Journal Title
XX(X):1–40

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

# mULLER: A Modular Monad-Based Semantics of the Neurosymbolic ULLER Framework

Daniel Romero Schellhorn[1] and Till Mossakowski[2]

**Abstract**

**ULLER** (Unified Language for LEarning and Reasoning) offers a unified first-order logic (FOL) syntax, enabling its knowledge bases to be used directly across a wide range of neurosymbolic systems. The original specification endows this syntax with three pairwise independent semantics: classical, fuzzy, and probabilistic, each accompanied by dedicated semantic rules. We show that these seemingly disparate semantics are all instances of one categorical framework based on *monads*, the very construct that models side effects in functional programming. This enables the *modular* addition of new semantics and systematic translations between them. As example, we outline the addition of generalised quantification in Logic Tensor Networks (LTN) to arbitrary (also infinite) domains by extending the Giry monad to probability spaces. In particular, our approach allows a modular implementation of ULLER in Python and Haskell, of which we have published initial versions on GitHub.

**Keywords**

Neurosymbolic AI, Category Theory, Monad, Probability Theory, Fuzzy Logic, Semantics

## 1 Introduction

Neurosymbolic integration is a rapidly developing branch of AI. In the past, numerous heterogeneous approaches have emerged, each with its own code base. Van Krieken et al. (2024) introduces ULLER, a unified neurosymbolic library that aspires to play for neurosymbolic systems the role that TensorFlow and PyTorch play for deep-learning workflows. Their theoretical core is the concept of a NeSy system: standard first-order logic enriched with neural components. In particular, formulas of the form

$$x := m(T_1, \ldots, T_n)F \qquad (T_i \text{ terms, } F \text{ a formula involving } x, m \text{ a neural model})$$

[01]University of Osnabrück, Osnabrück, Germany, d.schellhorn@uni-osnabrueck.de
[2]University of Osnabrück, Osnabrück, Germany, till.mossakowski@uni-osnabrueck.de

are used to integrate neural models $m$ into logical formulas. These formulas go beyond classical first-order logic. Instead, they perform computations that may return multiple values and typically involve non-determinism or probability distributions as illustrated in the following toy example in Van Krieken et al. (2024):

$$\begin{array}{l}\forall x \in \mathit{ImageData} \\ \quad (n_1 := \mathit{classify}(x.\mathit{im}_1) \\ \quad\quad (n_2 := \mathit{classify}(x.\mathit{im}_2) \\ \quad\quad\quad (n_1 + n_2 = x.\mathit{sum}) \ ))\end{array}$$

This classifies two images of digits and checks whether the sum of the resulting numbers is as specified in the dataset. The resulting (e.g. fuzzy or probabilistic) truth value can be used in a loss function. A shorthand notation for this is:

$$\forall x \in \mathit{ImageData}\big(n_1 := \mathit{classify}(x.\mathit{im}_1), n_2 := \mathit{classify}(x.\mathit{im}_2)\ (n_1 + n_2 = x.\mathit{sum})\ \big)$$

To simplify notation and to stress the relation to dynamic logic Harel et al. (2001); Mossakowski et al. (2010), we henceforth use the notation

$$\forall x \in \mathit{ImageData}\big[n_1 := \mathit{classify}(x.\mathit{im}_1)\big]\big[n_2 := \mathit{classify}(x.\mathit{im}_2)\big] n_1 + n_2 = x.\mathit{sum}$$

or shorthand

$$\forall x \in \mathit{ImageData}\big[n_1 := \mathit{classify}(x.\mathit{im}_1), n_2 := \mathit{classify}(x.\mathit{im}_2)\big] n_1 + n_2 = x.\mathit{sum}$$

While the notion of NeSy system in Van Krieken et al. (2024) is very powerful, it also has several shortcomings:

- There is no uniform inductive definition of truth, i.e. of the truth value of a sentence in an interpretation. Rather, the notion of NeSy system has the inductive interpretation function as a component, meaning that classical, probabilistic and fuzzy NeSy systems employ three different inductive definitions of truth. Parts of these definitions of truth are copied verbatim from one NeSy system to another, other parts need to be replaced. This duplication of semantic rules is not modular. By contrast, we aim at a truly uniform inductive definition of truth value that is independent of the NeSy system and hence can be reused for different NeSy systems, such that the NeSy system itself is a parameter of the inductive definition of truth.
- The case of continuous probability distributions (involving probability kernels or Markov kernels) is not covered faithfully, because in this case, measurable spaces are required to properly define the mentioned Markov kernels. However, measurable spaces are not considered in Van Krieken et al. (2024) and probability measures are confused with density functions in the monadic formula of the probabilistic semantics.
- The treatment of logical connectives is not uniform across NeSy systems, i.e. the different sets of connectives are not considered as instances of a common abstract (algebraic) notion. Also, quantifiers in probabilistic semantics are defined using possibly infinite products, without requiring a suitable order structure on domains and without discussing convergence.

- The high-level concepts of semantics and computation are not properly separated. Computation (namely sampling) is mixed into the semantics at least in two places. The first one in the classical semantics, where the possibly multi-valued arg max can only be properly evaluated using sampling. We conceptualise the arg max differently as a transition *between* semantics. The second time is in "Sampling Semantics", which is not really semantics but computation (sampling).

We argue that ULLER is conceptually robust and show that a monadic formulation resolves all of the foregoing issues. In particular, ULLER formulas of the form

$$[x := m(T_1, \ldots, T_n)]F \qquad (T_i \text{ terms, } F \text{ a formula involving } x, m \text{ a neural model})$$

can be modelled using Moggi's notion of computational monad Moggi (1991), which has been introduced to model side effects in (functional) programming. Although monads originate in category theory, we first present them using a set-theoretic approach that does not involve any category theory. The generalisation to an arbitrary category, which is needed for continuous probabilities and some aspects of infinite domains comes only in later sections. We call our categorical approach "monadic ULLER", "modular ULLER", or simply "*mULLER*"[1].

This paper is organised as follows. Section 2 introduces NeSy frameworks and their algebraic prerequisites without use of category theory. Based on that, section 3 introduces the syntax and semantics of mULLER. Section 4 discusses several examples of set-based NeSy frameworks, including classical, fuzzy, and probabilistic ones. Section 5 discusses translations between NeSy systems. Section 6 generalises set-based NeSy frameworks to categorical NeSy frameworks, which is needed for continuous probability distributions and infinite domains. Section 7 defines the categorical semantics of mULLER and provides examples of categorical NeSy systems. Section 8 discusses related work, and section 9 outlines our implementation of mULLER in Python and Haskell. Finally, section 10 concludes the paper and outlines future work. Appendix A contains a brief introduction to the needed concepts of category theory.

## 2 Set-Based NeSy Frameworks

A neurosymbolic framework (NeSy framework) is a general framework for NeSy systems combining neural models with symbolic logic, and it provides the semantic background for the specific logic involved. Examples are the logics behind DeepProbLog Manhaeve et al. (2021) or Logic Tensor Networks Badreddine et al. (2022).

The notion of NeSy framework is not defined in Van Krieken et al. (2024). Rather, they define a notion of NeSy system, which is quite ad-hoc, because it simultaneously makes two choices: (1) a choice of a particular interpretation with functions, predicates and neural models (e.g., probabilities for traffic lights, or neural networks learning addition of digit images), and (2) a choice of semantic rules for interpreting terms (which also involves a choice of the logic, e.g. classical or probabilistic or fuzzy). This causes semantic rule duplication.

[1] *muller*: noun, a heavy tool of stone or iron used to grind and mix material.

Our notion of NeSy framework provides a means to disentangle these two choices. Moreover, our approach makes semantic rules independent not only of particular interpretations, but also of the choice of logic (classical, probabilistic, or fuzzy).

In the sequel, we first introduce some background on monads, which are the key concept of our approach, and on the algebraic structure needed to model the space of truth values. Then we go on to define the notions of NeSy framework and NeSy system.

## 2.1 Set-Based Monads

We interpret formulas $[x := m(T_1, \ldots, T_n)]F$ involving neural models $m$ (= certain computations) using Moggi's notion of computational monad Moggi (1991) in the form of Kleisli triples:

**Definition 1.** *A **Kleisli triple*** (monad) $(\mathcal{T}, \eta, (-)^*)$ *consists of:*

- *A mapping $\mathcal{T}$, mapping sets $X$ to sets $\mathcal{T}X$ (of computations with values from $X$),*
- *A family of functions: $\eta_X : X \to \mathcal{T}X$ for each set $X$ (construing a value $a \in X$ as stateless computation $\eta_X(a) \in \mathcal{T}X$),*
- *A function that assigns to each function $f : X \to \mathcal{T}Y$ a function $f^* : \mathcal{T}X \to \mathcal{T}Y$ (called the* Kleisli extension*), needed for sequential composition of computations,*

*such that the following axioms hold:*

1. $(\eta_X)^* = \mathrm{id}_{\mathcal{T}X}$,
2. $f^* \circ \eta_X = f$ *for all* $f : X \to \mathcal{T}Y$,
3. $(g^* \circ f)^* = g^* \circ f^*$ *for all* $f : X \to \mathcal{T}Y$ *and* $g : Y \to \mathcal{T}Z$.

Given computations $ma : \mathcal{T}A$ and $mb(x) : \mathcal{T}B$, we can compose them to $(\lambda x : A.mb(x))^*(ma)$[2] of type $\mathcal{T}B$. In Haskell's do-notation, this is written as **do** $x \leftarrow ma$; $mb(x)$. Categorically, the Kleisli morphism $f : A \to \mathcal{T}B$ is lifted to $f^* : \mathcal{T}A \to \mathcal{T}B$. For $ma \in \mathrm{Ob}(\mathcal{T}A)$ and $mb := f$, we then have $f^*(ma) = mb^*(ma)$, also written as $ma >>= f$ or $ma >>= mb$ in fish notation.[3]

**Example 1.** ***Non-empty Powerset monad*** $\mathcal{P}_{\neq\emptyset}$ *For a set $X$:*

$$\mathcal{P}_{\neq\emptyset}X := \{A \subseteq X \mid A \neq \emptyset\} \quad \textit{(non-empty subsets of } X\textit{)},$$

$$\eta_X(x) := \{x\}, \quad f^*(A) := \bigcup_{a \in A} f(a) \quad \big(f : X \to \mathcal{P}_{\neq\emptyset}Y, A \in \mathcal{P}_{\neq\emptyset}X\big).$$

[2] In this lambda notation the variable x of type A is sent to mb(x) of type $\mathcal{T}B$.

[3] See https://ncatlab.org/nlab/show/monad+%28in+computer+science%29#DoNotation for a comprehensive comparison of all different notations.

**Example 2.** ***Probability distribution monad (Kleisli triple)*** $\mathcal{D}$.[4]

$$\mathcal{D}X := \left\{ \rho : X \to [0,1] \text{ finitely supported} \;\middle|\; \sum_{x \in X} \rho(x) = 1 \right\} \quad \text{(prob. distributions on } X\text{)},$$

$$\eta_X(x) := \delta_x,\ \delta_x(y) = \begin{cases} 1, & x = y \\ 0, & x \neq y \end{cases} \qquad f^*(\rho)(y) := \sum_{x \in X} f(x)(y) \cdot \rho(x) \qquad \big(f : X \to \mathcal{D}Y,\ \rho \in \mathcal{D}X\big).$$

*$\delta_x$ is the probability distribution that assigns all probability mass to $x$. $f^*(\rho)$ corresponds to a two-level random process: first $x$ is drawn from $\rho$, then $y$ is drawn from $f(x)$. This results in a marginal distribution of $Y$ for the joint distribution $\hat{\rho}(x,y) := f(x)(y) \cdot \rho(x)$.* ***do*** *$x \leftarrow ma$; $mb(x)$ can be interpreted as "sample $x$ from $ma$ and then proceed with $mb(x)$".*

## 2.2 Double Monoid Bounded Lattices (2Mon-BLat)

We need an algebraic structure to model the space of truth values. We weaken the notion of BL algebra of Hájek (1998) from fuzzy logic as follows:

**Definition 2.** *A* ***double monoid bounded lattice (2Mon-BLat)*** $\mathcal{R}$ *is a tuple*

$$\big(S,\ \leq,\ \bot,\ \top,\ \otimes,\ 0,\ 1,\ \oplus,\ \to,\ \neg\big)$$

*in which $S$ is a set, $\mathcal{L} := (S, \leq)$ a bounded lattice, while $\bot \in S$ and $\top \in S$ are its bottom and top elements.*[5] *Also $(S, \otimes, 1)$ and $(S, \oplus, 0)$ are monoids, $\to$ is a map $S \times S \to S$, and $\neg$ is a map $S \to S$.*

Also, we want to allow different aggregation operations other than infinite meet and join to cover the quantifiers of Logic Tensor Networks Badreddine et al. (2022), motivating the following definition:[6]

**Definition 3.** *An* ***aggregated 2Mon-BLat (aggr-2Mon-BLat)*** *has for each set $X$ two order-preserving maps:*

$$\mathrm{aggr}^{\forall}_X, \mathrm{aggr}^{\exists}_X :\ \mathcal{L}^X \longrightarrow \mathcal{L}.$$

*In case of a complete lattice, $\mathrm{aggr}^{\forall}_X$ can be chosen as meet $\bigwedge_X$ and $\mathrm{aggr}^{\exists}_X$ as join $\bigvee_X$.*

[4] Note that the sums below are only finite if one excludes all the zero addenda.

[5] In many cases we have $\bot$ is neutral element for $\oplus$ and $\top$ is neutral element for $\otimes$, for example inside of the unit interval $[0,1]$. In some cases, like Gödel logic, we even have $\oplus = \vee$ and $\otimes = \wedge$. Also, $\to$ is normally chosen as right adjoint to $\otimes$ or as $x \to y := \neg x \oplus y$. In the first case $\neg$ can be defined as implication to zero, in the second one it is defined a priori. Check Table 3 for details.

[6] This is inspired by the notion of *aggregated functions* in Badreddine and Spranger (2021).

### 2.3 Definition of Set-Based NeSy Framework

Given some basic notion of truth $\Omega$, our NeSy systems work on the monadic space of truth values $\mathcal{T}\Omega$, which is required to be an aggregated 2Mon-BLat. If $\mathcal{T}$ is the identity monad, $\mathcal{T}\{0,1\}$ is just the two-element set $\{0,1\}$ of classical truth values. If $\mathcal{T}$ is the distribution monad, $\mathcal{T}\{0,1\}$ is isomorphic to the unit interval $[0,1]$, regarded as the space of probabilistic or fuzzy truth values.

**Definition 4.** *A **NeSy framework** $\mathcal{F} = (\mathcal{T}, \mathcal{R})$ consists of*

1. *a* monad $\mathcal{T}$*,*
2. *an* aggr-2Mon-BLat $\mathcal{R}$ *on* $\mathcal{T}\Omega$ *for some set* $\Omega$*.*

*Here, $\Omega$ is a set acting as truth basis,*[7] *and $\mathcal{T}\Omega$ is the monadic space of truth values.*

Examples are given in Table 1 and discussed in more detail in section 4. Note that further examples arise by varying the 2Mon-BLat $\mathcal{R}$ on $[0,1]$. Examples requiring category theory are given in Table 4.

**Table 1.** NeSy Framework Examples (set-based)

| **Logic/Theory** | $\mathcal{T}$ | $\Omega$ | $\mathcal{T}\Omega$ | $\mathcal{R}$ | **Subsection** |
|---|---|---|---|---|---|
| Classical | Identity | $\{0,1\}$ | $\{0,1\}$ | Boolean Alg. | 4.1 |
| Three-valued LP | Powerset $\mathcal{P}_{\neq\emptyset}$ | $\{0,1\}$ | $\{0,B,1\}$ | Kleene/Priest Alg. | 4.1 |
| Distributional | Distribution $\mathcal{D}$ | $\{0,1\}$ | $[0,1]$ | Product BL–Alg. | 4.2 |
| Finitary LTN$_p$ | Distribution $\mathcal{D}$ | $\{0,1\}$ | $[0,1]$ | Product SBL-Alg. | 4.3 |
| Classical Fuzzy | Identity | $[0,1]$ | $[0,1]$ | Classical BL–Alg. | – |

**Proposition 1.** *Assume that we can lift lattices along $\mathcal{T}$, i.e. for a lattice structure on $X$, we can construct a lattice on $\mathcal{T}X$ such that $\eta$ preservers $\bot$ and $\top$.*
*If $\mathcal{R} = (\Omega, \leq, \bot, \top, \otimes, 0, 1, \oplus, \to, \neg)$ is a* 2Mon-BLat*, then $\mathcal{T}\Omega$ is so, too, in a canonical way.*

**Proof.** We define $(\mathcal{T}\Omega, \leq', \bot', \top', \otimes', 0, 1, \oplus', \to', \neg')$ as follows:

- $\leq'$ is the lifting of $\leq$,
- $\bot' := \eta(\bot), \qquad \top' := \eta(\top)$,
- $0' := \eta(0)$,
- $1' := \eta(1)$,
- $a \otimes' b := \mathbf{do}\ \ x \leftarrow a; y \leftarrow b;\ \ \eta(x \otimes y)$,
- $a \oplus' b := \mathbf{do}\ \ x \leftarrow a; y \leftarrow b;\ \ \eta(x \oplus y)$,

[7] Similar to the basis of a vector space.

- $a \to' b := \mathbf{do}\ \ x \leftarrow a; y \leftarrow b;\ \ \eta(x \to y)$,
- $\neg' a := \mathbf{do}\ \ x \leftarrow a;\ \ \eta(\neg x)$.

Associativity and unit laws of $\otimes'$ and $\oplus'$ follows from the corresponding properties of the monoids for $\otimes$ and $\oplus$ and those of the monad. □

In particular, this means that the canonical Boolean algebra structure on $\{T, F\}$ can be lifted to $\mathcal{T}\{T, F\}$ for any monad $\mathcal{T}$.

# 3 Syntax and Semantics of mULLER

## 3.1 Syntax of First-Order Logic

The ULLER language of Van Krieken et al. (2024) features computational function symbols that can be realised e.g. by neural networks. In a similar spirit, we here add computational predicate symbols, which are also realised by neural networks, for example in Logic Tensor Networks Badreddine et al. (2022).

**Definition 5.** *A **NeSy signature** $\Sigma$ consists of*

- *a set $S$ of* sorts *of $\Sigma$,*
- *two disjoint sets* Pred, MPred *of* predicate symbols *and* computational predicate symbols *of form $p : s_1, \ldots, s_n$, where $p$ is a name and each $s_i \in S$ a sort,*
- *two disjoint sets* Func, mFunc *of* function symbols *and* computational function symbols *of form $f : s_1, \ldots, s_n \to s$, , where $f$ is a name and $s, s_i \in S$ are sorts.*

*(Computational) predicate symbols with no arguments are called (computational) propositional symbols* (Prps *and* mPrps *respectively*). *Function symbols with one argument are called properties* (Prop), *those with none are called constants* (Const).

Concerning syntax, we largely follow the definitions given in Van Krieken et al. (2024). That is, given a *signature* $\Sigma$ of non-logical symbols and set of variables V, we can define the syntax of first-order logic (FOL) formulas over $\Sigma$ and V as a context-free grammar:

**Terms:**

$T ::= x : s$ $\quad [x \in \mathrm{V}, s \in S]$

$T ::= c \mid T.\mathsf{prop} \mid f(T, \ldots, T)$ $\quad [c \in \mathrm{Const}, \mathsf{prop} \in \mathrm{Prop}, f \in \mathrm{Func}]$

**Atomic Formulas:**

$F ::= R \mid P(T, \ldots, T)$ $\quad [R \in \mathrm{Prps}, P \in \mathrm{Pred}]$

$F ::= N \mid M(T, \ldots, T)$ $\quad [N \in \mathrm{mPrps}, M \in \mathrm{MPred}]$

**Compound Formulas:**

$F ::= \bot \mid \top \mid F \to\ F \mid \neg F \mid F \| F \mid F \& F \mid (F)$

$F ::= \exists x : s\ (F) \mid \forall x : s\ (F)$ $\quad [x \in \mathrm{V}, s \in S]$

$F ::= [x := m(T, \ldots, T)]F$ $\quad [m \in \mathrm{mFunc}]$

### 3.2 Tarskian Semantics

**Definition 6.** *A **NeSy interpretation** $\mathcal{I}$ on $\Sigma$ of $(\mathcal{T},\mathcal{R})$, for a* NeSy signature $\Sigma$ *and a* NeSy framework $(\mathcal{T},\mathcal{R})$*, is given by*

- *a set $\mathcal{I}(s)$ for every sort $s$,*
- *a function $\mathcal{I}(f):\mathcal{I}(s_1)\times\ldots\times\mathcal{I}(s_n)\to\mathcal{I}(s)$ for every (normal) function symbol $f:s_1,\ldots,s_n\to s\in$* Func*,*
- *a function $\mathcal{I}(m):\mathcal{I}(s_1)\times\ldots\times\mathcal{I}(s_n)\to\mathcal{T}(\mathcal{I}(s))$ for every computational function symbol $m:s_1,\ldots,s_n\to s\in$* mFunc*,*
- *a function $\mathcal{I}(P):\mathcal{I}(s_1)\times\ldots\times\mathcal{I}(s_n)\to\Omega$ for every predicate symbol $P:s_1,\ldots,s_n\in$* Pred*,*
- *and a function $\mathcal{I}(M):\mathcal{I}(s_1)\times\ldots\times\mathcal{I}(s_n)\to\mathcal{T}\Omega$ for every computational predicate symbol $M:s_1,\ldots,s_n\in$* MPred*.*

**Definition 7.** *A **NeSy system** can be defined as a triple $(\mathcal{T},\mathcal{R},\mathcal{I})$, where $(\mathcal{T},\mathcal{R})$ is a* NeSy framework *and $\mathcal{I}$ a* NeSy interpretation *of that same* NeSy framework*.*

*Note that in this notation the category $\mathcal{C}$ on which the monad $\mathcal{T}$ is defined, the underlying set of basic truth values $\Omega$, with which the aggr-2Mon-BLat $\mathcal{R}$ is defined, and the signature $\Sigma$ on which the interpretation $\mathcal{I}$ is defined, are suppressed since they are implicit.*

Compared to Van Krieken et al. (2024), we have added computational predicate symbols, because LTN and other NeSy frameworks use these. However note that for probabilistic logic and weighted model counting, ULLER makes independence assumptions due to the nature of its notion of interpretation.[8] While computational function symbols enable the use of conditional probabilities, computational predicate symbols are always independent of each other. Hence, ULLER supports a certain combination of probabilistic and fuzzy logic, and so do LTNs. For details, see section 7.1.

Also, we have dropped uniformity of the notion of interpretation—it now becomes dependent on the monad at hand. This is necessary for faithfully distinguishing finitely supported and continuous probability distributions and for dealing with LTN-style quantification on infinite domains. Still, computational symbols can be realised by neural networks in all of these cases. However, the details of the mapping from neural networks to interpretations of computational symbols differ.

In Van Krieken et al. (2024), based on an interpretation, the notion of NeSy system provides a Tarskian inductive definition $[\![\cdot]\!]$ of the semantics of formulas and thus it implicitly also defines the semantics of the logical symbols. The drawback of this approach is that the Tarskian semantics $[\![\cdot]\!]$ is inherently tied to the specific NeSy system.

We can modularise matters here, because we first give a semantics of the logical symbols via a NeSy framework, and based on that, the interpretation provides the semantics of the non-logical

[8] As already hinted at in the original ULLER paper Van Krieken et al. (2024) and looked at in more detail in van Krieken et al. (2024) and van Krieken et al. (2025), the independence assumption can prevent NeSy predictors from correctly modelling uncertainty.

symbols. Hence, the inductive definition of the Tarskian semantics $[\![\cdot]\!]$ needs to be given only once, and this definition holds across all NeSy frameworks and systems.

**Definition 8.** *The **Tarskian semantics** $[\![\cdot]\!]$ of a* NeSy system $(\mathcal{T}, \mathcal{R}, \mathcal{I})$ *is given by:*

$$\textbf{\textit{Formulas:}}\ [\![F]\!]_{\mathcal{I}} : \mathcal{V}_F \to \mathcal{T}\Omega, \quad \textbf{\textit{Terms:}}\ [\![T]\!]_{\mathcal{I}} : \mathcal{V}_T \to \mathcal{I}(s_T).$$

**Table 2.** Inductive definition of the Tarskian semantics

| **Syntax** | **Set Semantics** $[\![\cdot]\!]_{\mathcal{I},\nu}$ |
|---|---|
| **Terms** | |
| $[\![x : s]\!]$ | $\nu(x)$ |
| $[\![c]\!],\ [\![T.\mathsf{prop}]\!],\ [\![f(\vec{T})]\!]$ | $\mathcal{I}(c),\ \mathcal{I}(\mathsf{prop})([\![T]\!]),\ \mathcal{I}(f)\big([\![\vec{T}]\!]\big)$ |
| **Atomic formulas** | |
| $[\![P]\!],\ [\![R(\vec{T})]\!]$ | $\eta_\Omega(\mathcal{I}(P)),\ \eta_\Omega\big(\mathcal{I}(R)([\![\vec{T}]\!])\big)$ |
| $[\![N]\!],\ [\![M(\vec{T})]\!]$ | $\mathcal{I}(N),\ \mathcal{I}(M)\big([\![\vec{T}]\!]\big)$ |
| **Compound formulas** | |
| $[\![\bot]\!],\ [\![\top]\!]$ | $\bot_{\mathcal{R}},\ \top_{\mathcal{R}}$ |
| $[\![F \to G]\!],\ [\![\neg F]\!]$ | $[\![F]\!] \to_{\mathcal{R}} [\![G]\!],\ \neg_{\mathcal{R}}[\![F]\!]$ |
| $[\![F \| G]\!],\ [\![F \& G]\!]$ | $[\![F]\!] \oplus_{\mathcal{R}} [\![G]\!],\ [\![F]\!] \otimes_{\mathcal{R}} [\![G]\!]$ |
| $[\![\exists x{:}s\, F]\!],\ [\![\forall x{:}s\, F]\!]$ | $\mathrm{aggr}^{\exists}_{\mathcal{I}(s)}(\lambda a.[\![F]\!]_{\nu[x\mapsto a]}),\ \mathrm{aggr}^{\forall}_{\mathcal{I}(s)}(\lambda a.[\![F]\!]_{\nu[x\mapsto a]})$ |
| $[\![[x := m(\vec{T})]F]\!]$ | $\textbf{do}\ \ a \leftarrow \mathcal{I}(m)([\![\vec{T}]\!]);\ [\![F]\!]_{\nu[x\mapsto a]}$ |

*Here, we define $\mathcal{V}_T := \prod_{x:t\in\Gamma_T} \mathcal{I}(t)$, where $\Gamma_T$ is the context of $T$ and $s_T$ is the (unique) sort of the term $T$. Analogously $\mathcal{V}_F := \prod_{x:t\in\Gamma_F} \mathcal{I}(t)$. $\vec{T}$ stands for $T_1, \ldots, T_n$. We work with variable valuations $\nu \in \mathcal{V}_T$ (or $\nu \in \mathcal{V}_F$) in local (term or formula) contexts, noting that elements of $\prod_{x:t\in\Gamma_T} \mathcal{I}(t)$ map variables $x : t$ to values in $\mathcal{I}(t)$. We write $[\![T]\!]_{\mathcal{I},\nu} = [\![T]\!]_{\mathcal{I}}(\nu)$ and $[\![F]\!]_{\mathcal{I},\nu} = [\![F]\!]_{\mathcal{I}}(\nu)$. That said, we mostly omit $\mathcal{I}$ and $\nu$ if clear from the context.*

## 4 Examples of Set-Based Semantics

In the sequel, we will discuss some NeSy frameworks in more detail and spell out how the semantic rules look when instantiated. We often implicitly define parts of the 2Mon-BLat through the semantic rules. E.g. the $\mathrm{aggr}^{\exists}$ and $\mathrm{aggr}^{\forall}$ functions are implicitly defined by listing semantic rules for the quantifiers.

### 4.1 Classical and Three-valued Semantics

Classical semantics is simply given by the identity monad and the Boolean algebra on $\Omega = \{0, 1\}$, which results in classical first-order logic.

Our classical semantics is deterministic, while Van Krieken et al. (2024) use probability distributions, causing the need for selection of values with highest probability, done via argmax. We will model this as NeSy transformation in section 5 and need a non-deterministic NeSy framework as target of this transformation. The (non-empty) powerset monad models non-deterministic computations, cf. multialgebras Walicki and Meldal (1994). These result in non-deterministic truth values as in:

*Logic of Paradox Semantics* For the Logic of Paradox Priest (2008) with the non-empty powerset monad $\mathcal{P}_{\neq\emptyset}$, we have $\mathcal{T} = \mathcal{P}_{\neq\emptyset}$, $\Omega = \{0, 1\}$ (equivalently $\{F, T\}$), and $\mathcal{T}\Omega = \mathcal{P}_{\neq\emptyset}(\{0, 1\}) = \{\{0\}, \{1\}, \{0, 1\}\}$. The three truth values correspond to: $\{0\} \equiv F$ (false only), $\{1\} \equiv T$ (true only), and $\{0, 1\} \equiv B$ (both true and false). Following the uniform Tarskian semantics:

$$[\![[x := m(\vec{T})]F]\!] := \bigcup_{a \in \mathcal{I}(m)([\![\vec{T}]\!])} [\![F]\!]_{\nu[x \mapsto a]} \tag{1}$$

$$[\![\exists x{:}s\ F]\!] := \sup_{a \in \mathcal{I}(s)} [\![F]\!]_{\nu[x \mapsto a]}, \quad [\![\forall x{:}s\ F]\!] := \inf_{a \in \mathcal{I}(s)} [\![F]\!]_{\nu[x \mapsto a]} \tag{2}$$

$$[\![F \| G]\!] := \max([\![F]\!], [\![G]\!]), \quad [\![F \& G]\!] := \min([\![F]\!], [\![G]\!]) \tag{3}$$

$$[\![F \to G]\!] := \max([\![\neg F]\!], [\![G]\!]), \quad [\![\neg F]\!] := \begin{cases} \{0, 1\} & \text{if } [\![F]\!] = \{0, 1\} \\ (\{0, 1\} \setminus [\![F]\!]) & \text{else} \end{cases} \tag{4}$$

$$[\![\bot]\!] := \{0\}, \quad [\![\top]\!] := \{1\} \tag{5}$$

where the operations implement Priest's Logic of Paradox with the lattice ordering $\{0\} <_{\text{LP}} \{0, 1\} <_{\text{LP}} \{1\}$ (i.e., F < B < T).

### 4.2 Distributional Semantics

The distributional semantics corresponds to the third row in Table 1, where we use the distribution monad $\mathcal{D}$ over the classical truth basis $\{0, 1\}$, yielding the truth space $[0, 1]$ equipped with a Product BL-algebra structure. This framework provides the semantic foundation for probabilistic logic programming systems and neural-symbolic approaches that work with probability distributions over truth values. In this setting, computational predicates and function symbols return probability distributions rather than deterministic values. We need to restrict interpretations to finite domains. Finite quantification just iterates conjunction or disjunction. For infinite quantifiers, check section 7.1.

$$[\![[x := m(\vec{T})]F]\!] := \sum_{a \in \mathcal{I}(s_m)} [\![F]\!]_{\nu[x \mapsto a]} \cdot \rho_m(a \mid \vec{T}) \tag{6}$$

$$[\![\exists x{:}s\ F]\!] := 1 - \prod_{a \in \mathcal{I}(s)} (1 - [\![F]\!]_{\nu[x \mapsto a]}), \quad [\![\forall x{:}s\ F]\!] := \prod_{a \in \mathcal{I}(s)} [\![F]\!]_{\nu[x \mapsto a]} \tag{7}$$

$$[\![F \| G]\!] := [\![F]\!] + [\![G]\!] - [\![F]\!] \cdot [\![G]\!], \quad [\![F \& G]\!] := [\![F]\!] \cdot [\![G]\!] \tag{8}$$

$$[\![F \to G]\!] := \max\big(1, [\![G]\!]/[\![F]\!]\big), \quad [\![\neg F]\!] := 1 - [\![F]\!] \tag{9}$$

$$[\![\bot]\!] := 0, \quad [\![\top]\!] := 1. \tag{10}$$

### 4.3 Finitary LTN$_p$ Semantics

The finitary Logic Tensor Networks (LTN) semantics corresponds to the fourth row in Table 1, employing the distribution monad $\mathcal{D}$ over the classical truth basis $\{0, 1\}$ with the truth space $[0, 1]$ equipped with a Product Real Logic structure, what we call the S-Product Algebra in Table 3. It is the same as the Product Algebra in the distributional semantics, only that we use S-Implication (strong implication) instead of R-Implication (residual implication). Moreover, aggregation differs as well: following Badreddine et al. (2022), quantification is performed using $p$-norms, where the parameter $p$ controls the "softness" of the logical operations. For existential quantification, we compute the $p$-norm of truth values, while for universal quantification, we use the dual formulation $1 - \|1 - \cdot\|_p$:

$$[\![\exists x{:}s\ F]\!] := \Big(\frac{1}{|\mathcal{I}_s|} \sum_{a \in \mathcal{I}_s} [\![F]\!]^p_{\nu[x \mapsto a]}\Big)^{1/p}, \tag{11}$$

$$[\![\forall x{:}s\ F]\!] := 1 - \Big(\frac{1}{|\mathcal{I}_s|} \sum_{a \in \mathcal{I}_s} (1 - [\![F]\!]_{\nu[x \mapsto a]})^{\,p}\Big)^{1/p}, \tag{12}$$

$$[\![F \to G]\!] := 1 - [\![F]\!] + [\![F]\!] \cdot [\![G]\!]. \tag{13}$$

This is however only possible for finite domains. For the infinite case and additional quantifier variants, we refer to section 7.2.

### 4.4 Sampling "Semantics"

While Van Krieken et al. (2024) have introduced a sampling semantics, we think that the semantics should define probabilities, while an implementation can work with e.g. Monte Carlo sampling in order to obtain an approximation that is easier to implement (and, in the case of quantification over infinite domains, unavoidable). Hence, we do not discuss sampling semantics here. But we expect that a Monte Carlo convergence theorem can be stated and proved.

## 5 NeSy Transformations

The original ULLER paper Van Krieken et al. (2024) uses a uniform notion of interpretation. This leads to the problem that classical semantics needs to extract values with maximal probability

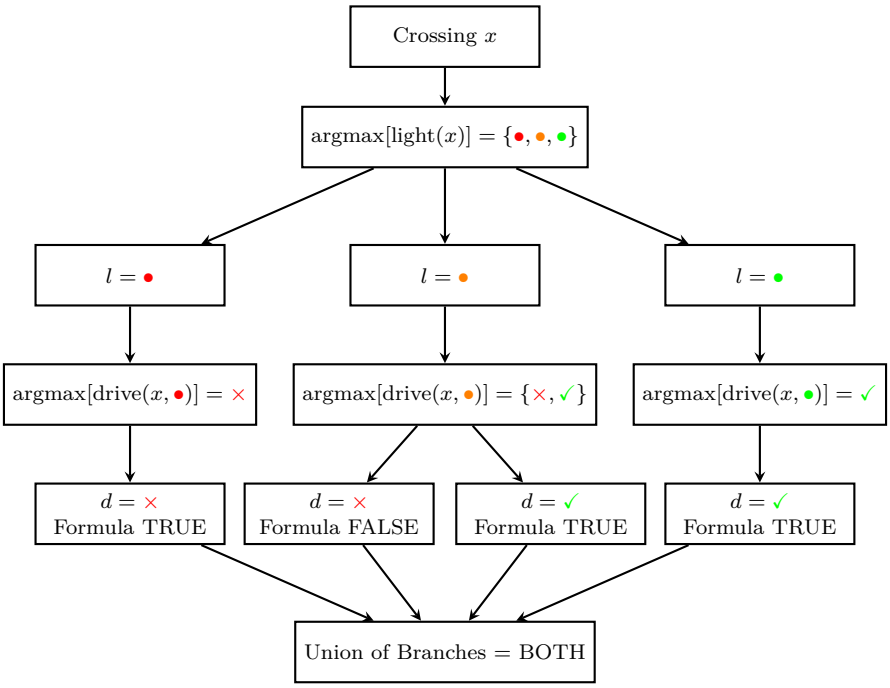



**Figure 1.** Argmax transformation flowchart for traffic light example from Van Krieken et al. (2024) showing how the Logic of Paradox handles argmax operations across different light colours. The diagram illustrates the branching logic when argmax produces multiple values (tie in amber case) and how results are combined by the universal quantifier in the LP algebra. In this example, the traffic lights are uniformly distributed, and the probability of continuing driving (green check mark) is 0.1, 0.5, and 0.9 for red, amber, and green light, respectively. The formula is $\forall x$:Crossing $[l := \text{light}(x),\ d := \text{drive}(x, l)](d \neq \checkmark, l = \bullet)$, that is "For every crossing, only continue driving if the light is green".

(using arg max) from a distribution, which is not possible if there is a tie[9]. Here, we propose an alternative way of dealing with this problem: namely, using a NeSy transformation, we can move e.g. from an interpretation in a probabilistic NeSy framework to one in a classical framework. Dealing with ties can be done using a non-deterministic semantics.

**Definition 9.** *A **NeSy transformation** $\alpha : \mathcal{F} \to \mathcal{F}'$ between two NeSy frameworks is a family of functions*[10] $\alpha_\Sigma : \text{Intp}_\mathcal{F}(\Sigma) \to \text{Intp}_{\mathcal{F}'}(\Sigma)$. *Here, the* interpretation function $\text{Intp}_\mathcal{F} : \Sigma \mapsto \text{Intp}(\mathcal{F}, \Sigma)$ *sends a signature to the set of interpretations on $\mathcal{F}$ for that signature. We write $\alpha$ for $\alpha_\Sigma$ if $\Sigma$ is clear from context.*

[9] Or in case of an infinite distribution (see section 7.1)
[10] For those interested in category theory; this is in fact a natural transformation, hence the name.

*Argmax transformation: From distributional to non-deterministic semantics* For a given distributional interpretation $\mathcal{I}(m)$ of a computational function symbol $m$, we can define a non-deterministic interpretation $\alpha(\mathcal{I})(m)$ of $m$ by defining, where $s_m$ is the sort of $m$, and likewise, $M$ is a computational predicate symbol:

$$\alpha(\mathcal{I})(m) := \operatorname*{arg\,max}_{a \in \mathcal{I}(s_m)} \big[\mathcal{I}(m)(a)\big], \qquad \alpha(\mathcal{I})(M) := \operatorname*{arg\,max}_{b \in \Omega = \{0,1\}} \big[\mathcal{I}(M)(b)\big],$$

and for all other symbols set $\alpha(\mathcal{I}) := \mathcal{I}$. This definition is *not* possible in the general probabilistic case, because probability measures often return zero on *all* single values. It is also a non-deterministic interpretation since it returns a *set* of values instead of a single value. The resulting semantics is three-valued, as in section 4.1 and we apply it to the traffic light example from Van Krieken et al. (2024) in Fig. 1. This arg max transformation is just one example of many possible NeSy transformations.

Then, in the practical implementation of ULLER, we use random sampling (see section 4.4) over a uniform distribution to obtain a single value from the set of values $\alpha(\mathcal{I})(m)$. This gives a precise foundation for the use of arg max in the classical semantics in Van Krieken et al. (2024).

## 6 Categorical NeSy Frameworks

So far, in this paper, we have not made use of category theory. Indeed, we have introduced set-based notions of monad and of Double monoid bounded lattice. They do not rely on category theory, nor do the central definitions of mULLER, in particular, the notions of NeSy framework, of signature, interpretation and formula, nor the rules of the Tarskian semantics.

By contrast, in this and the next section, we will heavily make use of category theory. (The subsequent Sections 8 and 9 implicitly use the results of this section, but do not directly make uses of category theory. Reader not interested in category theory may skip to Section 8.) The main purpose of the use of category theory is the possibility to work with continuous probability distributions.[11] Another motivation is the need for quantification structures for infinite domains, which are common in first-order logic. In these cases, we need to work with structured objects and structure-preserving maps, like measurable spaces and measurable functions. Without organising such spaces and maps into a category, we still can use most of the set-theoretic rules of the semantics. However, we would need to restrict to structure-preserving maps in places like the definition of interpretation (Def. 6), and, more severely, would need to prove that the semantic rules in Def. 8 again yield structure-preserving maps. When using category theory, we can avoid such proofs and base our theory on a certain structure that is required for the involved categories.

That said, we still can use the set-theoretic semantics rules for convenience, also for categories of sets with structure[12], just because the categorical version of the rules tells us that the resulting maps will be structure-preserving. The only set-theoretic rules of the semantics that we cannot simply re-use are the rules for quantifiers. Here, the aggregation functions $\mathrm{aggr}^{\exists}$ and $\mathrm{aggr}^{\forall}$

[11] The lack of which is described as a major shortcoming of traditional NeSy systems by Smet et al. (2023).
[12] Technically, these set-based categories $\mathbf{C}$ are constructs in the sense of Adámek et al. (1990), which means they come with a faithful functor $U : \mathbf{C} \to \mathbf{Set}$. $U$ maps objects of $\mathbf{C}$ to their underlying sets.

generally need to make use of the extra structure that the "sets with structure" have, see e.g. the aggregation functions for probabilistic semantics defined in section 7.1 below need to make use of the probability distribution coming with the universe.

A brief introduction to category theory is given in Appendix A.

## 6.1 More (on) Monads

**Definition 10.** *A **Kleisli triple** or **monad** on a category $\mathcal{C}$ involves a mapping of objects $\mathcal{T} : \mathrm{Ob}(\mathcal{C}) \to \mathrm{Ob}(\mathcal{C})$, a family of morphisms $\eta_X : X \to \mathcal{T}X$ for each object $X$ in $\mathcal{C}$ (called the* unit*), and a function that assigns to each morphism $f : X \to \mathcal{T}Y$ a morphism $f^* : \mathcal{T}X \to \mathcal{T}Y$ such that the axioms hold as in Def. 1. Note that a set-based Kleisli triple is then a Kleisli triple on the category* **Set**.

**Definition 11.** Strong Kleisli triple. *A Kleisli triple $(\mathcal{T}, \eta, (-)^*)$ on a Cartesian category $\mathcal{C}$ is called **strong** if there is a natural transformation*

$$\mathcal{S}_{A,B} : A \times \mathcal{T}B \longrightarrow \mathcal{T}(A \times B)$$

*satisfying the naturality condition: for any morphisms $f : A \to A'$ and $g : B \to B'$,*

$$\mathcal{T}(f \times g) \circ \mathcal{S}_{A,B} = \mathcal{S}_{A',B'} \circ (f \times \mathcal{T}g),$$

*and such that*

$$\mathcal{S}_{A,B} \circ (\mathrm{id}_A \times \eta_B) = \eta_{A \times B}, \qquad \mathcal{S}_{A,B} \circ (\mathrm{id}_A \times f^*) = (id \times f)^* \circ \mathcal{S}_{A,C}.$$

**Example 3.** ***Identity monad $\mathcal{I}$ on* Set *(category of sets and functions).*** *For a set $X$:*

$$\mathcal{I}X := X \quad \textit{(identity functor)},$$

$$\eta_X(x) := x, \quad f^*(x) := f(x) \quad \big(f : X \to Y, x \in X\big).$$

**Example 4.** ***Powerset monad $\mathcal{P}$ on* Set *(category of sets and functions).*** *For a set $X$:*

$$\mathcal{P}X := \{A \subseteq X\} \quad \textit{(powerset of } X\textit{)},$$

$$\eta_X(x) := \{x\}, \quad f^*(A) := \bigcup_{a \in A} f(a) \quad \big(f : X \to \mathcal{P}Y, A \subseteq X\big).$$

**Example 5.** Sub-Distribution Monad $\mathcal{S}$. *The sub-distribution monad $\mathcal{S}$ is similar to the distribution monad $\mathcal{D}$ but it allows for finitely supported measures that do not sum up to 1, that means:*

$$\mathcal{S}X := \Big\{ \rho : X \to [0,1] \;\Big|\; \sum_{x \in X} \rho(x) \leq 1,\ \rho \textit{ countably additive and has finite support} \Big\},$$

$$\eta_X(x) := \delta_x, \delta_x(y) = \begin{cases} 1, & x = y \\ 0, & x \neq y \end{cases} \quad f^*(\rho)(y) := \sum_{x \in X} \rho(x) f(x)(y) \quad \big(f : X \to \mathcal{S}Y,\ \rho \in \mathcal{S}X\big).$$

**Example 6.** ***Giry monad $\mathcal{G}$ on* Meas*, the category of measurable spaces and maps.*** *For a measurable space $(X, \Sigma_X)$ and Dirac measure $\delta_x$ on $X$ for $x \in X$:*

$$\mathcal{G}(X, \Sigma_X) := \left\{ \rho : X \to [0,1] \;\middle|\; \rho(X) = 1,\ \rho \text{ countably additive} \right\} \quad \text{(prob. measures on } X\text{)},$$

$$\eta_{(X,\Sigma_X)}(x) := \delta_x, \quad \delta_x(A) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases}$$

$$f^*(\rho)(A) := \int_X f(x)(A) d\rho(x) \quad \big(f : X \to \mathcal{G}Y, A \subseteq Y \text{ measurable}\big).$$

*$\delta_x$ is the probability distribution that assigns all probability mass to $x$.*

**Example 7.** ***Infinite Giry monad $\mathcal{G}_\infty$ on* Meas*.*** *For a measurable space $(X, \Sigma_X)$:*

$$\mathcal{G}_\infty(X, \Sigma_X) := \left\{ \mu : \Sigma_X \to [-\infty, \infty] \mid \mu(\emptyset) = 0,\ \mu \text{ countably additive} \right\},$$

$$\eta_{(X,\Sigma_X)}(x) := \delta_x, \qquad \delta_x(A) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases},$$

$$f^*(\mu)(A) := \int_X f(x)(A)\, d\mu(x) \quad \big(f : X \to \mathcal{S}Y,\ A \subseteq Y \text{ measurable}\big).$$

*Here the integral is the Lebesgue–Stieltjes integral with respect to the extended signed measure $\mu$. Writing the Jordan decomposition $\mu = \mu^+ - \mu^-$ and using linearity of the integral, one checks that the monad laws hold; thus $\mathcal{S}$ extends the Giry monad by allowing negative and (possibly) infinite total mass.*

**Proposition 2.** and definition of the measure-space monad $\mathcal{M}$. *We can define a measure monad $\mathcal{M}$ as a monad $(\mathcal{T}, \eta, \mu)$*[13] *on the category* **Measr** *of measure spaces with $\eta, \mu$ being the unit and multiplication of the Giry monad $\mathcal{G}$ on* **Meas***. For $\rho$ being probability measures, we can define a probability-space monad $\mathcal{O}$ on the category* **Prob** *of probability spaces. The same construction can be applied to obtain an infinite measure-space monad $\mathcal{M}_\infty$ on* **Measr***.*

$$\begin{aligned} \mathcal{M}((X,\rho)) &:= (\mathcal{G}(X), \rho^\eta), \\ \rho^\eta &:= B \longmapsto \rho(\eta^{-1}(B)),\ \text{for } B \subseteq \mathcal{G}(X) \text{ measurable}, \\ \eta^{\mathcal{M}} &:= \eta, \quad \mu^{\mathcal{M}} := \mu. \end{aligned}$$

**Proof.** If $\rho$ is a probability measure, we know that $\rho^\eta(\mathcal{G}(X)) = \rho(\eta^{-1}(\mathcal{G}(X))) = \rho(X) = 1$, countable additivity follows alike. Now we only need to check whether $\eta^{\mathcal{M}}, \mu^{\mathcal{M}}$ are measure preserving, which follows for the unit by definition:

[13] Here we use the traditional category theoretical definition of monad as $(\mathcal{T}, \eta, \mu)$, where $\mu$ is monad multiplication. This is however equivalent to the definition as Kleisli triple.

$$\eta^{\mathcal{M}} : (X, \rho) \longrightarrow \big(\mathcal{G}(X),\, \rho^{\eta}\big)$$

$$\rho\big((\eta^{\mathcal{M}})^{-1}(B)\big) \;=\; \rho\big(\eta^{-1}(B)\big) \;=\; \rho^{\eta}(B).$$

Now, for the multiplication we have:

$$\mu^{\mathcal{M}} : \big(\mathcal{G}^2(X),\, \rho{\circ}\eta^{-1}{\circ}\eta_{\mathcal{G}}^{-1}\big) \longrightarrow \big(\mathcal{G}(X),\, \rho{\circ}\eta^{-1}\big)$$

since

$$M^2(X) \;=\; M\big(\big(\mathcal{G}(X), \rho^{\eta}\big)\big) \;=\; \big(\mathcal{G}^2(X),\, (\rho^{\eta})^{\eta}\big),$$

and that means we can write for any measurable set $A \subseteq \mathcal{G}^2(X)$:

$$(\rho^{\eta})^{\eta}(A) \;=\; \rho^{\eta}\big(\eta_{\mathcal{G}}^{-1}(A)\big) \;=\; \rho\big(\eta^{-1} \circ \eta_{\mathcal{G}}^{-1}(A)\big).$$

Therefore we show the measure-preserving property as follows

$$\rho{\circ}\eta{\circ}\eta_{\mathcal{G}}^{-1}\big(\mu^{-1}(B)\big) \;=\; \rho{\circ}\eta^{-1}\big(\eta_{\mathcal{G}}^{-1}(\mu^{-1}(B))\big) \;=\; \rho{\circ}\eta^{-1}(B),$$

because we know the following fact from the monad laws:

$$A = \eta_{\mathcal{G}}^{-1}\big(\mu^{-1}(B)\big) \iff A = \mu\big(\eta_{\mathcal{G}}(A)\big) = B.$$

## 6.2 The 2Mon-BLat Algebra: A Comprehensive Overview

In this subsection, we provide a comprehensive overview of the different algebraic structures that can serve as algebras on the truth space $\mathcal{T}\Omega$ in our neurosymbolic framework, along with their associated operations types and logical properties.

Table 3 provides a comprehensive comparison of the fundamental operations across different algebraic structures, showing how each algebra defines its basic operations.

**Table 3.** Overview of aggregated 2Mon-BLat

| **Algebra** | **Set** | $\bot$ | $\top$ | $\oplus$ | $\otimes$ | $\to$ | $\neg$ | $\text{aggr}^{\exists}$ |
|---|---|---|---|---|---|---|---|---|
| Boolean | $\{0,1\}$ | 0 | 1 | max | min | $I_B$ | $\neg_R$ | sup |
| $\text{LTN}_p$ | $[0,1]$ | 0 | 1 | $S_P$ | $T_P$ | $I_{SP}$ | $\neg_C$ | $\lVert\cdot\rVert_p$ |
| $\text{LTN}_q$ | $[0,1]$ | 0 | 1 | $S_P$ | $T_P$ | $I_{SP}$ | $\neg_C$ | $P\exists_q$ |
| Product | $[0,1]$ | 0 | 1 | $S_P$ | $T_P$ | $I_P$ | $\neg_R$ | $P\exists$ |
| S-Prod. | $[0,1]$ | 0 | 1 | $S_P$ | $T_P$ | $I_{SP}$ | $\neg_C$ | $P\exists$ |
| Priest | $\{F,B,T\}$ | $F$ | $T$ | max | min | $I_{KD}$ | $\neg_C$ | sup |

*t-conorms and t-norms ($\oplus$ and $\otimes$):*

- $S_P$ (Probabilistic sum): $xS_Py = x + y - xy$
- $T_P$ (Product): $xT_Py = xy$

*Implications ($\rightarrow$):*

- $I_P$ (Product/Goguen): $I_P(x,y) = \begin{cases} 1 & \text{if } x \leq y \\ y/x & \text{otherwise} \end{cases}$
- $I_B$ (Boolean): $I_B(x,y) = \begin{cases} 0 & \text{if } x = 1, y = 0 \\ 1 & \text{otherwise} \end{cases}$
- $I_S$ (General S-implication): $I_S(x,y) = \neg x \oplus y$
- $I_{KD}$ (Kleene-Dienes/Material): $I_{KD}(x,y) = \max(1-x, y)$
- $I_{SP}$ (S-Product): $I_{SP}(x,y) = 1 - x + xy$

*Negations ($\neg$):*

- $\neg_R$ (Residual): $\neg_R x = x \rightarrow 0$ (includes Heyting/intuitionistic negation)
- $\neg_C$ (Classical/1-Involutive): $\neg_C x = 1 - x$
- $\neg_V$ (0-Involutive): $\neg_V x = 0 - x = -x$

*Aggregations ($\text{aggr}^{\exists}$):*

- $P\exists$ (Infinitary Probabilistic Sum): $P\exists(x) = 1 - \exp\Big(\mathbb{E}_{a\sim\mu}\Big[\ln(1 - [\![F]\!]_{\nu[x\mapsto a]})\Big]\Big)$
- $P\exists_q$ ($(\mu,q)$-approximated $P\exists$): $P\exists_q(x) = 1 - \exp\Big(\mathbb{E}_{a\sim\mu}\Big[\ln\big(1 - [\![F]\!]_{\nu[x\mapsto a]}\big)^q\Big]^{\frac{1}{q}}\Big)$ for $1/2 \leq q \leq 1$, with $P\exists_q \rightarrow P\exists$ as $q \rightarrow 1$. Here $\mu$ can be any measure and it depends on the context, in $\text{LTN}_q$ it depends on the measure space of the sort of the quantified variable at hand.

The quantification aggregations employ logarithmic and exponential transforms because they provide the natural generalisation of the product (or probabilistic sum) to infinitary domains. While finite probabilistic sums can be computed directly using products, extending to infinite domains requires the use of expectations, and the logarithmic and exponential transforms enable this generalisation while preserving the essential structure of probabilistic aggregation.

## 6.3 Definition of Categorical NeSy Frameworks

**Definition 12.** *An **aggr-2Mon-BLat internal to a Cartesian category** $\mathcal{C}$ on an object $A$ in $\mathcal{C}$ consists of a lattice on $A$ internal[14] to $\mathcal{C}$, morphisms $\oplus, \otimes, \rightarrow : A \times A \rightarrow A$; $\bot, \top, 0, 1 : 1_{\mathcal{C}} \rightarrow A$ and for any objects $B$ and $C$ maps $\text{aggr}^{\forall}_{B,C}, \text{aggr}^{\exists}_{B,C} : \mathcal{C}(B \times C, A) \rightarrow \mathcal{C}(B, A)$, such that the axioms of Def. 2 hold when appropriately interpreted in $\mathcal{C}$[15].*

[14] For an explanation check https://ncatlab.org/nlab/show/internalization and https://ncatlab.org/nlab/show/monoid+in+a+monoidal+category, since this is out of scope for this paper.
[15] $1_{\mathcal{C}}$ is a terminal object.

Note that our categorical handling of aggregation differs from that in the set-theoretic setting. A full analogy to the set-theoretic case would require aggregation morphisms $A^X \to X$, which would need a Cartesian closed category. This requirement seems too strong for our purposes, since it is not met in many examples and only is required for interpreting higher-order logics. However, given a Cartesian closed category (like **Set**) with aggregation $aggr : A^X \to X$, we can define aggregation in the sense of Def. 12 as mapping $f : B \times C \to A$ to $B \xrightarrow{\Lambda(f)} A^C \xrightarrow{aggr} A$. Here, $\Lambda(f)$ is currying, defined as follows in **Set**: $\Lambda(f)(x)(y) = f(x, y)$. In the sequel, we will rely on this definition also for **Set**-based categories (constructs Adámek et al. (1990)) that are not Cartesian closed, noting that the definition works even if $A^C$ is just a set and not an object in the category. Hence, in examples, we will define aggregation mostly as in Def. 2, but in some cases, we make use of the structure of the object $C$.

**Definition 13.** *A **NeSy framework** $(\mathcal{T}, \mathcal{R})$ consists of*

1. *a **strong monad** $\mathcal{T}$ with strength $\mathcal{S}$ on a Cartesian category $\mathcal{C}$,*[16]
2. *an **aggr-2Mon-BLat** $\mathcal{R}$ in $\mathcal{C}$ on $\mathcal{T}\Omega$ for some object $\Omega$.*

*Here, $\Omega$ is a set acting as truth basis,*[17] *and $\mathcal{T}\Omega$ is the monadic space of truth values.*

Examples are given in Table 4 and their semantics are discussed in section 7. Note that further examples arise by varying the 2Mon-BLat $\mathcal{R}$ on $[0, 1]$.

**Table 4.** NeSy Framework Examples (categorical)

| **Logic/Theory** | $\mathcal{C}$ | $\mathcal{T}$ | $\Omega$ | $\mathcal{T}\Omega$ | $\mathcal{R}$ | **Sem.** |
|---|---|---|---|---|---|---|
| Simple Prob. | **Meas** | Giry $\mathcal{G}$ | $\{0, 1\}$ | $[0, 1]$ | Product BL–Alg. | §7.1 |
| Standard Borel | **BorelMeas** | Giry $\mathcal{G}\vert_{\mathbf{BorelMeas}}$ | $\{0, 1\}$ | $[0, 1]$ | Product BL–Alg. | §7.1 |
| Probabilistic | **Measr** | Measr-space $\mathcal{M}$ | $\{0, 1\}$ | $[0, 1]$ | Product BL–Alg. | §7.1 |
| Infinitary $\mathrm{LTN}_p$ | **Prob** | Prob-space $\mathcal{O}$ | $\{0, 1\}$ | $[0, 1]$ | Product SBL-Alg. | §7.2 |
| $\mathrm{STL}_r$ | **Measr** | $\infty$-Measr-space $\mathcal{M}_\infty$ | $\{1\}$ | $\overline{\mathbb{R}}$ | approx. $\mathbb{R}$ | §7.3 |

## 7 Categorical Semantics

The categorical notion of interpretation differs from the set-theoretic definition (Definition 6) only in that *sets* are replaced by *objects* in the category $\mathcal{C}$ and *functions* are replaced by *morphisms* in $\mathcal{C}$. This generalisation allows the framework to work in any category with suitable structure, not just the category of sets and functions.

**Definition 14.** Tarskian semantics $[\![\cdot]\!]$ of formulas. *Given a NeSy framework $(\mathcal{T}, \mathcal{R})$ and a NeSy interpretation $\mathcal{I}$ we can determine the interpretation morphisms:*

$$\textbf{\textit{Formulas:}}\ [\![F]\!]_{\mathcal{I}} : \mathcal{V}_F \to \mathcal{T}\Omega, \quad \textbf{\textit{Terms:}}\ [\![T]\!]_{\mathcal{I}} : \mathcal{V}_T \to \mathcal{I}(s_T) :$$

[16] Note that **Set** is a Cartesian closed category, and every monad on **Set** is strong.
[17] Similar to the basis of a vector space.

**Table 5.** Inductive definition of the Tarskian semantics

| **Syntax** | **Categorical Semantics** $[\![\cdot]\!]_{\mathcal{I}}$ | **Set Semantics** $[\![\cdot]\!]_{\mathcal{I},\nu}$ |
|---|---|---|
| **Terms** | | |
| $[\![x : s]\!]$ | $\mathrm{id}_{\mathcal{I}(s)}$ | $\nu_s(x)$ |
| $[\![c]\!]$ | $\mathcal{I}(c)$ | $\mathcal{I}(c)$ |
| $[\![T.\mathsf{prop}]\!]$ | $\mathcal{I}(\mathsf{prop}) \circ [\![T]\!]$ | $\mathcal{I}(\mathsf{prop})([\![T]\!])$ |
| $[\![f(\vec{T})]\!]$ | $\mathcal{I}(f) \circ \langle [\![\vec{T}]\!]_i \circ \pi_i \rangle_i$ | $\mathcal{I}(f)\big([\![\vec{T}]\!]\big)$ |
| **Atomic formulas** | | |
| $[\![P]\!]$ | $\eta_\Omega \circ \mathcal{I}(P)$ | $\eta_\Omega(\mathcal{I}(P))$ |
| $[\![N]\!]$ | $\mathcal{I}(N)$ | $\mathcal{I}(N)$ |
| $[\![R(\vec{T})]\!]$ | $\eta_\Omega \circ \mathcal{I}(R) \circ \langle [\![\vec{T}]\!]_i \circ \pi_i \rangle_i$ | $\eta_\Omega\big(\mathcal{I}(R)([\![\vec{T}]\!])\big)$ |
| $[\![M(\vec{T})]\!]$ | $\mathcal{I}(M) \circ \langle [\![\vec{T}]\!]_i \circ \pi_i \rangle_i$ | $\mathcal{I}(M)\big([\![\vec{T}]\!]\big)$ |
| **Compound formulas** | | |
| $[\![\top]\!], [\![\bot]\!]$ | $1_{\mathcal{R}}, 0_{\mathcal{R}}$ | $1_{\mathcal{R}}, 0_{\mathcal{R}}$ |
| $[\![\neg F]\!]$ | $\neg_{\mathcal{R}} \circ [\![F]\!]$ | $\neg_{\mathcal{R}}([\![F]\!])$ |
| $[\![F \to G]\!]$ | $\to_{\mathcal{R}} \circ \langle [\![F]\!] \circ \pi_F, [\![G]\!] \circ \pi_G \rangle$ | $[\![F]\!] \to_{\mathcal{R}} [\![G]\!]$ |
| $[\![F \| G]\!]$ | $\oplus_{\mathcal{R}} \circ \langle [\![F]\!] \circ \pi_F, [\![G]\!] \circ \pi_G \rangle$ | $[\![F]\!] \oplus_{\mathcal{R}} [\![G]\!]$ |
| $[\![F \& G]\!]$ | $\otimes_{\mathcal{R}} \circ \langle [\![F]\!] \circ \pi_F, [\![G]\!] \circ \pi_G \rangle$ | $[\![F]\!] \otimes_{\mathcal{R}} [\![G]\!]$ |
| $[\![\exists x{:}s\, F]\!]$ | $\mathrm{aggr}^{\exists}_{V_{F\setminus x:s},\mathcal{I}(s)}([\![F]\!])$ | $\mathrm{aggr}^{\exists}_{\mathcal{I}(s)}(\lambda a.[\![F]\!]_{\nu[x\mapsto a]})$ |
| $[\![\forall x{:}s\, F]\!]$ | $\mathrm{aggr}^{\forall}_{V_{F\setminus x:s},\mathcal{I}(s)}([\![F]\!])$ | $\mathrm{aggr}^{\forall}_{\mathcal{I}(s)}(\lambda a.[\![F]\!]_{\nu[x\mapsto a]})$ |
| $[\![[x := m(\vec{T})]F]\!]$ | $[\![F]\!]^* \circ \mathcal{S} \circ \big\langle \pi_{V_{F\setminus x:s}}, \mathcal{I}(m) \circ \langle [\![\vec{T}]\!]_i \circ \pi_i \rangle_i \big\rangle$ | **do** $a \leftarrow \mathcal{I}(m)([\![\vec{T}]\!])$; $[\![F]\!]_{\nu[x\mapsto a]}$ |

**Remark 1.** *We define* $\mathcal{V}_T := \prod_{x:t\in\Gamma_T} \mathcal{I}(t)$. *Here* $\Gamma_T$ *is the context of* $T$ *and* $s_T$ *is the (unique) sort of the term* $T$. *Analogously* $\mathcal{V}_F := \prod_{x:t\in\Gamma_F} \mathcal{I}(t)$. *Note that if* $T_1$ *is a subterm of* $T_2$, *there is a projection* $\pi_{T_1,T_2} : \mathcal{V}_{T_2} \to \mathcal{V}_{T_1}$, *and analogously for formulas.* $\vec{T}$ *stands for* $T_1, \ldots, T_n$. *Moreover,* $\langle [\![\vec{T}]\!]_i \circ \pi_i \rangle_i = \langle [\![T_1]\!] \circ \pi_1, \ldots, [\![T_n]\!] \circ \pi_n \rangle$ *and* $[\![\vec{T}]\!] = ([\![T_1]\!], \ldots, [\![T_n]\!])$. *The categorical semantics ensures that all involved and resulting functions are morphisms in* $\mathcal{C}$, *i.e. are measurable in case that* $\mathcal{C} = \mathbf{Meas}$, *etc. With a purely set-theoretic semantics, we would need to prove measurability (or other properties) separately for each NeSy framework.*

*That said, besides the general categorical case,* [18] *for better understandability, we also translate the equations to their meaning in the category of sets. We work with variable valuations* $\nu \in \mathcal{V}_T$ *(and* $\nu \in \mathcal{V}_F$*), noting that elements of* $\prod_{x:t\in\Gamma_T} \mathcal{I}(t)$ *map variables* $x : t$ *to values in* $\mathcal{I}(t)$*. We write* $[\![T]\!]_{\mathcal{I},\nu} = [\![T]\!]_{\mathcal{I}}(\nu)$ *and* $[\![F]\!]_{\mathcal{I},\nu} = [\![F]\!]_{\mathcal{I}}(\nu)$*. That said, we mostly omit* $\mathcal{I}$ *and* $\nu$ *if clear from the context.*

## 7.1 Probabilistic Semantics

**Definition 15.** *A* ***probabilistic NeSy framework*** *is a tuple* $(\mathcal{T}, \mathcal{R})$ *with* $\mathcal{T} = \mathcal{M}$ *the Giry monad on the category* **Measr** *of measure spaces and* $\mathcal{R}$ *a suitable 2Mon-BLat, for example the Product BL-Algebra on* $[0, 1]$*.*

The interpretation of a function $f$ of arity $n$ is a (measure preserving) *Markov kernel*, which is a measurable map $X \xrightarrow{q} \mathcal{M}(Y)$ where $\mathcal{M}$ denotes the *measure monad* on the category **Measr** of measure spaces.

Our definition of a probabilistic semantics largely follows that in the original ULLER paper Van Krieken et al. (2024). A central design decision of ULLER is the use of first-order interpretations and the use of probability distributions to interpret computational function symbols. This means that ULLER (and therefore also mULLER) is (like Logic Tensor Networks) not built on weighted model counting, i.e. on probability distributions over the set of interpretations. That said, it is still possible to capture certain aspects of weighted model counting in ULLER and mULLER, as we will see in section 7.4 below.

Connectives in the probabilistic semantics of ULLER are interpreted assuming independence of probabilities for atomic formulas. Hence, our distributional semantics can be seen as a special case of a fuzzy semantics, where the t-norm is the probabilistic product and the t-conorm is the probabilistic sum. This means that we can use the same equations as in the fuzzy semantics (and as in LTNs), but with a different motivation. Moreover, this explains why there is no essential difference between these probabilistic and fuzzy semantics.

Let us derive from our general semantic in definition 8 the interpretation of monadic formulas in probabilistic semantics. For the probabilistic NeSy framework, we define the aggregation morphisms $\text{aggr}^{\forall}_{B,C}, \text{aggr}^{\exists}_{B,C}$ required by Def. 12 as follows: for any measure spaces $B$ and $C$,

$$\text{aggr}^{\forall}_{B,C}(f) := \exp \circ\, \mathbb{E}_{c\sim\mu_C}[\ln \circ f(\cdot, c)]$$
$$\text{aggr}^{\exists}_{B,C}(f) := (1 - \exp) \circ\, \mathbb{E}_{c\sim\mu_C}[1 - \ln \circ f(\cdot, c)]$$

where $f : B \times C \to [0, 1]$ and $\mu_C$ is the measure on $C$. In the following examples, we provide implicit definitions of these aggregation operations through their concrete realisations. In set-theoretic notation, the semantics of computational formulas can be written as, where we use the

[18] Logician's note: We don't differentiate properly between additive and multiplicative connectives/units/quantifiers for an easier presentation coherent with the NeSy literature. However, this could easily be adapted to obtain something like Girad's linear logic Girard (1995), following the Zeitgeist of his transcendental syntax.

notation $\rho_m(\cdot|\vec{T}) := \mathcal{I}(m)([\![\vec{T}]\!])$:

$$\begin{aligned}
[\![[x := m(\vec{T})]F]\!] &= \textbf{do}\ \ a \leftarrow \mathcal{I}(m)([\![\vec{T}]\!]);\ [\![F]\!]_{\nu[x\mapsto a]} \\
&= \int_{a\in\mathcal{I}(s_m)} [\![F]\!]_{\nu[x\mapsto a]}\, d\mathcal{I}(m)([\![\vec{T}]\!])(a) \\
&= \int_{a\in\mathcal{I}(s_m)} [\![F]\!]_{\nu[x\mapsto a]}\, d\rho_m(a|\vec{T}) \\
&= \mathbb{E}_{a\sim\rho_m(\cdot|\vec{T})}\big[[\![F]\!]_{\nu[x\mapsto a]}\big] \\
&= \sum_{a\in\mathcal{I}(s_m)} [\![F]\!]_{\nu[x\mapsto a]} \cdot \rho_m(a|\vec{T}) \quad \text{(if } \mathcal{I}(s_m) \text{ is finite)}
\end{aligned}$$

We evaluate in the Product Algebra to obtain, where $\mu_s$ is the measure given by the measure space of $\mathcal{I}(s)$

$$[\![[x := m(\vec{T})]F]\!] := \mathbb{E}_{a\sim\rho_m(\cdot|\vec{T})}\big[[\![F]\!]_{\nu[x\mapsto a]}\big] \tag{14}$$

$$[\![\exists x{:}s\ F]\!] := 1 - \exp\big(\mathbb{E}_{a\sim\mu_s}\big[\ln\big(1 - [\![F]\!]_{\nu[x\mapsto a]}\big)\big]\big) \tag{15}$$

$$[\![\forall x{:}s\ F]\!] = \exp \mathbb{E}_{a\sim\mu_s}\big[\ln [\![F]\!]_{\nu[x\mapsto a]}\big] \tag{16}$$

$$[\![F \| G]\!] := [\![F]\!] + [\![G]\!] - [\![F]\!]\cdot[\![G]\!], \quad [\![F \& G]\!] := [\![F]\!]\cdot[\![G]\!], \tag{17}$$

$$[\![F \to G]\!] := \max\big(1, [\![G]\!]/[\![F]\!]\big), \quad [\![\neg F]\!] := 1 - [\![F]\!] \tag{18}$$

$$[\![\bot]\!] := 0, \quad [\![\top]\!] := 1. \tag{19}$$

$\forall$ *as weighted products (finite case).* Let $\mathcal{I}(s)$ be finite and define the random variable $X(a) := [\![F]\!]_{\nu[x\mapsto a]}$. The infinitary probabilistic $\forall$ (Eq. (16)) evaluated on $F$ with a weighted counting measure on $X$ yields

$$[\![\forall x{:}s\ F]\!] \ = \ \exp\Big(\sum_{a\in\mathcal{I}(s)} w_a \ln X(a)\Big) \ = \ \Big(\prod_{a\in\mathcal{I}(s)} \exp(\ln(X(a)^{w_a}))\Big) \ = \ \prod_{a\in\mathcal{I}(s)} X(a)^{w_a}.$$

By contrast, the finite product aggregator of the original ULLER semantics is obtained by setting $w_a = 1$ for all $a \in \mathcal{I}(s)$:

$$[\![\forall x{:}s\ F]\!] \ = \ \prod_{a\in\mathcal{I}(s)} X(a)^1.$$

However, our definition of the $\forall$ quantifier is more general even in the finite case, as it allows for meaningful examples like the weighted counting measure on $\mathcal{I}(s)$ yielding the geometric mean with $n := \#\mathcal{I}(s)$, the number of elements in $\mathcal{I}(s)$, yielding

$$[\![\forall x{:}s\ F]\!] \ = \ \prod_{i=1}^{n} X(a_i)^{1/n}.$$

In the case above of the geometric mean and also in many other (continuous) cases, one actually does choose only *one* universal quantifier to be constant on all sorts, as for the existential

quantifier. In this case one can actually work within a simpler framework using the *normal* Giry monad on the category **Meas** of measurable spaces:

**Definition 16.** *A **simple probabilistic NeSy framework** is a tuple* $(\mathcal{T}, \mathcal{R})$ *with* $\mathcal{T} = \mathcal{M}$ *the Giry monad on the category* **Meas** *of measurable spaces and* $\Omega$ *the truth basis, normally* $\Omega = \{0, 1\}$.

Now, if one works in standard Borel spaces[19] (as one most often does in practice), one obtains an even more practical version of this simple probabilistic NeSy framework. This is very useful in practice since one can see an uncountable standard Borel space just as a set that is canonically equipped with the Borel sigma-algebra and Lebesgue measure. Also a finite or countable standard Borel space can be seen as a set that is canonically equipped with the discrete sigma-algebra and counting measure.

**Definition 17.** *A **simple standard Borel NeSy framework** is a tuple* $(\mathcal{T}, \mathcal{R})$ *with* $\mathcal{T} = \mathcal{M}$ *the Giry monad on the category* **BorelMeas** *of standard Borel spaces and* $\Omega$ *the truth basis, normally* $\Omega = \{0, 1\}$.

Additionally, if one chooses the measures for universal and existential quantification to be given by a density functions $f$ (most often the case in practice), one obtains an implementation-friendly Lebesgue-probabilistic NeSy semantics for the quantifiers $\forall$ and $\exists$:

$$[\![\forall x{:}s\ F]\!] = \exp \int_{a \in I(s)} \ln [\![F]\!]_{\nu[x \mapsto a]}\, f(a)\, da \tag{20}$$

$$[\![\exists x{:}s\ F]\!] = 1 - \exp \int_{a \in I(s)} \ln(1 - [\![F]\!]_{\nu[x \mapsto a]})\, f(a)\, da \tag{21}$$

In the same spirit, if the semantics of the monadic formula 14 should only use probability measures admitting a density w.r.t. the Lebesgue measure[20], one can work with a simplified Giry monad on the category **BorelMeas**, sending a standard Borel space to the standard Borel space of probability measures on that space admitting a density w.r.t. the Lebesgue measure. Combined with the quantifiers defined w.r.t Lebesgue densities from eq. (20) and eq. (21), this makes the implementation simpler and more efficient and is the default choice in practice anyway.

Actually, depending on the use case, one might want to use a probability monad on one of the following categories[21], listed in increasing complexity in Table 6 summarising typical suitability.

[19] Standard Borel spaces are measurable spaces where the underlying set is isomorphic to $\mathbb{R}$ or is finite or countable. All finite powers of the real line and all intervals, as measurable spaces, are isomorphic to the real line, as explained at https://ncatlab.org/nlab/show/standard+Borel+space. Also a measure space (or probability space) is called a standard Borel measure space (**BorelMeas**) (standard Borel probability space (**BorelProb**)) if and only if its underlying measurable space is standard Borel as above.

[20] This is hidden in the notation of the semantics of the monadic formula in the original ULLER semantics in Van Krieken et al. (2024), Eq. 20; although there the density is mistakenly confused with the corresponding measure.

[21] Which are by far not all, see https://ncatlab.org/nlab/show/monads+of+probability,+measures,+and+valuations for more examples.

**Table 6.** Suitability of base categories for probability monads

| Category | FS | Finite | $\cong \mathbb{Z}$ | $\cong \mathbb{R}$ | +DQ | +DPQ | +HO |
|---|---|---|---|---|---|---|---|
| **Set** | ✓ | ✓ | ✓ | ✓ | × | × | ✓ |
| **BorelMeas** | ○ | ○ | ✓ | ✓ | × | × | × |
| **BorelMeasr** | ○ | ○ | ○ | ○ | ✓ | ○ | × |
| **BorelProb** | ○ | ○ | ○ | ○ | ○ | ✓ | × |
| **Meas** | ○ | ○ | ✓ | ✓ | × | × | × |
| **Measr** | ○ | ○ | ○ | ○ | ✓ | ○ | × |
| **Prob** | ○ | ○ | ○ | ○ | ○ | ✓ | × |
| **QBS** | ○ | ○ | ✓ | ✓ | × | × | ✓ |

Here, × stands for *incompatible*, ○ for *compatible but generally not recommended* and ✓ for *compatible and generally recommended*. Also, the short form "FS" stands for "finitely supported", "DQ" for "sort-dependent quantifiers", "DPQ" for "sort-dependent probability quantifiers" and "HO" for "higher-order logic". Hence the "FS" columns denotes that the monad only constructs finitely-supported probability measures, the "Finite" column denotes that the monad constructs probability measures on finite measurable spaces, the "$\cong \mathbb{Z}$" column denotes that the monad constructs probability measures on measurable spaces isomorphic to $\mathbb{Z}$, the "$\cong \mathbb{R}$" column denotes that the monad constructs probability measures on measurable spaces isomorphic to $\mathbb{R}$, the "+DQ" column denotes that the monad allows for sort-dependent quantifiers via measures, the "+DPQ" column denotes that the monad allows for sort-dependent quantifiers via probability measures and the "+HO" column denotes that the monad allows for higher-order logic in Cartesian closed categories.

## 7.2 Infinitary LTN$_p$ Semantics

**Definition 18.** *A **LTN-like NeSy framework** is a tuple $(\mathcal{T}, \mathcal{R})$ with $\mathcal{T} = \mathcal{O}$ the probability monad on the category* **Prob** *of probability spaces and $\Omega$ the truth basis, normally $\Omega = \{0, 1\}$, and $\mathcal{R}$ a suitable 2Mon-BLat, for example the Product Real Algebra from Product Real Logic as in Badreddine et al. (2022).*

*Setting* Stable product real logic of Logic Tensor Networks Badreddine et al. (2022) uses $p$-means for finite quantification. The hyperparameter $p$ is usually increased during training, because this moves from mean (tolerant to outliers) towards the maximum[22] (logically stricter). However, since domains are generally infinite, we also need to aggregate infinite many truth–scores $(x_i)_{i \in I} \subseteq [0, 1]$. The power–mean extends from the finite case to an *integral* form that is well defined whenever the data are $L^p$-integrable. Let $(X, \mathcal{A}, \rho)$ be a probability space and $f : X \to [0, 1] \subseteq \mathbb{R}$ a measurable map with $\int_X f \, d\rho \leq 1$. Because $0 \leq f \leq 1$, one automatically has $f \in L^p(\rho)$ for every real $p$, so $p$-means are always defined. This bounded–by–one assumption reflects the fact that in our logical reading a truth-score never exceeds 1.

[22] This holds for existential quantification. Universal quantification $\forall$ is defined through $\neg\exists x{:}s\neg$, and converges to the minimum.

The infinitary $\mathrm{LTN}_p$ semantics is a modification of the probabilistic semantics, which is motivated by replacing the quantifiers in equation (15) and (16). We generalise this: by working in the category **Prob**, for any sort $s$ we have to provide a probability measure $\rho_s$ on $\mathcal{I}(s)$. This enables us to obtain a $p$-means for infinite[23] domains:

$$M_p(a_1,\dots,a_n) := \Big(\frac{1}{n}\sum_{i=1}^{n} a_i^p\Big)^{1/p}, \quad M_p(f;\rho_s) := \Big(\int_{x\in X} f(x)^p \, d\rho_s(x)\Big)^{1/p}, \tag{22}$$

and these extend to $M_0(a_1,\dots,a_n) := \Big(\prod_{i=1}^n a_i\Big)^{\frac{1}{n}}$ and $M_0(f;\rho_s) := \exp\big(\int \ln f \, d\rho_s\big)$. For $p\to\infty$ we recover the supremum. Take $X=\{1,\dots,N\}$ with counting measure $1/N$, then $M_p(f;\rho_s)$ reduces to $M_p(a_1,\dots,a_n)$ or choose weights $w_i$ summing up to 1 for $i=1,\dots,N$ to obtain the weighted $p$-mean. The aggregated 2Mon-BLat is similar to the probabilistic one, except for the Reichenbach implication as implication and the following aggregation functions. For a hyperparameter $1\le p<\infty$ of $\mathrm{LTN}_p$, let $\mathrm{aggr}^{\exists}_{\mathcal{I}(s)}(f) := M_p(f;\rho_s)$ and $\mathrm{aggr}^{\forall}_{\mathcal{I}(s)}(f) := 1 - M_p(\lambda x.f(1-x);\rho_s)$. As a result, equation (22) now becomes:

$$[\![\exists x{:}s\ F]\!] = \Big(\int_{a\in\mathcal{I}(s)} \big([\![F]\!]_{\nu[x\mapsto a]}\big)^p \, d\rho_s(a)\Big)^{1/p}.$$

As in Badreddine et al. (2022), this is for $1\le p<\infty$, where for $p\to\infty$ we recover the supremum. However, we also propose a different pair of quantifiers with $1/2\le q\le 1$, where for $q\to 1$ the universal quantifier converges to the product mean (geometric mean), while the existential quantifier converges to the probabilistic sum:

$$[\![\forall x{:}s\ F]\!] := \exp\Big(\Big(\int_{a\in\mathcal{I}(s)} \big(\ln[\![F]\!]_{\nu[x\mapsto a]}\big)^q \, d\rho_s(a)\Big)^{1/q}\Big), \quad [\![\exists x{:}s\ F]\!] := [\![\neg\forall x{:}s\ \neg F]\!].$$

It is worth noting that our probability measure $\rho_s$ depend on the sort $s$ of the variable $x$ in the quantifier, since it is given by the probability measure of the probability space of $\mathcal{I}(s)$. This stands in contrast to Ślusarz et al. (2023), where the probability measure depends directly on the variable $x$.

In practice, just as in the case of the simple probabilistic NeSy framework, one can also work with a simple Lebesgue NeSy framework, and choose the measures for universal and existential quantification to be given by a Lebesgue-density function $f$ to obtain:

$$[\![\forall x{:}s\ F]\!] := \exp\Big(\Big(\int_{a\in\mathcal{I}(s)} \big(\ln[\![F]\!]_{\nu[x\mapsto a]}\big)^q f(a)\, da\Big)^{1/q}\Big), \quad [\![\exists x{:}s\ F]\!] := [\![\neg\forall x{:}s\ \neg F]\!],$$

or in the case of the original LTN quantifiers:

$$[\![\exists x{:}s\ F]\!] = \Big(\int_{a\in\mathcal{I}(s)} \big([\![F]\!]_{\nu[x\mapsto a]}\big)^p f(a)\, da\Big)^{1/p}, \quad [\![\forall x{:}s\ F]\!] := [\![\neg\exists x{:}s\ \neg F]\!].$$

[23] This defends the idea of LTN against the criticism of Ślusarz et al. (2023), that the domains are finite.

### *7.3 Infinitary STL Semantics*

Signal Temporal Logic (STL) is a temporal logic for expressing properties of signals. STL is particularly useful for modelling and analysing the temporal aspects of real-time systems, such as the timing and sequencing of events.

In the semantics of STL, we do not have any implication connective, nor neutral elements. We still need to interpret the syntactic implication connective as some form of semantical implication and the syntactic $\bot$ and $\top$ as $-\infty$ and $\infty$, the latter as in Ślusarz et al. (2023). Also keep in mind, that this does not touch the truth designations of $-\infty$ as absolute falsity and $\infty$ as absolute truth. That is, STL works with these degrees of truth and falsity, which could be taken together to form one single degree (of probability), and which the absolute degrees are infinite. Therefore we model STL with a truth basis containing only one element: $\{1\}$, a basis element which is then scaled by to form the extended real numbers as truth space. A consequence of this that there are no meaningful non-computational predicates in STL.

What also can not be ignored is that STL does not directly uses a 2Mon-BLat, but only approximates one. It works within the normal extended real numbers algebra $(\tilde{\mathbb{R}}, \max, \min, +, *)$ and then goes on to approximate the min and max operations. The are many different ways to do this, but one of the most recent ones is to use the $\mathsf{A}^r$ and $\mathsf{O}^r$ operators as defined in Varnai and Dimarogonas (2020). Additionally, STL is not concerned with the operations $\otimes$ and $\oplus$ of the 2Mon-BLat, these are not used in the semantics of STL, and are just kept to be the standard operations $+$, $*$ of the extended real numbers algebra.

For these reasons, in order to faithfully model STL, in a way that makes it comparable to other semantics, we would need to extend our syntax and semantics, and we would also need to allow to approximate 2Mon-BLats. This however, is out of scope for this paper, and will be discussed in a future work, and yet we still give a first sketch:

$$[\![[x := m(\vec{T})]F]\!] := \mathbb{E}_{a\sim\mu_m(\cdot|\vec{T})}\left[[\![F]\!]_{\nu[x\mapsto a]}\right] \tag{23}$$

$$[\![\exists x{:}s\ F]\!] := \mathsf{O}^r_{a\in\mathcal{I}(s)}\left([\![F]\!]_{\nu[x\mapsto a]}\right), \quad [\![\forall x{:}s\ F]\!] := \mathsf{A}^r_{a\in\mathcal{I}(s)}\left([\![F]\!]_{\nu[x\mapsto a]}\right), \tag{24}$$

$$[\![F \| G]\!] := \mathsf{O}^r([\![F]\!], [\![G]\!]), \quad [\![F \& G]\!] := \mathsf{A}^r([\![F]\!], [\![G]\!]) \tag{25}$$

$$[\![F \to G]\!] := [\![\neg F \| G]\!] = \mathsf{O}(-[\![F]\!], [\![G]\!]), \quad [\![\neg F]\!] := -[\![F]\!] \tag{26}$$

$$[\![\bot]\!] := -\infty, \quad [\![\top]\!] := \infty. \tag{27}$$

The STL robustness metrics are defined as in Ślusarz et al. (2023) and originally in Varnai and Dimarogonas (2020):

$$\mathsf{A}^r_{a\in M}(a) = \begin{cases} \frac{\sum_a a_{min} e^{\tilde{a}} e^{r\tilde{a}}}{\sum_a e^{r\tilde{a}}} & \text{if } a_{min} < 0 \\ \frac{\sum_a a e^{-r\tilde{a}}}{\sum_a e^{-r\tilde{a}}} & \text{if } a_{min} > 0 \\ 0 & \text{if } a_{min} = 0 \end{cases}$$

where $r \in \mathbb{R}^+$ (constant), $a_{min} = \min_{a\in M}(a)$, and $\tilde{a} = \frac{a - a_{min}}{a_{min}}$. $\mathsf{A}^r$ is an approximation of the min operation, and for $r \to \infty$ it converges to it. Therefore, its notation is similar to the notation of the min operation, with $\mathsf{A}^r_{b\in N}(f(b)) := \mathsf{A}^r(\mathrm{im}(f)) := \mathsf{A}^r_{a\in\mathrm{im}(f)}(a)$. The operator $\mathsf{O}^r_{a\in M}$ is defined

as $-\mathsf{A}^r_{a\in M}(-a)$.[24] For infinite domains, the minimum is replaced by the infimum $\inf_{a\in M}(a)$, and the summations $\sum_{a\in M}$ are replaced by integrals $\int_{a\in M} d\mu_s(a)$, where $\mu_s$ is the measure given by the measure space of $\mathcal{I}(s)$.

### 7.4 Weighted Model Counting and Weighted Model Integration

ULLER can model certain aspects of weighted model counting (WMC) in a probabilistic semantics. However, instead of summing up literal or model weights, one needs to sum up weights of variable valuations. In the case of ULLER Van Krieken et al. (2024), we have the following definition. Given an interpretation $\mathcal{I}$ and a formula $F$ that is classical (i.e. without computational symbols) with context $\Gamma_F := \{x_1 : s_1, \ldots, x_n : s_n\}$, the domains of the variables are given by $\mathcal{I}(s_1), \ldots, \mathcal{I}(s_n)$[25]. This yields the weighted model count (WMC) as follows:

$$\begin{aligned}\mathrm{WMC}(F, w) &= \sum_{\vec{a}\in\mathcal{I}(s_1)\times\cdots\times\mathcal{I}(s_n)} w(\vec{a})[\![F]\!]_{\nu[x_1\mapsto a_1,\ldots,x_n\mapsto a_n]} \\ &= \sum_{a_1\in\mathcal{I}(s_1)} \cdots \sum_{a_n\in\mathcal{I}(s_n)} w(a_1,\ldots,a_n)\,[\![F]\!]_{\nu[x_1\mapsto a_1,\ldots,x_n\mapsto a_n]}\end{aligned}$$

If the weight function factorises, i.e. the random variables $x_1, \ldots, x_n$ are assumed **independent**[26] —then for every assignment $(a_1, \ldots, a_n) \in \mathcal{I}(s_1) \times \cdots \times \mathcal{I}(s_n)$:

$$w(a_1, \ldots, a_n) \;=\; \prod_{i=1}^{n} \rho_{f_i}(a_i).$$

Consequently, the weighted model count becomes

$$\mathrm{WMC}(F, f_1, \ldots, f_n) \;=\; \sum_{a_1\in\mathcal{I}(s_1)} \cdots \sum_{a_n\in\mathcal{I}(s_n)} \left(\prod_{i=1}^{n} \rho_{f_i}(a_i)\right) [\![F]\!]_{\nu[x_1\mapsto a_1,\ldots,x_n\mapsto a_n]}.$$

This WMC can be expressed in the language of NeSy systems as follows (Van Krieken et al. (2024), p. 234):

$$[x_1 := f_1(), \ldots, x_n := f_n()]F$$

In the linearly **dependent** case, rewrite it via the chain rule,

$$w(a_1, \ldots, a_n) \;=\; \prod_{i=1}^{n} \rho_{f_i}\big(a_i \mid a_1, \ldots, a_{i-1}\big),$$

[24] See Varnai and Dimarogonas (2020) for details.
[25] In the original ULLER paper these were written as $\Omega_1 := \mathcal{I}(s_1), \ldots, \Omega_n := \mathcal{I}(s_n)$.
[26] As in Van Krieken et al. (2024) (p. 16).

to make the conditional dependencies explicit. In this case, the WMC becomes

$$\sum_{a_1 \in \mathcal{I}(s_1)} \cdots \sum_{a_n \in \mathcal{I}(s_n)} \left( \prod_{i=1}^{n} \rho_{f_i}(a_i \mid a_1, \ldots, a_{i-1}) \right) [\![F]\!]_{\nu[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]}.$$

In the **continuous** case we obtain weighted model integration[27] (WMI) as follows:

$$\int_{a_1 \in \mathcal{I}(s_1)} \cdots \int_{a_n \in \mathcal{I}(s_n)} [\![F]\!]_{\nu[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]} d\rho_{f_n}(a_n \mid a_1, \ldots, a_{n-1}) \cdots d\rho_{f_1}(a_1)$$

Finally, we can also express even more general dependencies than linear ones. Given any Bayesian network with a set of variables $x_1, \ldots, x_n$, we can express this in ULLER as follows:

$$\sum_{a_1 \in \mathcal{I}(s_1)} \cdots \sum_{a_n \in \mathcal{I}(s_n)} \left( \prod_{i=1}^{n} \rho_{f_i}(a_i \mid \text{parents}(a_i)) \right) [\![F]\!]_{\nu[x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]}.$$

and in ULLER, this is expressed as (assuming that the $x_i$ are topologically ordered, i.e. $x_i$ can only a parent of $x_j$ if $i < j$):

$$[x_1 := f_1(), x_2 := f_2(\text{parents}(x_2)), \ldots, x_n := f_n(\text{parents}(x_n))]F.$$

## 8 Related Work and Their Examples

Monad-based dynamic logic Mossakowski et al. (2010) is similar to our approach, but differs in some important aspects. In Mossakowski et al. (2010), not the whole of $\mathcal{T}\Omega$ is used as the space of truth values, but only a subset of it, namely the pure computations $p$ with truth-valued result. These are discardible, i.e. they can be left out in a sequence of computations, and copyable, i.e. deterministic. The latter means that $[x := p, y := p]x = y$ holds. In the non-empty powerset monad, the distribution and the Giry monads, all computations are discardible, but only those in the image of $\eta$ are copyable. That is, only $T$ and $F$ (in the non-empty non-determinism monad) and $\delta_T$ and $\delta_F$ (in the distribution and Giry monads) are copyable. However, in the non-determinism monad, we clearly want $B \equiv \{T, F\}$ as a truth value, and in the distribution and the Giry monads, we want all probabilities $[0, 1]$ as a truth values (and this space is isomorphic to probability distributions over $\{T, F\}$). Hence, we do not want a copyability assumption in our semantics. It seems that this is common in monads used for knowledge representation, as opposed to monads used for programming language semantics as in Mossakowski et al. (2010). Note that both views may even be useful for one and the same monad (e.g. the non-empty powerset monad), depending on its use.

[27] Compare with Morettin et al. (2017).

### 8.1 Example: Weather Prediction of DeepSeaProbLog

As an illustration of how existing neurosymbolic systems using continuous probability distributions[28] can be expressed in mULLER, consider the following weather prediction example from DeepSeaProbLog Smet et al. (2023). The original DeepSeaProbLog program uses neural distributional facts to model humidity detection and temperature prediction:

```
humid(Data) ~ bernoulli(humid_detector(Data)).
temp(Data, T) ~ normal(temperature_predictor(Data)).

good_weather(Data) :- humid(Data) =:= 1, temp(Data) < 0.
good_weather(Data) :- humid(Data) =:= 0, temp(Data) > 15.

query(good_weather(world)).
```

Here, `humid_detector` and `temperature_predictor` output distribution parameters, while `world` represents a specific dataset. In mULLER syntax, this last query becomes the following formula (later denoted by $F$) with $\mathbf{world} := \mathcal{I}(\mathbf{data_1})$:

$$\begin{aligned}&[h := \mathit{bernoulli}(\mathit{humid\_detector}(\mathbf{data_1}))]\\ &\quad [t := \mathit{normal}(\mathit{temperature\_predictor}(\mathbf{data_1}))]\\ &\qquad (h = 1 \wedge t < 0) \vee (h = 0 \wedge t > 15)\end{aligned}$$

where `humid_detector` is a function returning parameters for a Bernoulli distribution, and `temperature_predictor` returns parameters $(\mu, \sigma)$ for a normal distribution.[29] The nested monadic assignments capture the same probabilistic dependencies as the original program, demonstrating how mULLER's uniform syntax can express diverse neurosymbolic paradigms. The semantic evaluation $[\![F]\!]_{\mathcal{I},\nu}$ of this formula yields the probability distribution over truth values, corresponding to the query result in DeepSeaProbLog.

*DeepSeaProbLog semantics vs mULLER (same query).* Given the signature

$$\begin{aligned}&\mathit{Worlds},\ \mathit{Unit\_Interval},\ \mathit{Reals}^2 \in \Sigma,\\ &\mathbf{data1} \in \text{Const}\\ &\mathit{humid\_detector}, \mathit{temperature\_predictor} \in \text{Func}\\ &\mathit{Bernoulli},\ \mathit{Normal} \in \text{mFunc},\end{aligned}$$

[28] Note that a proper treatment of these requires category theory, see sections 6.1 and 6.

[29] In Smet et al. (2023) the `humid_detector` returns a probability distribution over $[0, 1]$ and `temperature_predictor` returns a probability distribution over $\mathbb{R}^2$, that is that they are also monadic functions, that are composed in the Kleisli category with the monadic functions `Bernoulli` and `Normal`, respectively. For simplicity of presentation, we do not explicitly mention this in the syntax and assume they are just deterministic functions.

define the interpretation function

$$\begin{aligned}
&\mathbf{world} := \mathcal{I}(\mathbf{data_1}) \in \mathit{Worlds} := \mathcal{I}(\mathit{Data}) \\
&\mathcal{I}(\mathit{temperature_predictor}) : \mathit{Worlds} \to \mathbb{R}^2, \qquad \mathcal{I}(\mathit{humid_detector}) : \mathit{Worlds} \to [0,1]; \\
&\mathcal{N} := \mathcal{I}(\mathit{Normal}) : \mathbb{R}^2 \to \mathcal{G}([0,1]), \qquad \mathcal{B} := \mathcal{I}(\mathit{Bernoulli}) : [0,1] \to \mathcal{G}([0,1]).
\end{aligned}$$

and label

$$\begin{aligned}
&(\mu, \sigma) = \mathcal{I}(\mathit{temperature_predictor})(\mathbf{world}), \qquad p = \mathcal{I}(\mathit{humid_detector})(\mathbf{world}); \\
&T \sim \mathcal{N}(\mu, \sigma^2), \qquad H \sim \mathcal{B}(p).
\end{aligned}$$

Then define events $A := \{H = 1,\ T < 0\}$ and $B := \{H = 0,\ T > 15\}$ (disjoint). DeepSeaProbLog assigns the query probability as an expectation of an indicator under the joint measure (here product measure by independence of PCFs):

$$P_{\mathrm{DSP}}(\mathit{good\_weather}) = \mathbb{E}[\mathbf{1}_A + \mathbf{1}_B] = p \int_{\mathbb{R}} \mathbf{1}(t < 0)\, \varphi_{\mu,\sigma}(t)\, dt + (1-p) \int_{\mathbb{R}} \mathbf{1}(t > 15)\, \varphi_{\mu,\sigma}(t)\, dt,$$

where $\varphi_{\mu,\sigma}(t) := \dfrac{1}{\sqrt{2\pi}\,\sigma} \exp\!\left(-\dfrac{(t-\mu)^2}{2\sigma^2}\right)$ is the normal density. In mULLER, the monadic rule for $[x := m(\cdot)]F$ evaluates to an expectation (Eq. (14)), hence for $F$ above

$$\begin{aligned}
[\![F]\!] &= \mathbb{E}_H\big[\mathbb{E}_T\big[\mathbf{1}_{\{H=1,T<0\}} + \mathbf{1}_{\{H=0,T>15\}} \,\big|\, H\big]\big] \\
&= \mathbb{E}_H\big[\mathbf{1}_{\{H=1\}}\, \mathbb{E}_T[\mathbf{1}_{\{T<0\}}] + \mathbf{1}_{\{H=0\}}\, \mathbb{E}_T[\mathbf{1}_{\{T>15\}}]\big] \\
&= \mathbb{E}_H[\mathbf{1}_{\{H=1\}}] \int_{-\infty}^{0} \varphi_{\mu,\sigma}(t)\, dt + \mathbb{E}_H[\mathbf{1}_{\{H=0\}}] \int_{15}^{\infty} \varphi_{\mu,\sigma}(t)\, dt \\
&= p \int_{-\infty}^{0} \varphi_{\mu,\sigma}(t)\, dt + (1-p) \int_{15}^{\infty} \varphi_{\mu,\sigma}(t)\, dt \\
&= p\, \Phi\!\left(\frac{0-\mu}{\sigma}\right) + (1-p)\left(1 - \Phi\!\left(\frac{15-\mu}{\sigma}\right)\right) \\
&= P_{\mathrm{DSP}}(\mathit{good_weather}).
\end{aligned}$$

Here, $\Phi$ denotes the standard normal cumulative distribution function. Thus, for this program our semantics coincides with DeepSeaProbLog's possible-world semantics while avoiding world enumeration.

Beyond single-instance queries, the distributional quantifiers from Table 2 are meaningful in this setting. Three typical use cases are:

- **All stations (universal aggregation).** Probability that all weather stations have good weather:

$$\begin{aligned}
&\forall s \in \mathit{WeatherStations} \\
&\quad [h := \mathit{bernoulli}(\mathit{humid_detector}(\mathbf{World}_s))] \\
&\qquad [t := \mathit{normal}(\mathit{temperature_predictor}(\mathbf{World}_s))] \\
&\qquad\quad (h = 1 \wedge t < 0) \vee (h = 0 \wedge t > 15)
\end{aligned}$$

  which evaluates via $\prod_s(\cdot)$ in the Product algebra.

- **Exists a region (existential aggregation).** Probability that at least one region has good weather:

$$\begin{aligned}&\exists r \in \mathit{Regions}\\ &\quad [h := \mathit{bernoulli}(\mathit{humid\_detector}(\mathbf{World}_r))]\\ &\quad\quad [t := \mathit{normal}(\mathit{temperature\_predictor}(\mathbf{World}_r))]\\ &\quad\quad\quad (h = 1 \wedge t < 0) \vee (h = 0 \wedge t > 15)\end{aligned}$$

  which evaluates via $1 - \prod_r (1 - \cdot)$ (probabilistic sum).

- **Always over time (universal over time).** Probability that the weather is good for all time slots:

$$\begin{aligned}&\forall \tau \in \mathit{TimeSlots}\\ &\quad [h := \mathit{bernoulli}(\mathit{humid\_detector}(\mathbf{World}_\tau))]\\ &\quad\quad [t := \mathit{normal}(\mathit{temperature\_predictor}(\mathbf{World}_\tau))]\\ &\quad\quad\quad (h = 1 \wedge t < 0) \vee (h = 0 \wedge t > 15)\end{aligned}$$

  again aggregating with a product over $\tau$.

*On $\forall$-aggregation: product vs. infimum vs. LTN.* For distributional semantics there are several meaningful choices for aggregating universal quantification:

- **Product** (probabilistic $\forall$) as in (16): $[\![\forall x : s\, F]\!] = \prod_{a \in \mathcal{I}(s)} [\![F]\!]_{\nu[x \mapsto a]}$. This reads as "probability that *all* independent[30] events hold". It is however extremely sensitive: a single zero (e.g., one faulty station reporting bad weather) collapses the product to 0 and, even without zeros, the value decays exponentially with the number of stations, at least for continuous values.

- **Infimum/min** (classical fuzzy $\forall$): $\inf_a [\![F]\!]_{\nu[x \mapsto a]}$. This captures a strict worst-case reading and does not shrink when many stations are near 1, but it is still killed by a single zero and ignores the distribution of the other values.

- **LTN-style $p$-mean of complements** (Sec. 4.3, App. 7.2): $[\![\forall x : s\, F]\!] = 1 - \left(\frac{1}{|\mathcal{I}(s)|} \sum_a (1 - [\![F]\!]_{\nu[x \mapsto a]})^p\right)^{1/p}$ (or its measure-theoretic analogue). This provides a tunable continuum between averaging ($p$ small) and the infimum ($p \to \infty$), and is typically more robust to isolated outliers. A weighted variant $\sum_a w_a (1 - [\![F]\!]_a)^p$ with $\sum_a w_a = 1$ can encode station reliability.

In sensor networks with occasional false alarms (one station outputs 0 while thousands report values near 1), product and infimum both collapse to 0. A moderate-$p$ LTN aggregator (optionally weighted) avoids single-sensor catastrophic failure while still converging to the strict infimum as $p \to \infty$. When a strict "all must hold" interpretation is intended and measurements are trusted as independent, the product is appropriate.

[30] See paragraph 7.1 for a critical discussion of independence.

*Infinite (time) domains.* For countably or uncountably infinite sets of (time) points $T$, instead of the finite set of TimeSlots from the previous example, we use the measure-theoretic aggregations from Sec. 7.1 (Eqs. (15)–(16)) or the infinitary LTN aggregations from Sec. 7.2. Given a (probability) measure $\rho_T$ on $T$ and writing $\phi(\tau) := [\![F]\!]_{\nu[\tau\mapsto\tau]} \in [0,1]$:

$$[\![\forall \tau{:}T\ F]\!] = \exp\Big(\mathbb{E}_{\tau\sim\rho_T}[\ln \phi(\tau)]\Big), \qquad [\![\exists \tau{:}T\ F]\!] = 1 - \exp\Big(\mathbb{E}_{\tau\sim\rho_T}[\ln(1-\phi(\tau))]\Big).$$

These reduce to certain (depending on the measure $\rho_T$) finite products/probabilistic sums when $T$ is finite with the counting measure. Zeros at isolated times yield $\ln 0 = -\infty$ and therefore a value 0 for $\forall$ (consistent with the convention $0 \cdot \infty = 0$).

Alternatively, the infinitary LTN scheme provides a robust family of aggregators:

$$[\![\forall \tau{:}T\ F]\!] = 1 - \Big(\int_T (1-\phi(\tau))^p \, d\rho_T(\tau)\Big)^{1/p}, \qquad [\![\exists \tau{:}T\ F]\!] = \Big(\int_T \phi(\tau)^p \, d\rho_T(\tau)\Big)^{1/p}.$$

For rare glitches (bad weather reported at a few time instants), these aggregators with moderate $p$ (or with $\rho_T$ down-weighting unlikely times) avoid collapse to 0, while $p \to \infty$ recovers the strict infimum. When a "almost everywhere" reading is desired, one may also use the essential infimum/supremum w.r.t. $\rho_T$ to ignore measure-zero anomalies.

## 8.2 Comparison to LDL (Logic of Differentiable Logics)

The Logic of Differentiable Logics (LDL) approach Ślusarz et al. (2023) differs from our framework in several fundamental aspects. First, LDL incorporates comparison operators directly into their syntax, whereas our approach treats them as part of the non-logical signature, providing greater flexibility in language design. Second, LDL restricts their type system to Bool, Real, Vector, and Index types, while our framework maintains a more general type-theoretic foundation. Third, LDL's syntax includes constructs such as lambda-terms, let-terms, repetitions of expressions, vector constructors, and vector look-ups, which introduces complexity that our approach avoids through a more streamlined logical structure.

While both approaches employ typed languages, LDL does not utilise an abstract signature to achieve language generality. Furthermore, LDL lacks the algebraic structure provided by our 2Mon-BLat framework, which endows our semantics with a clearly defined mathematical structure while maintaining sufficient flexibility to accommodate different logical semantics (STL as an approximation). Additionally, LDL makes the restrictive assumption that all distinct random variables are independent, limiting its applicability to scenarios where this independence assumption holds.

A key distinction lies in the quantification mechanisms: LDL employs variable-dependent quantification, whereas our framework utilises sort-dependent quantification, which furthermore reduces complexity by providing all the (sort-dependent) information necessary for quantification by defining the interpretation function.

Finally, LDL does not require associativity of the logical operators, enabling the use of non-associative and/or operators characteristic of Signal Temporal Logic (STL). This non-associativity requirement constitutes another reason why STL does not naturally fit into our

2Mon-BLat model and should instead be modelled as approximating a 2Mon-BLat structure, a topic to be addressed in subsequent work.

Actually, it is not absolutely necessary to require associativity, as we did in Def. 2 of our 2Mon-BLat. However unifying nearly all semantics under this algebraic umbrella and seeing STL (and potentially others) as approximations of the 2Mon-BLat algebra, gives us a richer structure of our semantics and reveals otherwise suppressed algebraic laws.

*LDL quantifiers* Recall Defs. 3 and 12 and this discussion following the latter: an aggregated **2Mon-BLat** provides maps $\mathrm{aggr}^{\forall}_X, \mathrm{aggr}^{\exists}_X : \mathcal{L}^X \to \mathcal{L}$, where additionally, a structure on $X$ may be used. In LDL, the context $Q$ maps a bound variable $x{:}s$ to a random variable $Q[x]$ on $\mathcal{I}(s)$ with density $p_X$. For a measurable $g : \mathcal{I}(s) \to \mathbb{R}$ write

$$x_{\min} := \arg\min_{a \in \mathcal{I}(s)} g(a), \qquad x_{\max} := \arg\max_{a \in \mathcal{I}(s)} g(a).$$

For a random variable $X$ with density $p_X$, LDL defines in Ślusarz et al. (2023)(p. 9) (and we adopt) the quantifier aggregations exactly as

$$\mathrm{E}_{\min}[g(X)] := \lim_{\gamma \to 0} \int_{x \in B_\gamma^{x_{\min}}} p_X(x)\, g(x)\, dx, \qquad \mathrm{E}_{\max}[g(X)] := \lim_{\gamma \to 0} \int_{x \in B_\gamma^{x_{\max}}} p_X(x)\, g(x)\, dx.$$

Hence we set

$$\mathrm{aggr}^{\forall}_{\mathcal{I}(s),Q[x]}(g) := \mathrm{E}_{\min}\big[g\big(Q[x]\big)\big], \qquad \mathrm{aggr}^{\exists}_{\mathcal{I}(s),Q[x]}(g) := \mathrm{E}_{\max}\big[g\big(Q[x]\big)\big].$$

Consequently, for any $F$ and valuation $\nu$, this is coherent with the definition of the quantifiers from Table 2:

$$[\![\forall x{:}s\, F]\!]_\nu = \mathrm{aggr}^{\forall}_{\mathcal{I}(s),Q[x]}\big(\lambda a.[\![F]\!]_{\nu[x \mapsto a]}\big), \qquad [\![\exists x{:}s\, F]\!]_\nu = \mathrm{aggr}^{\exists}_{\mathcal{I}(s),Q[x]}\big(\lambda a.[\![F]\!]_{\nu[x \mapsto a]}\big).$$

*Variable–dependent vs. sort–dependent quantification.* In our probabilistic semantics (Sec. 7.1), each sort $s$ comes with a fixed measure $\mu_s$ and quantification integrates with respect to $\mu_s$ (independent of the variable name). LDL instead equips each *bound variable* $x{:}s$ with its own random variable $Q[x]$ (which has a density and may differ between variables of the same sort). This design choice serves two purposes:

- It allows *simultaneous* use of different distributions on the same sort in one formula, e.g. $x{:}s$ drawn from a data distribution $Q[x]$ and $y{:}s$ drawn from an adversarial or reweighted distribution $Q[y]$.
- It enables *contextual* or *conditional* sampling: $Q[x]$ can depend on the surrounding bound context $\Gamma$ or external parameters, effectively acting as a Markov kernel $\rho_x(\cdot \mid \Gamma)$.

By contrast, our sort–based variant fixes a measure $\mu_s$ and uses the same aggregator for every variable $x{:}s$: for the chosen aggregation operators (product/infimum/LTN, etc.). This is much simpler, since it avoids the complexity of managing multiple distributions for different variables of the same sort. Moreover, it seems natural to associate the carrier sets of an interpretation with probability distributions. However, mULLER still fulfils all 3 goals stated in Sec. 1 of Ślusarz et al. (2023) and is even more modular and well-separated:

1. mULLER formally covers a sufficient fragment of first-order logic to express key properties in machine learning verification, such as robustness.
2. The syntax, semantics and pragmatics of mULLER are well-separated. And even more so than in LDL, since we clearly disentangle the concepts of signature, syntax as context-free grammar, interpretation of a NeSy framework, and Tarskian semantics of a NeSy system in an 2Mon-BLat algebra.
3. mULLER has a unified, general syntax and semantics able to express multiple different DLs and is modular on the choice of DL, by introducing the modularity of choosing the monad, the truth space, and the 2Mon-BLat algebra by choosing a NeSy framework.

*mULLER encoding of the LDL robustness example.* Following (Ślusarz et al. 2023, Ex. 3.1), we consider an image-classification setting where inputs are $28 \times 28$ grayscale images flattened to vectors in $\mathbb{R}^{784}$. Hence our input sort is $\mathrm{Vec}_{784}$ with $\mathcal{I}(\mathrm{Vec}_{784}) = \mathbb{R}^{784}$. A classifier typically outputs a vector of class scores/logits in $\mathbb{R}^m$ (e.g., $m = 10$ for MNIST). Accordingly, we take the network as a function

$$f : \mathbb{R}^{784} \longrightarrow \mathbb{R}^m \qquad (m \geq 1 \text{ fixed}).$$

The robustness property we encode states $\ell_\infty$-robustness around a reference image $\hat{x}$: whenever an input $x$ lies in the $\ell_\infty$-ball $B_\infty(\hat{x}, \varepsilon)$, the output $f(x)$ must lie in the $\ell_\infty$-ball $B_\infty(f(\hat{x}), \delta)$. This is exactly what our predicate $\mathrm{bounded}(\cdot,\cdot,\cdot)$ and the formula $\Phi_{\varepsilon,\delta,\hat{x}}$ capture below. Regarding the first goal mentioned above, we can encode the LDL robustness example as follows: Fix sorts $\mathrm{Vec}_{784}$, $\mathrm{Index}_{784}$, $\mathrm{Vec}_m$, $\mathrm{Index}_m$, Real with interpretations $\mathcal{I}(\mathrm{Vec}_{784}) = \mathbb{R}^{784}$, $\mathcal{I}(\mathrm{Index}_{784}) = \{0, \ldots, 783\}$, $\mathcal{I}(\mathrm{Vec}_m) = \mathbb{R}^m$, $\mathcal{I}(\mathrm{Index}_m) = \{0, \ldots, m-1\}$, $\mathcal{I}(\mathrm{Real}) = \mathbb{R}$. Let normal function symbols

$$\begin{aligned} &\mathrm{at}_{784} : \mathrm{Vec}_{784} \times \mathrm{Index}_{784} \to \mathrm{Real}, && \mathrm{at}_m : \mathrm{Vec}_m \times \mathrm{Index}_m \to \mathrm{Real}, \\ &\mathrm{abs} : \mathrm{Real} \to \mathrm{Real}, && \mathrm{leq} : \mathrm{Real} \times \mathrm{Real} \to \Omega, \end{aligned}$$

be given, where $\mathrm{at}_{784}(v, i)$ and $\mathrm{at}_m(w, j)$ read components, abs is absolute value, and $\mathrm{leq}(a, b)$ is the crisp predicate $[a \leq b]$. Let $f{:}\mathrm{Vec}_{784} \to \mathrm{Vec}_m$ be a computational function symbol (the network). Define the derived predicates

$$\mathrm{bounded\_in}(v, u, a) := \forall i{:}\mathrm{Index}_{784}\ \mathrm{leq}\big(\mathrm{abs}(\mathrm{at}_{784}(v, i) - \mathrm{at}_{784}(u, i)),\ a\big),$$

$$\mathrm{bounded\_out}(w, z, a) := \forall j{:}\mathrm{Index}_m\ \mathrm{leq}\big(\mathrm{abs}(\mathrm{at}_m(w, j) - \mathrm{at}_m(z, j)),\ a\big).$$

For parameters $\varepsilon, \delta \in \mathbb{R}$ and a fixed input $\hat{x} \in \mathbb{R}^{784}$, the LDL robustness property of $f$ is encoded in mULLER as the family of formulas

$$\Phi_{\varepsilon,\delta,\hat{x}} := \forall x{:}\mathrm{Vec}_{784}\ \Big(\mathrm{bounded\_in}(x, \hat{x}, \varepsilon)\ \to\ \mathrm{bounded\_out}\big(f(x), f(\hat{x}), \delta\big)\Big).$$

Under the LDL quantifier aggregator (par. 8.2) for the bound variable $x$, its semantics is

$$\begin{aligned} [\![\Phi_{\varepsilon,\delta,\hat{x}}]\!]_\nu &= \mathrm{aggr}^{\forall}_{\mathcal{I}(\mathrm{Vec}_{784}),\, Q[x]}\big(\lambda a.[\![\mathrm{bounded\_in}(x, \hat{x}, \varepsilon) \to \mathrm{bounded\_out}(f(x), f(\hat{x}), \delta)]\!]_{\nu[x \mapsto a]}\big) \\ &= \mathrm{E}_{\min}\big[g(Q[x])\big] \end{aligned}$$

where the semantics of $g(a) := \lambda a.[\![\text{bounded\_in}(x, \hat{x}, \varepsilon) \rightarrow \text{bounded\_out}(f(x), f(\hat{x}), \delta)]\!]_{\nu[x \mapsto a]}$ is given by the semantics of the implication operator in a certain 2Mon-BLat. In conclusion we have modelled our example to coincide with the LDL semantics.

## 9 Implementation of NeSy Frameworks

We have implemented NeSy frameworks in the `mULLER` library, which is available at https://github.com/cherryfunk/mULLER. We provide implementations in Haskell and Python. The library comes with predefined NeSy frameworks, but also allows users to define their own frameworks, by providing a monad, a truth value space and a double monoid bounded lattice. The library also supports the definition of interpretations. A parser transforms formulas into an abstract syntax tree. There is a function implementing Tarskian semantics, i.e. the evaluation of formulas in a given interpretation. The library also includes a module for NeSy transformations, allowing users to apply transformations between different NeSy frameworks. For simplicity, we have not implemented a sort system, which means that the implementation is untyped.[31] Also, the integration of neural networks into interpretations has not been implemented yet, but we plan to do so in the future.

The rich Haskell type system allows us to express the semantics of NeSy frameworks in a type-safe way. The Python implementation takes this as a role model, but less type-safe, because Python is dynamically typed. We start with describing the Haskell implementation. We first introduce a type class for double monoid bounded lattices. Note that for simplicity, we have not implemented the lattice structure, because it is not used in the semantics.[32]

```haskell
class TwoMonBLat a where
  top, bot :: a
  neg :: a -> a
  conj, disj, implies :: a -> a -> a
```

Based on this, we define a type class for aggregated double monoid bounded lattices:

```haskell
class TwoMonBLat a => Aggr2MonBLat s a where
  -- for a structure on b and a predicate on b, aggregate truth values a
  aggrE, aggrA :: s b -> (b -> a) -> a
```

Note that aggregation takes into account the structure of our category $\mathbf{C}$ of sets with structure, represented as `s b` here. Usually, we use finite lists, and then aggregation is just iteration of disjunction or conjunction:

```haskell
-- the mainly used Aggr2MonBLat: no additional structure (just lists) + Booleans
instance Monad t => Aggr2MonBLat [] (t Bool) where
```

[31] In Haskell, we could use type families and heterogeneous lists to implement sorted interpretations.
[32] In the future, this needs to be added. LDL quantifiers use the lattice structure. Also, to use ULLER for neuro-symbolic learning and reasoning, we need the lattice structure, because we need to be able maximise and minimise over the truth values as described in Van Krieken et al. (2024).

```
  aggrE s f = foldr disj bot $ map f s
  aggrA s f = foldr conj top $ map f s
```

For infinite aggregation, we need `s b` to be some (probability) measure. For example, for implementing the Giry monad, we can use the `Integrator` monad Tobin (2018) from the `monad-bayes` package, which represents measure spaces in a very faithful way. the Giry monad can be defined as a submonad. Values of this monad can be constructed from values of the `Integrator` monad by normalising them:

```
newtype Giry a = Giry { runGiry :: Integrator a }
fromIntegrator :: Integrator a -> Giry a
fromIntegrator m = Giry $ normalize $ lift m
```

Then, aggregation can be defined as in equation 15 and 16. Note that we need a double integral (`runIntegrator`) here, because the integrand is itself an element of $\mathcal{T}Bool$, and we need to employ the isomorphism $Giry\ Bool \cong [0,1]$ for the `Giry` monad.

```
instance Aggr2MonBLat Integrator (Giry Bool) where
  aggrA meas f =
    Giry $ integrator $ \meas_fun ->
        exp $ runIntegrator (runIntegrator (log . meas_fun) . runGiry . f) meas
  aggrE meas f =
    neg (aggrA meas (neg . f))
```

However, the `Integrator` monad is not very efficient and does not provide sampling. Therefore, we also provide an instance for the `SamplerIO` monad from the same package, which implements the Giry monad using sampling. Here, using Monte Carlo integration, we can compute the infinite aggregation up to any given precision by increasing the number of samples. Note that first a value in the domain is sampled, then the computational predicate is applied to the value, resulting in an element of $\mathcal{T}Bool$, and finally a Boolean value is sampled from that. In the case of universal quantification, we aggregate the sampled Boolean values using logical conjunction. The probability of obtaining `True` in all cases is the product of the probabilities of obtaining `True` for each case. This finite product approximates the infinite product (expressed using exp and ln) in equation 15. A similar reasoning holds for existential quantification.

```
-- Expectation-style aggregation over a distribution
-- Here we approximate via Monte Carlo with no_samples samples
no_samples = 1000
aggregation :: Monad m => ([a] -> a) -> m b -> (b -> m a) -> m a
aggregation connective dist f = do
  samples <- sequence (replicate no_samples dist)
  vals    <- mapM f samples
  return (connective vals)
instance Aggr2MonBLat SamplerIO (SamplerIO Bool) where
  aggrE = aggregation or
  aggrA = aggregation and
```

Based on this type class and the predefined type constructor class for monads, we define a type class for NeSy frameworks, as well as various instances. The instances do not to define any methods, because these have already been defined in the superclasses.

```
class (Monad t, Aggr2MonBLat s (t omega)) => NeSyFramework t s omega
-- Classical instance using identity monad, Omega is Bool
instance NeSyFramework Identity [] Bool
-- Distribution instance, Omega is Bool
instance Num prob => NeSyFramework (Dist.T prob) [] Bool
-- Non-empty powerset instance (non-determinism)
instance NeSyFramework SM.Set [] Bool
-- Giry monad instance, using SamplerIO for both aggregation and the monad
instance NeSyFramework SamplerIO SamplerIO Bool
-- Giry monad instance, using Integrator for aggregation and Giry for the monad
instance NeSyFramework Giry Integrator Bool
```

Next, we show the type definition for NeSy interpretations, which is parameterised by the monad `t`, the structure `s` of the category **C**, the truth value space `omega` and a type `a` for the universe of discourse.

```
  data Interpretation t s omega a =
  Interpretation { universe :: s a,
                   funcs :: Map.Map Ident ([a] -> a),
                   mFuncs :: Map.Map Ident ([a] -> t a),
                   preds :: Map.Map Ident ([a] -> omega),
                   mpreds :: Map.Map Ident ([a] -> t omega) }
```

We refrain from showing the same number of details of the Python implementation here, because it is much more verbose than the Haskell implementation. The Python implementation follows a structure similar to that of the Haskell implementation, but uses Python's dynamic typing and built-in data structures, building on the `pymonad` package. Here is the Python code snippet for the NeSy framework type class:

```
class NeSyFramework[_T: ParametrizedMonad, _O, _R: Aggr2MonBLat]:
    """
    Class to represent a monadic NeSy framework consisting of a monad (T),
    a set Omega acting as truth basis (O),
    and an aggregated double monoid bounded lattice (R).
    This class ensures the following runtime constraint which is not
    representable in Pythons type system:
    - _R: Aggr2MonBLat[_T[_O]]
    """
    _monad: Type[_T]
    _logic: _R
    ...
```

As in the case of Haskell, also for Python, we provide two versions of the Giry monad, one based on integration and one based on sampling. Again, the integration version is more faithful to the mathematical definition of the Giry monad, but the sampling version is more efficient. The sampling version is based on the `numpy` library. An idea for a more abstract version would be to use the `PyMC` library, such that Bayesian inference becomes possible. However, a principal problem arises. For `PyMC`, only an internal monad can be defined, providing monadic lifting for functions depending on `TensorVariables`. However, in order to obtain a monad in Python, we would need to lift functions depending on normal Python variables. This is a topic for future work.

Once the code base of the original ULLER paper Van Krieken et al. (2024) is available, our code base could be used to enhance the ULLER implementation with our modular abstractions.

## 10 Conclusion

The ULLER language Van Krieken et al. (2024) aims at a unifying foundation for neurosymbolic systems. In this paper, we have developed a new semantics for ULLER, based on Moggi's formalisation of computational effects as monads. In contrast to the original semantics, our semantics is truly modular. It is based on a notion of NeSy framework that provides the structure of the computational effects and the space of truth values. This modularity will enable a cleaner, more modular implementation of ULLER in Python and an easier integration of new frameworks, as well as a structured method of translating between different frameworks. First implementations of our mULLER framework are available in Python and Haskell, see `https://github.com/cherryfunk/mULLER`. Note that the distributional and probabilistic NeSy frameworks will make parameterized interpretations in the sense of Van Krieken et al. (2024) differentiable and that this can be integrated by using a category of differentiable manifolds and functions.

Our work suggests an analogy between ULLER's formulas $[x := m(T_1, \ldots, T_n)]F$ and Haskell's do-notation $\mathbf{do}\ x \leftarrow m(T_1, \ldots, T_n); F$ for computational effects. Inspired by this analogy, one could extend ULLER to a language with computational terms and formulas that may be nested.

### Acknowledgements

We thank Rick Adamy for the idea to use monads in the context of ULLER, Kai-Uwe Kühnberger for initiating our collaboration, Emile van Krieken for useful discussions and Björn Gehrke for helping us with the implementation in Python.

From Daniel: Big thanks to my father and family Bircks for providing me the space-time to work on this paper, Alice for reminding me to eat, and to my mother and Leilani H. Gilpin for mental support.

### References

Adámek J, Herrlich H and Strecker G (1990) *Abstract and Concrete Categories*. Wiley, New York.

Awodey S (2010) *Category Theory*. Number 52 in Oxford Logic Guides, 2. ed edition. Oxford: Oxford Univ. Press. ISBN 978-0-19-958736-0 978-0-19-923718-0.

Badreddine S, d'Avila Garcez AS, Serafini L and Spranger M (2022) Logic tensor networks. *Artif. Intell.* 303: 103649. DOI:10.1016/J.ARTINT.2021.103649. URL https://doi.org/10.1016/j.artint.2021.103649.

Badreddine S and Spranger M (2021) Extending real logic with aggregate functions. In: d'Avila Garcez AS and Jiménez-Ruiz E (eds.) *Proceedings of the 15th International Workshop on Neural-Symbolic Learning and Reasoning as part of the 1st International Joint Conference on Learning & Reasoning (IJCLR 2021), Virtual conference, October 25-27, 2021*, *CEUR Workshop Proceedings*, volume 2986. CEUR-WS.org, pp. 115–125. URL https://ceur-ws.org/Vol-2986/paper9.pdf.

Girard JY (1995) Linear Logic: Its syntax and semantics. In: Girard JY, Lafont Y and Regnier L (eds.) *Advances in Linear Logic*, 1 edition. Cambridge University Press. ISBN 978-0-521-55961-4 978-0-511-62915-0, pp. 1–42. DOI:10.1017/CBO9780511629150.002.

Hájek P (1998) *Metamathematics of Fuzzy Logic*. Trends in Logic. Dordrecht: Springer Netherlands. ISBN 978-1-4020-0370-7 978-94-011-5300-3. DOI:10.1007/978-94-011-5300-3.

Harel D, Kozen D and Tiuryn J (2001) Dynamic logic. *ACM SIGACT News* 32(1): 66–69.

Mac Lane S (1978) *Categories for the Working Mathematician*, *Graduate Texts in Mathematics*, volume 5. New York, NY: Springer New York. ISBN 978-1-4419-3123-8 978-1-4757-4721-8. DOI: 10.1007/978-1-4757-4721-8.

Manhaeve R, Dumancic S, Kimmig A, Demeester T and Raedt LD (2021) Neural probabilistic logic programming in DeepProbLog. *Artif. Intell.* 298: 103504. DOI:10.1016/J.ARTINT.2021.103504. URL https://doi.org/10.1016/j.artint.2021.103504.

Moggi E (1991) Notions of computation and monads. *Information and Computation* 93(1): 55–92. DOI:10.1016/0890-5401(91)90052-4.

Morettin P, Passerini A and Sebastiani R (2017) Efficient Weighted Model Integration via SMT-Based Predicate Abstraction. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. Melbourne, Australia: International Joint Conferences on Artificial Intelligence Organization, pp. 720–728. DOI:10.24963/ijcai.2017/100.

Mossakowski T, Schröder L and Goncharov S (2010) A generic complete dynamic logic for reasoning about purity and effects. *Formal Aspects Comput.* 22(3-4): 363–384. DOI:10.1007/S00165-010-0153-4. URL https://doi.org/10.1007/s00165-010-0153-4.

Priest G (2008) *An introduction to non-classical logic: From if to is*. Cambridge University Press.

Ślusarz N, Komendantskaya E, Daggitt ML, Stewart R and Stark K (2023) Logic of Differentiable Logics: Towards a Uniform Semantics of DL. DOI:10.48550/arXiv.2303.10650.

Smet LD, Martires PZD, Manhaeve R, Marra G, Kimmig A and Raedt LD (2023) Neural Probabilistic Logic Programming in Discrete-Continuous Domains. DOI:10.48550/arXiv.2303.04660.

Spivak DI (2014) *Category Theory for the Sciences*. Cambridge, Massachusetts: The MIT Press. ISBN 978-0-262-02813-4.

Tobin J (2018) *Embedded Domain-Specific Languages for Bayesian Modelling and Inference*. PhD Thesis, University of Auckland.

Van Krieken E, Badreddine S, Manhaeve R and Giunchiglia E (2024) ULLER: A Unified Language for Learning and Reasoning. In: Besold TR, d'Avila Garcez A, Jimenez-Ruiz E, Confalonieri R, Madhyastha P and Wagner B (eds.) *Neural-Symbolic Learning and Reasoning*, volume 14979.

Cham: Springer Nature Switzerland. ISBN 978-3-031-71166-4 978-3-031-71167-1, pp. 219–239. DOI: 10.1007/978-3-031-71167-1_12.

van Krieken E, Minervini P, Ponti E and Vergari A (2025) Neurosymbolic Reasoning Shortcuts under the Independence Assumption. DOI:10.48550/arXiv.2507.11357.

van Krieken E, Minervini P, Ponti EM and Vergari A (2024) On the Independence Assumption in Neurosymbolic Learning. DOI:10.48550/arXiv.2404.08458.

Varnai P and Dimarogonas DV (2020) On Robustness Metrics for Learning STL Tasks. In: *2020 American Control Conference (ACC)*. pp. 5394–5399. DOI:10.23919/ACC45564.2020.9147692.

Walicki M and Meldal S (1994) Multialgebras, power algebras and complete calculi of identities and inclusions. In: *Workshop on the Specification of Abstract Data Types*. Springer, pp. 453–468.

## A Brief Introduction to Category Theory

We recall some basic notions of category theory. See Spivak (2014) for an introduction with a focus on application in database theory, Awodey (2010) for a logical and type-theoretic overview, and Mac Lane (1978) as a general reference.

**Definition 19.** *A Category* $\mathbf{C}$ *consists of*

- *a class* $|\mathbf{C}|$ *of objects,*
- *for any two objects* $A, B \in |\mathbf{C}|$ *a set* $\mathbf{C}(A, B)$ *of morphisms from* $A$ *to* $B$*.* $f \in \mathbf{C}(A, B)$ *is written as* $f : A \to B$ *(not necessarily a function),*
- *for any object* $A \in |\mathbf{C}|$ *an identity morphism* $\mathrm{id}_A \in \mathbf{C}(A, A)$*, i.e.* $\mathrm{id}_A : A \to A$*,*
- *for any* $A, B, C \in |\mathbf{C}|$ *a composition operation* $\circ : \mathbf{C}(B, C) \times \mathbf{C}(A, B) \to \mathbf{C}(A, C)$*, i.e. for* $f : A \to B, g : B \to C$*, we have* $g \circ f : A \to C$*,*

*such that*

- *identities are neutral elements for composition, i.e.* $f \circ \mathrm{id}_A = f = \mathrm{id}_B \circ f$*, and*
- *composition is associative, i.e.* $(f \circ g) \circ h = f \circ (g \circ h)$*.*

Examples of categories are:

- **Set**: Sets and functions.
  - $|\mathbf{Set}| = \{M \mid M \text{ is a set}\}$
  - $\mathbf{Set}(A, B) = \{f \mid f : A \to B \text{ is a function}\}$
  - Compositions and identities of functions.
- **Meas**: measurable spaces and measurable functions.[33]

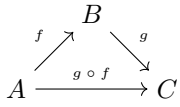
[33] More details at https://ncatlab.org/nlab/show/measurable+space.

  - $|\mathbf{Meas}| = \{(X, \Sigma_X) \mid \Sigma_X \text{ is a } \sigma\text{-algebra on } X\}$
  - $\mathbf{Meas}((X, \Sigma_X), (Y, \Sigma_Y)) = \{f : X \to Y \mid f^{-1}(B) \in \Sigma_X \text{ for all } B \in \Sigma_Y\}$
  - Compositions and identities of measurable functions.

- **Measr**: measure spaces and measure preserving functions.[34]
  - $|\mathbf{Measr}| = \{(X, \Sigma_X, \mu_X) \mid \mu_X \text{ is a measure on } (\Sigma_X, X)\}$
  - $\mathbf{Measr}((X, \Sigma_X, \mu_X), (Y, \Sigma_Y, \mu_Y)) = \{f : X \to Y \mid f \text{ is measure preserving}\}$
  - Compositions and identities of measure preserving functions.

- **Prob**: probability spaces and measure preserving functions.[35]
  - $|\mathbf{Prob}| = \{(X, \Sigma_X, \rho_X) \mid \rho_X \text{ is a probability measure on } (\Sigma_X, X)\}$
  - $\mathbf{Prob}((X, \Sigma_X, \rho_X), (Y, \Sigma_Y, \rho_Y)) = \{f : X \to Y \mid f \text{ is measure preserving}\}$
  - Compositions and identities of measure preserving functions.

Commutative diagrams are often used to visualise equalities between (compositions of) morphisms in categories. For example, the following diagram shows the composition of morphisms $f : A \to B$ and $g : B \to C$:

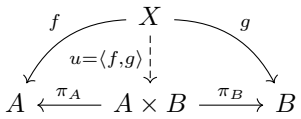



**Definition 20.** *An object* $1 \in |\mathbf{C}|$ *is called* terminal*, if for each* $A \in |\mathbf{C}|$ *there exists a unique morphism* $!_A : A \to 1_{\mathbf{C}}$.

Examples of terminal objects in **Set** are all singleton sets.

**Definition 21.** Products. *Let* $\mathcal{C}$ *be a category and let* $A, B$ *be objects in* $\mathcal{C}$. *A* product *of* $A$ *and* $B$ *is an object* $A \times B$ *together with two morphisms (called projections)* $\pi_A : A \times B \to A$ *and* $\pi_B : A \times B \to B$ *such that for any object* $X$ *with morphisms* $f : X \to A$ *and* $g : X \to B$*, there exists a unique morphism* $u : X \to A \times B$ *such that* $\pi_A \circ u = f$ *and* $\pi_B \circ u = g$*. This unique* $u$ *is noted as* $\langle f, g \rangle$ *and is called the* pairing *of* $f$ *and* $g$.

*This universal property can be depicted by the following commutative diagram:*




This easily generalises to products of finitely many objects. A category having a terminal object and binary products is called *Cartesian category*.

[34] More details at https://ncatlab.org/nlab/show/measure+space.
[35] More details at https://ncatlab.org/nlab/show/Prob.