_\lambda(e_2)),$$

and the type mapping $\alpha = \alpha$, $\tau_1 \to \tau_2 = \Pi x{:}\tau_1.\, \tau_2$ (with $x \notin \mathrm{FV}$).

Lemma A.1 (Context Lemma)**.** *If $\Gamma \vdash e : \tau$ in STLC, then (the pointwise-mapped) $\Gamma \vdash \mathcal{T}_\lambda(e) : \tau$ in SPC.*

Sketch. By induction on the typing derivation of e. Variables are immediate. Abstractions use Π-Intro, applications use Π-Elim in SPC. The type mapping preserves well-formedness. □

Lemma A.2 (Substitution Lemma)**.** *If $\Gamma, x{:}\tau \vdash e : \sigma$ and $\Gamma \vdash v : \tau$, then*

$$\Gamma \vdash \mathcal{T}_\lambda(e)[\,\mathcal{T}_\lambda(v)/x\,] : \sigma.$$

Sketch. By induction on the structure of e. The only nontrivial case is application of β in SPC:

$$\mathrm{Pop}(\mathrm{Sphere}(x{:}\tau.\, \mathcal{T}_\lambda(e)),\, \mathcal{T}_\lambda(v)) \;\to\; \mathcal{T}_\lambda(e)[\,\mathcal{T}_\lambda(v)/x\,],$$

which is well-typed by Π-Intro/Elim and the IH. □

A.2 Preservation Corollary

Corollary A.3 (Preservation for $\mathcal{T}_\lambda$)**.** *If* $\Gamma \vdash e : \tau$ *and* $e \to_\beta e'$*, then* $\Gamma \vdash \mathcal{T}_\lambda(e) : \tau$ *and* $\mathcal{T}_\lambda(e) \to \mathcal{T}_\lambda(e')$ *in SPC.*

Sketch. Single β-step in STLC corresponds to a single β-step in SPC by the translation and the Substitution Lemma. □

A.3 End-to-End Worked Example

Source term (prob. λ-calculus, internal choice) Let $\tau = \alpha$ be a base type and consider

$$e \;=\; \textbf{choice}(p,\ (\lambda x{:}\alpha.\, x)\, a,\ (\lambda x{:}\alpha.\, a)\, x_0)\,,$$

with constants $a : \alpha$, variable $x_0 : \alpha$, and $p : \mathsf{Prob}$.

Typing (source): $\cdot, p : \mathsf{Prob}, a : \alpha, x_0 : \alpha \;\vdash\; e : \alpha$.

Translation to SPC

$$\mathcal{T}_{\mathrm{prob}\,\lambda}(e) \;=\; \mathrm{Choice}\Big(p,\ \underbrace{\mathrm{Pop}(\mathrm{Sphere}(x{:}\alpha.\, x),\, a)}_{\text{branch } t},\ \underbrace{\mathrm{Pop}(\mathrm{Sphere}(x{:}\alpha.\, a),\, x_0)}_{\text{branch } u}\Big) \;:\; \alpha.$$

Operational trace in SPC

$$\begin{aligned}&\mathrm{Choice}\big(p,\ \mathrm{Pop}(\mathrm{Sphere}(x{:}\alpha.\, x),\, a),\ \mathrm{Pop}(\mathrm{Sphere}(x{:}\alpha.\, a),\, x_0)\big)\\ &\to \begin{cases}\mathrm{Pop}(\mathrm{Sphere}(x{:}\alpha.\, x),\, a) \to a & \text{with prob. } p,\\ \mathrm{Pop}(\mathrm{Sphere}(x{:}\alpha.\, a),\, x_0) \to a & \text{with prob. } 1-p.\end{cases}\end{aligned}$$

Hence in either branch the normal form is $a : \alpha$.

Denotational interpretation (Option B: distribution monad) In the presheaf topos $\mathcal{E} = [\mathsf{Sphere}^{op}, \mathsf{Set}]$ with distribution monad Dist,

$$\mathrm{Choice}(p, t, u) \;=\; p \cdot \delta_t + (1-p) \cdot \delta_u.$$

But $t = u = a$, so

$$\mathcal{T}_{\mathrm{prob}\,\lambda}(e) = p \cdot \delta_a + (1-p) \cdot \delta_a = \delta_a.$$

Thus the denotation is a Dirac mass at a, matching the operational normal form.

Variant with distinct outcomes. Let $\alpha = O = \{\mathsf{Safe}, \mathsf{Doom}\}$ and consider

$$e' = \textbf{choice}(p,\ \mathsf{Doom},\ \mathsf{Safe}) \quad : \; O.$$

Then

$$\mathcal{T}_{\mathrm{prob}\,\lambda}(e') = \mathrm{Choice}(p, \mathsf{Doom}, \mathsf{Safe}),$$

operationally sampling Doom w.p. p and Safe w.p. $1-p$, and

$$\mathcal{T}_{\mathrm{prob}\,\lambda}(e') = p\, \delta_{\mathsf{Doom}} + (1-p)\, \delta_{\mathsf{Safe}}.$$

A.4 Aggregation via Merge and Choice

For independent hazards $R_i = \text{Choice}(p_i, \mathsf{Doom}, \mathsf{Safe})$, define the SPC term

$$T_n \;=\; \text{FoldOr}\big(\text{Merge}(R_1, \ldots, R_n)\big).$$

Under independence,

$$\Pr[T_n = \mathsf{Doom}] \;=\; 1 - \prod_{i=1}^{n}(1 - p_i),$$

as established in the Independent-Channels Lemma; denotationally, this is the pushforward of the product distribution through the Boolean disjunction $\mathsf{anyDoom} : O^n \to O$.

References

[1] Robin Adams, Benedikt Ahrens, Andrej Bauer, et al. Foundations of probabilistic programming in type theory. *Logical Methods in Computer Science*, 16(4):1–29, 2020.

[2] Steve Awodey. *Category Theory*, volume 52 of *Oxford Logic Guides*. Oxford University Press, 2nd edition, 2010.

[3] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, revised edition, 1984.

[4] Yves Bertot and Pierre Castéran. Interactive theorem proving and program development: Coq'art. In *Text in Theoretical Computer Science*. Springer, 2004.

[5] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, 1940.

[6] Thierry Coquand and Gérard Huet. The calculus of constructions. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, pages 266–278. IEEE, 1988.

[7] Tobias Fritz. A synthetic approach to markov kernels, conditional independence, and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, 2020.

[8] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, 1987.

[9] Michèle Giry. A categorical approach to probability theory. In *Categorical Aspects of Topology and Analysis*, pages 68–85. Springer, 1982.

[10] Noah D. Goodman and Andreas Stuhlmüller. The design and implementation of probabilistic programming languages. *arXiv preprint arXiv:1401.3301*, 2014.

[11] Andrew D. Gordon, Thomas A. Henzinger, et al. Probabilistic programming. In *Proceedings of the Future of Software Engineering*, pages 167–181. ACM, 2014.

[12] Bart Jacobs and Jorik Mandemaker. A new look at probability: coalgebra and monads. *Mathematical Structures in Computer Science*, 25(3):651–680, 2015.

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

[14] Joachim Lambek and Philip J. Scott. *Introduction to higher order categorical logic*. Cambridge University Press, 1986.

[15] F. William Lawvere. Quantifiers and sheaves. *Actes du Congrès International des Mathématiciens*, 1:329–334, 1970.

[16] Saunders Mac Lane. Natural associativity and commutativity. *Rice University Studies*, 49(4):28–46, 1963.

[17] Per Martin-Löf. An intuitionistic theory of types: predicative part. In *Logic Colloquium '73*, pages 73–118. North-Holland, 1975.

[18] Robin Milner. *A Calculus of Communicating Systems*, volume 92. Springer, 1980.

[19] Robin Milner. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.

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

[21] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[22] Ulf Norell. Dependently typed programming in agda. *Lecture Notes in Computer Science*, 5674:230–266, 2009.

[23] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[24] Ross Street. The formal theory of monads. *Journal of Pure and Applied Algebra*, 2(2):149–168, 1972.

[25] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4365–4371, 2018.