# NeSyCat Theory: Semantics in Kleisli Categories for Neuro-Symbolic AI in HaskTorch

Daniel Romero Schellhorn
Universität Osnabrück
Osnabrück, Germany
d.p.schellhorn@gmail.com

The divide between the scientific communities of AI and Category Theory is deep. Even the new branch of neuro-symbolic AI has not yet acknowledged category theory, nor has category theory been applied directly to neuro-symbolic AI. The NeSyCat project aims at beginning to close this gap. Therefore it is *not* concerned with proving new results in category theory *about* AI, which would only widen the gap, but instead with applying the plethora of results that have been already proven in category theory. To do that, however, these results need to be structured in an implementation compatible way. This structure is constructed as a bridge specifically between the syntax-semantic duality of categorical logic and the Haskell type system, and yet it simultaneously acts as a blueprint for other programming languages and as a platform to easily implement additional results of category theory. In order to connect the working HaskTorch implementation to the theory, the style of alternating between logic/math and (non-pseudo) code directly summons the computational trilogy.

## 0 Introduction

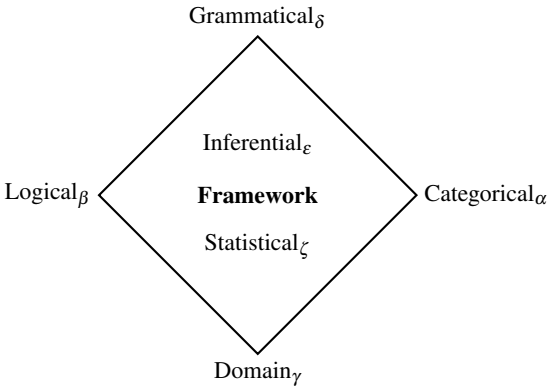


Figure 1: The six layers.

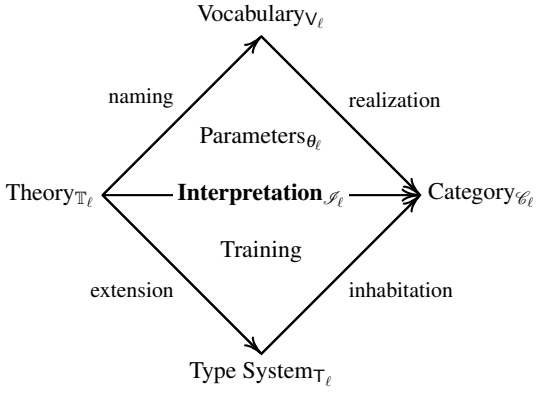


Figure 2: The six components at each layer $\ell$.

The NeSyCat framework and its HaskTorch implementation are organized around six **layers** (Figure 1), each identified by a Greek letter. Each layer specifies six interrelated components (Figure 2): a **theory** $\mathbb{T}_\ell$, a **vocabulary** $\mathsf{V}_\ell$, a **category** $\mathscr{C}_\ell$, a **type system** $\mathsf{T}_\ell$, **parameters** $\theta_\ell$, and **training**. These are connected by five morphisms: *interpretation* $\mathscr{I}_\ell$ (theory $\to$ category), *naming* (theory $\to$ vocabulary), *realization* (vocabulary $\to$ category), *extension* (theory $\to$ type system), and *inhabitation* (type system $\to$ category).

The *categorical layer* ($\alpha$) chooses the ambient category (**Cat** or **Hask**) and the paradigm-specific monads. The *logical layer* ($\beta$) equips a truth sort $\tau$ with a 2Mon-BLat (Def. 2.2) of connectives and quan-

To appear in EPTCS.

tifiers; its category $\mathscr{C}_\beta$ must be cartesian closed—concretely **Set** for set theory, **QBS** [16] for measure theory, or **Dflg** [18] for geometry (where smooth morphisms are required for gradient-based training). The *domain layer* ($\gamma$) declares the non-logical sorts and function symbols; its category $\mathscr{C}_\gamma$ (e.g. $\mathbf{Set}_{\mathbb{R}}$, $\mathbf{Meas}_{\mathbb{R}}$, $\mathbf{Tens}_{\mathbb{R}}$) need not be cartesian closed. The *grammatical layer* ($\delta$) generates formulas via a context-free grammar combining domain and logical symbols; its category $\mathscr{C}_\delta$ (**Set** for sets, **QBS** for measure theory, or **ΔGen** [11, 36] for geometry) is the overarching CCC into which both $\mathscr{C}_\beta$ and $\mathscr{C}_\gamma$ embed via faithful functors. The *inferential layer* ($\varepsilon$) turns formulas into a combined loss $J(\theta) = (1-\lambda)\cdot J_{\text{data}} + \lambda \cdot J_{\text{know}}$. The *statistical layer* ($\zeta$) provides evaluation metrics (accuracy, F1, precision). Each layer defines its own theory of the following form:

**Definition 0.1.** *A **theory***[1] $\mathbb{T} = (\Sigma, \mathbb{A})$ *[3] is formed by a set of **axioms** $\mathbb{A}$ on a **signature** $\Sigma$ consisting of:*

- *A class of **sorts*** Sor.
- *A class of **function symbols*** Fun. *For each* $f \in \text{Fun}$ *with* $\text{dom}(f) = (S_1, \ldots, S_n) \in \text{List}(\text{Sor})$ *and* $\text{cod}(f) = T \in \text{Sor}$ *we write* $f : S_1, \ldots, S_n \to T$.
- *A class of **relation symbols*** Rel. *For each* $R \in \text{Rel}$ *with* $\text{ari}(R) = (S_1, \ldots, S_n) \in \text{List}(\text{Sor})$ *we write* $R : S_1, \ldots, S_n$.
- *A class of **aggregation symbols*** Agg. *For each* $A \in \text{Agg}$ *with* $\text{ovr}(A) = S \in \text{Sor}$ *we write* $A : S$.

Throughout, we illustrate each layer with a **binary classification** running example (circle-in-square on $[0,1]^2$), using two universes simultaneously: `MeasU` (measure theory, `Dist` monad) for probabilistic evaluation and `GeomU` (geometry, `Identity` monad) for gradient-based training. Formulas are written once, polymorphically over the universe, and automatically specialize to both paradigms.

# 1 Categorical Layer

The categorical layer provides the ambient mathematical universe in which all subsequent layers are interpreted. It decomposes into sub-components, mirroring the general pipeline of the framework: category, theory, type system, vocabulary, and parameters, as training is done jointly with the other layers. As our ambient category we take the 2-category **Cat** of categories, functors and natural transformations.[2] Concretely, this is realized by the category **Hask** of Haskell types and functions, which is cartesian closed: identity is `id`, composition is `(.)`, and the exponential adjunction is witnessed by `curry`/`uncurry`.

**Definition 1.1.** *A **categorical theory** $\mathbb{T}_\alpha$ consists of the following, obeying some **categorical axioms**:*

- *A class of **category symbols*** Catg *as the class of categorical sorts.*
- *A class of **functor symbols*** Fntr *as the class of categorical function symbols.*
- *A class of **distributor symbols*** Dstr *as the class of categorical relation symbols.*
- *A class of **limit symbols*** Limt *as the class of categorical aggregation symbols.*

In Haskell, the categorical theory is captured by the `Universe` type class. Each universe $u$ determines a category constraint `Cat u` (which types are objects) and a Kleisli monad `M u` (the functorial structure for probabilistic or deterministic computation):

[1] Following Awodey [3] and nLab [31], we use the word **theory** to mean the pair $(\Sigma, \mathbb{A})$ of a signature and a set of axioms—i.e., what Johnstone [20] calls a signature with a presentation of axioms. In the stricter categorical sense of Lawvere [2], a "theory" is the syntactic category generated by such a presentation.

[2] This might be expanded to the virtual double category $\mathbb{V}$-**Cat** of categories enriched in a monoidal category $\mathbb{V}$, as in [1].

**CATEGORICAL THEORY: THE UNIVERSE CLASS** StarTheory.hs

```
class Universe u where
  type Cat u :: Type -> Constraint
  type M u :: Type -> Type
```

The abstract theory is interpreted by providing concrete instances. For our running example, two universes are defined, corresponding to the two principal paradigms of the framework:

- `GeomU` (geometry): the `Identity` monad over typed tensors (`TensObj`), for deterministic, gradient-based computation.
- `MeasU` (measure theory): the `Dist` monad over plain data types (`DataObj`), for probabilistic, sampling-based computation.

**CATEGORICAL INTERPRETATION: GEOMU AND MEASU** StarIntp.hs

```
instance Universe GeomU where
  type Cat GeomU = TensObj
  type M GeomU = Identity
instance Universe MeasU where
  type Cat MeasU = DataObj
  type M MeasU = StarVocab.Dist
```

The type system specifies which Haskell kinds, type constructors, and type-level predicates are valid inhabitants of the categorical framework. It acts as a type-theoretic whitelist: only declared instances may participate as objects (types), functors (type constructors), or relations (type classes, see in code).

- `CatObjTyp`: categorical object types (the object kinds). Inhabitants: the unit kind `()`, the standard kind `Type`, and product kinds `(Type, Type)`.
- `CatFunTyp`: categorical functor types (type constructors). Inhabitants: the terminal `()` and initial `Void` constant functors, any `Monad m` (as an endofunctor), and the bifunctors `(,)` (product), and `(->)` (exponential).

**CATEGORICAL TYPE SYSTEM: ALLOWED OBJECTS, FUNCTORS, AND RELATIONS** StarTypes.hs

```
class CatObjTyp (o :: k)
instance CatObjTyp () -- = 0-ary object kind
instance CatObjTyp Type -- = 1-ary object kind
instance CatObjTyp (Type, Type) -- = 2-ary object kind
class CatFunTyp (f :: k)
instance {-# OVERLAPPABLE #-} (Monad m) => CatFunTyp m
instance CatFunTyp (,) -- product
instance CatFunTyp (->) -- exponential
```

The categorical vocabulary $\mathsf{V}_\alpha$ provides the named monad symbols that are available in the framework: $\{\mathrm{Ident}, \mathrm{Dist}, \mathrm{Giry}\}$. Both `Dist` and `Giry` are defined as *free monads* via GADTs—they represent probability distributions as syntax trees rather than eagerly computing cartesian products. `Dist` enforces finite support; `Giry` additionally supports countably infinite (Poisson, Geometric) and continuous (Normal, Uniform, Beta, etc.) distributions.

**CATEGORICAL VOCABULARY: DIST (FINITE SUPPORT)** StarVocab.hs

```
data Dist a where
  Pure :: a -> Dist a
```

```
Bind :: Dist x -> (x -> Dist a) -> Dist a
FiniteSupp :: [(a, Double)] -> Dist a
FinUniform :: [a] -> Dist a
```

The realization step maps each monad symbol in the vocabulary to a lawful monad in Haskell by providing `Functor`, `Applicative`, and `Monad` instance declarations. For a free monad, the key insight is that `>>=` does not compute—it extends the syntax tree via the `Bind` constructor. Evaluation (sampling, integration, expectation) is deferred to a separate interpretation step (Sec. 2). The `Giry` monad follows the same free monad pattern (`GPure`/`GBind` instead of `Pure`/`Bind`) as the `Dist` monad:

**CATEGORICAL REALIZATION: DIST MONAD INSTANCE** `Dist.hs`

```
instance Monad Dist where
  return :: a -> Dist a
  return = pure
  (>>=) :: Dist a -> (a -> Dist b) -> Dist b
  (>>=) = Bind
```

## 2 Logical Layer

The **logical category** $\mathscr{C}_\beta$ must be *cartesian closed* ($(- \times A) \dashv (A \to -)$ for every object $A$) and *monad-closed*. Concretely: $\mathscr{C}_\beta = \mathbf{Set}$ for set theory, $\mathscr{C}_\beta = \mathbf{QBS}$ (quasi-Borel spaces [16]) for the Giry monad, or $\mathscr{C}_\beta = \mathbf{Dflg}$ (diffeological spaces [18]) for geometry. Only the geometry paradigm requires smooth ($C^\infty$) morphisms, since it is the paradigm used for gradient-based training[3]; the set-theoretic and measure-theoretic paradigms, used for classical reasoning and probabilistic evaluation respectively, do not. The minimal requirements for a logical vocabulary are given by the theory of a **2Mon-BLat** [34].

**Definition 2.1.** *A **logical theory** $\mathbb{T}_\beta$ consists of the following, obeying some **logical axioms**:*

- *A **truth symbol** $\tau$.*
- *A set of **connective symbols** Conn each with an arity $n \in \mathbb{N}_0$. For a connective symbol $*$ with arity $n$ we write $* : \tau^n \to \tau$. It at least contains the nullary verum symbol $\top : \tau^0 \to \tau$.*
- *A set of **comparison symbols** Comp each with an arity $n \in \mathbb{N}_0$. For a comparison symbol $\prec$ with arity $n$ we write $\prec : \tau^n$. It at least contains the two-ary sequent symbol $\vdash : \tau^2$ ([20, p. 812]).*
- *A set of **quantifier symbols** Quan.*

**Definition 2.2.** *A theory of a **double monoid bounded lattice (2Mon-BLat)** is given as a logical vocabulary $\Upsilon$ with the following symbols:*

$$\big(\tau, \vdash, \bot, \top, \oplus, \otimes, \vec{0}, \vec{1}\big),$$

*having the following arities:*

$$\vdash : \tau^2 \to \tau, \quad \bot : 1 \to \tau, \quad \top : 1 \to \tau, \quad \oplus : \tau^2 \to \tau, \quad \otimes : \tau^2 \to \tau, \quad \vec{0} : 1 \to \tau, \quad \vec{1} : 1 \to \tau.$$

[3] Three classes of activation functions arise: (i) *smooth* ($C^\infty$) activations (sigmoid, tanh, GELU, SiLU, softmax, softplus) fit directly into **Dflg**; (ii) *differentiable but not smooth* ($C^k$, $k < \infty$) activations such as ELU ($C^1$ for $\alpha$=1) formally leave **Dflg**, and by Boman's theorem [9] no cartesian closed category of $C^k$ spaces exists; (iii) *non-differentiable* activations (ReLU, Leaky ReLU) are piecewise linear, differentiable only almost everywhere. Classes (ii) and (iii) are handled in practice via Clarke subgradients [8]; a rigorous categorical treatment—potentially via a combined smooth-measurable category [17]—is future work.

*The dual* $(\tau, \vdash, \oplus, \vec{0})$ *and* $(\tau, \vdash, \otimes, \vec{1})$ *obey the axioms of ordered monoids and* $(\tau, \vdash, \bot, \top)$ *obeys the axioms of a bounded lattice. The bounded lattice axioms then inherently give rise to the two dual meet and join connective symbols* $\vee : \tau^2 \to \tau$ *and* $\wedge : \tau^2 \to \tau$ *(which may or may not have the same interpretations as the symbols* $\oplus$ *and* $\otimes$*).*

**LOGICAL THEORY: 2MONBLAT** `TwoMonBLatTheory.hs`

The logical theory is a multi-parameter type class, polymorphic over the universe `u` and truth value type `tau`. A functional dependency `tau -> u` ensures each truth type is interpreted in exactly one universe:

```
class TwoMonBLatTheory u tau | tau -> u where
  vdash :: tau -> tau -> Bool
  vee :: ParamsLogic tau -> tau -> tau -> tau
  bot :: tau
  wedge :: ParamsLogic tau -> tau -> tau -> tau
  top :: tau
  neg :: tau -> tau
  implies :: ParamsLogic tau -> tau -> tau -> tau
  oplus :: tau -> tau -> tau
  v0 :: tau
  otimes :: tau -> tau -> tau
  v1 :: tau
```

To aggregate over truth values—for example via the classical quantifiers $\forall$ and $\exists$, the fuzzy quantifiers $\bigvee$ and $\bigwedge$, or averaging sums and integrals as in LTN [5]—we extend the 2Mon-BLat to:

**Definition 2.3.** *A theory of an* ***aggregated double monoid bounded lattice (A2Mon-BLat)*** *is a* 2Mon-BLat *equipped with a dual pair of* quantifier symbols Quan $:= (\bigoplus, \bigotimes)$. *If the bounded lattice is complete, its axiom additionally give rise to the dual quantifier symbols as the meet and join connective symbols* $\bigvee$ *and* $\bigwedge$ *(which may or may not have the same interpretations as the symbols* $\bigoplus$ *and* $\bigotimes$*).*

**QUANTIFIER THEORY: A2MONBLAT** `A2MonBLatTheory.hs`

The quantifier class derives the monad from the universe (`M u`). Quantifiers are *guarded*: a `Guard u a` type family specifies the subset of the domain to quantify over (a list for `MeasU`, a batch tensor for `GeomU`):

```
bigWedge :: ParamsLogic tau -> Guard u a -> (a -> M u tau) -> M u tau
bigVee :: ParamsLogic tau -> Guard u a -> (a -> M u tau) -> M u tau
bigOplus :: Guard u a -> (a -> M u tau) -> M u tau
bigOtimes :: Guard u a -> (a -> M u tau) -> M u tau
```

**Definition 2.4.** *An* ***interpretation*** $\mathscr{I}_\Upsilon$ *of a logical vocabulary in a* cartesian closed *category* $\mathscr{C}_\beta$ *(with pullbacks, when comparisons are used) is given by:*

- *an object* $\Omega := \mathscr{I}(\tau)$ *(the interpretation of the truth symbol* $\tau$*),*
- *a morphism* $\mathscr{I}(*) : \Omega^n \to \Omega$ *for each connective symbol* $* : \tau^n \to \tau$ *with arity n,*
- *a subobject* $\mathscr{I}(\prec) \hookrightarrow \Omega^n$ *for each comparison symbol* $\prec$*:* $\tau^n$*,*
- *a family*[4] *of morphisms* $\mathscr{I}(Q)$ *for each quantifier symbol Q such that for each object B, we have:*

$$\mathscr{I}(Q)_B : \Omega^B \to \Omega.$$

[4]For classical quantifiers this is indeed a natural transformation exactly by the Beck-Chevalley condition [20, 25]. However, since we allow for more general fuzzy and smooth quantifier interpretations, this condition is rather an axiom which can be chosen to be present or not.

**TENSOR INTERPRETATION (SMOOTH LOGIC, GEOMU)** `Tensor.hs`

The module `Tensor` interprets the 2Mon-BLat symbols as pointwise batch operations on $\mathbb{R}$-valued tensors.

```
instance TwoMonBLatTheory GeomU Omega where
  type ParamsLogic Omega = Torch.Tensor
  vdash a b = Torch.asValue a <= (Torch.asValue b :: Float)
  vee betaT a b =
    let pa = a `Torch.mul` betaT
        pb = b `Torch.mul` betaT
     in F.logaddexp pa pb `Torch.div` betaT
  neg = negate
```

The **logical parameters** $\theta_\beta$ control the sharpness of the smooth approximations to min and max. A single parameter $\beta > 0$ governs the LogSumExp smoothing: $\vee(a,b) = \frac{1}{\beta}\log(e^{\beta a} + e^{\beta b})$. As $\beta \to \infty$, this recovers the hard maximum; smaller $\beta$ yields a softer approximation. The smooth minimum $\wedge$ is obtained by De Morgan duality: $\wedge(a,b) = -\vee(-a,-b)$.

# 3 Domain Layer

Building on [34], the domain layer specializes the logical framework to a concrete application domain—here, binary classification. The domain category $\mathscr{C}_\gamma$ need not be cartesian closed since exponentials live in the grammatical category $\mathscr{C}_\delta$, not in $\mathscr{C}_\gamma$ itself. For practical reasons we restrict $\mathscr{C}_\gamma$ to be a *concrete* category with finite products whose objects have at most cardinality $\mathfrak{c} := |\mathbb{R}|$. Concretely: $\mathscr{C}_\gamma = \mathbf{Set}_\mathbb{R}$ (sets with cardinality $\leq \mathfrak{c}$) for the set-theoretic paradigm, $\mathscr{C}_\gamma = \mathbf{Meas}_\mathbb{R}$ (standard Borel measurable spaces[5]) for the Giry monad, or $\mathscr{C}_\gamma = \mathbf{Tens}_\mathbb{R}$ (shaped tensor spaces $\mathbb{R}^{n_1\times\cdots\times n_k}$ with differentiable maps) for geometry. Crucially, we distinguish between Tarski and Kleisli function and relation symbols, since the latter are interpreted in the Kleisli category of the domain monad $\mathscr{M}_\gamma$:

**Definition 3.1.** *A **domain theory** $\mathbb{T}_\gamma$ is given by:*

- *A class of **sorts*** Sor.
- *A class of **function symbols*** $\mathrm{Fun} = \mathrm{Fun}^{\mathrm{Tarski}} \sqcup \mathrm{Fun}^{\mathrm{Kleisli}}$ *each with a fixed domain* $\mathrm{dom}(f) = [S_1,\ldots,S_n] \in \mathrm{List}(\mathrm{Sor})$ *and codomain* $\mathrm{cod}(f) = T \in \mathrm{Sor}$. *For a function symbol $f$ with domain $S_1,\ldots,S_n$ and codomain $T$ we write $f : S_1,\ldots,S_n \to T$ if it is a tarski function symbol, and $f : S_1,\ldots,S_n \nrightarrow T$ if it is a kleisli function symbol.*
- *A class of **relation symbols*** $\mathrm{Rel} = \mathrm{Rel}^{\mathrm{Tarski}} \sqcup \mathrm{Rel}^{\mathrm{Kleisli}}$ *each with a fixed arity* $\mathrm{ari}(R) = (S_1,\ldots,S_n) \in \mathrm{List}(\mathrm{Sor})$. *For a relation symbol $R$ with arity $(S_1,\ldots,S_n)$ we write $R : S_1,\ldots,S_n \to \tau$ if it is a tarski relation symbol, and $R : S_1,\ldots,S_n \nrightarrow \tau$ if it is a kleisli relation symbol.*
- *A class of **variable symbols*** Var *each over a fixed sort* $\mathrm{ovr}(x) = S \in \mathrm{Sor}$. *For a variable symbol $x$ over sort $S$ we write $x : S$.*

For binary classification, the domain signature is parameterized by a single universe $\mathfrak{u}$, which determines both the sorts (via associated type families) and the monad. Neither the monad nor the parameter space $\Theta$ are sorts—the monad comes from the universe, and $\Theta$ lives in the parameter manifold $\mathsf{Para}(\mathscr{C})$ [13, 14], an instance of the actegory framework [19]:

$$\mathbb{T}_\gamma = \{\underbrace{\mathsf{Point}}_{\mathrm{Sor}};\ \underbrace{\mathsf{labelA} : \mathsf{Point} \to \tau}_{\mathrm{Rel}^{\mathrm{Tarski}}};\ \underbrace{\mathsf{classifierA}_\theta : \mathsf{Point} \nrightarrow \tau}_{\mathrm{Rel}^{\mathrm{Kleisli}}}\}$$

[5] By Kuratowski's isomorphism theorem [22], every uncountable standard Borel space is Borel-isomorphic to $\mathbb{R}$, so up to isomorphism the objects of $\mathbf{Meas}_\mathbb{R}$ are exactly the finite, countable, and continuum-sized spaces.

**DOMAIN THEORY: BINARY CLASSIFICATION** BinaryTheory.hs

Sorts (`Point`, `Omega`) are associated type families of `u`. The classifier's monad comes from the universe (`M u`)—no separate monad parameter. `ParamsMLP` ($\Theta$) is a fixed external type from Para:

```
class (Universe u) => BinarySorts u where
  type Point u :: Type
  type Omega u :: Type
class (BinarySorts u) => BinaryFun u where
  labelA :: Point u -> Omega u
class (BinaryFun u, Monad (M u)) => BinaryKlFun u where
  classifierA :: ParamsMLP -> Point u -> M u (Omega u)
```

The extension step assigns concrete Haskell types to the names, one instance per universe. These are type family instances mapping names (`Point`, `Omega`) to concrete types:

**SORT EXTENSION: MEASU (MEASURE THEORY)** BinaryDataExtension.hs

```
instance BinarySorts MeasU where
  type Point  MeasU = (Float, Float)  -- R^2 as a Cartesian product
  type Omega  MeasU = BoolLogic.Omega  -- = Bool
```

**SORT EXTENSION: GEOMU (GEOMETRY)** BinaryTensExtension.hs

```
instance BinarySorts GeomU where
  type Point  GeomU = Torch.Tensor  -- shape: [2], dtype: Float
  type Omega  GeomU = TensLogic.Omega  -- = Torch.Tensor
```

**Definition 3.2.** *An **interpretation** $\mathscr{I}_\gamma$ of a domain theory $\mathbb{T}_\gamma$ in a domain monad $\mathscr{M}_\gamma$ on a domain category $\mathscr{C}_\gamma$ with finite products is given by, with $\mathscr{I}(S_1,\ldots,S_n) := \mathscr{I}(S_1)\times\ldots\times\mathscr{I}(S_n)$:*

- *an object $\mathscr{I}(S)$ for each sort $S \in \mathrm{Sor}_\gamma$;*
- *a morphism $\mathscr{I}(f) : \mathscr{I}(S_1,\ldots,S_n) \to \mathscr{I}(T)$ for each tarski function symbol $f : S_1\ldots S_n \to T$, and a morphism $\mathscr{I}(f) : \mathscr{I}(S_1,\ldots,S_n) \to \mathscr{M}_\gamma\mathscr{I}(T)$ for each kleisli function symbol $f : S_1\ldots S_n \to T$;*
- *a morphism $\mathscr{I}(R) : \mathscr{I}(S_1,\ldots,S_n) \to \Omega$ for each tarski relation symbol $R : S_1\ldots S_n$, and as above a morphism $\mathscr{I}(R) : \mathscr{I}(S_1,\ldots,S_n) \to \mathscr{M}_\gamma\Omega$ for each kleisli relation symbol $R : S_1\ldots S_n$;*
- *setting $\mathscr{I}(x) := \mathrm{id}_{\mathscr{I}(S)}$ for each variable symbol $x : S$.*

**DOMAIN INTERPRETATION: GEOMU** BinaryReal.hs

```
instance BinaryKlFun GeomU where
  classifierA :: ParamsMLP -> Point GeomU -> Identity (Omega GeomU)
  classifierA paramMLP ptTensor =
    Identity (hThetaReal paramMLP ptTensor)
```

The domain type system defines which Haskell types are valid objects in each universe. Two marker type classes act as type membership predicates: `DataObj` for the measure theory (`Bool`, `Integer`, `Float`, etc. and products, lists) and `TensObj` for the geometry paradigm (tensors, products ):

**DOMAIN TYPE SYSTEM: TENSOBJ (GEOMETRY)** Tens.hs

```
class TensObj a
instance TensObj Torch.Tensor
instance (TensObj a, TensObj b) => TensObj (a, b)
instance TensObj ()
```

# 4 Grammatical Layer

The **grammatical category** $\mathscr{C}_\delta$ is the cartesian closed category in which formulas live, combining domain sorts with logical operators. It is equipped with a strong **grammatical monad** $\mathscr{M}_\delta$[6]. Both $\mathscr{C}_\beta$ and $\mathscr{C}_\gamma$ embed into $\mathscr{C}_\delta$ via faithful functors and the grammatical monad $\mathscr{M}_\delta$ restricts to the domain monad $\mathscr{M}_\gamma$. Concretely: $\mathscr{C}_\delta = \mathbf{Set}$ for set theory, $\mathscr{C}_\delta = \mathbf{QBS}$ with probability monad for measure theory, and $\mathscr{C}_\delta = \Delta\mathbf{Gen}$ with identity monad for geometry, with faithful embeddings: $\mathbf{Set}_\mathbb{R} \hookrightarrow \mathbf{Set} = \mathbf{Set}$, $\mathbf{Meas}_\mathbb{R} \hookrightarrow \mathbf{QBS} = \mathbf{QBS}$ [16], and $\mathbf{Tens}_\mathbb{R} \hookrightarrow \Delta\mathbf{Gen} \xleftarrow{\text{Dtplg}} \mathbf{Dflg}$ [17][7]. By slight abuse of notation, we write $\Omega := \mathscr{I}(\tau)$ for the truth object in $\mathscr{C}_\beta$, $\mathscr{C}_\gamma$, and $\mathscr{C}_\delta$ interchangeably (see App. E and Figure 5).

**Definition 4.1.** *A **grammatical theory** $\mathbb{T}_\delta$ is given by:*

- *A class of **contexts*** Cntxt.
- *A class of **terms** $\Xi$. Each term $\xi \in \Xi$ has a fixed input* $\mathrm{in}(\xi) := [x_1 : S_1, \ldots, x_n : S_n] \in$ Cntxt*, given by the context of its free variables, and output* $\mathrm{out}(\xi) := [y : T] \in$ Cntxt *given by its type $\xi : T$. For a term $\xi$ with input $[x_1 : S_1, \ldots, x_n : S_n]$ and output $[y : T]$ we write $\xi : [x_1 : S_1, \ldots, x_n : S_n] \to [y : T]$.*
- *A class of **formulas** $\Phi$. Each formula $\phi \in \Phi$ has a canonical context $[\phi] := [x_1 : S_1, \ldots, x_n : S_n] \in$* Cntxt*, given by the context of its free variables. For a formula $\phi$ with the canonical context $[x_1 : S_1, \ldots, x_n : S_n]$ we write $\phi : [x_1 : S_1, \ldots, x_n : S_n]$.*
- *The subclass of **variable terms** $\Xi_{\mathrm{var}} \subseteq \Xi$. Each variable term $\xi \in \Xi_{\mathrm{var}}$ is a variable symbol $x \in$* Var*, hence it has input* $\mathrm{in}(\xi) := [x : \mathrm{ovr}(x)]$ *and output* $\mathrm{out}(\xi) := [x : \mathrm{ovr}(x)]$*, abbreviated as* $[x]$.

The **grammatical theory** $\mathbb{T}_\delta$ is given by the following context-free grammar, generating the classes $\Xi$ and $\Phi$ of terms and formulas, while the contexts are of the usual form:

**Definition 4.2.** *A **context** [20] $\vec{x}$ is a finite list $[x_1, \ldots, x_n]$ of* distinct *variables, including the empty context $[]$. These can equivalently be seen as linear ordered subsets of* Var*, so we can use set operations (union, intersection, difference, etc.) on them.*[8]

| Grammar Rule | Symbols | Description | Metatype |
|---|---|---|---|
| $\xi ::= x$ | $x \in \mathrm{Var}$ | *Variable Term* | $\xi$ |
| $\xi ::= f(\vec{\xi})$ | $f \in \mathrm{Fun}$ | *Functional Term* | $\vec{\xi} \to \xi$ |
| $\xi ::= \xi'[\vec{x} := \vec{\xi}]$ | $\vec{x} \in \mathrm{Var}^{\vec{\xi}}$ | *Substitution Term* | $\xi', \vec{\xi} \to \xi$ |
| $\phi ::= R(\vec{\xi})$ | $R \in \mathrm{Rel}$ | *Atomic Formula* | $\vec{\xi} \to \phi$ |
| $\phi ::= *(\vec{\phi})$ | $* \in \mathrm{Conn}$ | *Compound Formula* | $\vec{\phi} \to \phi$ |
| $\phi ::= Q\vec{x}(\phi)$ | $Q \in \mathrm{Qua}$ | *Quantified Formula* | $\phi \to \phi$ |
| $\phi ::= \phi[\vec{x} := \vec{\xi}]$ | $\vec{x} \in \mathrm{Var}^{\vec{\xi}}$ | *Substitution Formula* | $\phi, \vec{\xi} \to \phi$ |

[6] This compositional semantics is closely related to the DisCoCat framework for natural language [40].

[7] $\Delta$-generated spaces are equivalently D-topological spaces: topological spaces whose topology is final with respect to all continuous maps $\mathbb{R}^n \to X$. They form a convenient CCC [36] that is coreflective in **Top** and a full subcategory of **Dflg** [11].

[8] Formulas in context are handled in Def. B.3, as they can easily be formed via the weaking rule: Precompose with the projection from a larger context $\Gamma$ to the respective smaller context of the term or formula.

**Definition 4.3.** *A **sequent** [20] is a certain* comparison of formulas *in context, namely one of the form $\vec{x}.\vec{\phi} \vdash \phi'$ where $\vec{x}$ is a suitable context, $\vec{\phi}$ is a list of formulas and $\phi'$ a formula in context, while $\vdash: \tau^2$ is the sequent symbol from Def. 2.1.*

In Haskell, the formula is written **once**, polymorphic over the universe `u`. The pointwise predicate expresses "the classifier agrees with the label":

$$\varphi_\theta(x) \;=\; \big(\mathsf{labelA}(x) \to \mathsf{classifierA}_\theta(x)\big) \;\wedge\; \big(\neg\,\mathsf{labelA}(x) \to \neg\,\mathsf{classifierA}_\theta(x)\big)$$

**ABSTRACT POINTWISE PREDICATE** `BinaryFormulas.hs`

The predicate is polymorphic over `u`. The single constraint `BinaryKlFun u` provides both sorts and monad. The parameter $\theta$ (`paramMLP`) is threaded through from Para:

```
binaryPredicate lp paramMLP pt = do
  pred <- classifierA @u paramMLP pt
  let label = labelA @u pt
  return (wedge lp (implies lp label pred) (implies lp (neg label) (neg pred)))
```

The full sentence is a guarded universal quantifier $\forall_{x\in S}.\,\varphi_\theta(x)$ via `bigWedge`. The guard (`Guard u a`) is the subset $S$ to quantify over. For `GeomU`, the guard is a batch tensor and reduction uses LogSumExp; for `MeasU`, the guard is a list of points and the quantifier folds over them:

**ABSTRACT MONADIC SENTENCE** `BinaryFormulas.hs`

```
binarySentence lp guard paramMLP =
  bigWedge lp guard
    (binaryPredicate @u lp paramMLP)
```

**Definition 4.4.** *The **Kleisli grammatical interpretation**[9] $\mathscr{I}_\delta^{\text{Kleisli}}$, written as $[\![\cdot]\!]^{\text{Kleisli}}$ or $[\![\cdot]\!]$, extends the Tarski interpretation (Def. B.1) to a grammatical monad $\mathscr{M}_\delta$ on $\mathscr{C}_\delta$ with commutator* com *(App. A.1):*

$$\begin{aligned} \Xi = \textbf{\textit{Terms:}} \qquad & [\![\xi]\!] : \mathscr{I}_\gamma(\text{in}(\xi)) \to \mathscr{M}_\gamma \mathscr{I}_\gamma(\text{out}(\xi)), \\ \Phi = \textbf{\textit{Formulas:}} \qquad & [\![\phi]\!] : \mathscr{I}_\gamma([\phi]) \to \mathscr{M}_\gamma \Omega. \end{aligned}$$

| ***Syntax*** | ***Semantics*** | ***Type of Morphism*** |
|---|---|---|
| $[\![x]\!]$ | $\mathscr{I}_\gamma(x)$ | $\mathscr{I}_\gamma([x]) \to \mathscr{M}_\gamma \mathscr{I}_\gamma([x])$ |
| $[\![f(\vec{\xi})]\!]$ | $\mathscr{I}_\gamma(f)^{\cdot \mathscr{M}_\gamma} \circ \text{com} \circ [\![\vec{\xi}]\!]$ | $\mathscr{I}_\gamma(\text{in}(\vec{\xi})) \to \prod_i \mathscr{M}_\gamma \mathscr{I}_\gamma(\text{out}(\vec{\xi}_i)) \to \mathscr{M}_\gamma \mathscr{I}_\gamma(\text{dom}(f)) \to \mathscr{M}_\gamma \mathscr{I}_\gamma(\text{cod}(f))$ |
| $[\![R(\vec{\xi})]\!]$ | $\mathscr{I}_\gamma(R)^{\cdot \mathscr{M}_\gamma} \circ \text{com} \circ [\![\vec{\xi}]\!]$ | $\mathscr{I}_\gamma(\text{in}(\vec{\xi})) \to \prod_i \mathscr{M}_\gamma \mathscr{I}_\gamma(\text{out}(\vec{\xi}_i)) \to \mathscr{M}_\gamma \mathscr{I}_\gamma(\text{on}(R)) \to \mathscr{M}_\gamma \Omega$ |
| $[\![*(\vec{\phi})]\!]$ | $\mathscr{I}_\beta(*)^{\cdot \mathscr{M}_\beta} \circ \text{com} \circ [\![\vec{\phi}]\!]$ | $\mathscr{I}_\gamma([\vec{\phi}]) \to \left(\mathscr{M}_\beta \Omega\right)^{\text{ari}(*)} \to \mathscr{M}_\beta \left(\Omega^{\text{ari}(*)}\right) \to \mathscr{M}_\beta \Omega$ |
| $[\![Q\vec{x}(\phi)]\!]$ | $\mathscr{I}_\beta(Q)_{\mathscr{I}_\gamma([\vec{x}])} \circ \Lambda([\![\phi]\!])$ | $\mathscr{I}_\gamma([\phi] \setminus [\vec{x}]) \to \left(\mathscr{M}_\beta \Omega\right)^{\mathscr{I}_\gamma([\vec{x}])} \to \mathscr{M}_\beta \Omega$ |

Instantiating the formula fixes the universe, that is the category $\mathscr{C}_\delta$ and the monad $\mathscr{M}$ on it. For training: `@GeomU` yields a differentiable scalar (monad = `Identity`). For evaluation: `@MeasU` yields a probability distribution Dist Bool (monad = `Dist`):

[9] To unify the semantics, the interpretations of Tarski symbols are composed with the unit $\eta$ of the monad.

**CONCRETE AXIOM: GEOMU (TRAINING)** BinaryIntpTens.hs

```
binaryAxiomTens :: Torch.Tensor -> Torch.Tensor -> ParamsMLP -> Omega GeomU
binaryAxiomTens betaT guard paramMLP =
  runIdentity (binarySentence @GeomU betaT guard paramMLP)
```

**CONCRETE AXIOM: MEASU (EVALUATION)** BinaryIntpData.hs

```
binaryAxiomData :: [Point MeasU] -> ParamsMLP -> M MeasU (Omega MeasU)
binaryAxiomData guard paramMLP = binarySentence @MeasU () guard paramMLP
```

# 5 Inferential Layer

The inferential layer turns the formula into an optimization problem, viewing gradient-based learning as functorial composition in the parameterized category Para [13]. Here, the domain parameters $\theta_\gamma$ are the model parameters (e.g., MLP weights). Training optimizes a combined objective, where the loss function combines a *knowledge loss* $J_{\text{know}}$ (how unsatisfied is the axiom?) with a *data loss* $J_{\text{data}}$ (how far is the prediction from the label?). It jointly minimizes the objective $J(\theta)$ over $\theta = (\theta_\alpha, \theta_\beta, \theta_\gamma)$[10] via Adam for 1000 epochs, producing optimal parameters $\theta^*$. The parameter $\lambda \in [0,1]$ interpolates between pure data ($\lambda = 0$) and pure knowledge supervision ($\lambda = 1$):

$$J(\theta) = (1-\lambda)\cdot J_{\text{data}}(\theta) + \lambda \cdot J_{\text{know}}(\theta), \qquad J_{\text{know}} = \mathsf{softplus}\big([\![\forall x.\, \varphi_\theta(x)]\!]\big)$$

**INFERENCE THEORY** InferenceTheory.hs

The loss function symbols are declared abstractly:

```
class InferenceFun cat where
  type Loss cat :: *
  lossKnow :: cat -> Loss cat
  lossData :: cat -> cat -> Loss cat
  lossComb :: Loss cat -> Loss cat -> cat -> Loss cat
```

**INFERENCE INTERPRETATION: TENSREAL** InferenceIntpTens.hs

The interpretation assigns: lossKnow $\mapsto$ softplus, lossData $\mapsto$ crossEntropy, lossComb $\mapsto$ convex:

```
instance InferenceFun Torch.Tensor where
  type Loss Torch.Tensor = Torch.Tensor
  lossKnow = softplus
  lossData = crossEntropy
  lossComb = convex
```

# 6 Statistical Layer

The statistical theory $\mathbb{T}_\zeta$ specifies evaluation metric symbols: accuracy, F1-score, precision, recall, and confidence. The bridge between `MeasU` and `GeomU` is provided by `encPoint` (data point $\to$ tensor) and `decOmega` (logit $\to$ probability distribution). After training produces $\theta^* =$ `paramMLPOpti`, we evaluate through the `MeasU` pipeline:

$$\texttt{classifierA}^{\texttt{Meas}}_{\theta^*}(x) \;=\; \texttt{decOmega}\big(\texttt{MLP}_{\theta^*}(\texttt{encPoint}(x))\big)$$

[10] The categorical parameters $\theta_\alpha$ are not implemented yet, however they are expected to be used in the form of parameterized monads [41] (see App. C.1).

where `decOmega` applies $\sigma$ (sigmoid) internally and wraps the result as `DistBool`. We extract $P(\texttt{True})$ via integration and compare against the ground truth $\texttt{labelA}^{\texttt{Meas}}$ to compute metric scores. We determine the optimal sharpness parameter $\beta$ by sweeping $\beta \in \{0.5, 0.75, 1.0, \ldots, 3.0\}$ with $\lambda = 1$ (pure axiom). Each configuration is run 20 times (50 train / 50 test, 1000 epochs, Adam with $lr = 0.001$). Figure 3 shows that low $\beta$ values ($\leq 0.5$) produce high variance (F1 std $> 0.24$), because the smooth connectives lack sufficient sharpness to separate classes. The optimal sharpness is $\beta^* = 1.75$ (Acc $0.954 \pm 0.037$, F1 $0.919 \pm 0.055$), after which performance fluctuates without further improvement.

With $\beta^* = 1.75$ fixed, sweeping $\lambda$ from 0 (pure data) to 1 (pure axiom) reveals that increasing the knowledge weight monotonically improves both accuracy and F1 while reducing variance. Pure data ($\lambda$=0) is the worst setting (Acc $0.879 \pm 0.089$, F1 $0.715 \pm 0.290$), occasionally collapsing to all-negative predictions—a failure mode structurally impossible under axiom-driven training. Pure axiom ($\lambda$=1) achieves the best accuracy ($0.958 \pm 0.040$) and F1 ($0.920 \pm 0.072$).

We compare against LTNtorch [7], using identical dataset (100 points in $[0,1]^2$, circle-in-square), architecture (2→16→16→1 MLP with ELU), optimizer (Adam, $lr$=0.001), and training budget (1000 epochs, pure knowledge loss). LTNtorch uses $p$-Mean Error ($p$=2) aggregation with $\neg x = 1 - x$; NeSyCat uses LogSumExp ($\beta$=1.75) on unbounded logits with Softplus loss. All metrics are evaluated on a 50-point test set, averaged over 20 runs:

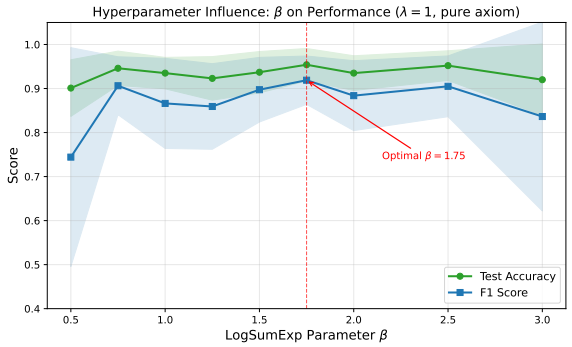


Figure 3: Beta sweep ($\lambda$=1, pure axiom, 20 runs).

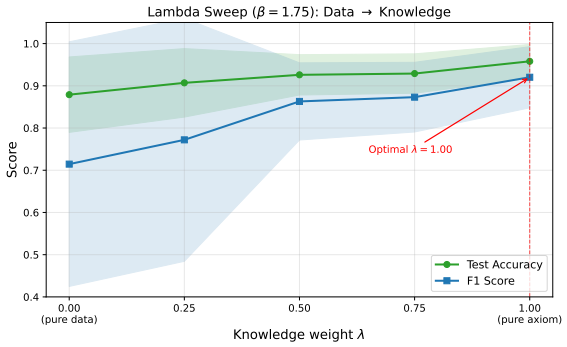


Figure 4: Lambda sweep ($\beta$=1.75, 20 runs).

Table 1: NeSyCat HaskTorch vs LTNtorch on binary classification (mean ± std, 20 runs, test-only)

| | Train Acc | Test Acc | F1 | Precision | $\bar{P}_+$ | ms/epoch |
|---|---|---|---|---|---|---|
| LTNtorch | 0.993 | $0.940 \pm 0.058$ | $0.905 \pm 0.085$ | $0.903 \pm 0.121$ | 0.890 | $0.40 \pm 0.01$ |
| **NeSyCat** | **1.000** | $\mathbf{0.958 \pm 0.040}$ | $\mathbf{0.920 \pm 0.072}$ | $\mathbf{0.938 \pm 0.063}$ | 0.876 | $\mathbf{0.29 \pm 0.01}$ |

NeSyCat achieves perfect train accuracy, higher test accuracy (+1.8%), F1 (+1.5%), precision (+3.5%), all with lower variance, and is 28% faster per epoch despite calling the same LibTorch C++ backend via FFI—the speedup comes from full-batch tensor operations without Python interpreter overhead. The key advantage is LogSumExp semantics on unbounded logits: by operating outside the $[0,1]$ simplex and applying the Softplus loss, gradients for confidently classified points vanish smoothly, focusing optimization on boundary examples.

# 7 Related & Future Work

The whole paper is heavily based on our first NeSyCat paper [34], still to be published.

The benchmark in Section 6 only initiates the comparison with Logic Tensor Networks [5], available as both a TensorFlow [6] and PyTorch [10] implementation. Equally important is the relationship to probabilistic logic programming: DeepProbLog [27] embeds naturally into NeSyCat by choosing the distribution monad at the categorical layer and interpreting its probabilistic facts as Kleisli morphisms. DeepStochLog [42] extends this to stochastic definite-clause grammars, while DeepSeaProbLog [37] introduces continuous quantifiers—both of which map onto NeSyCat's quantifier framework.

On the categorical side, the lens category [12] and the parameterized category Para [13, 14] provide the natural setting for learnable morphisms, formalized as an instance of *actegories* (actions of monoidal categories) in the sense of [19]. String diagrams [21, 33] offer a graphical syntax for morphism composition that we plan to implement in React Flow [43], enabling visual construction of NeSyCat pipelines that automatically generate executable code. Extending *parameterized monads* [41] (see App. C.1) to actegories offers a path to learning the parameters of monads, and to expanding the classical confusion matrix to a *confusion square* for richer statistical evaluation at the $\zeta$-layer. Additionally, the two modes of Tarski ($A \to B$) and Kleisli ($A \to \mathscr{M}B$) symbols and their interpretations can be extended by a third mode of Eilenberg-Moore symbols, interpreted in the Eilenberg-Moore category as morphisms $\mathscr{M}A \to \mathscr{M}B$.

With respect to logic, spatial and temporal NeSy logics [28] can be integrated by treating time and space as *lower parameters* in string diagrams in contrast to the *upper parameters* of neural weights [14]. The resulting spatiotemporal parameter manifold admits a geometric interpretation analogous to relativistic spacetime manifolds [15], where the metric structure governs the inferential dynamics. Higher-order logic can be implemented in diffeological spaces [17] and connected to NLP, by the HO DisCoCat framework [40], which provides a similar compositional semantics with monoidal signatures. The sequent calculus formulation of the comparison symbol $\vdash$ [20] offers a proof-theoretic perspective.

On the data and knowledge representation side, categorical database theory and ologs [38] together with categorical data structures for technical computing [32] provide the foundation for representing domain data in a PostgreSQL database while simultaneously maintaining knowledge axiom ontologies as nested TypeScript files, connected via an ORM such as Prisma or alternatively via CQL [35].

Beyond the current HaskTorch implementation, I plan JAX, PyTorch, and Julia ports, an Agda formalization exploiting dependent types for stronger static guarantees, and a deeper connection to probabilistic programming languages such as Anglican [39] via quasi-Borel spaces [16].

# 8 Conclusion

We have presented the NeSyCat framework, which structures neurosymbolic AI into six layers ($\alpha$–$\zeta$) with six components each, bridging categorical logic and the Haskell type system via the computational trilogy. Every definition—from semantic universes (`GeomU`, `MeasU`) through 2Mon-BLat truth values and polymorphic domain signatures to the Kleisli grammatical interpretation—was paired with running HaskTorch code and instantiated on a binary classification example, benchmarking against LTNtorch with higher accuracy (+1.8%), F1 (+1.5%), precision (+3.5%), and 28% faster execution. The concrete category choices (**Set**, **ΔGen**, **QBS**) were driven by each layer's requirements, and the resulting structure acts as both a blueprint for other languages and a platform for further categorical results. The implementation is available at `https://github.com/cherryfunk/NeSyCat-HaskTorch`.

## Acknowledgements

Professionally, I thank my supervisor Prof. Mossakowski for reminding me of how important a working implementation is and for laying the groundwork of the Haskell implementation, as well as my colleague Björn Gehrke for the many discussions on type systems. Furthermore, I thank Nathanael Arkor for answering many of my questions and Tobias Fritz for helping me believe that I am not crazy.

Personally, I would like to thank my dad for being such a good host, my aunt Nilda for reminding me how important references are and Alice for her patience. What is an Arrow without a Cat, Alice?

## References

[1] Nathanael Arkor & Dylan McDermott (2024): *The Formal Theory of Relative Monads*. *Journal of Pure and Applied Algebra* 228(9), p. 107676, doi:10.1016/j.jpaa.2024.107676. arXiv:2302.14014.

[2] Steve Awodey (2010): *Category Theory*, 2. ed edition. *Oxford Logic Guides* 52, Oxford Univ. Press, Oxford.

[3] Steve Awodey & Andrej Bauer: *Introduction to Categorical Logic*.

[4] Steve Awodey & Henrik Forssell (2013): *First-Order Logical Duality*. *Annals of Pure and Applied Logic* 164(3), pp. 319–348, doi:10.1016/j.apal.2012.10.016. arXiv:1008.3145.

[5] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini & Michael Spranger (2022): *Logic Tensor Networks*. *Artificial Intelligence* 303, p. 103649, doi:10.1016/j.artint.2021.103649. arXiv:2012.13635.

[6] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini & Michael Spranger (2022): *Logic Tensor Networks — TensorFlow Implementation*. `https://github.com/logictensornetworks/logictensornetworks`. GitHub repository.

[7] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini & Michael Spranger (2022): *Logic Tensor Networks*. *Artificial Intelligence* 303, p. 103649, doi:10.1016/j.artint.2021.103649. Available at `http://arxiv.org/abs/2012.13635`. ArXiv:2012.13635 [cs].

[8] Jérôme Bolte & Edouard Pauwels (2021): *Conservative Set Valued Fields, Automatic Differentiation, Stochastic Gradient Descent Methods and Deep Learning*. *Mathematical Programming* 188, pp. 19–51, doi:10.1007/s10107-020-01501-5.

[9] Jan Boman (1967): *Differentiability of a Function and of its Compositions with Functions of One Variable*. *Mathematica Scandinavica* 20, pp. 249–268, doi:10.7146/math.scand.a-10835.

[10] Tommaso Carraro, Luciano Serafini & Fabio Aiolli (2024): *LTNtorch: PyTorch Implementation of Logic Tensor Networks*, doi:10.48550/arXiv.2409.16045. arXiv:2409.16045.

[11] J. Daniel Christensen, Gord Sinnamon & Enxin Wu (2014): *The D-Topology for Diffeological Spaces*. *Pacific Journal of Mathematics* 272(1), pp. 87–110, doi:10.2140/pjm.2014.272.87.

[12] Brendan Fong & David I. Spivak (2019): *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, doi:10.1017/9781108668804.

[13] Brendan Fong, David I. Spivak & Rémy Tuyéras (2019): *Backprop as Functor: A compositional perspective on supervised learning*. In: *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, IEEE, pp. 1–13, doi:10.1109/LICS.2019.8785665.

[14] Bruno Gavranović, Paul Lessard, Andrew Dudzik, Tamara von Glehn, João G. M. Araújo & Petar Veličković (2024): *Position: Categorical Deep Learning Is an Algebraic Theory of All Architectures*. arXiv:2402.15332.

[15] Stephen W. Hawking & George F. R. Ellis (1973): *The Large Scale Structure of Space-Time*. Cambridge Monographs on Mathematical Physics, Cambridge University Press, doi:10.1017/CBO9780511524646.

[16] Chris Heunen, Ohad Kammar, Sam Staton & Hongseok Yang (2017): *A Convenient Category for Higher-Order Probability Theory*. In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–12, doi:10.1109/LICS.2017.8005137. arXiv:1701.02547.

[17] Mathieu Huot, Sam Staton & Matthijs Vákár (2020): *Correctness of Automatic Differentiation via Diffeologies and Categorical Gluing*. In: *Foundations of Software Science and Computation Structures (FoSSaCS 2020)*, *Lecture Notes in Computer Science* 12077, Springer, pp. 319–338. arXiv:2001.02209.

[18] Patrick Iglesias-Zemmour (2013): *Diffeology*. *Mathematical Surveys and Monographs* 185, American Mathematical Society, doi:10.1090/surv/185.

[19] George Janelidze & G. Max Kelly (2001): *A Note on Actions of a Monoidal Category*. *Theory and Applications of Categories* 9(4), pp. 61–91. Available at `http://www.tac.mta.ca/tac/volumes/9/n4/n4.pdf`.

[20] Peter T Johnstone (2002): *Sketches of an Elephant A Topos Theory Compendium*. Oxford University PressOxford, doi:10.1093/oso/9780198515982.001.0001.

[21] André Joyal & Ross Street (1991): *The Geometry of Tensor Calculus, I*. *Advances in Mathematics* 88(1), pp. 55–112, doi:10.1016/0001-8708(91)90003-P.

[22] Alexander S. Kechris (1995): *Classical Descriptive Set Theory*. *Graduate Texts in Mathematics* 156, Springer, doi:10.1007/978-1-4612-4190-4.

[23] Anders Kock (1970): *Monads on Symmetric Monoidal Closed Categories*. *Archiv der Mathematik* 21, pp. 1–10, doi:10.1007/BF01220868.

[24] Anders Kock (1972): *Strong Functors and Monoidal Monads*. *Archiv der Mathematik* 23, pp. 113–120, doi:10.1007/BF01304852.

[25] F. William Lawvere (1970): *Equality in Hyperdoctrines and Comprehension Schema as an Adjoint Functor*. In: *Applications of Categorical Algebra*, *Proceedings of Symposia in Pure Mathematics* 17, American Mathematical Society, pp. 1–14, doi:10.1090/pspum/017/0257175.

[26] Ernest G. Manes (1976): *Algebraic Theories*. Springer, doi:10.1007/978-1-4612-9860-1.

[27] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester & Luc De Raedt (2018): *DeepProbLog: Neural Probabilistic Logic Programming*, doi:10.48550/arXiv.1805.10872. arXiv:1805.10872.

[28] Jiayuan Mao, Zhezheng Luo, Chuang Gan, Joshua B Tenenbaum, Jiajun Wu, Leslie Pack Kaelbling & Tomer D Ullman: *Temporal and Object Quantification Networks*.

[29] Francisco Marmolejo & Richard J. Wood (2010): *Monads as Extension Systems — No Iteration is Necessary*. *Theory and Applications of Categories* 24(4), pp. 84–113.

[30] Eugenio Moggi (1991): *Notions of Computation and Monads*. *Information and Computation* 93(1), pp. 55–92, doi:10.1016/0890-5401(91)90052-4.

[31] nLab authors (2025): *Theory*. `https://ncatlab.org/nlab/show/theory`. NLab.

[32] Evan Patterson, Owen Lynch & James Fairbanks (2022): *Categorical Data Structures for Technical Computing*. *Compositionality* 4, p. 5, doi:10.32408/compositionality-4-5. arXiv:2106.04703.

[33] Robin Piedeleu & Fabio Zanasi (2023): *An Introduction to String Diagrams for Computer Scientists*, doi:10.48550/arXiv.2305.08768. arXiv:2305.08768.

[34] Daniel Romero Schellhorn & Till Mossakowski (2025): *NeSyCat: A Modular Monad-Based Semantics of the Neurosymbolic ULLER Framework*. *Neurosymbolic Artificial Intelligence*. To appear. Extended version of NeSy 2025 accepted paper, submitted and accepted to the Special Issue on NeSy 2025 Extended Papers.

[35] Patrick Schultz, David I. Spivak & Ryan Wisnesky (2017): *Algebraic Model Management: A Survey*. In: *Recent Trends in Algebraic Development Techniques (WADT 2016)*, Lecture Notes in Computer Science, Springer. arXiv:2301.04846.

[36] Kazuhisa Shimakawa, K. Yoshida & Tadayuki Haraguchi (2018): *Homology and Cohomology via Enriched Bifunctors*. *Kyushu Journal of Mathematics* 72(2), pp. 239–252, doi:10.2206/kyushujm.72.239.

[37] Lennert De Smet, Pedro Zuidberg Dos Martires, Robin Manhaeve, Giuseppe Marra, Angelika Kimmig & Luc De Raedt (2023): *DeepSeaProbLog: Neural Probabilistic Logic Programming in Discrete-Continuous Domains*, doi:10.48550/arXiv.2303.04660. arXiv:2303.04660.

[38] David I. Spivak (2014): *Category Theory for the Sciences*. The MIT Press, Cambridge, Massachusetts.

[39] David Tolpin, Jan-Willem van de Meent, Hongseok Yang & Frank Wood (2016): *Design and Implementation of Probabilistic Programming Language Anglican*. In: *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages (IFL 2016)*, ACM, doi:10.1145/3064899.3064910.

[40] Alexis Toumi & Giovanni de Felice (2025): *Higher-Order DisCoCat (Peirce-Lambek-Montague Semantics)*. *Electronic Proceedings in Theoretical Computer Science* 429, pp. 130–145, doi:10.4204/EPTCS.429.7. arXiv:2311.17813.

[41] Tarmo Uustalu (2003): *Generalizing Substitution*. *RAIRO – Theoretical Informatics and Applications* 37(4), pp. 315–336, doi:10.1051/ita:2003022.

[42] Thomas Winters, Giuseppe Marra, Robin Manhaeve & Luc De Raedt (2021): *DeepStochLog: Neural Stochastic Logic Programming*, doi:10.48550/arXiv.2106.12574. arXiv:2106.12574.

[43] xyflow (2025): *React Flow: Node-Based UIs in React*. `https://reactflow.dev`. Open-source library, `https://github.com/xyflow/xyflow`.

# A Monads and their Commutators

**Definition A.1.** *An **extension system** [29], also called a **Kleisli triple** [30] or **algebraic theory in extension form** [26], on a category $\mathscr{C}$ consists of:*

- *for every object $A \in \mathscr{C}$, an object $\mathscr{M}A \in \mathscr{C}$ and a morphism $\eta_A : A \to \mathscr{M}A$ (the* unit*),*
- *for every morphism $f : B \to \mathscr{M}A$, a morphism $f^{\cdot\mathscr{M}} : \mathscr{M}B \to \mathscr{M}A$ (the* Kleisli extension*),*

*satisfying: (1)* $(\eta_A)^{\mathscr{M}} = \mathrm{id}_{\mathscr{M}A}$, *(2)* $f^{\mathscr{M}} \circ \eta_B = f$, *(3)* $f^{\mathscr{M}} \circ g^{\mathscr{M}} = (f^{\mathscr{M}} \circ g)^{\mathscr{M}}$.
*This is equivalent to a monad* $(\mathscr{M}, \eta, \mu)$ *via* $\mu_A = (\mathrm{id}_{\mathscr{M}A})^{\mathscr{M}}$ *and* $f^{\mathscr{M}} = \mu_A \circ \mathscr{M} f$.

**Definition A.2.** *A* ***strong extension system*** *[24] on a cartesian closed category* $\mathscr{C}$ *is an extension system* $(\mathscr{M}, \eta, (-)^{\mathscr{M}})$ *equipped with an internal extension (the* bind*):*

$$\mathsf{bind}_{A,B} : \mathscr{M}B \times (\mathscr{M}A)^B \to \mathscr{M}A,$$

*internalizing the Kleisli extension:* $\mathsf{bind}(m, f) = f^{\mathscr{M}}(m)$. *This is equivalent to a* tensorial strength $\sigma_{A,B} : A \times \mathscr{M}B \to \mathscr{M}(A \times B)$.

**Definition A.3.** *A strong monad* $(\mathscr{M}, \eta, \mu, \sigma)$ *on a symmetric monoidal category is* ***commutative*** *[23] if its left-strength* $\sigma$ *and right-strength* $\tau$ *(induced from* $\sigma$ *and the braiding) satisfy:*

$$\mu \circ \mathscr{M}\tau \circ \sigma \;=\; \mu \circ \mathscr{M}\sigma \circ \tau \;:\; \mathscr{M}A \times \mathscr{M}B \to \mathscr{M}(A \times B).$$

The combined context map $\mathrm{com}_{A,B} : \mathscr{M}A \times \mathscr{M}B \to \mathscr{M}(A \times B)$ can be defined for *any* strong monad, but in two ways: evaluating the left argument first ($\mathrm{com}^L = \mu \circ \mathscr{M}\tau \circ \sigma$) or the right argument first ($\mathrm{com}^R = \mu \circ \mathscr{M}\sigma \circ \tau$). A strong monad is commutative precisely when $\mathrm{com}^L = \mathrm{com}^R$. For example, the Giry monad on **Meas** is commutative—essentially the content of Fubini's theorem.

In neuro-symbolic architectures, commutativity ensures that commutative logical connectives (like $\wedge$) remain commutative after monadic lifting. Without commutativity, the semantics of a formula would be sensitive to evaluation order or input wiring.

# B Grammar in Context and with Substitution

We spell out how the grammatical interpretation extends to terms and formulas *in context*—i.e., with an explicit ambient context $\Gamma$ of typed variables—and how substitution is handled compositionally. For brevity, we present only the *Tarski* grammatical interpretation (identity monad). The same constructions generalize directly to the *Kleisli* grammatical interpretation (Def. 4.4) by replacing each morphism $A \to B$ with a Kleisli morphism $A \to \mathscr{M}B$ and inserting the combined context map com and Kleisli extensions.

**Definition B.1.** *The* ***Tarski grammatical interpretation*** $\mathscr{I}_\delta^{\mathrm{Tarski}}$*, written as* $[\![\cdot]\!]^{\mathrm{Tarski}}$*, or here simply as* $[\![\cdot]\!]$*, is similar to the Kleisli interpretation from Def. 4.4, but only interprets the Tarski symbols:*

$$\Xi = \textbf{\textit{Terms:}} \qquad [\![\xi]\!] : \mathscr{I}(\mathrm{in}(\xi)) \to \mathscr{I}(\mathrm{out}(\xi)),$$
$$\Phi = \textbf{\textit{Formulas:}} \qquad [\![\phi]\!] : \mathscr{I}(\mathrm{ovr}(\phi)) \to \Omega.$$

| ***Syntax*** | ***Semantics*** | ***Type of Morphism*** |
|---|---|---|
| $[\![x]\!]$ | $\mathscr{I}(x)$ | $\mathscr{I}(\mathrm{in}(x)) \to \mathscr{I}(\mathrm{out}(x))$ |
| $[\![f(\vec{\xi})]\!]$ | $\mathscr{I}(f) \circ [\![\vec{\xi}]\!]$ | $\mathscr{I}(\mathrm{in}(\vec{\xi})) \to \mathscr{I}(\mathrm{dom}(f)) \to \mathscr{I}(\mathrm{cod}(f))$ |
| $[\![\xi'[\vec{x} := \vec{\xi}]]\!]$ | $[\![\xi']\!] \circ \left(\mathrm{id}_{\mathscr{I}([\xi'] \setminus [\vec{x}])} \times [\![\vec{\xi}]\!]\right)$ | $\mathscr{I}([\xi'] \setminus [\vec{x}]) \times \mathscr{I}([\vec{\xi}]) \to \mathscr{I}([\xi'] \setminus [\vec{x}]) \times \mathscr{I}([\vec{x}]) \to \mathscr{I}(\mathrm{out}(\xi'))$ |
| $[\![R(\vec{\xi})]\!]$ | $\mathscr{I}(R) \circ [\![\vec{\xi}]\!]$ | $\mathscr{I}(\mathrm{in}(\vec{\xi})) \to \mathscr{I}(\mathrm{on}(R)) \to \Omega$ |
| $[\![*(\vec{\phi})]\!]$ | $\mathscr{I}(*) \circ [\![\vec{\phi}]\!]$ | $\mathscr{I}(\mathrm{ovr}(\vec{\phi})) \to \Omega^{\mathrm{ari}(*)} \to \Omega$ |
| $[\![Q\vec{x}(\phi)]\!]$ | $\mathscr{I}(Q)_{\mathscr{I}([\vec{x}])} \circ \Lambda([\![\phi]\!])$ | $\mathscr{I}([\phi] \setminus [\vec{x}]) \to \Omega^{\mathscr{I}([\vec{x}])} \to \Omega$ |
| $[\![\phi[\vec{x} := \vec{\xi}]]\!]$ | $[\![\phi]\!] \circ \left(\mathrm{id}_{\mathscr{I}([\phi] \setminus [\vec{x}])} \times [\![\vec{\xi}]\!]\right)$ | $\mathscr{I}([\phi] \setminus [\vec{x}]) \times \mathscr{I}([\vec{\xi}]) \to \mathscr{I}([\phi] \setminus [\vec{x}]) \times \mathscr{I}([\vec{x}]) \to \Omega$ |

To extend this to formulas *in context*, we need the notion of a context and the weakening rule (precomposition with projections):

**Definition B.2.** *A **term-in-context** [20] $\vec{x}.\xi$ is a term $\xi$ in a* suitable *context $\vec{x}$, i.e. one which contains all variables mentioned in $\xi$. Also, a **formula-in-context** $\vec{x}.\phi$ is a formula $\phi$ in a* suitable *context $\vec{x}$, i.e. one which contains all free variables occuring in $\phi$, and similarly for a **comparison-in-context** $\vec{x}.\psi$.*

**Definition B.3.** *The following table shows the semantics of the grammar in context. The type of morphism is given by the following formula:*

$$\begin{aligned} \Xi = \textbf{\textit{Terms:}} &\qquad [\![\Gamma.\xi]\!] : \mathscr{I}(\Gamma) \to \mathscr{I}(\mathrm{in}(\xi)) \to \mathscr{I}(\mathrm{out}(\xi)), \\ \Phi = \textbf{\textit{Formulas:}} &\qquad [\![\Gamma.\phi]\!] : \mathscr{I}(\Gamma) \to \mathscr{I}([\phi]) \to \Omega. \end{aligned}$$

| ***Syntax*** | ***Semantics*** $\ldots \circ \pi$ *(proj. from* $\Gamma$*)* | ***Type of Morphism:*** $\mathscr{I}(\Gamma) \to \ldots$ |
|---|---|---|
| $[\![\Gamma.y]\!]$ | $\mathscr{I}(y)$ | $\mathscr{I}([y]) \to \mathscr{I}(\mathrm{of}(y))$ |
| $[\![\Gamma.f(\vec{\xi})]\!]$ | $\mathscr{I}(f) \circ [\![\vec{\xi}]\!]$ | $\mathscr{I}([\vec{\xi}]) \to \mathscr{I}(\mathrm{dom}(f)) \to \mathscr{I}(\mathrm{cod}(f))$ |
| $[\![\Gamma.\xi'[\vec{y} := \vec{\xi}]]\!]$ | $[\![\xi']\!] \circ \left(\mathrm{id}_{\mathscr{I}([\xi'] \setminus [\vec{y}])} \times [\![\vec{\xi}]\!]\right)$ | *too long, see footnote*[11] |
| $[\![\Gamma.R(\vec{\xi})]\!]$ | $\mathscr{I}(R) \circ [\![\vec{\xi}]\!]$ | $\mathscr{I}([\vec{\xi}]) \to \mathscr{I}(\mathrm{on}(R)) \to \Omega$ |
| $[\![\Gamma.*(\vec{\phi})]\!]$ | $\mathscr{I}(*) \circ [\![\vec{\phi}]\!]$ | $\mathscr{I}([\vec{\phi}]) \to \Omega^{\mathrm{ari}(*)} \to \Omega$ |
| $[\![\Gamma.Q\vec{y}(\phi)]\!]$ | $\mathscr{I}(Q)_{\mathscr{I}([\vec{y}])} \circ \Lambda_{\mathscr{I}([\vec{y}])}([\![\phi]\!])$ | $\mathscr{I}([\phi] \setminus [\vec{y}]) \to \Omega^{\mathscr{I}([\vec{y}])} \to \Omega$ |
| $[\![\Gamma.\phi[\vec{y} := \vec{\xi}]]\!]$ | $[\![\phi]\!] \circ \left(\mathrm{id}_{\mathscr{I}([\phi] \setminus [\vec{y}])} \times [\![\vec{\xi}]\!]\right)$ | *too long, see footnote*[12] |

Here $\mathscr{I}(\Gamma) := \mathscr{I}(S_1) \times \cdots \times \mathscr{I}(S_n)$ is the finite product in $\mathscr{C}$ determined by the context. The morphisms like $\pi_{\mathscr{I}([y])} : \mathscr{I}(\Gamma) \to \mathscr{I}([y])$ are the corresponding canonical product projections (dropping all components except the indicated variable(s)). The shorthand $[\![\xi]\!]$ denotes the semantics of a term $\xi$ in its canonical context given by its free variables $[\xi]$ as in Def. 4.1. With that in mind, $[\![\vec{\xi}]\!] := \langle [\![\vec{\xi}_i]\!] \rangle_i$ is the unique morphism of the universal property of the product of terms in canonical context, and for a list of formulas $\vec{\phi}$ analogously $[\![\vec{\phi}]\!] := \langle [\![\vec{\phi}_i]\!] \rangle_i$. The semantics of the quantified formulas is for the case of $[y] \subseteq \Gamma$, since otherwise $[\![\Gamma.Qy(\phi)]\!] := [\![\Gamma.\phi]\!]$ and similarly for the substitution formula $[\vec{y}] \subseteq \Gamma$ since adding or removing variables not in $\Gamma$ does not change the semantics.

# C Parameterized Monads

**Definition C.1.** *A **parameterized monad** $\mathscr{T}_\theta$ on a concrete category $\mathscr{C}$ with f.p.*[13] *is a monad specified by an ordered semiring $\mathbb{X}$ internal to $\mathscr{C}$. The monad $\mathscr{T}_\theta$ is then given as follows, see Table 2 for examples:*

$$\begin{aligned} &\mathscr{T}_\theta : \Theta \times \mathscr{C} \to \mathscr{C} \\ &\qquad (\theta, A) \mapsto \{f \in \mathscr{C}[A, \mathbb{X}^{\theta_0}] \mid \theta_1 \leq \|f\|_{\theta_2} \leq \theta_3\} \text{ (equipped with } \mathscr{C}\text{-structure).} \end{aligned}$$

[11] $\mathscr{I}([\xi'] \setminus [\vec{y}]) \times \mathscr{I}([\vec{\xi}]) \to \mathscr{I}([\xi'] \setminus [\vec{y}]) \times \mathscr{I}([\vec{y}]) \to \mathscr{I}(\mathrm{out}(\xi'))$

[12] $\mathscr{I}([\phi] \setminus [\vec{y}]) \times \mathscr{I}([\vec{\xi}]) \to \mathscr{I}([\phi] \setminus [\vec{y}]) \times \mathscr{I}([\vec{y}]) \to \Omega$

[13] Alternatively, this can be expanded to actegories [14].

This definition is related to, but distinct from, the notion of *parameterized monads* due to [41], who defines a parameterized monad on $\mathscr{C}$ as a triple $(T', \eta', -^\circ)$ with $T' : |\mathscr{C}| \times \mathscr{C} \to \mathscr{C}$—essentially a functor $\mathscr{C} \to \mathbf{Monad}(\mathscr{C})$. In Uustalu's formulation, the parameter is an object of $\mathscr{C}$ itself. In our Definition C.1, the parameter $\theta \in \Theta$ need *not* be an object of $\mathscr{C}$: it lives in a separate parameter manifold $\mathsf{Para}(\mathscr{C})$ [13, 14], which is an *actegory* [19]—an action of a monoidal category $(\Theta, \times)$ of parameter spaces on $\mathscr{C}$. This is more general: the parameter space can be any object of any monoidal category acting on $\mathscr{C}$, not just an object of $\mathscr{C}$ itself. This actegory perspective also provides a natural setting for the *lens* pattern [12] underlying backpropagation, unifying the parameterization of monads and interpretations within a single categorical framework.

Table 2: Examples of Parameterized Monads

| $\mathscr{T}_\theta$ | $\mathscr{C}$ | $\mathbb{X}^{\theta_0}$ | $\theta_1, \theta_2, \theta_3$ |
|---|---|---|---|
| Identity | **Set** | $(\mathbb{N}, \leq, 0, 1, +, \cdot)$ | $1 \leq \lVert\cdot\rVert_1 \leq 1$ |
| Powerset $\mathscr{P}$ | **Set** | $(\mathbb{N}^2, \leq, 0, 1, +, \cdot)$ | $0 \leq \lVert\cdot\rVert_\infty \leq 1$ |
| n.e. Powerset $\mathscr{P}_{\neq\emptyset}$ | **Set** | $(\mathbb{N}^2, \leq, 0, 1, +, \cdot)$ | $1 \leq \lVert\cdot\rVert_\infty \leq 1$ |
| Sub-Distribution $\mathscr{S}$ | **Set** | $(\mathbb{R}^2_{\geq 0}, \leq, 0, 1, +, \cdot)$ | $0 \leq \lVert\cdot\rVert_1 \leq 1$ |
| Distribution $\mathscr{D}$ | **Set** | $(\mathbb{R}^2_{\geq 0}, \leq, 0, 1, +, \cdot)$ | $1 \leq \lVert\cdot\rVert_1 \leq 1$ |
| Giry $\mathscr{G}$ | **BorelMeas** | $(\mathbb{R}^2_{\geq 0}, \leq, 0, 1, +, \cdot)$ | $1 \leq \lVert\cdot\rVert_1 \leq 1$ |

# D Categorical 2Mon-BLat

Just as the logical vocabulary uses a single sort $\tau$ together with connective, comparison, and quantifier symbols to describe ordered algebraic theories (e.g., monoids, lattices), the categorical vocabulary uses for example a single category symbol $\mathscr{C}$ together with functor, distributor, and limit symbols. The key insight is that the *same ordered algebraic theories* appear at both levels—the difference lies solely in the ambient category and the choice of the dualizing object [4]: a theory interpreted in **Set** with dualizing object $\{0, 1\}$ yields an ordered algebraic structure on a set, while the same theory interpreted in **Cat** with dualizing object **Set** yields a structured category.

**Definition D.1.** *A **categorical 2Mon-BLat** is a categorical theory $\mathbb{T}_\alpha$ with the following symbols:*

$$\big(\mathscr{C},\ \mathrm{id},\ \mathrm{Hom},\ \bot,\ \top,\ \oplus,\ \otimes,\ \vec{0},\ \vec{1}\big),$$

*having the following arities:*

$$\mathrm{id} : \mathscr{C} \to \mathscr{C}, \quad \mathrm{Hom} : \mathscr{C}^2, \quad \bot : \mathscr{C}, \quad \top : \mathscr{C}, \quad \oplus : \mathscr{C}^2 \to \mathscr{C}, \quad \otimes : \mathscr{C}^2 \to \mathscr{C}, \quad \vec{0} : \mathscr{C}, \quad \vec{1} : \mathscr{C}.$$

*The dual $(\mathscr{C}, \oplus, \vec{0})$ and $(\mathscr{C}, \otimes, \vec{1})$ obey the axioms of* monoidal categories*, and $(\mathscr{C}, \bot, \top)$ satisfies axioms of a categoy with finite coproducts and products: $\bot$ is initial symbol, $\top$ is terminal symbol, and $\sqcup : \mathscr{C}^2 \to \mathscr{C}$ and $\sqcap : \mathscr{C}^2 \to \mathscr{C}$ are coproduct and product symbols (which may or may not coincide with $\oplus, \otimes, \vec{0}, \vec{1}$).*

Note that the logical 2Mon-BLat (Def. 2.2) requires the connectives to be *monotone* with respect to the comparison $\vdash$ (i.e., $a \vdash a'$ and $b \vdash b'$ implies $a \otimes b \vdash a' \otimes b'$). At the categorical layer, this monotonicity is subsumed by *functoriality*: since $\otimes$ is a bifunctor, given morphisms $f : A \to A'$ and $g : B \to B'$, one

automatically obtains $f \otimes g : A \otimes B \to A' \otimes B'$. Functoriality is a categorification of monotonicity, carrying strictly more data: not merely the existence of a relationship, but the actual morphisms witnessing it.

**Definition D.2.** *A **categorical A2Mon-BLat** is a categorical 2Mon-BLat equipped with a dual pair of* limit symbols Limt $:= (\bigoplus, \bigotimes)$. *If the category has arbitrary coproducts and products, its axioms additionally give rise to the dual coproduct and product functor symbols $\coprod$ and $\prod$ (which may or may not coincide with $\bigoplus$ and $\bigotimes$).*

This definition mirrors exactly the logical 2Mon-BLat and A2Mon-BLat theories (Defs. 2.2 and 2.3)—the only difference is the sort: the logical theory operates on a truth symbol $\tau$, while the categorical theory operates on a category symbol $\mathscr{C}$. Whether the tensor $\otimes$ coincides with the product $\sqcap$ (equivalently, whether $\oplus$ coincides with $\sqcup$) distinguishes two fundamental cases which parallels the logical vocabulary: in classical logic, $\otimes = \wedge$ and $\oplus = \vee$ (the "cartesian" case); in linear or fuzzy logics, $\otimes$ and $\oplus$ are independent connectives (the "general" case).

**Example D.3** (Cartesian Monoidal Category)**.** *A categorical 2Mon-BLat in which $\otimes = \sqcap$, $\oplus = \sqcup$, $\vec{0} = \bot$ and $\vec{1} = \top$ is called* cartesian monoidal. *Here the monoidal structure adds no independent information beyond what the categorical limits and colimits already provide.*

**Example D.4** (Monoidal Category)**.** *In general, it is possible that $\otimes \neq \sqcap$ (or products may not exist at all). It carries a potentially independent monoidal structure: the tensor $\otimes$ with unit $\vec{1}$ is an additional bifunctor, not necessarily determined by limits.*

# E Alchemical Excurse: The Category Theorist's Stone

The four categories $\mathscr{C}_\alpha, \mathscr{C}_\beta, \mathscr{C}_\gamma, \mathscr{C}_\delta$ are related as follows. The ambient category $\mathscr{C}_\alpha$ (e.g. **Cat** or **Hask**) contains $\mathscr{C}_\beta$, $\mathscr{C}_\gamma$, and $\mathscr{C}_\delta$ as objects. The logical category $\mathscr{C}_\beta$ and the domain category $\mathscr{C}_\gamma$ both embed into the grammatical category $\mathscr{C}_\delta$ via faithful functors $J_\beta : \mathscr{C}_\beta \to \mathscr{C}_\delta$ and $J_\gamma : \mathscr{C}_\gamma \to \mathscr{C}_\delta$.

The truth object $\Omega := \mathscr{I}(\tau)$ plays a distinguished role: it must be an object of *all three* categories simultaneously. Formally, $\Omega$ lies in $\mathrm{im}(J_\beta) \cap \mathrm{im}(J_\gamma) \subseteq \mathscr{C}_\delta$. This is necessary because predicates are morphisms $\mathscr{I}(S_1) \times \cdots \times \mathscr{I}(S_n) \to \Omega$ whose domain sorts live in $\mathscr{C}_\gamma$ while $\Omega$ carries the logical structure of $\mathscr{C}_\beta$—both must be composable in $\mathscr{C}_\delta$. By slight abuse of notation, we write $\Omega$ for the truth object in all three categories interchangeably, identifying it with its preimages under $J_\beta$ and $J_\gamma$.

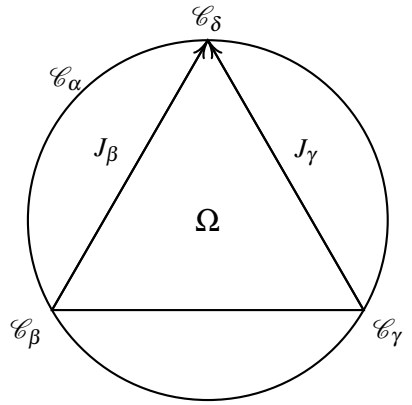


Figure 5: The Category Theorist's Stone.