Journal Title
XX(X):2–57

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

Mapping the Neuro-Symbolic Landscape: A Formal, Unified Framework for Design and Comparison

Ole Fenske[1]*, Daniel Romero Schellhorn[2]*, Sebastian Bader[1], Till Mossakowski [2] and Thomas Kirste [1]

Abstract

Neuro-Symbolic AI (NeSy) has produced a vast range of architectures, thus resulting in a landscape that is exceptionally difficult to navigate. Systems differ along at least four independent dimensions: the symbolic interface they expose (syntax), the mathematical space in which symbols are grounded (semantics), the inference procedures they invoke (inference), and the structural patterns by which neural and symbolic components are wired together (integration). Each of these dimensions admits a wide range of choices, and existing systems exercise them in highly heterogeneous combinations. This heterogeneity is compounded by three structural limitations that have pervaded the survey literature: (1) dimension descriptions are informal and non-comparable across systems; (2) the four central design dimensions (syntax, semantics, inference, and integration) are treated as independent, obscuring the relations between them; and (3) taxonomies developed or used in different surveys often use different terms for the same aspects or vice versa. Together, these factors have made principled comparison and design of NeSy architectures unnecessarily difficult. This paper introduces the *NeSy design space*: a franework which allows for a general comparibility by showing that several different systems can be seen as instances of the same design space. In this regard, the presented work defines for the four commonly used dimensions (e. g., syntax, semantics, inference, integration) its own taxonomy by grounding them in a unified, formal mathematical framework and at the same time makes the relations between the single dimensions explicit, thus providing a comprehensive and framework for the design and comparison of NeSy systems.

Keywords

Neurosymbolic, Neuro-Symbolic, Neural-Symbolic, Symbolic-Subsymbolic, Hybrid AI

Prepared using sagej.cls *[Version: 2017/01/17 v1.20]*

1 Introduction

It is by now well established that neither pure neural learning nor pure symbolic reasoning is sufficient for building AI systems that are simultaneously robust, interpretable, and capable of systematic generalisation (Garcez et al. 2019; Besold et al. 2017). Neural networks achieve state-of-the-art performance on high-dimensional perceptual tasks yet remain opaque, data-hungry, and brittle under distribution shift. Symbolic AI, comprising logic programming, constraint reasoning, and probabilistic graphical models, provides compositionality, interpretability, and data efficiency, but lacks the capacity to learn from raw observations at scale. Neuro-Symbolic AI (NeSy) addresses this complementarity by integrating both paradigms within a single system. The field has produced a growing body of concrete approaches (Yang et al. 2017; Cohen et al. 2017; Rocktäschel and Riedel 2017; Valkov et al. 2018; Manhaeve et al. 2018; Evans and Grefenstette 2018; Šourek et al. 2018; Mao et al. 2018; Xu et al. 2018; Dai et al. 2019; Raissi et al. 2019; Dong et al. 2019; Weber et al. 2019; Fischer et al. 2019; Si et al. 2019; Marra Giuseppe et al. 2020; Riegel et al. 2020; Huang et al. 2021; Tsamoura et al. 2021; Rajaby Faghihi et al. 2021; Marra and Kuželka 2021; Winters et al. 2021; Badreddine et al. 2022; Pryor et al. 2023; Smet et al. 2023; van Krieken et al. 2023; Shindo et al. 2023; Skryagin et al. 2023; Yang et al. 2023; Smet et al. 2024; van Krieken et al. 2024; Schellhorn and Mossakowski 2025; Derkinderen et al. 2025; Kikaj et al. 2025; Dickens et al. 2025), each combining neural and symbolic approaches in distinct ways.

This proliferation has motivated a corresponding body of surveys and position papers (Bader and Hitzler 2005; Kersting et al. 2011; Besold et al. 2017; Garcez et al. 2019; Raedt et al. 2019; De Raedt et al. 2020; Belle 2020; Ferrone and Zanzotto 2020; Garcez and Lamb 2020; Lamb et al. 2021; Sarker et al. 2021; van Bekkum et al. 2021; Dash et al. 2022; Hitzler et al. 2022; Kautz 2022; von Rueden et al. 2023; Yu et al. 2023; Bhuyan et al. 2024; Marra et al. 2024; Bougzime et al. 2025; Cui et al. 2025; Pryor and Getoor 2025; Wang et al. 2025b), each organising the landscape along different axes: application domain, direction of information flow, type of symbolic knowledge, or the stage of the learning pipeline at which integration occurs. While collectively valuable, these surveys share three structural limitations that this paper addresses.

Informal dimension descriptions. Existing surveys do characterise NeSy systems along recognisable dimensions. On the semantics side, systems are attributed Boolean, fuzzy, or probabilistic semantics (De Raedt et al. 2020; Marra et al. 2024); on the syntax side, they are classified by their choice of symbolic language — propositional logic, relational or full first-order logic, or extensions thereof (Bader and Hitzler 2005; Garcez et al. 2019; De Raedt et al.

[01]University of Rostock, Hybrid Methods in Artificial Intelligence and Machine Learning, GER
[2]University of Osnabrück, Hybrid Artificial Intelligence, GER
*These authors contributed equally to this work and share first authorship.
[0]
Corresponding author:
Ole Fenske, University of Rostock, Hybrid Methods in Artificial Intelligence and Machine Learning, GER
[0]Email: ole.fenske@uni-rostock.de

Prepared using sagej.cls

2020; Yu et al. 2023; Bhuyan et al. 2024; Marra et al. 2024; Wang et al. 2025b); inference is organised along axes such as model-theoretic versus proof-theoretic reasoning (De Raedt et al. 2020; Marra et al. 2024), or deduction, induction, and abduction (Garcez et al. 2019; Bhuyan et al. 2024) following Peirce (1878); and integration is structured around the typologies introduced by Kautz (2022) and refined in subsequent work (van Bekkum et al. 2021; von Rueden et al. 2023; Cui et al. 2025; Bougzime et al. 2025; Pryor and Getoor 2025). Yet in all these cases the characterisation remains informal: semantic distinctions are attributed without specifying a shared mathematical structure; syntactic classifications are described by example rather than by a precise language definition; inference categories are labelled by outcome rather than grounded in a formal account of the operation; and integration patterns are characterised by metaphor and data-flow diagrams. No shared formal vocabulary exists against which any of these claims could be verified or compared across systems.

Independent treatment of dimensions. The four dimensions are, moreover, treated as independent axes of classification. Surveys evaluate existing NeSy systems either on mutiple dimensions but in total isolation, or focus on a single dimension, thus falling short in making relations between the dimensions explicit. Surveys focused on integration patterns (van Bekkum et al. 2021; von Rueden et al. 2023; Cui et al. 2025; Bougzime et al. 2025; Pryor and Getoor 2025) classify systems by how neural and symbolic components communicate without specifying what kind of value flows between them — a question that is determined by the semantic choice. The same independence holds in the other direction: syntactic expressivity is assessed without noting that a richer symbolic language constrains the set of tractable inference methods (Bader and Hitzler 2005; Garcez et al. 2019; De Raedt et al. 2020; Yu et al. 2023; Bhuyan et al. 2024; Marra et al. 2024; Wang et al. 2025b), and inference taxonomies are developed without grounding them in the semantics framework they presuppose (Garcez et al. 2019; De Raedt et al. 2020; Marra et al. 2024; Bhuyan et al. 2024). The result is that the relations between dimensions, which are not incidental but constitutive of what a NeSy architecture is, remain invisible in the existing literature.

Terminological fragmentation. The surveys that do exist employ mutually inconsistent vocabularies, both within and across communities. The same structural pattern is labelled "Nesting", "Hypothesis Integration", or "Interfacing" depending on the source; "inference" denotes forward evaluation of a neural network in machine learning but logical derivation in symbolic AI; and "learning" can refer to weight optimisation, rule induction, or parameter estimation depending on the community of origin. The inverse problem is equally prevalent: the same term is applied to structurally distinct operations, making direct comparison impossible without manual re-alignment. Because no prior survey establishes a shared formal vocabulary that maps these usages onto one another, claims about the properties of NeSy systems are not transportable across the literature.

Contributions. This paper introduces the *NeSy design space*: a unified, formal framework that characterises any NeSy system along four dimensions (Table 1). The framework is deliberately scoped to logic-based symbolic representations — propositional, first-order, and their extensions to temporal, spatial, and higher-order settings. NeSy systems whose symbolic component is an automaton, a procedural or functional program, or a domain-specific language are outside the current scope; extending the framework to cover these is identified as future work (Sec. 6).

The need for such a framework is not merely our observation: Belle et al. (2024) explicitly ask whether "a unified mathematical language to bridge the gap between different system-building paradigms" is required, which was later on reiterated (Belle and Marcus 2025) — yet neither provides one. The present paper does.

Table 1. Overview of the NeSy Design Space

Dimension	Key Question	Intuition within NeSy
Syntax	What can we express symbolically?	Defines the interface (the shared symbolic language) between the paradigms.
Semantics	What is the meaning of the symbols?	Determines in what mathematical space the single symbols are interpreted.
Inference	How to infer (new) knowledge from this meaning?	Specifies how we reason and learn within the single paradigms.
Integration	How do the different paradigms communicate?	Describes how the different paradigms are connected and how information flows between them.

Syntax (Sec. 2) formalises the symbolic interface as a language $\Lambda = (\mathcal{L}, \Sigma)$, separating the domain-agnostic logical vocabulary $\mathcal{L}$ (connectives, quantifiers) from the domain-specific non-logical vocabulary Σ (sorts, function symbols, relation symbols). This separation makes the expressivity of a NeSy system explicit and enforces type safety at the interface between neural and symbolic components. We classify existing systems along a spectrum from propositional to full first-order logic, with extensions to temporal, spatial, and higher-order settings.

Semantics (Sec. 3) grounds symbolic expressions in computation via a Kleisli-categorical framework. We show that Boolean, fuzzy, and probabilistic semantics are all instances of a single structure parameterised by a category $\mathcal{C}$, a monad $\mathcal{T}$, and a truth object Ω. Neural components are modelled as parameterised Kleisli arrows within this framework, occupying a formally precise role that prior accounts have described only informally. This unification makes the treatment of different semantical choices directly commensurable across systems.

Inference (Sec. 4) provides a unified taxonomy of knowledge derivation across logical, probabilistic, and neural paradigms. Following Peirce, we organise inference along three orthogonal dimensions: type (deduction, induction, abduction), method (proof-theoretic vs. model-theoretic), and faithfulness (exact vs. approximate). Furthermore, we show that the same formal operations appear under different names in logic, probabilistic, and neural settings. This resolves the terminology clash between communities in which "inference" means forward evaluation in machine learning but logical derivation in symbolic AI, subsuming both under a single vocabulary grounded in the syntax and semantics framework of Sec. 2-3.

Integration (Sec. 5) formalises the structural relationship between paradigms via three patterns: **Nesting**, in which one paradigm invokes the other as a subroutine; **Co-routine**, in which both paradigms iteratively refine a shared state; and **Compilation**, in which one paradigm is translated offline into a form consumed by the other at runtime. Each pattern is defined precisely as a composition of Kleisli arrows, grounding and extending the Kautz

typology (Kautz 2022) in the same categorical language as the semantics dimension. This makes the dependence between dimensions precise: the monad $\mathcal{T}$ determines what kind of value flows between components, and thus which integration patterns are applicable — the integration choice is not independent of the semantic choice.

Paper structure. Section 2 develops the syntax dimension. Section 3 introduces the categorical semantics framework. Section 4 presents the inference taxonomy. Section 5 defines the three integration patterns. Section 6 concludes and identifies directions for future work.

Throughout the paper, the MNIST digit-addition task (Manhaeve et al. 2018) serves as a running example, instantiating each dimension concretely and allowing the reader to get a better intuition for the abstract definitions. Additional definitions and insights are provided in the Appendix.

2 Syntax

The syntax dimension answers a single question: what can a NeSy system express symbolically? This matters for two distinct reasons. First, it determines the system's expressive power: whether it can state universally quantified rules, refer to typed data, or construct compound terms. Second, and less obviously, it defines the formal contract at the neural–symbolic boundary: every symbol in the language has a declared type, and that type constrains what any component (neural or logical) may produce or consume. We use the term *symbolic* throughout this paper to refer specifically to this syntactic layer.

The importance of this layer has not gone unnoticed in the survey literature. Several surveys (Bader and Hitzler 2005; Garcez et al. 2019; De Raedt et al. 2020; Yu et al. 2023; Bhuyan et al. 2024; Marra et al. 2024; Wang et al. 2025b) characterise NeSy systems by their choice of symbolic language. Bader and Hitzler (2005) distinguishes propositional from first-order logic; De Raedt et al. (2020) and Marra et al. (2024) additionally consider relational logic; Garcez et al. (2019) incorporates temporal logic; Yu et al. (2023), Bhuyan et al. (2024), and Wang et al. (2025b) extend the picture further to knowledge graphs and, in the case of Bhuyan et al. (2024), also higher-order logic. Yet across all these works the treatment of syntax remains informal: syntactic choices are described intuitively or by example rather than through a shared formal definition. In particular, no prior work provides a unified framework that makes explicit what the components of a symbolic language are, how they interact, and what constraints they place on the rest of the system.

We fill this gap by formalising the symbolic interface of a NeSy system as a language $\Lambda = (\mathcal{L}, \Sigma)$, separating the domain-agnostic logical vocabulary $\mathcal{L}$ (connectives, quantifiers, truth basis) from the domain-specific non-logical vocabulary Σ (sorts, function symbols, relation symbols, variables). This separation makes the expressive power of a system explicit and provides a common vocabulary for comparison. Table 2 gives an immediate overview of the syntactic landscape of NeSy AI. Each row is a standard language type; each column is one ingredient a symbolic language can include. Reading the table top to bottom traces a progression from minimal to maximally expressive: Propositional Logic has none of the structural ingredients (every formula is a flat combination of propositions) while full First-Order Logic (FOL) includes all of them.

Table 2. Syntax classification (Sor = sort symbols, Fun = function symbols, Rel = relation symbols, Var = variable symbols, Con = connectives, Qua = quantifiers)

Syntax Type	**Sor**	**Fun**	**Rel**	**Var**	**Con**	**Qua**
Propositional (P)	×	×	0-ary	×	✓	×
Relational FOL (RFOL)	One/Many	×	✓	✓	✓	✓
|- Grounded	One/Many	×	✓	×	✓	×
|- Single Sorted	One	×	✓	✓	✓	✓
|- Many Sorted	Many	×	✓	✓	✓	✓
First-Order Logic (FOL)	One/Many	✓	✓	✓	✓	✓
|- Grounded	One/Many	✓	✓	×	✓	×
|- Single Sorted	One	✓	✓	✓	✓	✓
|- Many Sorted	Many	✓	✓	✓	✓	✓

To read this table without the formalism: sorts are data types (e.g., `Image`, `Digit`); function symbols are typed transformations between them (e.g., `digit` :`Image` → `Digit`); relation symbols are the predicates over which the system reasons (e.g., =: `Natural`^{2}); variables and quantifiers jointly enable universally stated rules ($\forall x \forall y \texttt{add}(x, y) = \texttt{digit}(x) + \texttt{digit}(y)$); and connectives are the logical operators ($\neg, \wedge, \vee, \rightarrow$) for combining formulas. The critical column for NeSy are `Fun` and `Rel`: function or relation symbols are where neural modules are plugged in. Two downstream consequences of syntax choice are worth highlighting before the formalism:

Expressivity vs. inference tractability. Moving from Propositional to Relational FOL to full FOL increases what can be stated, but also the cost of reasoning over it. Grounded variants (where variables have been instantiated against a fixed domain) reduce the inference problem to the propositional level, enabling the use of propositional solvers and model counters. This reduction is straightforward for RFOL (no function symbols): without function symbols, the Herbrand universe consists solely of constants and is finite, so all ground atoms can be enumerated into a propositional formula. For FOL with function symbols, grounding terminates only if the function symbols are non-recursive and the domain is finite. These conditions are satisfied by most practical NeSy systems, where function symbols act as data constructors over a bounded set of inputs (e.g., `digit` applied to a fixed image dataset). This is why systems like DeepProbLog (Manhaeve et al. 2018) and Markov Logic Networks (Richardson and Domingos 2006) operate on grounded propositional formulae despite being specified at the first-order level. Table 2 makes this explicit via the Grounded sub-rows: grounding eliminates **Var** and **Qua**, and under the finite-domain conditions above, collapses the specification into a (typically large) propositional formula on which exact or approximate inference is then performed.

Type safety at the neural–symbolic boundary. Declaring `digit` : `Image` → `Digit` as a function symbol commits the system to a fixed input/output type for the neural module that realises it. The symbol itself is neutral (e. g. it can be implemented as a CNN, a lookup table, or a decision tree) but its syntactic type is fixed and governs what the rest of the system may assume about it. This is where syntax acts as a contract rather than mere grammar. The remainder of this section substantiates these points formally. Section 2.1 defines the logical vocabulary $\mathcal{L}$

and non-logical vocabulary Σ precisely. Section 2.2 covers temporal, spatial, and higher-order extensions. Section 2.3 instantiates the framework on representative NeSy systems, locating each in Table 2 explicitly.

2.1 Logical and non-logical vocabulary

Formally, we decouple the logical machinery from the domain-specific schema by defining a language Λ as a tuple $(\mathcal{L}, \Sigma)$. Here, $\mathcal{L}$ is the logical vocabulary and Σ is the non-logical vocabulary, with each composed of different components. This definition separates the domain-agnostic logical scaffolding $\mathcal{L}$ (connectives, quantifiers, truth basis) from the domain-specific non-logical vocabulary Σ (sorts, functions, relations).

The logical vocabulary determines the complexity of the reasoning engine (e.g., whether the system supports quantification or merely propositional logic):

Definition 1. *A **logical vocabulary** $\mathcal{L}$ consists of a truth symbol τ, a set of connective symbols, each with a fixed arity $n \in \mathbb{N}$ (number of arguments), and (optionally) a set of quantifier symbols; all satisfying a set of logical axioms.*

Example 1. *The standard Boolean logical vocabulary is given by the truth symbol* `Bool` *(later interpreted as the truth value object $\Omega = \{0, 1\}$) and the set of connective symbols $\{\neg, \wedge, \vee, \rightarrow\}$ with arities $\mathrm{ary}(\neg) = 1$ and $\mathrm{ary}(\wedge) = \mathrm{ary}(\vee) = \mathrm{ary}(\rightarrow) = 2$; satisfying the axioms classical logic. When extended with a set of quantifier symbols $\{\forall, \exists\}$ (universal and existential quantifiers) we get the vocabulary of first-order logic.*

On the other hand, the non-logical vocabulary Σ defines the symbol sets of an application. In a NeSy context, it lists the types of data the system processes (Sorts), the transformations it applies (Functions), and the properties it can evaluate (Relations):

- *Sort* symbols provide the types for variables, functions, and relations. At the syntactic level, a sort is just a name (e.g. `Image` or `Digit`); only once an interpretation $\mathcal{I}$ is fixed does it map to a concrete domain $\mathcal{I}(S)$ (e.g. $\mathbb{R}^{28\times 28}$ or $\{0, \ldots, 9\}$).

- *Variable* symbols are typed placeholders $x : S$ used to state rules generically. Operationally, $x :$ `Image` can be instantiated by a batch of encoded images.

- *Function* symbols are syntactic constructors for terms, specified only by their input/output sorts, e.g. $\texttt{digit} : \texttt{Image} \rightarrow \texttt{Digit}$. Only after choosing an interpretation does the symbol become a concrete map $\mathcal{I}(\texttt{digit}) : \mathcal{I}(\texttt{Image}) \rightarrow \mathcal{I}(\texttt{Digit})$—in NeSy, this may be a neural module (e.g. a CNN) or a symbolic procedure.

- *Relation* symbols are syntactic constructors for formulas, e.g. $= : \texttt{Natural}^2$. Under an interpretation they denote maps into the truth-value object Ω fixed by the logical vocabulary (crisp, fuzzy, probabilistic, ...).

Definition 2. *A **non-logical vocabulary** Σ[1] is defined as a tuple of sets and type annotations:*

- Sor *is a set of sorts (types),*
- Var *is a set of variable symbols of the form* $x : S$*,*
- Fun *is a set of function symbols of the form* $f : S_1, \ldots, S_n \to T$*,*
- Rel *is a set of relation symbols of the form* $R : S_1, \ldots, S_n$*.*

To get a better intuition for this definition, we use the MNIST digit addition task, which is used as a standard benchmark across multiple NeSy frameworks (Manhaeve et al. 2018; Badreddine et al. 2022; Winters et al. 2021):

Example 2. *Given digit image pairs, the only supervision is their sum* `add`*; we must learn a classifier* `digit` *from this indirect signal. The non-logical vocabulary* Σ *is:*

- $\text{Sor} = \{\texttt{Image}, \texttt{Digit}, \texttt{Natural}\}$, $\quad \text{Var} = \{x, y \colon \texttt{Image}\}$,
- $\text{Fun} = \{\texttt{digit}\colon \texttt{Image} \to \texttt{Digit},\ \texttt{add}\colon \texttt{Image}^2 \to \texttt{Natural},\ +\colon \texttt{Digit}^2 \to \texttt{Natural}\}$,
- $\text{Rel} = \{= : \texttt{Natural}^2\}$.

The available data consists of triples (x_i, y_i, n_i) *where* n_i *is the observed sum, expressed by* `add`*. The addition axiom*

$$\forall x \, \forall y \;\; \texttt{add}(x, y) \;=\; \texttt{digit}(x) + \texttt{digit}(y)$$

decomposes the data compositionally: `add` *is observed, while* `digit` *is the unknown module to be learned. By evaluating this against observed sums, the system can learn* `digit` *via backpropagation through the symbolic constraint. Syntactically, these are merely typed symbols; only when an interpretation is fixed (Section 3) are they realized as neural networks or arithmetic operations.*

Appendix A provides the full context-free grammar for well-formed terms and formulas (Definition 18) and makes variable scoping precise via contexts (Definition 19).

2.2 Temporal, Spatial and Higher-Order Syntax

Temporal and spatial logic. Classical logic describes a single static snapshot of the world. To model dynamics such as video streams or robot trajectories, the vocabulary Λ can be extended in two ways. The first makes time explicit by adding a sort `Time` to Sor, so predicates carry a time argument (e. g. `is_alive` : `Entity` $\times$ `Time`) enabling timestamped reasoning as in event-calculus-based systems (Fenske et al. 2025). The second keeps time implicit by introducing modal temporal operators as quantifier symbols (e. g. **G** (always) and **F** (eventually) in Linear Temporal Logic (Mao et al. 2021)) which absorb the temporal dimension into the semantics

[1] This is also called a *(many-sorted) signature* in the logic/model-theory literature: the collection of non-logical symbols together with their arities/types (Goguen and Burstall 1992). We added variable symbols to not let them be defined in an "in-between the lines" or "ad-hoc" manner, like in many textbooks, e.g. Johnstone (2002).

of the operators rather than exposing a `Time` sort. Semantically, this shifts the interpretation from a single static world $\mathcal{I}$ to a trajectory $(\mathcal{I}_t)_{t\in\mathrm{Time}}$; the neurosymbolic automata of Manginas et al. (2025) illustrate this pattern, monitoring the evolving world state at each step to trigger transitions. Spatial syntax follows the same principle: a sort `Space` can be added to model spatially distributed data, or combined with `Time` into a spatiotemporal manifold for reasoning about events in four dimensions.

Higher-Order Logic. First-order logic quantifies over *individuals*; many learning tasks instead require reasoning about *relations or functions themselves* (e.g., searching for a relation R that best explains a set of observations). This motivates **Higher-Order Logic (HOL)**, in which variables range over relations and functions rather than over domain elements.

To preserve type safety, we introduce relational and functional types as first-class citizens: $[S_1, \ldots, S_n \to \tau]$ for relations and $[S_1, \ldots, S_n \to T]$ for functions of that signature. Higher-order variables are typed accordingly,

$$U : [S_1, \ldots, S_n \to \tau], \qquad F : [S_1, \ldots, S_n \to T],$$

with the bracket notation distinguishing them from first-order variables $(x : S)$ and from the fixed symbols of Def. 2. **Kinds** classify these sets ($\mathrm{Kind} := \{\mathrm{Sor}, \mathrm{Fun}, \mathrm{Rel}\}$), and **type constructors** handle structured data:

- $\mathtt{List}, \mathcal{P} : \mathrm{Sor} \to \mathrm{Sor}$ for sequences and power objects;
- $\prod, \sum : \mathrm{Sor} \times \mathrm{Sor} \to \mathrm{Sor}$ for product and sum types (records and variants);
- $(\cdot \to \cdot) : \mathrm{Sor} \times \mathrm{Sor} \to \mathrm{Sor}$ for function types $A \to B$, paired at the term level with λ-abstraction (e.g., $\lambda x{:}A.\, t : A \to B$).

This expressivity underpins **program synthesis** and **higher-order inductive logic programming (HOILP)**, whose goal is to synthesize a function or relation symbol satisfying a given specification. Some modern ILP approaches exploit higher-order (meta-level) templates to "invent" new predicates by expanding the non-logical vocabulary during training (Cropper et al. 2019), though ILP in general does not require HOL. Finally, variables ranging over sorts $(S : \mathrm{Sor})$ enable **parametric polymorphism**, yielding rules that are generic across data types (as in Haskell).

2.3 Common languages of NeSy AI

The formal vocabulary framework of the preceding sections can express a wide range of syntactic choices. As already mentioned, NeSy systems cluster around a few recurrent language types (cf. Table 2). We illustrate this with three representative systems, spelling out their logical and non-logical vocabularies explicitly.

DeepProbLog (Manhaeve et al. 2018) DeepProbLog extends ProbLog with neural predicates, requiring full first-order logic (FOL), however with a caveat: it only includes function symbols, called functors, which act purely as data constructors ($\mathrm{Fun} = \{s : \mathtt{Entity} \to \mathtt{Entity}, \mathtt{list} : \mathtt{Entity}^2 \to \mathtt{Entity}\}$), in that sense it is actually more of a relational language (RFOL). *Logical*

vocabulary $\mathcal{L}$: truth symbol `Bool`; connectives $\{\neg, \wedge, \vee, \leftarrow\}$ (the latter for definite clauses); quantifiers $\{\forall, \exists\}$ (implicit through Prolog's universal rule heads and existential rule bodies). *Non-logical vocabulary* Σ: standard Prolog is untyped, so it operates over a single implicit sort ($\text{Sor} = \{\texttt{Entity}\}$), which constitutes the Herbrand universe of terms. For example it defines Relation symbols ($\text{Rel} = \{= : \texttt{Entity}^2, \texttt{addition} : \texttt{Entity}^3\}$), and variables ($\text{Var} = \{X, Y, Z : \texttt{Entity}\}$). *Syntax type*: Single-sorted FOL. The step from RFOL (like Scallop) to FOL is characterized by the inclusion of Fun: these functors allow the creation of infinite term universes like Peano integers or recursive lists.

Scallop (Huang et al. 2021) Scallop programs are Datalog rules—a fragment of relational first-order logic (RFOL) without function symbols. *Logical vocabulary* $\mathcal{L}$: truth symbol `Bool`; connectives $\{\neg, \wedge, \vee\}$; quantifiers $\{\forall, \exists\}$ (implicit through Datalog's closed-world semantics and rule heads). *Non-logical vocabulary* Σ: a single implicit sort ($\text{Sor} = \{\texttt{Entity}\}$), no function symbols ($\text{Fun} = \emptyset$), n-ary relation symbols (e.g. $\texttt{edge} : \texttt{Entity}^2$, $\texttt{path} : \texttt{Entity}^2$), and universally quantified variables ($\text{Var} = \{x, y, z : \texttt{Entity}\}$). *Syntax type*: Single-sorted RFOL. The key syntactic constraint is the absence of function symbols: all reasoning is over ground tuples of entities. Neural predicates (e.g. a CNN classifying an image into an entity) are introduced as extensional facts whose soft probabilities are provided by external neural modules. Syntactically they are ordinary relation symbols, but their extensions are neural.

Logic Tensor Networks (Badreddine et al. 2022) LTN uses a many-sorted first-order logic where every symbol is grounded into tensor space. *Logical vocabulary* $\mathcal{L}$: truth symbol τ (interpreted as $[0, 1]$ under fuzzy semantics); connectives are fuzzy t-norms and t-conorms $\{\neg_G, \wedge_P, \vee_P, \rightarrow_R\}$ (Gödel negation, product t-norm, product t-conorm, Reichenbach implication); quantifiers $\{\forall, \exists\}$ are realized as differentiable aggregators (e.g. p-Mean). *Non-logical vocabulary* Σ: multiple sorts ($\text{Sor} = \{\texttt{Image}, \texttt{Label}, \ldots\}$), function symbols ($\text{Fun} = \{\texttt{embed} : \texttt{Image} \rightarrow \texttt{Label}\}$), relation symbols ($\text{Rel} = \{\texttt{is\_digit} : \texttt{Image} \times \texttt{Label}\}$), and variables ($\text{Var} = \{x : \texttt{Image}, l : \texttt{Label}\}$). *Syntax type*: Many-sorted FOL. The distinctive syntactic choice is the combination of many sorts with a non-classical logical vocabulary: the fuzzy connectives and aggregator-based quantifiers are part of $\mathcal{L}$, not Σ.

3 Semantics

If Syntax is the contract a NeSy system exposes, Semantics is its fulfilment: it determines what the symbolic expressions actually mean by mapping them into mathematical objects in which reasoning and learning take place. The semantics dimension answers a question Syntax deliberately leaves open. For example, $\texttt{digit} : \texttt{Image} \rightarrow \texttt{Digit}$ names a function, but says nothing about whether it is realised as a crisp classifier, a probability distribution, or a fuzzy membership score. That choice is semantic, and it has far-reaching consequences for how components can be composed, how uncertainty propagates, and what the training objective looks like.

Despite its centrality, this dimension has received the least systematic attention in the NeSy survey literature. Most surveys classify systems by their syntactic language or integration pattern while leaving their semantic choices implicit or described only informally. Two notable exceptions are De Raedt et al. (2020) and Marra et al. (2024), which catalogue the three broad

semantic families (boolean, fuzzy, and probabilistic) and relate them to the probabilistic logic programming tradition. More recently, Schellhorn and Mossakowski (2025) introduced with mULlER a categorical, monad-based framework that subsumes all three families within a single algebraic structure; the present section builds directly on that work. Table 3 organises the three broad semantic families as a choice of a category $\mathcal{C}$, a monad $\mathcal{T}$ and a truth basis Ω:

Table 3. Semantics (where $\Omega := \mathcal{I}(\tau)$ and C = the mathematical universe all components share; $\mathcal{T}$ = the monad, encoding what kind of uncertainty flows between components; Ω (Basis) = the truth value type; $\mathcal{T}\Omega$ (Space) = the actual truth space after applying $\mathcal{T}$; $\mathcal{A}$ = the axioms the truth space must satisfy).

Theory	$\mathcal{C}$ **(Category)**	$\mathcal{T}$ **(Monad)**	Ω **(Basis)**	$\mathcal{T}\Omega$ **(Space)**	$\mathcal{A}$ **(Axioms)**
Logic (Set Theory)					
Classical	**Set**	Id	$\{0,1\}$	$\{0,1\}$	Boolean
Fuzzy	**Set**	Id	$[0,1]$	$[0,1]$	BL
Three-valued	**Set**	$\mathcal{P}_{\neq\emptyset}$	$\{0,1\}$	$\{F,B,T\}$	Kleene
Four-valued	**Set**	$\mathcal{P}$	$\{0,1\}$	$\{F,N,B,T\}$	Belnap
Probability (Probability Theory)					
Finite	**Set**	$\mathcal{D}$	$\{0,1\}$	$[0,1]$	Product-BL
Continuous	**BorelMeas**	$\mathcal{G}$	$\{0,1\}$	$[0,1]$	Product-BL
Neural (Tensor Algebra)					
Tensor	**Tens**	Id	$\overline{\mathbb{R}}$	$\overline{\mathbb{R}}$	Tensor

To read this table without the formalism: $\mathcal{C}$ specifies the mathematical universe in which all data lives. Plain sets (**Set**) for logic and discrete probability, measurable spaces (**BorelMeas**) for continuous probability, tensor spaces (**Tens**) for purely neural computation. Ω determines the set of truth values we use when working in a given category, and $\mathcal{T}$ specifies what kind of computational effect flows between components: the identity monad (Id) means plain deterministic functions; the distribution monad ($\mathcal{D}$) means probability distributions over outputs; the Giry monad ($\mathcal{G}$) extends this to continuous densities. When applied to the category $\mathcal{C}$ it gives rise to the the truth space $\mathcal{T}\Omega$, which is used for the actual inference computation in a NeSy system. Classical logical semantics has crisp $\{0,1\}$ truth values, fuzzy logic uses $[0,1]$, probabilistic semantics operates on $[0,1]$ as well but arrived at differently (as an expectation over $\mathcal{D}$ rather than a graded membership). $\mathcal{A}$ specifies the algebraic axioms this truth space must satisfy, which in turn determine which logical connectives are available and how they are defined.

For comparing the semantics of different NeSy approaches this has the benfit that the different semantical choices are not longer distinct from each other, but can be viewed as choosing different realizations of the same underlying mathematical structure. For example, the difference between fuzzy and probabilistic semantics is not a fundamental one, but rather a choice of how to interpret the truth values and the connectives. This allows us to compare different NeSy systems on a more principled basis, and to understand (I) what uncertainty exactly quantifies, (II) how the classical

logical connectives are generalised to handle this uncertainty, and (III) what the resulting training objective looks like when we maximise the satisfaction of axioms under this semantics.

The remainder of this section develops the formal machinery and applies it to some representative examples. Section 3.1 defines the categorical semantics framework (category $\mathcal{C}$, monad $\mathcal{T}$, truth object Ω, and the Kleisli interpretation) precisely. Section 3.2 instantiates the framework on representative NeSy systems.

3.1 Categorical Semantics

Categories as a shared universe. The central challenge for NeSy semantics is that neural, symbolic, and probabilistic components naturally live in different mathematical worlds: a CNN operates on tensor spaces, a logic program operates on sets of ground atoms, and a probabilistic model operates on measurable spaces. Composing them naively breaks down as soon as the output type of one component does not match the input type of the next. Category theory provides the right language to resolve this uniformly. A *category* $\mathcal{C}$ consists of a collection of *objects* (data types) and, for each pair of objects, a collection of *morphisms* (computations between them), together with an associative composition law and identities. Fixing $\mathcal{C}$ enforces that every component of the system (e. g. neural modules, symbolic rules, probabilistic kernels) shares the same data types and composition rule. Three concrete choices cover the NeSy landscape:

- **Set** (plain sets and functions): the default universe for classical and fuzzy semantics, and for finite-support probability.
- **BorelMeas** (Borel measurable spaces and measurable maps): the natural universe for general probability measures and Markov kernels.
- **Tens** (tensor spaces and maps between them): the universe for neural computation; differentiability enters separately through the parameterised family $\theta \mapsto I_\theta$, not as a requirement on morphisms.

Grounding the non-logical vocabulary. Given a language $\Lambda = (L, \Sigma)$ and a category $\mathcal{C}$, an *interpretation* grounds abstract symbols into concrete mathematical objects: it is this map that turns $\texttt{digit} : \mathsf{Image} \to \mathsf{Digit}$ into a specific CNN, or $+$ into arithmetic addition. The non-logical vocabulary Σ maps directly onto the structure of $\mathcal{C}$: each sort $S \in \mathit{Sor}$ becomes an object $I(S) \in \mathrm{Ob}(\mathcal{C})$, and each function symbol $f : S_1, \ldots, S_n \to T$ becomes a morphism

$$I_\theta(f) \;:\; I(S_1) \times \cdots \times I(S_n) \;\longrightarrow\; I(T)$$

in $\mathcal{C}$, where this map may carry learnable parameters θ. Crucially, at the syntactic level $\texttt{digit} : \mathsf{Image} \to \mathsf{Digit}$ is compatible with *any* realisation (e. g. a lookup table, a decision tree, or a CNN). The choice of realisation is made here, in the interpretation.

Definition 3. Parameterised Interpretation. *A* parameterised interpretation I_θ *of a non-logical vocabulary* Σ *in a category* $\mathcal{C}$ *assigns:*

- *an object* $I(S) \in \mathrm{Ob}(\mathcal{C})$ *to each sort* $S \in \mathit{Sor}$,

- *a morphism* $I_\theta(f) : I(S_1) \times \cdots \times I(S_n) \to I(T)$ *to each function symbol* $f : S_1, \ldots, S_n \to T$*, where* $I_\theta(f)$ *may be a neural network parameterised by* θ*,*
- *a morphism* $I_\theta(R) : I(S_1) \times \cdots \times I(S_n) \to \Omega$ *to each relation symbol* $R : S_1, \ldots, S_n$*, where* Ω *is a distinguished truth object introduced below.*

The truth object and logical vocabulary. Once Σ is grounded, we need operators to *combine* truth values — to build complex formulas from atomic ones. This requires a distinguished object $\Omega := I(\tau) \in \mathrm{Ob}(\mathcal{C})$ and morphisms acting on it. The logical vocabulary L is interpreted around Ω: each connective $* : \tau^n \to \tau$ becomes a morphism $I(*) : \Omega^n \to \Omega$, and each quantifier Q becomes a family of morphisms $I(Q)_B : \Omega^B \to \Omega$ for each object B, where Ω^B is the exponential object in $\mathcal{C}$ (requiring $\mathcal{C}$ to be cartesian closed). These morphisms must satisfy the axioms of a *double monoid bounded lattice* (**2Mon-BLat**): the operations $\otimes$ (conjunction, identity $\vec{1}$) and $\oplus$ (disjunction, identity $\vec{0}$) form two monoids, and $(\Omega, \bot, \top)$ forms a bounded lattice. This is the minimal algebraic requirement for a coherent truth space, and classical Boolean, fuzzy, and probabilistic semantics are all instances of it. For more information on 2MonBLats, see App. B.

Definition 4. Logical. *A* logic $I(L)$ *for a logical vocabulary* L *assigns:*

- *a truth object* $\Omega := I(\tau) \in \mathrm{Ob}(\mathcal{C})$*,*
- *a morphism* $I(*) : \Omega^n \to \Omega$ *for each connective symbol* $*$ *of arity* n*,*
- *a family* $I(Q)_B : \Omega^B \to \Omega$ *for each quantifier symbol* Q *and object* B*,*

all satisfying the 2Mon-BLat axioms.

The classical Boolean logic sets $\Omega = \{0, 1\}$ with $I(\wedge) = \min$, $I(\vee) = \max$, $I(\neg)(v) = 1 - v$, $I(\forall)_B = \inf_{b \in B}$, $I(\exists)_B = \sup_{b \in B}$. The Tarski semantics — a plain assignment of sets and functions to symbols — is the special case $\mathcal{C} = \mathbf{Set}$ with the identity monad; we return to the general monad-based treatment momentarily.

Example 3. MNIST-Addition: deterministic grounding. *Continuing Example 2, with* $\mathcal{C} = \mathbf{Set}$ *and Boolean logic, the MNIST-Addition vocabulary is grounded as:*

- Sorts*:* $I(\mathsf{Image}) = \mathbb{R}^{28 \times 28}$, $I(\mathsf{Digit}) = \{0, \ldots, 9\}$, $I(\mathsf{Natural}) = \mathbb{N}$.
- Functions*:* $I_\theta(\texttt{digit}) : \mathbb{R}^{28 \times 28} \to \{0, \ldots, 9\}$ *is a CNN parameterised by* θ*;* $I(+) : \{0, \ldots, 9\}^2 \to \mathbb{N}$ *is standard addition.*
- Relation*:* $I(=) : \mathbb{N}^2 \to \{0, 1\}$, $I(=)(a, b) = 1$ *iff* $a = b$*.*

The addition axiom $I(\texttt{add})(x, y) = I(\texttt{digit})(x) + I(\texttt{digit})(y)$ *becomes a constraint on* θ*: the system must learn* ***digit*** *from the indirect supervision provided by observed sums alone. However, as discussed next, the argmax output of the CNN makes this constraint non-differentiable, motivating the probabilistic extension.*

The composition problem and monads. The interpretation above gives *deterministic* semantics: every symbol maps to a plain morphism. This is already powerful enough to express the Boolean MNIST-Addition system, but it breaks for the probabilistic variant that is actually used in practice. To make learning tractable, the CNN must return a *distribution* over digit classes via softmax rather than a single hard label via argmax. The output type of $I_\theta(\texttt{digit})$ is then $\mathcal{D}(\{0,\dots,9\})$ (e. g., a distribution over $\mathcal{I}(\texttt{Digit})$) and $I(+) : \{0,\dots,9\}^2 \to \mathbb{N}$ cannot directly consume it. Plain function composition no longer applies because the output type of $I_\theta(\texttt{digit})$ and the input type of $I(+)$ no longer match. This structural problem arises whenever a neural component produces uncertain outputs that must be composed with deterministic symbolic operations and it recurs across the full NeSy landscape. A monad $\mathcal{T}$ on $\mathcal{C}$ resolves this uniformly by replacing plain morphisms $X \to Y$ with *Kleisli arrows* $X \to \mathcal{T}Y$, where $\mathcal{T}Y$ is the space of effectful (e.g. uncertain) computations over X.

Definition 5. Monad / Kleisli Triple. *A* monad $(\mathcal{T}, \eta, (-)^*)$ *on a category* $\mathcal{C}$ *consists of:*

- *an endofunctor* $\mathcal{T}$ *mapping each object* X *to a computation space* $\mathcal{T}X$*,*
- *a unit* $\eta_X : X \to \mathcal{T}X$ *treating a pure value as a trivial computation (e.g. a Dirac delta),*
- *a* Kleisli extension $(-)^*$ *lifting* $f : X \to \mathcal{T}Y$ *to* $f^* : \mathcal{T}X \to \mathcal{T}Y$*, enabling composition* $g^* \circ f : X \to \mathcal{T}Z$ *for* $g : Y \to \mathcal{T}Z$*,*

subject to the monad laws: $(\eta_X)^* = \mathrm{id}_{\mathcal{T}X}$*,* $f^* \circ \eta_X = f$*, and* $(g^* \circ f)^* = g^* \circ f^*$*.*

The three components play distinct roles in resolving the composition problem. $\mathcal{T}$ designates, for each object X, a computation space $\mathcal{T}X$ already present in $\mathcal{C}$. For example $\mathcal{D}(\{0,\dots,9\})$, the 9-simplex of distributions over digit classes. It is the *type* that an effectful computation over X must return; no specific distribution is produced until a concrete Kleisli arrow is applied to a concrete input. η bridges the deterministic and effectful worlds: a plain morphism such as $I(+) : \{0,\dots,9\}^2 \to \mathbb{N}$ is lifted to $\eta_{\mathbb{N}} \circ I(+) : \{0,\dots,9\}^2 \to \mathcal{D}(\mathbb{N})$ by wrapping each output as a point mass $\delta_{d_1+d_2}$, making every deterministic morphism a valid Kleisli arrow and ensuring that purely symbolic components can participate in a probabilistic computation without modification. $(-)^*$ is the composition rule: given $f : X \to \mathcal{T}Y$ and $g : Y \to \mathcal{T}Z$, the Kleisli composition $g^* \circ f : X \to \mathcal{T}Z$ propagates the uncertainty produced by f through g. For $\mathcal{T} = \mathcal{D}$ this is exactly the law of total probability:

$$(g^* \circ f)(x)(z) \;=\; \sum_y f(x)(y) \cdot g(y)(z).$$

The monad laws, such as associativity and identity of $(-)^*$, guarantee that chaining arbitrarily many neural and symbolic components produces coherent results regardless of the order of composition. We additionally require $\mathcal{T}$ to be *commutative*: when two independent effectful computations $f : X \to \mathcal{T}Y$ and $g : Z \to \mathcal{T}W$ are run in parallel, the order in which their effects are combined does not matter. For the distribution monad $\mathcal{D}$ this is the statement that

$$\sum_y \sum_w p(y)\, q(w)\, (\cdots) = \sum_w \sum_y p(y)\, q(w)\, (\cdots)$$

— i.e., the order of summation is irrelevant. This condition is necessary for the composition of multiple neural components to be well-defined; the formal statement is given in App. C. Moreover, App. D instantiates Definition 5 explicitly for all monads in Table 3.

The **Kleisli category** $\mathrm{Kl}(\mathcal{T})$ has the same objects as $\mathcal{C}$ but takes Kleisli arrows $X \to \mathcal{T}Y$ as its morphisms, with composition governed by $(-)^*$. This is the central structural payoff: by adopting Kleisli arrows as the uniform notion of morphism, neural modules, symbolic rules, and probabilistic kernels all become arrows of the *same* category, composable by the same rule.

Definition 6. Kleisli Interpretation. *A* Kleisli interpretation I_θ *of* $\Lambda = (\mathcal{L}, \Sigma)$ *in* $(\mathcal{C}, \mathcal{T}, \Omega)$ *extends Definition 3 by replacing every plain morphism with a Kleisli arrow:*

- $I_\theta(f) : I(S_1) \times \cdots \times I(S_n) \to \mathcal{T}(I(T))$ *for each function symbol* $f : S_1, \ldots, S_n \to T$,
- $I_\theta(R) : I(S_1) \times \cdots \times I(S_n) \to \mathcal{T}\Omega$ *for each relation symbol* $R : S_1, \ldots, S_n$,

with the logical vocabulary interpreted on $\mathcal{T}\Omega$ *(the* truth space*) via the 2Mon-BLat structure. Deterministic morphisms are embedded into* $\mathrm{Kl}(\mathcal{T})$ *via* η*. The Tarski interpretation is recovered as the special case* $\mathcal{T} = \mathrm{Id}$.

The formal recursive evaluation $[\![\cdot]\!]_I$ over the grammar of Λ — the Kleisli semantics — is given in App. C.

Example 4. Distributional semantics for MNIST. *Take* $\mathcal{C} = \mathbf{Set}$, $\mathcal{T} = \mathcal{D}$ *(distribution monad),* $\Omega = \{0, 1\}$, $\mathcal{T}\Omega \cong [0, 1]$ *with Product-BL axioms:* $I(\wedge) = x \cdot y$, $I(\vee) = x + y - xy$, $I(\forall)_B = \prod_{b \in B}$. *The CNN becomes a Kleisli arrow* $I_\theta(\texttt{digit}) : \mathbb{R}^{28\times 28} \to \mathcal{D}(\{0, \ldots, 9\})$*; standard addition is lifted via* η *to* $\eta_{\mathbb{N}} \circ I(+) : \{0, \ldots, 9\}^2 \to \mathcal{D}(\mathbb{N})$*. Kleisli composition via* $(-)^*$ — *which for* $\mathcal{D}$ *is the law of total probability* — *then gives the formula probability:*

$$[\![\texttt{add}(img_0, img_1) = \texttt{digit}(img_0) + \texttt{digit}(img_1)]\!]_{\mathcal{I}} = \sum_{d_1 + d_2 = 8} \rho_0(d_1) \cdot \rho_1(d_2)$$

where $\rho_0 = I_\theta(\texttt{digit})(\texttt{img}_0)$ *and* $\rho_1 = I_\theta(\texttt{digit})(\texttt{img}_1)$*. Under the Product BL algebra (*$\otimes$ *= product), the universal quantifier* $\forall$ *is interpreted as a product* $\bigotimes = \prod$ *over the domain. Thus the full axiom evaluates to the the product of all per-pair probabilities. In practice, DeepProbLog (Manhaeve et al. 2018) maximizes* $\sum_{(x,y)} \log [\![\cdot]\!]_{\mathcal{I}}$ *via stochastic gradient descent, which is equivalent. The gradient with respect to* θ *provides the learning signal for* $\mathcal{I}_\theta(\texttt{digit})$.

What this buys. The framework makes three dependencies explicit that existing surveys leave implicit. First, the *category* $\mathcal{C}$ determines which data types are compatible — a system working in **Tens** cannot directly compose with one working in **BorelMeas** without an explicit bridge morphism. Second, the *monad* T determines what flows between components: probabilities ($\mathcal{D}$, $\mathcal{G}$), sets of alternatives ($\mathcal{P}$), or plain values (Id). Integration patterns that require a specific type of value to flow between paradigms are only well-defined for compatible choices of T — a constraint the current literature does not make precise, and one that we make explicit in Section 5. Third, the *truth-space algebra* on $T\Omega$ determines the training objective, making it possible to compare how different systems handle uncertainty at the level of their mathematical structure rather than their implementation. Moreover, the unification achieved here directly addresses an

open problem acknowledged in the community: Belle and Marcus (2025) note that whether fuzzy or probabilistic semantics is better suited for NeSy systems remains a matter of debate, without identifying a shared mathematical structure. The Kleisli-categorical framework presented here resolves this by showing both are instantiations of the same mathemaical basis.

3.2 Implications for NeSy AI

NeSy systems typically use more than one of these semantics at the same time, for example by having a symbolic component working in the **Set** category with Boolean semantics and a neural component with Tensor semantics. In such cases, we can use the categorical framework to model the interaction between these different semantic layers. This allows us to capture the full complexity of NeSy systems in a rigorous mathematical way. The key takeaway here is that the semantics for NeSy systems can be fully defined by choosing a category $\mathcal{C}$ with a monad $\mathcal{T}$ on it, as well as an interpretation of a logical vocabulary respecting the axioms $\mathcal{A}$ (e.g., Boolean, Kleene, Fuzzy, or Product algebra, or a even more general semiring).[2]

We now illustrate how three representative NeSy systems instantiate this framework, each representing a different theory of Table 3.

Logic Tensor Networks (LTNs) of Badreddine et al. (2022) adopts a **fuzzy** logic: it works in $\mathcal{C} = \mathbf{Set}$ with the identity monad $\mathcal{T} = \mathrm{Id}$, so there are no computational effects. The truth basis is $\Omega = [0,1]$ and, since $\mathrm{Id}([0,1]) = [0,1]$, the truth space coincides with the basis. The logical axioms follow a Product Real Logic BL algebra (see Appendix B): $\mathcal{I}(\wedge) = \otimes_P$ (product t-norm, $a \otimes b = a \cdot b$), $\mathcal{I}(\vee) = \oplus_P$ (probabilistic sum, $a \oplus b = a + b - ab$), $\mathcal{I}(\neg) = \neg_G$ (Gödel negation, $\neg a = 1 - a$), and $\mathcal{I}(\rightarrow) = \rightarrow_R$ (Reichenbach implication, $a \rightarrow b = 1 - a + ab$). Quantifiers are realized as p-means $A_{p\mathrm{M}}$. Concretely, LTN grounds `digit` as a predicate[3] $\mathcal{I}_\theta(\texttt{digit}) : \mathbb{R}^{28\times 28} \times \{0,\dots,9\} \to [0,1]$, defined by $\mathcal{I}_\theta(\texttt{digit})(x,d) = \mathrm{softmax}(\mathrm{CNN}_\theta(x))_d$. Since the individual digit labels are latent (only the sum n is observed), the addition axiom requires existential quantification over all digit pairs consistent with n, using a guard $d_1 + d_2 = n$ to restrict the domain:

$$\begin{aligned}&[\![\exists d_1, d_2 : d_1 + d_2 = n \ \big(\texttt{digit}(img_0, d_1) \wedge \texttt{digit}(img_1, d_2)\big)]\!]_{\mathcal{I}_\theta} \\ &= \left(\frac{1}{|S_n|} \sum_{d_1+d_2=n} \big(\mathcal{I}_\theta(\texttt{digit})(img_0, d_1) \cdot \mathcal{I}_\theta(\texttt{digit})(img_1, d_2)\big)^p\right)^{\frac{1}{p}}\end{aligned}$$

where $S_n = \{(d_1, d_2) \mid d_1 + d_2 = n\}$ and p is the aggregation exponent for the quantifier. Then the outer quantifier $\forall \mathrm{Diag}(x, y, n)$, LTN's *diagonal quantification* is applied: it zips the variables along matched training triples (x_i, y_i, n_i) and aggregates analogously.

DeepProbLog (Manhaeve et al. 2018) uses a **finite** (e.g., discrete probabilistic) semantics: $\mathcal{C} = \mathbf{Set}$, $\mathcal{T} = \mathcal{D}$ (distribution monad), $\Omega = \{0,1\}$ lifted to $\mathcal{D}\{0,1\} \cong [0,1]$, with Product BL axioms ($\mathcal{I}(\wedge) = xy$, $\mathcal{I}(\vee) = x + y - xy$, $\mathcal{I}(\forall)_B = \prod_{b\in B}$). Like LTN, `digit` is a relation symbol

[2] As descibed in more detail in Derkinderen et al. (2024); Wang et al. (2025a); Schellhorn and Mossakowski (2025).
[3] Equivalently, one may view `digit` as a function $\mathcal{I}_\theta(\texttt{digit}) : \mathbb{R}^{28\times 28} \to [0,1]^{10}$ (a softmax vector) and recover the predicate via fuzzy equality: $[\![\texttt{digit}(x) = d]\!]_{\mathcal{I}_\theta} = \mathrm{softmax}(\mathrm{CNN}_\theta(x))_d$. Both formulations yield the same computation.

$\mathcal{I}_\theta(\texttt{digit}) : \mathbb{R}^{28\times 28} \times \{0,\dots,9\} \to \mathcal{D}(\{0,1\})$, specified via a *neural annotated disjunction* (nAD) assigning $\mathrm{softmax}(\mathrm{CNN}_\theta(img))_d$ to each atom $\texttt{digit}(img, d)$, with exactly one d true per image.

Importantly, DeepProbLog does not evaluate an explicit FOL formula. It uses a *proof-theoretic* pipeline: the logic program (Horn clauses with implicit universal quantifiers) is *grounded* via SLD-resolution into a propositional Boolean expression, compiled into a Sentential Decision Diagram (SDD), on which *weighted model counting* (WMC) computes $P(q) = \sum_{C_M : q \in \omega_{C_M}} P(\omega_{C_M})$ as a sum over possible worlds (Smet et al. 2023). In our notation, this is equivalent to direct marginalization (cf. Ex. 4):

$$\begin{aligned}&[\![\exists d_1, d_2 : d_1 + d_2 = n \ \left(\texttt{digit}(img_0, d_1) \wedge \texttt{digit}(img_1, d_2)\right)]\!]_{\mathcal{I}_\theta} \\ &= \sum_{d_1+d_2=n} \mathcal{I}_\theta(\texttt{digit})(img_0, d_1) \cdot \mathcal{I}_\theta(\texttt{digit})(img_1, d_2)\end{aligned}$$

The two formulations yield the same number but differ in mechanism: DeepProbLog compiles to a propositional circuit and counts models; in Example 4 this is evaluated via Kleisli composition. Training maximizes $\sum_{(x,y)} \log P(q)$ via gradient descent through the SDD.

DeepSeaProbLog (Smet et al. 2023) DeepSeaProbLog extends DeepProbLog to **continuous probability**: $\mathcal{C} = \mathbf{BorelMeas}$, $\mathcal{T} = \mathcal{G}$ (Giry monad), same Product BL axioms. *Distributional facts* allow neural networks to output continuous distributions—e.g. a temperature sensor modeled as $\texttt{temp} \sim \mathcal{N}(\mu_\theta, \sigma^2_\theta)$ with neural parameters. The possible-worlds probability becomes an integral: $P(\omega_{C_M}) = \int [\prod_{c_i \in C_M} \mathbb{1}(c_i) \prod_{c_i \in \mathcal{C}_M \setminus C_M} \mathbb{1}(\bar{c}_i)] \, \mathrm{d}P_{\mathcal{F}_D}$, where c_i are probabilistic comparison formulas (PCFs, e.g. $\texttt{temp} > 15$).

Crucially, continuous outputs are Kleisli arrows $\mathcal{I}_\theta(m) : A \to \mathcal{G}(B)$ into the Giry monad, so the appropriate construct is the *Kleisli bind* $[v := m(\vec{T})]\, F$ rather than the existential quantifier $\exists v{:}s\, F$ used in the discrete case (Schellhorn and Mossakowski 2025): the quantifier aggregates over a domain, whereas the do-notation applies the Kleisli extension $[\![F]\!]^*$ to the specific measure output by the Kleisli arrow. Each do-binding invokes the Kleisli extension, which in $\mathcal{G}$ *is* integration:

$$\begin{aligned}&\textbf{do}\ v_1 \leftarrow m_1(\vec{T}_1);\ \textbf{do}\ v_2 \leftarrow m_2(\vec{T}_2);\ \mathbb{1}(\varphi(v_1, v_2)) \\ &= \int \int \mathbb{1}(\varphi(v_1, v_2)) \, \mathrm{d}\rho_\theta(v_2 \mid \vec{T}_2) \, \mathrm{d}\rho_\theta(v_1 \mid \vec{T}_1)\end{aligned}$$

In $\mathcal{D}$ on **Set**, the Kleisli bind is a finite sum; in $\mathcal{G}$ on **BorelMeas**, it is an integral, thus necessitating σ-algebras and integral approximation.

4 Inference

Having established what a NeSy system can express (Sec.2) and what its expressions mean (Sec.3), the inference dimension answers the third question: how does it derive new knowledge? This question immediately runs into a structural problem: the word inference means fundamentally different things in the three paradigms that NeSy systems combine. In logic, it denotes deriving conclusions from premises : a syntactic or semantic operation over formulas. In

machine learning, it denotes evaluating a trained model on new inputs, while learning occupies the role logicians call induction. In probabilistic modelling, it denotes computing marginal or posterior distributions. These are not merely terminological differences; they reflect distinct conceptions of reasoning native to each paradigm's semantic context, and no prior survey provides a shared vocabulary that maps them onto one another.

Table 4 provides that vocabulary. It organises all paradigm-specific inference procedures along three orthogonal axes that apply uniformly across paradigms:

Table 4. Taxonomy of inference methods across logical, probabilistic, and neural paradigms. (P=Proof-theoretic, M=Model-theoretic, H=Hybrid (P&M), E=Exact, A=Approximate).

Paradigm	Type	Method	Faithful.	Examples
Logical	Deduction	P	E	Resolution
		M		SAT Solving, Model Counting (MC)
	Induction	P	A	Rule Learning (ILP)
		M		Constraint-based ILP
	Abduction	P	A	Abductive Proof Search
		M		Diagnosis, ASP abduction
Probabilistic	Deduction	P	E	Proof tree evaluation
		M	E	Weighted MC, Belief Propagation
		M	A	Sampling, Variational Inference
	Induction	P		Expectation Maximization (EM)
		M	A	EM, Maximum Likelihood Estimation
		H		Structure learning
	Abduction	P	A	Viterbi decoding
		M	E/A	MAP inference
Neural	Deduction	M	E^*	Forward propagation
	Induction	M	A	Backpropagation, SGD
	Abduction	M	A	Variational Auto Encoder, GAN inversion, Normalizing Flows

Existing surveys partition this space along one axis at a time, leaving the others implicit. Yu et al. (2023) and Wang et al. (2025b) organise by functional role — *learning for reasoning*, *reasoning for learning*, *reasoning–learning* — which captures intent but says nothing about the formal structure of the procedures involved. De Raedt et al. (2020) and Marra et al. (2024) focus on the method axis, distinguishing structure learning from parameter learning and noting the proof-theoretic/model-theoretic divide. Garcez et al. (2019) and Bhuyan et al. (2024) adopt the type axis, organising by deduction, induction, and abduction. None connects all three axes, grounds them in a shared semantic context $(\mathcal{C}, \mathcal{T})$, or shows how they relate across paradigms. Table 4 does all three simultaneously. For a detailed treatment of each paradigm's inference algorithms

organized along these axes, see Appendix E. But the three axes can be also understood without these further details and formalism:

Type — *what is unknown?* Following Peirce (1878), any inference can be cast as a syllogism with three roles: a Rule (a general principle), a Case (a particular instance), and a Result (a conclusion). Deduction knows the Rule and Case and derives the Result. The direction of classical logic and neural forward passes alike. Induction knows Cases and Results and infers the Rule. This is training, ILP, and parameter estimation. Abduction knows the Rule and Result and infers the most plausible Case, which covers diagnosis, latent-space inference, and program synthesis.

Method — *how is it computed?* Proof-theoretic inference operates purely at the syntactic level, deriving new formulas from axioms via structural rules, without ever constructing an interpretation. Model-theoretic inference operates at the semantic level: it constructs or evaluates an interpretation $\mathcal{I}_\theta$ and checks formula satisfaction through $[\![\cdot]\!]_{\mathcal{I}_\theta}$. Crucially, model-theoretic inference always operates within a specific semantic context $(\mathcal{C}, \mathcal{T})$ — which is why the same inference type (e.g., deduction) manifests as exact enumeration in $(\mathbf{Set}, \mathcal{D})$ but as a differentiable forward pass in $(\mathbf{Tens}, \mathrm{Id})$.

Faithfulness — *how precisely?* An inference procedure is exact if it computes the true semantic value; approximate if it produces an estimate that may deviate. For proof-theoretic methods, exactness corresponds to soundness and completeness of the calculus. For model-theoretic methods, it corresponds to whether the evaluation algorithm computes $[\![\varphi]\!]_{\mathcal{I}_\theta}$ exactly or via sampling, variational bounds, or beam search. Neural deduction is marked Exact* because it is exact relative to the learned parameters, though those parameters are themselves the product of approximate induction.

Two NeSy-specific consequences are worth stating before the formalism.

- Most NeSy systems chain multiple inference types across semantic contexts. A single pipeline typically combines model-theoretic deduction in $(\mathbf{Tens}, \mathrm{Id})$ for neural predicate evaluation, exact or approximate model-theoretic deduction in $(\mathbf{Set}, \mathcal{D})$ for probabilistic aggregation, and approximate model-theoretic induction crossing back into $(\mathbf{Tens}, \mathrm{Id})$ for gradient-based parameter update. Table 4 makes these chains legible as sequences of rows rather than system-specific descriptions.

- The terminology clash is resolved by the type axis. What machine learning calls "inference" is deduction (Rule + Case $\to$ Result, given fixed θ). What machine learning calls "training" is induction (Case + Result $\to$ Rule, learning θ). Both are instances of the same trichotomy operating in $(\mathbf{Tens}, \mathrm{Id})$ with model-theoretic method.

The remainder of this section develops the taxonomy formally. Section 4.1 defines the three inference types. Section 4.2 defines proof-theoretic and model-theoretic methods, grounded in the semantics of Section 3. Section 4.3 defines faithfulness. Section 4.4 applies the full taxonomy to DeepProbLog, Scallop, and LTN.

4.1 Types of Inference

Our classification of inference types follows Peirce (1878), who identified three irreducible forms of reasoning by analyzing the syllogistic figures. In any syllogism, Peirce distinguished three roles:

- **Rule**: The major premise — a general principle (e.g., "all beans from this bag are white").
- **Case**: The minor premise — a particular instance (e.g., "these beans are from this bag").
- **Result**: The conclusion (e.g., "these beans are white").

Deduction proceeds from Rule and Case to Result — the standard first-figure syllogism. Peirce's key insight was that the second and third syllogistic figures yield two further, logically irreducible modes of reasoning: **abduction** infers the Case from a Rule and Result ("these beans are white and all beans from this bag are white, so these beans are probably from this bag"), while **induction** infers the Rule from Cases and Results ("these beans are from this bag and they are white, so perhaps all beans from this bag are white"). In summary:

Type	Given	Unknown	Direction
Deduction	Rule + Case	Result	Rule, Case $\rightarrow$ Result
Induction	Case + Result	Rule	Case, Result $\rightarrow$ Rule
Abduction	Rule + Result	Case	Rule, Result $\rightarrow$ Case

Later, Peirce came to view these not merely as different kinds of argument but as three *stages of inquiry*: first **abduction** proposes a hypothesis to explain a surprising fact, then **deduction** traces the necessary consequences of that hypothesis, and finally **induction** puts those consequences to the test. Each stage retains its own logical validity and is irreducible to the others, yet all three are necessary for any complete process of inquiry. This pattern maps naturally onto NeSy workflows: abduction generates candidate explanations or structures, deduction evaluates their consequences through the system's semantics, and induction tunes the model parameters from data.

Peirce's abstract trichotomy is universal, but what concretely fills the three roles varies by setting:

Paradigm	Rule	Case	Result	Induction learns...
Logic	Theory $\mathbb{T}$	Var. assignm. a	$[\![\varphi]\!]_{\mathcal{I}_\theta}(a) \in \{0,1\}$	New axioms (ILP)
Probabilistic	Distributions P	Query	Probability $\in [0,1]$	P via EM / MLE
Neural	Parameters θ	Input x	$[\![\varphi]\!]_{\mathcal{I}_\theta}(x) \in \mathcal{T}\Omega$	θ via backprop
NeSy	Model $\mathcal{I}_\theta$	Var. assignm. a	$[\![\varphi]\!]_{\mathcal{I}_\theta}(a) \in \mathcal{T}\Omega$	θ, s.t. $\mathcal{I}_\theta \models \mathbb{T}$

In classical logic, the Rule *is* the theory: deduction derives new theorems $\mathbb{T} \vdash \psi$, and induction (ILP) discovers new axioms. In neural systems, the Rule is the parameter vector θ: deduction is a forward pass, induction is training. In NeSy systems, both coexist: the interpretation $\mathcal{I}_\theta$ is the computational rule, while the axioms $\mathbb{T}$ constrain which interpretations are admissible. NeSy induction learns θ subject to $\mathbb{T}$.

Example 5. *Consider the MNIST addition system from Ex. 2 with the axiom* $\forall x \forall y \ \texttt{add}(x, y) = \texttt{digit}(x) + \texttt{digit}(y)$. *The* ***Rule*** *is the interpretation* $\mathcal{I}_\theta$*, including the CNN* $\mathcal{I}_\theta(\texttt{digit})$*; the axiom constrains which* θ *are admissible. The* ***Case*** *is the assignment* $x \mapsto img_0$, $y \mapsto img_1$, *and the* ***Result*** *is* $[\![\texttt{add}(img_0, img_1) = \texttt{digit}(img_0) + \texttt{digit}(img_1)]\!]_{\mathcal{I}_\theta}$*: in the Boolean setting this equals* 1 *(cf. Ex. 3); in the distributional setting, a probability in* $[0, 1]$ *(cf. Ex. 4).*

4.2 Methods of Inference

The type tells us *what* we are solving for, but not *how*. The "how" corresponds directly to the two layers introduced in Sec. 2 and Sec. 3: syntax and semantics. This yields two fundamentally different computational strategies. **Proof-theoretic** inference operates purely at the syntactic level, deriving new formulas from axioms via rules of inference without ever interpreting them. **Model-theoretic** inference operates at the semantic level: it requires an interpretation (what Johnstone (2002) calls a Σ-*structure*) and evaluates formulas through $[\![\cdot]\!]_{\mathcal{I}_\theta}$. An interpretation that satisfies all axioms of a theory is called a *model* of that theory.

Definition 7. Proof-theoretic inference. *Proof-theoretic inference operates purely on the syntax of* Λ *(Sec. 2), without invoking the semantic evaluation* $[\![\cdot]\!]_{\mathcal{I}_\theta}$*. Given a theory* $\mathbb{T} \subseteq \Phi_\Lambda$ *(a set of formulas taken as axioms), a deduction system provides rules of inference, such as structural rules, equality rules, and introduction/elimination rules for the connectives and quantifiers. This allows deriving new formulas from* $\mathbb{T}$*. We write* $\mathbb{T} \vdash \varphi$ *to denote that* φ *is derivable from* $\mathbb{T}$*.*

Definition 8. Model-theoretic inference. *Model-theoretic inference operates through the semantic evaluation* $[\![\cdot]\!]_{\mathcal{I}_\theta}$ *(Def. 26). An interpretation* $\mathcal{I}_\theta$ satisfies *a formula* φ *(written* $\mathcal{I}_\theta \models \varphi$*) if* $[\![\varphi]\!]_{\mathcal{I}_\theta}(a) = \top$ *for all valid assignments* a*. An interpretation that satisfies every formula in a theory* $\mathbb{T}$ *is called a* model *of* $\mathbb{T}$*.*

Example 6. *In the MNIST addition system of Ex. 5, proof-theoretic deduction derives* $\mathbb{T} \vdash \texttt{add}(\texttt{img}_0, \texttt{img}_1) = \texttt{digit}(\texttt{img}_0) + \texttt{digit}(\texttt{img}_1)$ *by universal elimination — the formula holds. If* $\mathcal{I}_\theta$ *were a perfect model of* $\mathbb{T}$*, model-theoretic evaluation of the same formula would agree (by soundness). In practice,* $[\![\texttt{add}(\texttt{img}_0, \texttt{img}_1) = \texttt{digit}(\texttt{img}_0) + \texttt{digit}(\texttt{img}_1)]\!]_{\mathcal{I}_\theta}$ *may evaluate to less than* 1*, because* $\mathcal{I}_\theta$ *does not yet perfectly satisfy the axiom. Induction (training) is exactly the process of adjusting* θ *so that* $\mathcal{I}_\theta$ *better satisfies* $\mathbb{T}$*.*

4.3 Faithfulness of Inference

The method determines the computational pathway, but not how reliably it reaches the correct answer. We say an inference procedure is *faithful* to the degree that its output matches the true semantic value. Two properties are relevant:

1. *Soundness and completeness of the calculus* (for proof-theoretic methods): A deduction system is *sound* if every derivable formula is semantically valid, meaning that $\mathbb{T} \vdash \varphi$ implies $\mathcal{I}_\theta \models \varphi$ for every model $\mathcal{I}_\theta$ of $\mathbb{T}$ (Johnstone 2002, Prop. D1.3.2). It is *complete* if every semantically valid formula is derivable.

2. *Exactness of the computation* (for both methods): Whether the procedure computes the correct answer without approximation.

Definition 9. Exact inference. *An inference is* exact *if the procedure yields the correct answer:*

- Proof-theoretic*: The calculus is sound and complete, and the search procedure terminates with the correct derivability judgment:* $\mathbb{T} \vdash \varphi$ *iff* $\mathcal{I}_\theta \models \varphi$ *for every model* $\mathcal{I}_\theta$ *of* $\mathbb{T}$.
- Model-theoretic*: The evaluation algorithm computes* $[\![\varphi]\!]_{\mathcal{I}_\theta}(a) \in \mathcal{T}\Omega$ *exactly.*

Definition 10. Approximate inference. *An inference is* approximate *if the procedure produces an estimate that may deviate from the true answer:*

- Proof-theoretic*: The calculus may be incomplete, or the search is truncated (e.g., bounded depth), yielding only a subset of derivable formulas.*
- Model-theoretic*: The evaluation produces an estimate* $\hat{y} \approx [\![\varphi]\!]_{\mathcal{I}_\theta}(a)$ *(e. g., via sampling, variational bounds, or beam search).*

Example 7. *In classical propositional logic with* $\mathbb{T} = \{p \to q,\ p\}$*, both methods derive* q*: proof-theoretically via modus ponens (*$\mathbb{T} \vdash q$*), model-theoretically by checking that every model satisfying* $\mathbb{T}$ *also satisfies* q *(*$\mathbb{T} \models q$*). Soundness guarantees that derivability implies semantic validity (*$\mathbb{T} \vdash q \Rightarrow \mathbb{T} \models q$*); completeness guarantees the converse (*$\mathbb{T} \models q \Rightarrow \mathbb{T} \vdash q$*), see Sec. 4.3.*

Example 8. *In the Boolean setting, Prolog-style resolution derives* $\mathbb{T} \vdash \texttt{add}(img_0, img_1) = 5$ *exactly (*exact proof-theoretic*). In the distributional setting, the CNN forward pass exactly (of course only up to the quality of the CNN model) computes the digit distributions* $\mathcal{I}_\theta(\texttt{digit})(img_0)$ *and* $\mathcal{I}_\theta(\texttt{digit})(img_1)$*; the result then requires aggregating over all digit pairs* (d_1, d_2) *with* $d_1 + d_2 = 5$*, weighted by those distributions. Enumerating all such pairs yields* exact model-theoretic *deduction; estimating the sum via Monte Carlo sampling yields* approximate model-theoretic *deduction.*

4.4 Common choices for NeSy AI

As already mentioned, NeSy systems rarely use a single inference technique. Instead, in line with Peirce's idea of seeing deduction, induction, and abduction as complementary stages of inference, they chain paradigm-specific steps. For example, a neural component evaluates atomic predicates (model-theoretic deduction), a logical or probabilistic engine aggregates over the theory (proof-theoretic or exact model-theoretic deduction), and gradient-based optimization adjusts the parameters (model-theoretic induction). To illustrate the multitude of possible combinations of the single inference techniques we discuss three representative NeSy systems here:

DeepProbLog (Manhaeve et al. 2018) chains three paradigm-specific inference steps, each operating in a distinct semantic context. *Deduction*: The system first uses SLD resolution (exact, proof-theoretic) to build the relevant proof trees of the ProbLog program. This step is independent of any $(\mathcal{C}, \mathcal{T})$ choice, as it operates purely on syntax. These proof trees are then compiled into a Sentential Decision Diagram (SDD), on which weighted model counting computes the query probability exactly (exact, model-theoretic deduction in $(\mathbf{Set}, \mathcal{D})$). The atomic neural predicates $\mathcal{I}_\theta(\texttt{digit})(img)$ that supply the leaf probabilities are evaluated via CNN forward passes (exact model-theoretic deduction in $(\mathbf{Tens}, \mathrm{Id}_{\mathbf{Tens}})$). *Induction*: The gradient of

the query probability with respect to θ is computed via backpropagation through the SDD and the neural networks (approximate, model-theoretic), adjusting θ so that $\mathcal{I}_\theta$ better satisfies $\mathbb{T}$. This inductive step crosses back from the probabilistic context $(\mathbf{Set}, \mathcal{D})$ into $(\mathbf{Tens}, \mathrm{Id})$, following the gradient through the Kleisli composition. *Overall*: Deduction is exact (sound+complete calculus, exact aggregation); induction is approximate (gradient-based optimisation with no convergence guarantee).

Scallop (Huang et al. 2021) uses Datalog-style provenance to make logical deduction differentiable. *Deduction*: A Datalog engine derives all consequences of the program via semi-naïve evaluation — exact, proof-theoretic deduction, independent of the semantic context. Each derived fact carries a provenance tag that records which neural predictions contributed to it, computed as CNN forward passes in $(\mathbf{Tens}, \mathrm{Id}_{\mathbf{Tens}})$ (exact, model-theoretic deduction). The provenance is then aggregated using a differentiable semiring (approximate, model-theoretic), producing a soft probability for each query within an algebraic structure that interpolates between $(\mathbf{Set}, \mathrm{Id})$ and $(\mathbf{Set}, \mathcal{D})$ depending on the semiring chosen. *Induction*: Gradients flow through the provenance semiring back to the neural components in $(\mathbf{Tens}, \mathrm{Id})$ (approximate, model-theoretic), updating θ. *Overall*: The logical derivation is exact and proof-theoretic; the aggregation and learning are approximate and model-theoretic. The key design choice is the provenance semiring, which governs the trade-off between faithfulness and differentiability and determines how tightly the aggregation step approximates inference in $(\mathbf{Set}, \mathcal{D})$.

Logic Tensor Networks (Badreddine et al. 2022) (LTN) takes a purely model-theoretic approach throughout, operating in the single semantic context $(\mathbf{Set}, \mathrm{Id})$ with truth basis $\Omega = [0, 1]$ and Product Real Logic axioms. *Deduction*: Formulas are evaluated by composing differentiable fuzzy t-norms and aggregators over the neural interpretation $\mathcal{I}_\theta$ — approximate model-theoretic deduction, since the fuzzy semantics only approximates classical logical validity. Crucially, the neural modules $\mathcal{I}_\theta(\texttt{digit})$ are computed as tensor operations in $(\mathbf{Tens}, \mathrm{Id}_{\mathbf{Tens}})$: although LTN presents a single unified semantics, the internal evaluation of neural predicates takes place in a tensor context that is then projected into $[0, 1]$ via softmax, making the boundary between $(\mathbf{Tens}, \mathrm{Id})$ and $(\mathbf{Set}, \mathrm{Id})$ implicit rather than explicit. *Induction*: The satisfaction of the theory $\mathbb{T}$ is differentiable, so θ is trained by gradient descent to maximise $[\![\mathbb{T}]\!]_{\mathcal{I}_\theta}$ (approximate, model-theoretic), following gradients back through the t-norm evaluations into $(\mathbf{Tens}, \mathrm{Id})$. *Overall*: Both deduction and induction are approximate and model-theoretic. There is no proof-theoretic component — all reasoning passes through the semantic evaluation $[\![\cdot]\!]_{\mathcal{I}_\theta}$ — and the system operates in a single nominal semantic context, though the neural/tensor substrate introduces a latent second context that is not made formally explicit.

5 Integration

The previous sections defined what a NeSy system can express (Syntax), what its expressions mean (Semantics), and how it derives new knowledge (Inference). The integration dimension answers the final question: *how are the neural and symbolic paradigms structurally wired together?*

This is the dimension that has attracted the most attention in the NeSy survey literature. The dominant reference point is Kautz (2022), whose six-type classification of NeSy architectures has become the de facto organising principle of the field, subsequently refined by Bougzime et al.

(2025); Pryor and Getoor (2025); van Bekkum et al. (2021); von Rueden et al. (2023) and Cui et al. (2025). All of these, however, share a common limitation: the choice of monad $\mathcal{T}$ — which determines what kind of value flows between components — is never made part of the integration definition itself. We close this gap by grounding the three patterns directly in the Kleisli-categorical framework of Section 3, making the inter-dimension dependency explicit: the integration pattern is not independent of the semantic choice — it is defined by it.

Kautz's six types are defined by the notation used for each architecture, where N denotes a neural component and L/P a logical or probabilistic one: **Type 1** (Symbolic Neuro symbolic) is a sequential encode–neural–decode pipeline; **Type 2** (Symbolic[Neuro]) embeds a neural component as a subroutine inside a symbolic solver; **Type 3** (Neuro | Symbolic) places both components in a coroutine relationship, each feeding the other; **Type 4** (Neuro: Symbolic → Neuro) uses symbolic knowledge to generate training data or a loss signal for a neural model; **Type 5** ($\text{Neuro}_{\text{Symbolic}}$) compiles symbolic rules directly into the neural architecture as fixed structures; and **Type 6** (Neuro[Symbolic]) embeds a symbolic reasoning engine as a subroutine inside a neural system. Our three patterns — Nesting, Co-routine, Compilation — consolidate and formally ground these six types, as shown in Table 5.

Table 5. Integration pattern classification: basis generators only. Sequential pipelines (Kautz Type 1), layered alternation, and other finite compositions of the basis are derived patterns; see §5.4. *Kautz types refer to Kautz (2022).*

Pattern	Variant	Direction	Kautz	Examples
Nesting	I: L/P[N]	L/P calls N	Type 2	DeepProbLog, Scallop
	II: N[L/P]	N calls L/P	Type 6	(rare; see meta-level)
	Meta-level	N outputs formula	—	LOGIC-LM, NS-CL
Co-routine	Iterative	N|L/P (loop)	Type 3	ABL
Compilation	To loss	$\text{N}\leftarrow_{\text{loss}}\text{L/P}$	Type 4	LTN, Semantic Loss
	To data	$\text{N}\leftarrow_{\text{data}}\text{L/P}$	Type 4	Knowledge distillation, A-NESI
	To gates	$\text{N}\leftarrow_{\text{gate}}\text{L}$	Type 5	LRNN, LTN

The three patterns differ in a single structural dimension, meaning when and how often the paradigms interact at runtime.

Nesting is a one-directional subroutine relationship: one paradigm calls the other as a black box and consumes its output. The caller drives the computation; the callee produces an intermediate result. Direction matters: Nesting I (symbolic calls neural) is the dominant pattern in logic-based NeSy, where a ProbLog or Datalog engine invokes neural modules to evaluate atomic predicates. Nesting II (neural calls symbolic) is rarer. The neural component uses a solver or simulator as a structured subroutine. The semantic constraint is that the output type of the callee must match the input type the caller expects; this is where the monad $\mathcal{T}$ choice becomes binding.

Co-routine is a bidirectional, iterative relationship: neither paradigm is subordinate. Each receives the output of the other as its input, and the two refine a shared state over multiple cycles. Iteration is genuinely new structure, a fixed-point combinator rather than finite Kleisli composition, so we treat Co-routine as a basis pattern rather than a derived one. Finite-depth

alternation arises instead from composing Nesting steps on a fixed schedule (see §5.4). Co-routine patterns arise naturally when learning and reasoning must inform each other (e. g., abductive learning, where a neural classifier and a logic engine iteratively correct each other's outputs).

Compilation is the only pattern that is not a runtime interaction. One paradigm is translated offline, before deployment, into a form the other can consume directly: a constraint loss term, a labelled dataset, or a fixed differentiable gate. At runtime, only one paradigm is active. This has a practical payoff: compilation eliminates the overhead of invoking a solver at inference time. But it has also a principled consequence: the compiled artefact bakes in a specific semantic choice that cannot be changed without recompilation.

Two cross-cutting NeSy consequences are worth stating before the formalism.

Integration pattern and semantic choice are mutually constraining. Nesting and Co-routine require the monad $\mathcal{T}$ to be compatible across both paradigms. If one component produces distributions ($\mathcal{T} = \mathcal{D}$) and the other expects plain values ($\mathcal{T} = \mathrm{Id}$), the composition is undefined without an explicit bridge. Compilation avoids this constraint by resolving the type mismatch offline, which is part of why it is so prevalent in practice.

Most real systems combine multiple patterns. NS-CL (Mao et al. 2018) combines Nesting I (perception modules feeding a probabilistic engine) with meta-level Nesting (a program generator outputting formulas). LTN (Badreddine et al. 2022) combines compilation to gates (formula structure as differentiable computation) with compilation to loss (formula satisfaction as training objective). Table 5 makes these combinations legible by treating patterns as independent structural choices rather than mutually exclusive categories.

Basis generators and derived compositions. The three patterns play the role of basis generators: every NeSy architecture in the literature can be written as a Kleisli term over them, just as every Python computation is built from `int`, `float`, `str` via composition. Categorical composition (Kleisli sequencing and do-notation (App. Def. 25), the commutator and strength (App. Def. 23, 24)) is the *generic combinator*. We therefore do not enumerate finite compositions as further primitives. Where a published system corresponds to a recurring composition (encode–neural–decode pipelines, K-fold alternating layers, ...), we collect those in §5.4 as derived examples, not as further definitions.

The remainder of this section formalises each basis pattern as a composition of Kleisli arrows. Section 5.1 defines the Nesting variants. Section 5.2 defines the iterative Co-routine basis pattern. Section 5.3 defines the three Compilation variants. Section 5.4 illustrates how the basis combines via categorical composition, with sequential pipelines and layered alternation as worked examples.

Throughout, we assume a fixed symbolic interface $\Lambda = (\mathcal{L}, \Sigma)$, a semantic category $\mathcal{C}$ with monad $\mathcal{T}$, and a (possibly parameterized) interpretation $\mathcal{I}_\theta$. Every function or relation symbol is interpreted as a Kleisli arrow (Sec. 3). We write **do** for monadic sequencing: given $f : X \to \mathcal{T}Y$ and $g : Y \to \mathcal{T}Z$,

$$(\textbf{do}\ \ y \leftarrow f(x);\ g(y)) \ :\ X \to \mathcal{T}Z$$

first runs f to produce an effectful result y, then passes it to g. For the identity monad ($\mathcal{T} = \mathrm{Id}$), this is just ordinary function composition. The formal grounding of this notation is given in Appendix C.

5.1 Nesting

Nesting is the pattern in which one paradigm *calls* the other as a subroutine. There is a clear caller–callee relationship: the outer component drives the computation and the inner component produces an intermediate result that the outer component consumes.

The first variant, corresponding to **Kautz Type 2**, is when the logical or probabilistic component drives the computation and the neural component is invoked as a subroutine.

Definition 11. Nesting I: L/P[N]. *Let $f : Z \to \mathcal{T}U$ be a neural-parameterized symbol (with parameters θ) and $g : X \times U \to \mathcal{T}Y$ a symbol with a fixed L/P interpretation. Their nesting is the composite:*

$$k_\theta(x, z) \;=\; \textbf{\textit{do}}\;\; u \leftarrow \mathcal{I}_\theta(f)(z);\; \mathcal{I}(g)(x, u).$$

The neural component f produces u (e.g., a class label or distribution), which the L/P component g then uses as input.

Example 9. DeepProbLog (Manhaeve et al. 2018). *A neural network classifies an image of a digit ($f_\theta : \texttt{Image} \to \mathcal{D}(\texttt{Digit})$), producing a distribution over digit labels. A ProbLog program (g) then reasons over these labels, e.g., computing $P(\texttt{addition}(d_1, d_2) = 5)$ by summing over all digit pairs. In our notation:*

$$k_\theta(query, img) \;=\; \textbf{\textit{do}}\;\; d \leftarrow f_\theta(img);\; g(query, d).$$

Nesting I extends directly to multiple neural subroutines (an *ensemble*). Given neural symbols $f_1, \ldots, f_n$ and an L/P combiner g:

$$k_\theta(x, z) \;=\; \textbf{do}\;\; u_1 \leftarrow f_1(z);\; \cdots;\; u_n \leftarrow f_n(z);\; g(x, u_1, \ldots, u_n).$$

This covers systems where multiple neural modules each handle a different aspect (e.g., separate perception modules in DeepProbLog (Manhaeve et al. 2018)). Conversely, the neural component can be the driver and the L/P component the subroutine (**Kautz Type 6**):

Definition 12. Nesting II: N[L/P]. *Let $g : Z \to \mathcal{T}U$ be a symbol with a fixed L/P interpretation and $f : X \times U \to \mathcal{T}Y$ a neural-parameterized symbol. Their nesting is:*

$$k_\theta(x, z) \;=\; \textbf{\textit{do}}\;\; u \leftarrow \mathcal{I}(g)(z);\; \mathcal{I}_\theta(f)(x, u).$$

The L/P component g produces an intermediate result u (e.g., a solved constraint, a retrieved fact), which the neural component f then processes.

A special case of Nesting II is the linear pipeline encode–neural–decode (**Kautz Type 1**). Because it is a finite composition of two Nesting II steps with $\mathcal{T} = \mathrm{Id}$ on the encoder and decoder, we treat it as a derived pattern and present it in §5.4.

All the nesting variants above exchange *semantic* objects, such as elements of sort interpretations $\mathcal{I}(U)$ or $\mathcal{T}(\mathcal{I}(U))$. A qualitatively different case arises when the neural component produces not a semantic value but a *syntactic* one: a formula $\varphi \in \Phi_\Lambda$ over the shared language Λ. This is a meta-level output, because formulas are themselves built from the function and relation symbols of Σ, which in turn reference sorts. They live one level above the sort interpretations.

To capture this in the Kleisli framework, one introduces a meta-sort $\texttt{Fml}$ with $\mathcal{I}(\texttt{Fml}) := \Phi_\Lambda$, so that the neural Kleisli arrow has the form $f_\theta : Z \to \mathcal{T}(\Phi_\Lambda)$. This yields a Nesting II in which the neural component generates a formula and the L/P engine evaluates it via $[\![\cdot]\!]$:

$$k_\theta(x, z) \;=\; \textbf{do}\;\; \varphi \leftarrow \mathcal{I}_\theta(f)(z);\; [\![\varphi]\!](x).$$

Example 10. LOGIC-LM (Pan et al. 2023). *A large language model (f_θ) receives a natural-language reasoning problem z and generates a first-order logic formalization $\varphi \in \Phi_\Lambda$. A deterministic logic solver then evaluates $[\![\varphi]\!]$ to produce the answer. In our notation:*

$$k_\theta(\mathit{query}, z) \;=\; \textbf{\textit{do}}\;\; \varphi \leftarrow f_\theta(z);\; [\![\varphi]\!](\mathit{query}).$$

The LLM produces syntax (a formula), not a semantic value. The semantic evaluation $[\![\varphi]\!]$ interprets this formula in a fixed L/P interpretation. The neural component is absent from the reasoning step; it merely generates *the formula that $[\![\cdot]\!]$ evaluates.*

This can be even taken one step further: the neural component can generate a *neurosymbolic program*, that itself is a Nesting I system:

Example 11. NS-CL (Mao et al. 2018). *A neural semantic parser f_{θ_s} receives a natural-language question q and generates a program $\varphi \in \Phi_\Lambda$ (meta-level Nesting II). This program is itself a Nesting I: the symbolic executor g drives the computation and calls neural concept classifiers $f_{\theta_1}, \cdots, f_{\theta_n}$ as subroutines at each atomic step with it's corresponding input $z_1, \ldots, z_n \in \mathbf{z}$. NS-CL thus nests a Nesting I inside a Nesting II:*

$$k_\theta(\mathit{query}, q, \mathbf{z}) \;=\; \textbf{\textit{do}}\;\; \varphi \leftarrow f_{\theta_s}(q);\; [\![\varphi]\!](\mathit{query}, \mathbf{z}).$$

$$[\![\varphi]\!](\mathit{query}, \mathbf{z}) \;=\; \textbf{\textit{do}}\;\; u_1 \leftarrow f_{\theta_1}(z_1);\; \cdots;\; u_n \leftarrow f_{\theta_n}(z_n);\; g(q, u_1, \ldots, u_n).$$

5.2 Co-Routine

A co-routine is the pattern in which neither paradigm is subordinate: each receives the output of the other as its input. This bidirectional dependency admits two structural variants: an *iterative form, in which the same components refine a shared state* over multiple cycles, and a *layered form, in which distinct components alternate in a directed acyclic pipeline* of fixed depth. This corresponds to **Kautz Type 3**.

Definition 13. Co-Routine I (iterative): N | L/P. *Let $f : X \times Y \to \mathcal{T}Z$ be a neural-parameterized symbol and $g : Z \to \mathcal{T}Y$ a symbol with a fixed L/P interpretation. The one-step update is:*

$$k_\theta(x, y) \;=\; \textbf{\textit{do}}\;\; z \leftarrow \mathcal{I}_\theta(f)(x, y);\; \mathcal{I}(g)(z).$$

After one step, k_θ returns an effectful state $\rho \in \mathcal{T}Y$ (not a concrete $y \in Y$). To iterate, we lift to the effectful state space:

$$F_\theta(x, \rho) \;=\; \textbf{\textit{do}}\;\; y \leftarrow \rho;\; k_\theta(x, y) \;\;:\;\; X \times \mathcal{T}Y \to \mathcal{T}Y.$$

Starting from $\rho^{(0)} = \eta(y_0)$ (wrapping an initial state via the monad unit), iteration proceeds as:

$$\rho^{(t+1)} \;:=\; F_\theta(x, \rho^{(t)}), \quad t = 0, 1, \ldots, n-1.$$

Iteration is genuinely additional structure: unlike sequential or layered patterns, it is not a finite Kleisli composition of Nesting steps but a fixed-point combinator, which is why we treat it as a basis generator in its own right.

Example 12. Abductive Learning (ABL) (Dai et al. 2019). *A neural classifier (f_θ) maps images to candidate digit labels. A logic engine (g) checks whether these labels satisfy domain constraints (e.g., arithmetic equations) and proposes corrections (abductive explanations). The corrected labels become the new training targets for f_θ, and the process repeats. Each iteration refines both the classifier's accuracy and the logical consistency of the outputs.*

A qualitatively different shape arises when distinct neural and symbolic components alternate along a *fixed, finite* schedule of depth K rather than iterating the same pair until convergence. Because such layered alternation is just a finite Kleisli composition of Nesting steps interleaved with deterministic symbolic derivations, it is a derived pattern and is presented in §5.4, with DeepGraphLog (Kikaj et al. 2025) as the worked example.

5.3 Compilation

Compilation is the pattern in which one paradigm is translated *once, offline* into a form that the other paradigm can use directly at runtime. The source paradigm is absent from the deployed system. The three variants below differ in the kind of artifact produced. Unlike Nesting and Iteration, Compilation does not act on Kleisli arrows but on the interpretation $\mathcal{I}_\theta$ itself before runtime; it is therefore irreducibly a third basis generator and not derivable from the other two. Throughout this subsection, formula evaluation is the algebra evaluation $[\![\varphi]\!]_{\mathcal{I}_\theta}$ in the truth space $\mathcal{T}\Omega$ developed in §3 (Boolean, Product-BL, fuzzy, etc.), not weighted model counting; the choice of algebra determines what *differentiable* means for the compiled artefact.

The first variant converts logical constraints into a differentiable penalty term added to the neural training objective (**Kautz Type 4**).

Definition 14. Compilation to loss: $\mathrm{N} \leftarrow_{\mathrm{loss}} \mathrm{L/P}$. *Given an axiomatization $\mathcal{A} = \{\psi_1, \ldots, \psi_M\} \subseteq \Phi_\Lambda$ and a penalty function $\mathsf{pen} : [0,1] \to \mathbb{R}_{\geq 0}$ (e.g., $\mathsf{pen}(v) = 1 - v$), the constraint loss is:*

$$J_{\mathrm{know}}(\theta) \;=\; \sum_{k=1}^{M} \mathsf{pen}\big([\![\psi_k]\!]_{\mathcal{I}_\theta}\big).$$

The total training objective combines supervised loss with the compiled constraints:

$$J(\theta) \;=\; J_{\mathrm{data}}(\theta) \;+\; \lambda\, J_{\mathrm{know}}(\theta).$$

After training, the logical constraints are not invoked at runtime.

Example 13. Semantic Loss (Xu et al. 2018) and LTN (Badreddine et al. 2022). *Xu et al. (2018) derive a semantic loss function from first principles that measures how close a neural output vector is to satisfying a propositional constraint, and use it for semi-supervised classification and structured prediction. LTN (Badreddine et al. 2022) generalizes this to first-order formulas (e.g., $\forall x.\, \mathtt{Cat}(x) \to \neg\, \mathtt{Dog}(x)$) evaluated under a fuzzy interpretation where connectives are differentiable (e.g., product t-norm). In both cases, the formula satisfaction scores*

become the constraint loss J_{know}, and gradient descent jointly trains the neural groundings to satisfy both data-driven loss and logical constraints.

Alternatively, an L/P reasoning procedure can label a dataset offline, and the neural component is then trained on this dataset (also **Kautz Type 4**).

Definition 15. Compilation to data: $\text{N} \leftarrow_{\text{data}} \text{L/P}$. *Given an axiomatization $\mathcal{A} \subseteq \Phi_\Lambda$ and a labeling procedure $\text{label}_\mathcal{A} : \mathcal{I}(X) \to \mathcal{I}(Y)$ derived from $\mathcal{A}$ (e.g., a solver or simulator), the compiled dataset is:*

$$D_{\text{train}} \;=\; \{(x,\, \text{label}_\mathcal{A}(x)) \mid x \in \mathcal{X}\}.$$

A neural predictor $f_\theta : X \to Y$ is then trained by minimizing the task loss on D_{train}. The L/P component is absent at test time.

Example 14. Knowledge distillation. *More generally, knowledge distillation (Hinton et al. 2015) compresses the outputs of a complex model (symbolic or otherwise) into a lightweight neural network. In a NeSy setting, a constraint solver or theorem prover generates labeled solutions for a combinatorial problem, and a neural network is trained to approximate these solutions from raw input features, enabling fast inference at test time without invoking the solver as done in A-NESI (van Krieken et al. 2023).*

Finally, a logical formula can be embedded directly into the neural network architecture as a fixed differentiable subcomputation (**Kautz Type 5**).

Definition 16. Compilation to gates: $\text{N} \leftarrow_{\text{gate}} \text{L}$. *For a quantifier-free formula ψ with atomic subformulas $\alpha_1, \ldots, \alpha_k$ and differentiable connectives $\mathcal{I}(*)$, define the* ***formula-evaluation gate****:*

$$g_\psi(t_1, \ldots, t_k) \;=\; \mathcal{I}(\psi)\big|_{\alpha_i \mapsto t_i},$$

the fixed map obtained by replacing each atom's truth value with the corresponding input and evaluating the connectives. The full computation is:

$$[\![\vec{x}.\psi]\!]_{\mathcal{I}_\theta} \;=\; g_\psi \circ \big\langle [\![\vec{x}.\alpha_1]\!]_{\mathcal{I}_\theta}, \ldots, [\![\vec{x}.\alpha_k]\!]_{\mathcal{I}_\theta} \big\rangle.$$

Neural modules compute the atomic truth degrees; g_ψ is a fixed differentiable layer through which gradients flow to the parameters θ.

Example 15. LRNN (Šourek et al. 2018) and LTN (Badreddine et al. 2022). *Lifted Relational Neural Networks (LRNN) (Šourek et al. 2018) compile a weighted logic program directly into a neural network architecture: each clause becomes a layer, and the logical structure determines the network's wiring. The result is a neural network whose topology is a fixed gate encoding the original logic program. LTN (Badreddine et al. 2022) provides a complementary example. Consider $\psi := P(x) \wedge \neg(Q(x) \vee R(x))$ with Łukasiewicz connectives ($\mathcal{I}(\neg)(a) = 1 - a$, $\mathcal{I}(\vee)(a,b) = \min(1, a+b)$, $\mathcal{I}(\wedge)(a,b) = \max(0, a+b-1)$). The gate $g_\psi : [0,1]^3 \to [0,1]$ is the piecewise-linear map*

$$g_\psi(t_1, t_2, t_3) = \max\big(0,\; t_1 + 1 - \min(1, t_2 + t_3) - 1\big).$$

Neural relation modules compute $t_i = [\![x.\alpha_i]\!]_{\mathcal{I}_\theta}(x)$, and g_ψ is a fixed layer in the computation graph. Note that LTN typically also uses compilation to loss (Def. 14) when training by maximizing formula satisfaction, combining two compilation variants.

5.4 Compositions of the basis patterns

The basis generators of §5.1–§5.3 (Nesting, Co-routine, Compilation) together with categorical composition (Kleisli sequencing and do-notation (App. Def. 25), the commutator and strength (App. Def. 23, 24)) suffice to express every NeSy architecture surveyed here. Most published systems are not single basis patterns but Kleisli composites of several. This subsection illustrates two recurring composition shapes: a linear encoder–neural–decoder pipeline (Kautz Type 1) and a fixed-depth alternation of distinct neural and symbolic components.

Sequential pipelines (encoder–neural–decoder). Given an encoder $\mathrm{enc}_S : \mathcal{I}^{\mathrm{rel}}(S) \to \mathbb{R}^d$, a neural map $h_\theta : \mathbb{R}^d \to \mathbb{R}^K$, and a decoder $\mathrm{dec}_T : \mathbb{R}^K \to \mathcal{T}(\mathcal{I}^{\mathrm{rel}}(T))$, the sequential integration of a symbol $f : S \to T$ is the Kleisli composite

$$\mathcal{I}^{\mathrm{rel}}_\theta(f) \;=\; \mathrm{dec}_T \circ h_\theta \circ \mathrm{enc}_S,$$

namely three Nesting II steps with $\mathcal{T} = \mathrm{Id}$ on the encoder and decoder. No new primitive is needed: the composition rule of $\mathcal{K}\ell(\mathcal{T})$ assembles the three arrows, and the bridge morphism $(\mathbf{Tens}, \mathrm{Id}) \to \mathcal{K}\ell(\mathcal{D})$ supplied by a softmax decoder injects the neural module into the Kleisli world (an instance of the NeSy transformation framework of Schellhorn and Mossakowski (2025)).

Example 16. Standard neural classification. *An image sort S is encoded into $\mathbb{R}^d$ via a CNN, a neural head h_θ maps to $\mathbb{R}^K$ logits, and softmax decodes to a distribution $\mathcal{D}(\{1, \ldots, K\})$. The symbolic interface ($f : \texttt{Image} \to \texttt{Class}$) is realised purely via this pipeline.*

Layered alternation. A second recurring composition alternates distinct neural evaluations with deterministic symbolic derivation along a fixed schedule of length K. By App. Def. 25, this is the Kleisli composite

$$\begin{aligned} k_\theta(q) = \mathbf{do}\quad & a_1 \leftarrow \mathcal{I}_\theta(f_1)(\omega_1); \quad \omega_2 := R_1^{\downarrow}(\omega_1, a_1); \\ & \ldots \\ & a_K \leftarrow \mathcal{I}_\theta(f_K)(\omega_K); \quad \mathcal{I}(g)(q, a_K), \end{aligned}$$

where each $f_k : \omega_k \to \mathcal{T}(\mathcal{A}_k)$ is a neural-parameterised Kleisli arrow on possible worlds, and $R_k^{\downarrow}(\omega_k, a_k) = \{h\theta \mid R_k \cup \omega_k \cup \{a_k\} \models h\theta,\ h\theta \text{ ground}\}$ is the deterministic ground-atom derivation of a rule set R_k, embedded into $\mathcal{K}\ell(\mathcal{T})$ via η. For $K = 1$ and $R_1 = \emptyset$ this collapses to Nesting I (Def. 11); for $K \geq 2$ it is a strict composition of Nesting II steps and is therefore not a new basis pattern. Unlike iterative Co-routine (Def. 13), which iterates the *same* pair until a fixed point, layered alternation chains *distinct* components in a directed acyclic pipeline of fixed depth K.

Example 17. DeepGraphLog Blocks World as composed Nesting (Kikaj et al. 2025). Categorical signature. *Category $\mathcal{K}\ell(\mathcal{D})$ over* **Set**, *with monad $\mathcal{T} = \mathcal{D}$ as in §3. Sorts* $\mathrm{Sor} = \{\texttt{Block}, \texttt{World}, \texttt{Bool}\}$, *where $\mathcal{I}(\texttt{World})$ is the set of finite ground-atom sets over the language. Constants $g, m, p, f : \mathbf{1} \to \texttt{Block}$ pick out four named blocks. Relation symbols* $\mathsf{glass}/1, \mathsf{on}/2, \mathsf{move}/2, \mathsf{tower}/0$ *are realised as Kleisli arrows into $\mathcal{D}\Omega$; their truth in a world ω is read off via $[\![\cdot]\!]_{\mathcal{I}}$ (App. Def. 26). The deterministic rules R_1 are morphisms in $\mathcal{K}\ell(\mathrm{Id})$, embedded into $\mathcal{K}\ell(\mathcal{D})$ via η.*

Setup. *Let $K = 2$. The initial world ω_1 collects deterministic base facts encoding the block configuration, e.g., $\omega_1 = \{\mathsf{glass}(g), \mathsf{on}(g, f), \mathsf{on}(m, p)\}$. The rules at layer 1 are*

$$R_1 = \{\ \mathsf{illegal}(X, Y) \text{:-}\ \mathsf{move}(X, Y),\ \mathsf{glass}(Y);\quad \mathsf{after\_move}(X, Y) \text{:-}\ \mathsf{move}(X, Y),\ \neg\,\mathsf{illegal}(X, Y)\ \}.$$

The neural Kleisli arrows are $\mathcal{I}_\theta(f_{\mathsf{move}}) : \mathcal{I}(\mathtt{World}) \to \mathcal{D}(\mathcal{A}_{\mathsf{move}})$ and $\mathcal{I}_\theta(f_{\mathsf{tower}}) : \mathcal{I}(\mathtt{World}) \to \mathcal{D}(\{0, 1\})$, each typically realised as a GNN whose input graph is the world ω_k.

Composition. *Instantiating the layered template,*

$$\begin{aligned} k_\theta(\mathsf{tower?}) = \textbf{\textit{do}} \quad & a_{\mathsf{move}} \leftarrow \mathcal{I}_\theta(f_{\mathsf{move}})(\omega_1);\ \omega_2 := R_1^{\downarrow}(\omega_1,\, a_{\mathsf{move}}); \\ & p_{\mathsf{tower}} \leftarrow \mathcal{I}_\theta(f_{\mathsf{tower}})(\omega_2);\quad \mathbf{return}(p_{\mathsf{tower}}). \end{aligned}$$

The first bind draws a move*-atom distribution from $\mathcal{I}_\theta(f_{\mathsf{move}})(\omega_1)$; applying $R_1^{\downarrow}$ deterministically derives which moves are* illegal *and which are* after_move*, producing ω_2. The atoms in $\omega_2 \setminus \omega_1$ are not present in ω_1: they exist only as logical consequences of R_1 given a_{move}. The second bind passes the derived world ω_2 to $\mathcal{I}_\theta(f_{\mathsf{tower}})$, which predicts whether a valid tower results.*

Where the marginalisation comes from. *The marginal over move-atom alternatives is supplied by the Kleisli extension $(-)^*$ of $\mathcal{D}$ at the first bind (App. Def. 26); no separate weighted-model-counting step is invoked. This is exactly the mechanism used in Example 4 (MNIST-Addition): the monad $\mathcal{D}$ supplies marginalisation, and the rule application $R_1^{\downarrow}$ is a deterministic morphism lifted into $\mathcal{K}\ell(\mathcal{D})$ via η.*

Sequential pipelines and layered alternation together illustrate the central claim of this section: the basis patterns of §5.1–§5.3, together with categorical composition, suffice to express the integration structure of every NeSy system surveyed here.

6 Conclusion and Future Directions

This paper introduced the NeSy design space which is organized around four key dimensions: *Syntax* formalises the symbolic interface as a logic-based language $\Lambda = (L, \Sigma)$, making expressive power explicit and enforcing type safety at the neural–symbolic boundary. *Semantics* grounds logical expressions via a Kleisli-categorical framework parameterised by (C, T, Ω), unifying Boolean, fuzzy, and probabilistic semantics as special cases and modelling neural components as parameterised Kleisli arrows within the same structure. *Inference* provides a unified three-dimensional taxonomy: type (deduction, induction, abduction), method (proof-theoretic vs. model-theoretic), and faithfulness (exact vs. approximate). These components can be applied uniformly across logical, probabilistic, and neural paradigms, resolving the cross-community terminology clash. *Integration* formalises three basic structural patterns (Nesting, Co-routine, Compilation) as compositions of Kleisli arrows, making the dependence between integration and semantic choices explicit. It was shown that a wide range of existing NeSy systems can be represented as instances of this design space, thus demonstrating its generality and unifying power.

While the proposed framework covers a wide range of existing NeSy systems, several avenues remain open.

Practical Application. The most immediate practical application of our design space is comparative analysis: given two systems positioned in the four-dimensional space, what does their relative position imply about their respective strengths and limitations (e.g., in terms of expressivity, computational efficiency, or interpretability)? We have illustrated this informally throughout the paper, but a principled comparison methodology (e. g. including metrics or partial orders on each dimension) remains future work. This would give the practitioner a principled basis for choosing the most appropriate NeSy system for a given task.

Syntax. The current treatment of temporal and spatial logics (Sec. 2.2) is deliberately introductory: it stops short of a fine-grained formal characterisation of modal temporal operators or metric spatial logics. Extending the framework to temporal, epistemic, and dynamic logics would also enable principled coverage of agent-oriented NeSy systems, a direction the community has identified as important but formally underdeveloped (Belle et al. 2024). Similarly, symbolic representations beyond logic (e.g., knowledge graphs, functional programs, domain-specific languages, etc.) are widely used in NeSy work but do not yet have a place in the current syntax taxonomy. Extending the non-logical vocabulary framework to accommodate these would broaden the reach of the design space substantially.

Semantics. At the moment the semantics of NeSy systems is formalised at the level of the neural–symbolic interface, but the internal semantics of the single components themselves is left implicit. A more complete treatment would model the internal semantics of such components as well, and also account for the interactions between the internal and external semantics (e.g., through representation bridges). This would allow the design space to capture not only how symbols are grounded in neural, logical or probabilistic representations, but also how those representations are structured and manipulated internally—for example, how compositionality is achieved within the neural component itself.

Inference. The inference taxonomy (Sec. 4) classifies methods with respect to the syntax and semantics dimensions, thus making it highly dependent on these dimensions. Diving deeper into these relationships could highlight how certain inference types or methods are more natural or efficient for certain syntactic or semantic choices, and conversely how certain syntactic or semantic choices enable or constrain certain inference types. Furthermore, being able to describe the inference in a complete representation-agnostic way (independent of the semantical/syntactical choice) would allow for a better comparability across different NeSy systems.

Integration. The three integration patterns are defined for a fixed class of compositions (Kleisli arrows of a single monad). A more abstract treatment would define more general integration *operators* (e.g., as morphisms in a suitable category of NeSy systems), allowing arbitrary combinations and the composition of patterns themselves. A closely related open question concerns the relationship between the *runtime* patterns (Nesting, Co-routine) and the *compile-time* pattern (Compilation): the current framework treats them as three alternatives, but in practice many systems apply compile-time transformations before runtime and then use a nesting or co-routine pattern at inference time. Making this temporal layering explicit—and connecting it to the inference-type taxonomy (e.g., compile-time induction vs. runtime deduction)—would give a more actionable view of the design choices available at each stage.

References

Andrieu C, de Freitas N, Doucet A and Jordan MI (2003) An Introduction to MCMC for Machine Learning. *Machine Learning* 50(1): 5–43. DOI:10.1023/A:1020281327116. URL https://doi.org/10.1023/A:1020281327116.

Apt KR and van Emden MH (1982) Contributions to the Theory of Logic Programming. *J. ACM* 29(3): 841–862. DOI:10.1145/322326.322339. URL https://dl.acm.org/doi/10.1145/322326.322339.

Bader S and Hitzler P (2005) Dimensions of Neural-symbolic Integration - A Structured Survey. DOI: 10.48550/arXiv.cs/0511042. URL http://arxiv.org/abs/cs/0511042. ArXiv:cs/0511042.

Badreddine S, d'Avila Garcez A, Serafini L and Spranger M (2022) Logic Tensor Networks. *Artificial Intelligence* 303: 103649. DOI:10.1016/j.artint.2021.103649. URL https://www.sciencedirect.com/science/article/pii/S0004370221002009.

Belle V (2020) Symbolic Logic Meets Machine Learning: A Brief Survey in Infinite Domains. In: *Scalable Uncertainty Management: 14th International Conference, SUM 2020, Bozen-Bolzano, Italy, September 23–25, 2020, Proceedings*. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-030-58448-1, pp. 3–16. DOI:10.1007/978-3-030-58449-8_1. URL https://doi.org/10.1007/978-3-030-58449-8_1.

Belle V, Fisher M, Russo A, Komendantskaya E and Nottle A (2024) Neuro-Symbolic AI + Agent Systems: A First Reflection on Trends, Opportunities and Challenges. In: Amigoni F and Sinha A (eds.) *Autonomous Agents and Multiagent Systems. Best and Visionary Papers*. Cham: Springer Nature Switzerland. ISBN 978-3-031-56255-6, pp. 180–200. DOI:10.1007/978-3-031-56255-6_10.

Belle V and Marcus G (2025) The future is neuro-symbolic: Where has it been, and where is it going? In: *Proceedings of the 40th AAAI Conference on Artificial Intelligence*. AAAI Press, pp. 1–8. URL https://www.research.ed.ac.uk/en/publications/the-future-is-neuro-symbolic-where-has-it-been-and-where-is-it-go/.

Belle V, Passerini A and Van Den Broeck G (2015) Probabilistic inference in hybrid domains by weighted model integration. In: *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15. Buenos Aires, Argentina: AAAI Press. ISBN 978-1-57735-738-4, pp. 2770–2776.

Bellodi E and Riguzzi F (2015) Structure learning of probabilistic logic programs by searching the clause space. *Theory and Practice of Logic Programming* 15(2): 169–212. DOI:10.1017/S1471068413000689. URL https://www.cambridge.org/core/journals/theory-and-practice-of-logic-programming/article/structure-learning-of-probabilistic-logic-programs-by-searching-the-clause-space/5FDD5D67FC43CC80A3F25C27E4062847.

Besold TR, Garcez Ad, Bader S, Bowman H, Domingos P, Hitzler P, Kuehnberger KU, Lamb LC, Lowd D, Lima PMV, de Penning L, Pinkas G, Poon H and Zaverucha G (2017) Neural-Symbolic Learning and Reasoning: A Survey and Interpretation. *arXiv:1711.03902 [cs]* URL http://arxiv.org/abs/1711.03902. ArXiv: 1711.03902.

Bhuyan BP, Ramdane-Cherif A, Tomar R and Singh TP (2024) Neuro-symbolic artificial intelligence: a survey. *Neural Computing and Applications* 36(21): 12809–12844. DOI:10.1007/s00521-024-09960-z. URL https://doi.org/10.1007/s00521-024-09960-z.

Bougzime O, Jabbar S, Cruz C and Demoly F (2025) Unlocking the Potential of Generative AI through Neuro-Symbolic Architectures: Benefits and Limitations. DOI:10.48550/arXiv.2502.11269. URL

http://arxiv.org/abs/2502.11269. ArXiv:2502.11269 [cs].

Cohen WW, Yang F and Mazaitis KR (2017) TensorLog: Deep Learning Meets Probabilistic DBs. DOI: 10.48550/arXiv.1707.05390. URL http://arxiv.org/abs/1707.05390. ArXiv:1707.05390 [cs].

Cropper A, Morel R and Muggleton SH (2019) Learning higher-order logic programs. DOI:10.48550/arXiv.1907.10953.

Cui Z, Gao T, Talamadupula K and Ji Q (2025) Knowledge-Augmented Deep Learning and its Applications: A Survey. *IEEE Transactions on Neural Networks and Learning Systems* 36(2): 2133–2153. DOI:10.1109/TNNLS.2023.3338619. URL https://ieeexplore.ieee.org/document/10359123.

Dai WZ, Xu Q, Yu Y and Zhou ZH (2019) Bridging machine learning and logical reasoning by abductive learning. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 253. Red Hook, NY, USA: Curran Associates Inc., pp. 2815–2826.

Darwiche A and Marquis P (2002) A knowledge compilation map. *J. Artif. Int. Res.* 17(1): 229–264.

Dash T, Chitlangia S, Ahuja A and Srinivasan A (2022) A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Scientific Reports* 12(1): 1040. DOI:10.1038/s41598-021-04590-0. URL https://www.nature.com/articles/s41598-021-04590-0.

Davis M, Logemann G and Loveland D (1962) A machine program for theorem-proving. *Commun. ACM* 5(7): 394–397. DOI:10.1145/368273.368557. URL https://doi.org/10.1145/368273.368557.

De Raedt L, Dries A, Thon I, Van den Broeck G, Verbeke M, Yang Q and Wooldridge M (2015) Inducing probabilistic relational rules from probabilistic examples. In: *Proceedings of 24th international joint conference on artificial intelligence (IJCAI)*, volume 2015. IJCAI-INT JOINT CONF ARTIF INTELL, pp. 1835–1842.

De Raedt L, Dumančić S, Manhaeve R and Marra G (2020) From Statistical Relational to Neuro-Symbolic Artificial Intelligence. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. Yokohama, Japan: International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-6-5, pp. 4943–4950. DOI:10.24963/ijcai.2020/688. URL https://www.ijcai.org/proceedings/2020/688.

De Raedt L, Kersting K, Natarajan S and Poole D (2016) *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Cham: Springer International Publishing. ISBN 978-3-031-00022-5 978-3-031-01574-8. DOI:10.1007/978-3-031-01574-8. URL https://link.springer.com/10.1007/978-3-031-01574-8.

De Raedt L, Kimmig A and Toivonen H (2007) Problog: a probabilistic prolog and its application in link discovery. In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., p. 2468–2473.

Derkinderen V, Manhaeve R, Adriaensen R, Praet LV, Smet LD, Marra G and Raedt LD (2025) The DeepLog Neurosymbolic Machine. DOI:10.48550/arXiv.2508.13697. URL http://arxiv.org/abs/2508.13697. ArXiv:2508.13697 [cs].

Derkinderen V, Manhaeve R, Zuidberg Dos Martires P and De Raedt L (2024) Semirings for probabilistic and neuro-symbolic logic programming. *International Journal of Approximate Reasoning* 171: 109130. DOI:10.1016/j.ijar.2024.109130.

Dickens C, Pryor C, Gao C, Albalak A, Augustine E, Wang W, Wright S and Getoor L (2025) A Mathematical Framework and a Suite of Learning Techniques for Neural-Symbolic Systems. DOI: 10.48550/arXiv.2407.09693. URL http://arxiv.org/abs/2407.09693. ArXiv:2407.09693 [cs].

Dong H, Mao J, Lin T, Wang C, Li L and Zhou D (2019) Neural Logic Machines. DOI:10.48550/arXiv.1904.11694. URL http://arxiv.org/abs/1904.11694. ArXiv:1904.11694 [cs].

Ellis K, Wong C, Nye M, Sable-Meyer M, Cary L, Morales L, Hewitt L, Solar-Lezama A and Tenenbaum JB (2020) DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning. DOI:10.48550/arXiv.2006.08381.

Evans R and Grefenstette E (2018) Learning Explanatory Rules from Noisy Data. *Journal of Artificial Intelligence Research* 61: 1–64. DOI:10.1613/jair.5714. URL https://jair.org/index.php/jair/article/view/11172.

Fenske O, Bader S and Kirste T (2025) Neurosymbolic Learning in Structured Probability Spaces: A Case Study. In: *Proceedings of The 19th International Conference on Neurosymbolic Learning and Reasoning*. PMLR, pp. 938–956. URL https://proceedings.mlr.press/v284/fenske25a.html.

Ferrone L and Zanzotto FM (2020) Symbolic, Distributed, and Distributional Representations for Natural Language Processing in the Era of Deep Learning: A Survey. *Frontiers in Robotics and AI* 6. DOI:10.3389/frobt.2019.00153. URL https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2019.00153/full.

Fierens D, Broeck GVd, Renkens J, Shterionov D, Gutmann B, Thon I, Janssens G and De Raedt L (2015) Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming* 15(3): 358–401. DOI:10.1017/S1471068414000076. URL http://arxiv.org/abs/1304.6810. ArXiv:1304.6810 [cs].

Fischer M, Balunovic M, Drachsler-Cohen D, Gehr T, Zhang C and Vechev M (2019) DL2: Training and Querying Neural Networks with Logic. In: *Proceedings of the 36th International Conference on Machine Learning*. PMLR, pp. 1931–1941. URL https://proceedings.mlr.press/v97/fischer19a.html.

Fung TH and Kowalski R (1997) The IFF proof procedure for abductive logic programming. *The Journal of Logic Programming* 33(2): 151–165. DOI:10.1016/S0743-1066(97)00026-5. URL https://www.sciencedirect.com/science/article/pii/S0743106697000265.

Garcez Ad, Gori M, Lamb LC, Serafini L, Spranger M and Tran SN (2019) Neural-Symbolic Computing: An Effective Methodology for Principled Integration of Machine Learning and Reasoning. *arXiv:1905.06088 [cs]* URL http://arxiv.org/abs/1905.06088. ArXiv: 1905.06088.

Garcez Ad and Lamb LC (2020) Neurosymbolic AI: The 3rd Wave. URL http://arxiv.org/abs/2012.05876. ArXiv:2012.05876 [cs].

Goguen JA and Burstall RM (1992) Institutions: Abstract model theory for specification and programming. *Journal of the ACM* 39(1): 95–146. DOI:10.1145/147508.147524.

Goodfellow I, Bengio Y and Courville A (2016) *Deep Learning*. Cambridge, MA: MIT Press. URL http://www.deeplearningbook.org.

Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A and Bengio Y (2014a) Generative adversarial nets. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14. Cambridge, MA, USA: MIT Press, p. 2672–2680.

Goodfellow IJ, Shlens J and Szegedy C (2014b) Explaining and harnessing adversarial examples. *CoRR* abs/1412.6572. URL https://api.semanticscholar.org/CorpusID:6706414.

Gutmann B, Thon I and De Raedt L (2011) Learning the parameters of probabilistic logic programs from interpretations. In: *Proceedings of ECML-PKDD 2011*, *Lecture Notes in Computer Science*, volume 6911. Springer, pp. 581–596.

Hinton G, Vinyals O and Dean J (2015) Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* .

Hitzler P, Eberhart A, Ebrahimi M, Sarker MK and Zhou L (2022) Neuro-symbolic approaches in artificial intelligence. *National Science Review* 9(6): nwac035. DOI:10.1093/nsr/nwac035. URL https://doi.org/10.1093/nsr/nwac035.

Huang J, Li Z, Chen B, Samel K, Naik M, Song L and Si X (2021) Scallop: From Probabilistic Deductive Databases to Scalable Differentiable Reasoning. In: *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc., pp. 25134–25145. URL https://proceedings.neurips.cc/paper/2021/hash/d367eef13f90793bd8121e2f675f0dc2-Abstract.html.

Johnstone PT (2002) *Sketches of an Elephant A Topos Theory Compendium*. Oxford University PressOxford. ISBN 978-0-19-851598-2 978-1-383-02287-2. DOI:10.1093/oso/9780198515982.001.0001.

Kautz HA (2022) The third AI summer: AAAI Robert S. Engelmore Memorial Lecture. *AI Magazine* 43(1): 105–125. DOI:10.1002/aaai.12036. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/aaai.12036. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aaai.12036.

Kersting K, Natarajan S and Poole D (2011) Statistical Relational AI : Logic , Probability and Computation. In: *International Conference on Logic Programming (ICLP)*. URL https://www.semanticscholar.org/paper/Statistical-Relational-AI-%3A-Logic-%2C-Probability-and-Kersting-Natarajan/49490378ad40fadbffe1e81c8b169c83a067164c.

Kikaj A, Marra G, Geerts F, Manhaeve R and Raedt LD (2025) DeepGraphLog for Layered Neurosymbolic AI. DOI:10.48550/arXiv.2509.07665. URL http://arxiv.org/abs/2509.07665. ArXiv:2509.07665 [cs].

Kingma DP and Ba J (2015) Adam: A Method for Stochastic Optimization. In: Bengio Y and LeCun Y (eds.) *Proceedings of ICLR 2015*. URL http://arxiv.org/abs/1412.6980.

Kingma DP and Welling M (2014) Auto-encoding variational bayes. In: Bengio Y and LeCun Y (eds.) *Proceedings of ICLR 2014*. URL http://dblp.uni-trier.de/db/conf/iclr/iclr2014.html#KingmaW13.

Koller D and Friedman N (2010) *Probabilistic Graphical Models: Principles and Techniques*. Adaptive Computation and Machine Learning, nachdr. edition. Cambridge, Mass.: MIT Press. ISBN 978-0-262-01319-2.

Lamb LC, d'Avila Garcez A, Gori M, Prates MO, Avelar PH and Vardi MY (2021) Graph neural networks meet neural-symbolic computing: a survey and perspective. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20. Yokohama, Yokohama, Japan. ISBN 978-0-9992411-6-5, pp. 4877–4884. URL https://dl.acm.org/doi/10.5555/3491440.3492119.

Law M, Russo A and Broda K (2014) Inductive learning of answer set programs. In: *Proceedings of the 14th European Conference on Logics in Artificial Intelligence - Volume 8761*. Berlin, Heidelberg:

Springer-Verlag. ISBN 9783319115573, p. 311–325. DOI:10.1007/978-3-319-11558-0_22. URL https://doi.org/10.1007/978-3-319-11558-0_22.

Lecun Y, Bottou L, Bengio Y and Haffner P (1998) Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11): 2278–2324. DOI:10.1109/5.726791.

Lehmann J (2009) Dl-learner: Learning concepts in description logics. *J. Mach. Learn. Res.* 10: 2639–2642.

Manginas N, Paliouras G and De Raedt L (2025) Nesya: neurosymbolic automata. In: *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI '25. ISBN 978-1-956792-06-5. DOI:10.24963/ijcai.2025/662. URL https://doi.org/10.24963/ijcai.2025/662.

Manhaeve R, Dumancic S, Kimmig A, Demeester T and De Raedt L (2018) DeepProbLog: Neural Probabilistic Logic Programming. In: *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc. URL https://papers.nips.cc/paper/2018/hash/dc5d637ed5e62c36ecb73b654b05ba2a-Abstract.html.

Mao J, Gan C, Kohli P, Tenenbaum JB and Wu J (2018) The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. URL https://openreview.net/forum?id=rJgMlhRctm.

Mao J, Luo Z, Gan C, Tenenbaum JB, Wu J, Kaelbling LP and Ullman TD (2021) Temporal and object quantification networks. In: Zhou ZH (ed.) *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. International Joint Conferences on Artificial Intelligence Organization, pp. 2804–2811. DOI:10.24963/ijcai.2021/386. URL https://doi.org/10.24963/ijcai.2021/386. Main Track.

Marques-Silva JaP and Sakallah KA (1999) Grasp: A search algorithm for propositional satisfiability. *IEEE Trans. Comput.* 48(5): 506–521. DOI:10.1109/12.769433. URL https://doi.org/10.1109/12.769433.

Marra G, Dumančić S, Manhaeve R and De Raedt L (2024) From statistical relational to neurosymbolic artificial intelligence: A survey. *Artificial Intelligence* 328: 104062. DOI:10.1016/j.artint.2023.104062. URL https://www.sciencedirect.com/science/article/pii/S0004370223002084.

Marra G and Kuželka O (2021) Neural markov logic networks. In: *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*. PMLR, pp. 908–917. URL https://proceedings.mlr.press/v161/marra21a.html.

Marra Giuseppe, Diligenti Michelangelo, Giannini Francesco, Gori Marco and Maggini Marco (2020) Relational Neural Machines. In: *Frontiers in Artificial Intelligence and Applications*. IOS Press. DOI:10.3233/FAIA200237. URL https://www.medra.org/servlet/aliasResolver?alias=iospressISBN&isbn=978-1-64368-100-9&spage=1340&doi=10.3233/FAIA200237.

Muggleton S (1995) Inverse entailment and progol. *New Gen. Comput.* 13(3–4): 245–286. DOI:10.1007/BF03037227. URL https://doi.org/10.1007/BF03037227.

Murphy KP, Weiss Y and Jordan MI (1999) Loopy belief propagation for approximate inference: an empirical study. In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, UAI'99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-614-2, pp. 467–475. URL https://dl.acm.org/doi/10.5555/2073796.2073849.

Pan L, Albalak A, Wang X and Wang WY (2023) Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295* .

Peirce CS (1878) Deduction, induction, and hypothesis. *Popular Science Monthly* 13: 470–482. URL https://en.wikisource.org/wiki/Popular_Science_Monthly/Volume_13/August_1878/Illustrations_of_the_Logic_of_Science_VI.

Pryor C, Dickens C, Augustine E, Albalak A, Wang W and Getoor L (2023) NeuPSL: Neural Probabilistic Soft Logic. DOI:10.48550/arXiv.2205.14268. URL http://arxiv.org/abs/2205.14268. ArXiv:2205.14268 [cs].

Pryor C and Getoor L (2025) Neural-Symbolic Architectural Axioms of Integration: A Manifesto. In: *Proceedings of The 19th International Conference on Neurosymbolic Learning and Reasoning*. PMLR, pp. 322–342. URL https://proceedings.mlr.press/v284/pryor25a.html.

Quinlan JR (1990) Learning logical definitions from relations. *Machine Learning* 5: 239–266. URL https://api.semanticscholar.org/CorpusID:6746439.

Raedt LD, Manhaeve R, Dumancic S, Demeester T and Kimmig A (2019) Neuro-Symbolic = Neural + Logical + Probabilistic. In: *NeSy@IJCAI*. pp. 1–10. URL https://api.semanticscholar.org/CorpusID:201872637.

Raissi M, Perdikaris P and Karniadakis G (2019) Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics* 378: 686–707. DOI:10.1016/j.jcp.2018.10.045. URL https://linkinghub.elsevier.com/retrieve/pii/S0021999118307125.

Rajaby Faghihi H, Guo Q, Uszok A, Nafar A and Kordjamshidi P (2021) DomiKnowS: A Library for Integration of Symbolic Domain Knowledge in Deep Learning. In: Adel H and Shi S (eds.) *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, pp. 231–241. DOI:10.18653/v1/2021.emnlp-demo.27. URL https://aclanthology.org/2021.emnlp-demo.27/.

Reiter R (1987) A theory of diagnosis from first principles. *Artificial Intelligence* 32(1): 57–95. DOI:10.1016/0004-3702(87)90062-2. URL https://www.sciencedirect.com/science/article/pii/0004370287900622.

Rezende D and Mohamed S (2015) Variational inference with normalizing flows. In: Bach F and Blei D (eds.) *Proceedings of the 32nd International Conference on Machine Learning*, *Proceedings of Machine Learning Research*, volume 37. Lille, France: PMLR, pp. 1530–1538. URL https://proceedings.mlr.press/v37/rezende15.html.

Richardson M and Domingos P (2006) Markov logic networks. *Mach. Learn.* 62(1-2): 107–136. DOI:10.1007/s10994-006-5833-1. URL https://doi.org/10.1007/s10994-006-5833-1.

Riegel R, Gray A, Luus F, Khan N, Makondo N, Akhalwaya IY, Qian H, Fagin R, Barahona F, Sharma U, Ikbal S, Karanam H, Neelam S, Likhyani A and Srivastava S (2020) Logical Neural Networks. *arXiv:2006.13155 [cs]* URL http://arxiv.org/abs/2006.13155. ArXiv: 2006.13155.

Robinson JA (1965) A machine-oriented logic based on the resolution principle. *J. ACM* 12(1): 23–41. DOI:10.1145/321250.321253. URL https://doi.org/10.1145/321250.321253.

Rocktäschel T and Riedel S (2017) End-to-end Differentiable Proving. In: *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper/2017/hash/b2ab001909a8a6f04b51920306046ce5-Abstract.html.

Rumelhart DE, Hinton GE and Williams RJ (1986) Learning representations by back-propagating errors. *Nature* 323(6088): 533–536. DOI:10.1038/323533a0. URL https://www.nature.com/articles/323533a0.

Sakama C and Inoue K (2000) Abductive logic programming and disjunctive logic programming: their relationship and transferability. *The Journal of Logic Programming* 44(1): 75–100. DOI:10.1016/S0743-1066(99)00073-4. URL https://www.sciencedirect.com/science/article/pii/S0743106699000734.

Sarker MK, Zhou L, Eberhart A and Hitzler P (2021) Neuro-Symbolic Artificial Intelligence: Current Trends. DOI:10.48550/arXiv.2105.05330. URL http://arxiv.org/abs/2105.05330. ArXiv:2105.05330 [cs].

Sato T (1995) A Statistical Learning Method for Logic Programs with Distribution Semantics. In: *Logic Programming: The 12th International Conference*. The MIT Press. ISBN 978-0-262-29143-9. DOI:10.7551/mitpress/4298.003.0069. URL https://doi.org/10.7551/mitpress/4298.003.0069. _eprint: https://direct.mit.edu/book/chapter-pdf/2304327/9780262291439_ccl.pdf.

Sato T and Kameya Y (2001) Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research* 15: 391–454.

Schellhorn DR and Mossakowski T (2025) mULLER: A Modular Monad-Based Semantics of the Neurosymbolic ULLER Framework. URL https://openreview.net/forum?id=zIsnMqATI6.

Shindo H, Pfanschilling V, Dhami DS and Kersting K (2023) $$\alpha$$ILP: thinking visual scenes as differentiable logic programs. *Machine Learning* 112(5): 1465–1497. DOI:10.1007/s10994-023-06320-1. URL https://doi.org/10.1007/s10994-023-06320-1.

Si X, Raghothaman M, Heo K and Naik M (2019) Synthesizing Datalog Programs using Numerical Relaxation. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. Macao, China: International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-4-1, pp. 6117–6124. DOI:10.24963/ijcai.2019/847. URL https://www.ijcai.org/proceedings/2019/847.

Skryagin A, Ochs D, Dhami DS and Kersting K (2023) Scalable Neural-Probabilistic Answer Set Programming. *Journal of Artificial Intelligence Research* 78: 579–617. DOI:10.1613/jair.1.15027. URL http://www.jair.org/index.php/jair/article/view/15027.

Smet LD, Martires PZD, Manhaeve R, Marra G, Kimmig A and Readt LD (2023) Neural probabilistic logic programming in discrete-continuous domains. In: *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*. PMLR, pp. 529–538. URL https://proceedings.mlr.press/v216/de-smet23a.html.

Smet LD, Venturato G, Raedt LD and Marra G (2024) Relational Neurosymbolic Markov Models. DOI:10.48550/arXiv.2412.13023. URL http://arxiv.org/abs/2412.13023. ArXiv:2412.13023 [cs].

Sundararajan M, Taly A and Yan Q (2017) Axiomatic attribution for deep networks. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17. JMLR.org, p. 3319–3328.

Thurley M (2006) sharpsat: counting models with advanced component caching and implicit bcp. In: *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing*, SAT'06. Berlin, Heidelberg: Springer-Verlag. ISBN 3540372067, p. 424–429. DOI:10.1007/11814948_38. URL https://doi.org/10.1007/11814948_38.

Tsamoura E, Hospedales T and Michael L (2021) Neural-Symbolic Integration: A Compositional Perspective. *Proceedings of the AAAI Conference on Artificial Intelligence* 35(6): 5051–5060. DOI: 10.1609/aaai.v35i6.16639. URL https://ojs.aaai.org/index.php/AAAI/article/view/16639.

Valkov L, Chaudhari D, Srivastava A, Sutton C and Chaudhuri S (2018) HOUDINI: lifelong learning as program synthesis. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18. Red Hook, NY, USA: Curran Associates Inc., pp. 8701–8712.

van Bekkum M, de Boer M, van Harmelen F, Meyer-Vitali A and Teije At (2021) Modular design patterns for hybrid learning and reasoning systems. *Applied Intelligence* 51(9): 6528–6546. DOI: 10.1007/s10489-021-02394-3. URL https://doi.org/10.1007/s10489-021-02394-3.

van Krieken E, Badreddine S, Manhaeve R and Giunchiglia E (2024) ULLER: A Unified Language for Learning and Reasoning. In: *Neural-Symbolic Learning and Reasoning: 18th International Conference, NeSy 2024, Barcelona, Spain, September 9–12, 2024, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag. ISBN 978-3-031-71166-4, pp. 219–239. DOI:10.1007/978-3-031-71167-1_12. URL https://doi.org/10.1007/978-3-031-71167-1_12.

van Krieken M, Thanapalasingam T, Tomczak JM, van Harmelen F and ten Teije A (2023) A-NESI: a scalable approximate method for probabilistic neurosymbolic inferencee. In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23. Red Hook, NY, USA: Curran Associates Inc., pp. 24586–24609.

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł and Polosukhin I (2017) Attention is all you need. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17. Red Hook, NY, USA: Curran Associates Inc. ISBN 978-1-5108-6096-4, pp. 6000–6010. URL https://dl.acm.org/doi/10.5555/3295222.3295349.

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L and Polosukhin I (2023) Attention Is All You Need.

von Rueden L, Mayer S, Beckh K, Georgiev B, Giesselbach S, Heese R, Kirsch B, Pfrommer J, Pick A, Ramamurthy R, Walczak M, Garcke J, Bauckhage C and Schuecker J (2023) Informed Machine Learning – A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems. *IEEE Transactions on Knowledge and Data Engineering* 35(1): 614–633. DOI:10.1109/TKDE.2021.3079836. URL https://ieeexplore.ieee.org/document/9429985. Conference Name: IEEE Transactions on Knowledge and Data Engineering.

Wainwright MJ and Jordan MI (2007) Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends® in Machine Learning* 1(1–2): 1–305. DOI:10.1561/2200000001.

Wang B, Mauá DD, den Broeck GV and Choi Y (2025a) A Compositional Atlas for Algebraic Circuits. DOI:10.48550/arXiv.2412.05481.

Wang W, Yang Y and Wu F (2025b) Towards Data-And Knowledge-Driven AI: A Survey on Neuro-Symbolic Computing. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 47(2): 878–899. DOI:10.1109/TPAMI.2024.3483273. URL https://ieeexplore.ieee.org/document/10721277.

Weber L, Minervini P, Münchmeyer J, Leser U and Rocktäschel T (2019) NLProlog: Reasoning with Weak Unification for Question Answering in Natural Language. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 6151–6161. DOI:10.18653/v1/P19-1618. URL https://www.aclweb.

org/anthology/P19-1618.

Winters T, Marra G, Manhaeve R and De Raedt L (2021) DeepStochLog: Neural Stochastic Logic Programming. URL http://arxiv.org/abs/2106.12574. ArXiv:2106.12574 [cs].

Xu J, Zhang Z, Friedman T, Liang Y and Broeck G (2018) A Semantic Loss Function for Deep Learning with Symbolic Knowledge. In: *Proceedings of the 35th International Conference on Machine Learning*. PMLR, pp. 5502–5511. URL https://proceedings.mlr.press/v80/xu18h.html.

Yang F, Yang Z and Cohen WW (2017) Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In: *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/0e55666a4ad822e0e34299df3591d979-Abstract.html.

Yang Z, Ishay A and Lee J (2023) NeurASP: Embracing Neural Networks into Answer Set Programming. DOI:10.48550/arXiv.2307.07700. URL http://arxiv.org/abs/2307.07700. ArXiv:2307.07700 [cs].

Yu D, Yang B, Liu D, Wang H and Pan S (2023) A survey on neural-symbolic learning systems. *Neural Networks* 166: 105–126. DOI:10.1016/j.neunet.2023.06.028. URL https://www.sciencedirect.com/science/article/pii/S0893608023003398.

Šourek G, Aschenbrenner V, Železný F, Schockaert S and Kuželka O (2018) Lifted relational neural networks: efficient learning of latent relational structures. *J. Artif. Int. Res.* 62(1): 69–100. DOI: 10.1613/jair.1.11203. URL https://doi.org/10.1613/jair.1.11203.

A Syntax: Formal Definitions

Throughout this appendix we fix a semantic category $\mathcal{C}$ that is *cartesian closed*, meaning it has finite products and function spaces — a standard assumption that ensures quantifiers are well-defined as higher-order operations (see Appendix C). The categories **Set**, **BorelMeas**, and **Tens** used in Table 3 all satisfy this requirement.

This section provides the formal definitions underlying Section 2. Definitions 1 and 2 from the main text remain unchanged; we extend them here with two things Section 2 omitted for readability: the formal grammar that generates well-formed terms and formulas from a language Λ, and the notion of a *context* that makes variable scoping precise.

First of all, using the Definitions 1 and 2 from the main text, we can now give a precise definition of what a *language* is in our framework:

Definition 17. Language. *A* language Λ *is a tuple* $(\mathcal{L}, \Sigma)$ *where* $\mathcal{L}$ *satisfies at least the axioms of a 2Mon-BLat (Definition 20) and* Σ *is a non-logical vocabulary (Definition 2).*

The grammar of terms and formulas. Given a language $\Lambda = (\mathcal{L}, \Sigma)$, we need to specify precisely which symbolic expressions are *well-formed*. In natural language processing, a context-free grammar defines which strings are valid sentences by giving rewrite rules: a sentence can be rewritten as a noun phrase followed by a verb phrase, a noun phrase can be rewritten as a determiner followed by a noun, and so on. We use exactly the same mechanism here, but for logical expressions rather than natural language sentences.

The grammar generates two classes of expressions: *terms* (Ξ), which denote data objects (e.g., a specific digit image, a computed sum), and *formulas* (Φ), which denote truth-valued statements

(e.g., "the digit is 3", "the sum equals 5"). Each rule in the grammar specifies one way to build a larger expression from smaller ones, together with the *metatype* of that expression — whether it is a term or a formula, and over what sorts.

Definition 18. First-Order Syntax. *The syntax of first-order logic in* Λ *is the context-free grammar:*

Grammar Rule	***Symbols***	***Description***	***Metatype***
$\xi ::= x$	$x \in \mathsf{Var}$	*Variable*	*Term*
$\xi ::= f(\vec{\xi})$	$f \in \mathsf{Fun}$	*Functional term*	$\vec{\xi} \to \xi$
$\varphi ::= R(\vec{\xi})$	$R \in \mathsf{Rel}$	*Atomic formula*	$\vec{\xi} \to \varphi$
$\varphi ::= *(\vec{\varphi})$	$* \in \mathsf{Conn}$	*Compound formula*	$\vec{\varphi} \to \varphi$
$\varphi ::= Qx(\varphi)$	$Q \in \mathsf{Quan}$	*Quantified formula*	$\varphi \to \varphi$

Terms are built from variables and function applications; formulas are built from relation applications, connectives, and quantifiers.

Reading this for MNIST-Addition (Example 2): the variable $x : \mathsf{Image}$ is a term by the first rule; $\mathsf{digit}(x)$ is a term by the second rule (applying the function symbol digit to the term x); the expression $\mathsf{digit}(x) + \mathsf{digit}(y) = \mathsf{add}(x, y)$ is a formula by the third and fourth rules (combining atomic formulas via the equality relation and connectives). The quantified addition axiom $\forall x \forall y\, \mathsf{add}(x, y) = \mathsf{digit}(x) + \mathsf{digit}(y)$ is then produced by two applications of the last rule.

Variable scoping via contexts. One subtlety the grammar leaves implicit is *which variables are in scope* when a formula is evaluated. In a programming language, a variable x can only be used inside the block where it was declared; using it outside produces an error. The same issue arises here: the formula $\mathsf{digit}(x) = 3$ only makes sense if x has been bound somewhere. A *context* makes this explicit.

Definition 19. Context and Terms/Formulas-in-Context. *A* context $\vec{x}$ *is a finite ordered list* $[x_1, \ldots, x_n]$ *of distinct variables, including the empty context* $[]$. *A* term-in-context $\vec{x}.\xi$ *is a term* ξ *whose every variable appears in* $\vec{x}$. *A* formula-in-context $\vec{x}.\varphi$ *is a formula* φ *whose every free variable appears in* $\vec{x}$.

The notation $\vec{x}.\varphi$ should be read as "formula φ with free variables drawn from $\vec{x}$." This is the form in which all semantic evaluations in Appendix C are stated: $[\![\vec{x}.\varphi]\!]$ maps an assignment of values to the variables in $\vec{x}$ to a truth value. For a *closed* formula (no free variables), $\vec{x}$ is empty and the evaluation produces a single truth value independently of any assignment — as one would expect.

Example 18. *In MNIST-Addition, the addition axiom has context* $[x{:}\mathsf{Image}, y{:}\mathsf{Image}]$. *The formula* $\mathsf{digit}(x) = 3$ *has context* $[x{:}\mathsf{Image}]$. *The closed formula* $\forall x \forall y\, \mathsf{add}(x, y) = \mathsf{digit}(x) + \mathsf{digit}(y)$ *has empty context* $[]$ *and evaluates to a single truth value under any interpretation.*

B Algebraic Foundations: 2Mon-BLat

B.1 Why the Truth Space Needs Structure

Section 2 defined a logical vocabulary $\mathcal{L}$ as a truth symbol τ, a set of connectives, and (optionally) quantifiers. But not every such collection makes logical sense: the connectives must combine truth values in a coherent way. The minimal requirement is that the truth space $\mathcal{T}\Omega$ carries the structure of a *double monoid bounded lattice*. We build up to this concept step by step.

A *monoid* is a set A equipped with a binary operation $\cdot : A \times A \to A$ and an identity element $e \in A$ such that the operation is associative and $e \cdot a = a \cdot e = a$ for all a. Familiar examples are $(\mathbb{R}, +, 0)$ and $(\mathbb{R}, \cdot, 1)$ — the real numbers under addition with identity 0, and under multiplication with identity 1.

A *double monoid* (2Mon) is simply a single set carrying two monoid structures simultaneously. In our context, the two operations are $\oplus$ (with identity $\vec{0}$, playing the role of disjunction) and $\otimes$ (with identity $\vec{1}$, playing the role of conjunction). No interaction between the two operations is required beyond their coexistence on the same set.

A *bounded lattice* is a set A equipped with a partial order $\leq$ together with:

- a bottom element $\bot \in A$ satisfying $\bot \leq a$ for all a (absolute falsehood),
- a top element $\top \in A$ satisfying $a \leq \top$ for all a (absolute truth),
- a meet $a \wedge b$ (greatest lower bound of a and b),
- a join $a \vee b$ (least upper bound of a and b).

A *double monoid bounded lattice* (2Mon-BLat) combines all of the above: the truth space is simultaneously a double monoid and a bounded lattice, with the two monoid operations and the lattice order mutually compatible. The lattice axioms additionally give rise to dual meet and join connectives $\wedge$ and $\vee$, which may or may not coincide with $\otimes$ and $\oplus$ depending on the instantiation:

Definition 20. 2Mon-BLat. *A theory of a* double monoid bounded lattice *(2Mon-BLat) is a logical vocabulary $\mathcal{L}$ (Definition 1) whose symbols $\{\tau, \bot, \top, \oplus, \otimes, \vec{0}, \vec{1}\}$ satisfy: $(\tau, \oplus, \vec{0})$ and $(\tau, \otimes, \vec{1})$ form monoids, and $(\tau, \bot, \top)$ forms a bounded lattice. The bounded lattice axioms give rise to dual meet and join connectives $\wedge$ and $\vee$ (which may or may not coincide with $\otimes$ and $\oplus$).*

This is the minimal algebraic requirement for a coherent truth space: it provides exactly what is needed to interpret connectives ($\otimes$ and $\oplus$), absolute falsehood ($\bot$), and absolute truth ($\top$). Classical Boolean logic on $\{0, 1\}$, fuzzy logic on $[0, 1]$, and probabilistic logic under Product-BL axioms are all instances — which is precisely what makes them commensurable within the framework of Section 3.

B.2 Concrete Instantiations

The following table collects the most relevant 2Mon-BLat instantiations encountered in the NeSy landscape. Each row specifies the complete tuple ($\mathcal{T}\Omega$, $\bot$, $\top$, $\otimes$, $\vec{1}$, $\oplus$, $\vec{0}$), where $\mathcal{T}\Omega$ is

the *truth space* in which the system actually reasons (cf. Table 3). The rightmost two columns indicate whether the monoid operations coincide with the lattice meet and join — i.e. whether $\otimes = \wedge$ and $\oplus = \vee$.

Table 6. Concrete 2Mon-BLat instantiations. T denotes a t-norm and S its induced t-conorm.

Algebra	$\mathcal{T}\Omega$	$\bot$	$\top$	$\otimes$	$\vec{1}$	$\oplus$	$\vec{0}$
Logic (Set Theory)							
Boolean	$\{0,1\}$	0	1	min	1	max	0
BL (generic t-norm)	$[0,1]$	0	1	$T(a,b)$	1	$S(a,b)$	0
Product BL / Real Logic	$[0,1]$	0	1	$a \cdot b$	1	$a+b-ab$	0
Łukasiewicz BL	$[0,1]$	0	1	$\max(0,\, a{+}b{-}1)$	1	$\min(1,\, a{+}b)$	0
Kleene (three-valued)	$\{0, \frac{1}{2}, 1\}$	0	1	min	1	max	0
Belnap (four-valued)	$\mathcal{P}(\{0,1\})$	$\varnothing$	$\{0,1\}$	$\cap$	$\{0,1\}$	$\cup$	$\varnothing$
Probability (Probability Theory)							
Product BL (discrete)	$[0,1]$	0	1	$a \cdot b$	1	$a+b-ab$	0
Product BL (continuous)	$[0,1]$	0	1	$a \cdot b$	1	$a+b-ab$	0
Neural (Tensor Algebra, pre-normalisation)							
Real arithmetic	$\overline{\mathbb{R}}$	$-\infty$	$+\infty$	$a \cdot b$	1	$a+b$	0

Boolean logic on $\{0,1\}$ is the degenerate case in which $\otimes = \wedge = \min$ and $\oplus = \vee = \max$. The monoid identities coincide with $\top$ and $\bot$: $\vec{1} = 1 = \top$ and $\vec{0} = 0 = \bot$. This is the truth space underlying classical logic programming systems such as Prolog and ASP (Table 3, row *Classical*).

BL algebras (generic t-norm) on $[0,1]$ generalise Boolean logic to graded truth: $\otimes$ is any left-continuous t-norm $T : [0,1]^2 \to [0,1]$ (associative, commutative, non-decreasing, with $T(a,1) = a$), and $\oplus$ is its induced t-conorm $S(a,b) = 1 - T(1{-}a,\, 1{-}b)$. Every BL algebra also has a residuum $a \to b := \sup\{c \mid T(a,c) \leq b\}$, which is the logical implication that completes the vocabulary $\mathcal{L}$. The two most common instantiations in NeSy are Product BL and Łukasiewicz BL; Gödel BL ($T = \min$) recovers Boolean-style idempotent conjunction on $[0,1]$ (Table 3, row *Fuzzy*).

Product BL / Product Real Logic on $[0,1]$ uses the product t-norm $a \otimes b := a \cdot b$ and the probabilistic sum $a \oplus b := a + b - ab$. This is the truth space used by LTN (Badreddine et al. 2022) and DeepProbLog (Manhaeve et al. 2018) (Table 3, rows *Fuzzy* and *Finite*). The two systems differ in their logical implication: LTN uses the Reichenbach implication $a \to b := 1 - a + ab$, whereas the strict Product BL residuum is $a \to b := \min(1,\, b/a)$. Both share the same $(\otimes, \oplus)$ pair and therefore the same 2Mon-BLat structure. Note also that this is precisely the $[0,1]$ instance identified on page 43: softmax normalisation projects neural logits into $[0,1]$, and the product t-norm and probabilistic sum are closed on $[0,1]$ with $\vec{1} = 1$ and $\vec{0} = 0$.

Łukasiewicz BL on $[0,1]$ uses the bounded sum $a \otimes b := \max(0, a+b-1)$ and its t-conorm $a \oplus b := \min(1, a+b)$. The residuum is $a \to b := \min(1, 1-a+b)$ (the Łukasiewicz implication), giving a linear, bounded form of conjunction weaker than the product t-norm. LRNN (Šourek et al. 2018) uses Łukasiewicz connectives when compiling weighted logic programs into differentiable gate networks.

Kleene three-valued logic on $\{0, \frac{1}{2}, 1\}$ adds an indeterminate truth value $\frac{1}{2}$ (neither true nor false) while keeping $\otimes = \min$ and $\oplus = \max$. It is the simplest finite extension of Boolean logic that allows partial information: $\frac{1}{2} \otimes 1 = \frac{1}{2}$, reflecting that conjunction with an unknown remains unknown. Via the non-empty powerset monad $\mathcal{P}_{\neq\varnothing}$, a predicate returns a non-empty subset of $\{0,1\}$: $\{0\} \equiv F$, $\{1\} \equiv T$, $\{0,1\} \equiv U$ (undefined/unknown), giving three distinguishable truth values (Table 3, row *Three-valued*).

Belnap four-valued logic on $\mathcal{P}(\{0,1\})$ treats truth values as sets of classical values: $\varnothing$ (no information, N), $\{0\}$ (false, F), $\{1\}$ (true, T), and $\{0,1\}$ (contradictory/both, B). The monoid operations are set intersection ($\otimes = \cap$) and set union ($\oplus = \cup$), with identities $\vec{1} = \{0,1\}$ and $\vec{0} = \varnothing$ (note that these coincide with $\top$ and $\bot$ under the subset order). Via the full powerset monad $\mathcal{P}$, a Kleisli arrow $\mathcal{I}(R) : X \to \mathcal{P}(\{0,1\})$ returns a set of possible classical truth values, and Kleisli composition is relational composition — $\mathcal{T}\Omega = \mathcal{P}(\{0,1\}) = \{\varnothing, \{0\}, \{1\}, \{0,1\}\}$ (Table 3, row *Four-valued*).

Neural / Real arithmetic on $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ uses standard multiplication ($\otimes := \cdot$, identity $\vec{1} = 1$) and addition ($\oplus := +$, identity $\vec{0} = 0$), with $\bot = -\infty$ and $\top = +\infty$. Neural components naturally output real-valued scores (logits, energies, or similarities). Worth to mention is, that $\overline{\mathbb{R}}$ fails the monoid closure requirement ($+\infty + (-\infty)$ and $+\infty \cdot 0$ are undefined), so this row in Table 3 should be understood as an idealised description of the underlying computation; the 2Mon-BLat requirement is fully satisfied only after normalisation (e.g., via `softmax`), which typically projects the output back into $[0,1]$, where the Product BL structure (or similar) takes over (Table 3, row *Tensor*).

C Semantics: Formal Definitions

This section provides the formal evaluation rules underlying Section 3. We fix a language $\Lambda = (\mathcal{L}, \Sigma)$, a cartesian closed category $\mathcal{C}$, a commutative monad T on $\mathcal{C}$, and a truth object $\Omega := I(\tau)$. We use the interpretations $I_{\mathcal{L}}$ and I_Σ from Section 3 (Definitions 3 and 6) without repeating them here. The goal is to define precisely how every term and formula in Λ evaluates to a morphism in $\mathcal{C}$ — first in the deterministic (Tarski) setting, then in the effectful (Kleisli) setting.

Tracking sorts through evaluation. When we evaluate a compound term like $\mathsf{digit}(x)$, the output sort is determined by the function symbol: $\mathsf{digit} : \mathsf{Image} \to \mathsf{Digit}$ produces a Digit. To make the evaluation rules type-correct, we need a function that reads off the output sort of any term:

Definition 21. Type of a Term. *The* type *function* $\mathrm{typ} : \Xi \to \mathsf{Sor}$ *is defined recursively:*

$$\mathrm{typ}(\xi) = \begin{cases} S & \text{if } \xi = x \in \mathsf{Var} \text{ with } x : S, \\ T & \text{if } \xi = f(\vec{\xi}) \text{ for } f \in \mathsf{Fun} \text{ with } f : \vec{S} \to T. \end{cases}$$

It extends to lists of terms by $\mathrm{typ}(\vec{\xi}) = [\mathrm{typ}(\xi_1), \ldots, \mathrm{typ}(\xi_n)]$.

This is bookkeeping, not mathematics: it says that the output sort of a term is determined by the outermost symbol. Its role is to ensure that whenever we write $I(f) \circ \langle [\![\vec{\xi}]\!] \rangle$, the input type of $I(f)$ matches the output type of the evaluated argument tuple.

Tarski semantics. We now define the deterministic evaluation — the special case $T = \mathrm{Id}$ where no uncertainty is present. The evaluation $[\![\cdot]\!]_I$ maps every term or formula in context to a morphism in $\mathcal{C}$.

Definition 22. Tarski Semantics. *The* Tarski semantics $[\![\cdot]\!]_I$ *for an interpretation* I *of* Λ *assigns to each term-in-context* $\vec{x}.\,\xi$ *a morphism* $I(\vec{x}) \to I(\mathrm{typ}(\xi))$*, and to each formula-in-context* $\vec{x}.\,\varphi$ *a morphism* $I(\vec{x}) \to \Omega$*, by the following recursive rules:*

Expression	***Evaluation***	***Type***
$[\![\vec{x}.\, y]\!]$	$\pi_{I(y)}$	$I(\vec{x}) \to I(y)$
$[\![\vec{x}.\, f(\vec{\xi})]\!]$	$I(f) \circ \langle [\![\vec{x}.\, \xi_i]\!] \rangle_i$	$I(\vec{x}) \to I(\mathrm{typ}(\vec{\xi})) \to I(\mathrm{cod}(f))$
$[\![\vec{x}.\, R(\vec{\xi})]\!]$	$I(R) \circ \langle [\![\vec{x}.\, \xi_i]\!] \rangle_i$	$I(\vec{x}) \to I(\mathrm{typ}(\vec{\xi})) \to \Omega$
$[\![\vec{x}.\, *(\vec{\varphi})]\!]$	$I(*) \circ \langle [\![\vec{x}.\, \varphi_i]\!] \rangle_i$	$I(\vec{x}) \to \Omega^n \to \Omega$
$[\![\vec{x}.\, Qy(\varphi)]\!]$	$I(Q)_{I(y)} \circ \Lambda_{I(y)}([\![\vec{x}.\, \varphi]\!]) \circ \pi_{I(\vec{x}\setminus y)}$	$I(\vec{x}) \to I(\vec{x}\setminus y) \to \Omega^{I(y)} \to \Omega$

Reading these rules left to right:

- A **variable** y evaluates to the projection $\pi_{I(y)}$ that extracts the value of y from the context tuple.
- A **function application** $f(\vec{\xi})$ first evaluates each argument ξ_i (producing a tuple via the pairing $\langle\cdot\rangle$), then applies $I(f)$ to the result.
- A **relation application** $R(\vec{\xi})$ works identically but outputs a truth value via $I(R)$.
- A **compound formula** $*(\vec{\varphi})$ evaluates each subformula, collects the truth values into a tuple, and applies the connective morphism $I(*)$.
- A **quantified formula** $Qy(\varphi)$ is the most involved. $\Lambda_{I(y)}([\![\vec{x}.\, \varphi]\!])$ is the *currying* of the evaluation: it turns the morphism $I(\vec{x}) \to \Omega$ into a morphism $I(\vec{x} \setminus y) \to \Omega^{I(y)}$, i.e., a function that maps each assignment of the remaining variables to a truth-valued function over $I(y)$. The quantifier $I(Q)_{I(y)}$ then aggregates this function into a single truth value, and $\pi_{I(\vec{x}\setminus y)}$ drops y from the input context.

Example 19. MNIST-Addition, deterministic. *With* $I_\theta(\mathsf{digit}) : \mathbb{R}^{28\times 28} \to \{0, \ldots, 9\}$ *(a CNN with argmax) and* $I(+) : \{0, \ldots, 9\}^2 \to \mathbb{N}$*, the addition axiom evaluates as:*

$$[\![[x, y].\ \mathsf{add}(x, y) = \mathsf{digit}(x) + \mathsf{digit}(y)]\!]_{I_\theta} = I(=) \circ \Big\langle I(\mathsf{add}),\ I(+) \circ \langle I_\theta(\mathsf{digit}) \circ \pi_x,\ I_\theta(\mathsf{digit}) \circ \pi_y \rangle \Big\rangle.$$

This morphism maps each image pair (x, y) *to* 1 *if the observed sum equals the predicted sum, and* 0 *otherwise.*

From deterministic to effectful: the commutator. Moving from Tarski to Kleisli semantics requires handling the case where multiple *independent* effectful computations must be combined. Consider evaluating $f(\xi_1, \xi_2)$ when both ξ_1 and ξ_2 are neural terms that return distributions. We need to pair a distribution over Y_1 with a distribution over Y_2 and produce a joint distribution over $Y_1 \times Y_2$ before we can apply $I(f)$. Plain product pairing no longer works because the outputs live in TY_1 and TY_2 rather than Y_1 and Y_2. The *commutator* resolves this:

Definition 23. Commutator. *A monad T on a cartesian category $\mathcal{C}$ is* commutative *if there exists a natural transformation*

$$\mathrm{com}_{A,B} : TA \times TB \to T(A \times B),$$

satisfying coherence conditions that ensure the order in which two independent effectful computations are combined does not matter.

For the distribution monad $\mathcal{D}$, this is the map $(p, q) \mapsto p \otimes q$ sending a pair of distributions to their product distribution — formally,

$$\mathrm{com}(p, q)(a, b) = p(a) \cdot q(b).$$

All monads in Table 3 are commutative.

Intuitively, commutativity is the statement that two independently drawn samples can be combined in either order without affecting the joint distribution. This is what allows the Kleisli semantics to evaluate the arguments ξ_1, ξ_2 of $f(\xi_1, \xi_2)$ independently and then merge their distributions via com before applying $I(f)^*$.

A related operation handles the specific case where one computation is pure (deterministic) and the other is effectful — the pattern that arises whenever a symbolic component with a fixed interpretation is combined with a neural one. The *strength* maps package this:

Definition 24. Strength. *For a commutative monad T on a cartesian category $\mathcal{C}$, the* left *and* right strength *maps are:*

$$\mathrm{st}_{A,B} := \mathrm{com}_{A,B} \circ \big(\eta_A \circ \pi_A,\ \pi_{TB}\big) : A \times TB \to T(A \times B),$$
$$\mathrm{st}'_{A,B} := \mathrm{com}_{A,B} \circ \big(\pi_{TA},\ \eta_B \circ \pi_B\big) : TA \times B \to T(A \times B).$$

Concretely, $\mathrm{st}_{A,B}$ takes a pure value $a \in A$ and an effectful computation $m \in TB$, lifts a into TA via the unit η (treating it as a trivial computation), and then applies com to produce a joint computation in $T(A \times B)$. This is the formal underpinning of every place in Section 5 where a deterministic symbolic component is composed with a probabilistic neural one.

Do-notation. The strength maps make it possible to define monadic sequencing in a context — the do-notation used throughout Section 5 — in a fully typed way:

Definition 25. Do-Notation. *Given $p : \Gamma \to \mathcal{T}A$ and $q : \Gamma \times A \to \mathcal{T}B$, the* contextual sequencing *is:*

$$\big(\mathbf{do}\ a \leftarrow p;\ q\big)\ :=\ q^* \circ \mathrm{st}_{\Gamma,A} \circ \langle \mathrm{id}_\Gamma,\ p\rangle\ :\ \Gamma \to \mathcal{T}B,$$

where q^* is the Kleisli extension of q (Definition 5). When $\Gamma = \mathbf{1}$ (no additional context), this reduces to ordinary Kleisli sequencing $q^* \circ p$.

Reading this step by step: $\langle \mathrm{id}_\Gamma, p\rangle$ pairs the current context $\gamma \in \Gamma$ with the effectful computation $p(\gamma) \in TA$; $\mathrm{st}_{\Gamma,A}$ lifts this pair into $T(\Gamma \times A)$, making the context available inside the effectful world; and q^* then passes the drawn value a together with γ to q. The monad laws guarantee that chaining multiple do-bindings is associative.

Example 20. Do-notation in DeepProbLog. *In DeepProbLog, the query computation for MNIST-Addition can be written as:*

$$k_\theta(query, img_0, img_1) = \textbf{do } d_0 \leftarrow I_\theta(\mathsf{digit})(img_0);\ \textbf{do } d_1 \leftarrow I_\theta(\mathsf{digit})(img_1);\ I(\mathsf{addition})(query, d_0, d_1).$$

The first bind draws a digit label d_0 from the CNN's output distribution over $\{0, \ldots, 9\}$ for img_0; the second bind draws d_1 for img_1; and $I(\mathsf{addition})$ then checks whether $d_0 + d_1$ equals the queried sum. Since the monad is $\mathcal{D}$, the Kleisli extension q^* applies the law of total probability at each bind, so the full computation marginalises over all digit pairs:

$$k_\theta(query, img_0, img_1) = \sum_{d_0+d_1=n} I_\theta(\mathsf{digit})(img_0)(d_0) \cdot I_\theta(\mathsf{digit})(img_1)(d_1).$$

Each do-bind corresponds to one application of Definition 25 with $\Gamma = \{img_0, img_1\}$ carrying the remaining context through the sequencing.

Kleisli semantics. We now lift the Tarski semantics to the effectful setting. Every plain morphism $X \to Y$ is replaced by a Kleisli arrow $X \to TY$, and composition is governed by com and $(-)^*$.

Definition 26. Kleisli Semantics. *The* Kleisli semantics $[\![\cdot]\!]_I$ *for a Kleisli interpretation I of Λ in a commutative monad $\mathcal{T}$ assigns to each term-in-context $\vec{x}.\,\xi$ a morphism $I(\vec{x}) \to \mathcal{T}\, I(\mathrm{typ}(\xi))$, and to each formula-in-context $\vec{x}.\,\varphi$ a morphism $I(\vec{x}) \to \mathcal{T}\Omega$, by:*

Expression	***Evaluation***
$[\![\vec{x}.\, y]\!]$	$\eta_{I(y)} \circ \pi_{I(y)}$
$[\![\vec{x}.\, f(\vec{\xi})]\!]$	$I(f)^* \circ \mathrm{com} \circ \langle [\![\vec{x}.\, \xi_i]\!]\rangle_i$
$[\![\vec{x}.\, R(\vec{\xi})]\!]$	$I(R)^* \circ \mathrm{com} \circ \langle [\![\vec{x}.\, \xi_i]\!]\rangle_i$
$[\![\vec{x}.\, *(\vec{\varphi})]\!]$	$I(*)^* \circ \mathrm{com} \circ \langle [\![\vec{x}.\, \varphi_i]\!]\rangle_i$
$[\![\vec{x}.\, Q\vec{y}(\varphi)]\!]$	$I(Q)_{I(\vec{y})} \circ \Lambda_{I(\vec{y})}([\![\vec{x}.\, \varphi]\!]) \circ \pi_{I(\vec{x}\setminus\vec{y})}$

The structure mirrors the Tarski table exactly, with two systematic changes. First, every plain morphism becomes a Kleisli arrow: $I(f)$ becomes $I(f)^*$, the Kleisli extension of $I(f)$ (Definition 5), which propagates the uncertainty already present in the input through f. Second, every place where the Tarski semantics uses product pairing $\langle\cdot\rangle$, the Kleisli semantics inserts com to merge the independent effectful computations before passing the result to the symbol. For the distribution monad $\mathcal{D}$, com is exactly the product distribution, so the combination of two independent CNN outputs is their joint distribution, and $I(f)^*$ then marginalises over this joint via the law of total probability.

The Tarski semantics is recovered as the special case $\mathcal{T} = \mathrm{Id}$: then $\eta = \mathrm{id}$, $(-)^* = (-)$, and com reduces to the canonical product pairing, collapsing every row back to the Tarski table.

Example 21. MNIST-Addition, distributional. *With* $\mathcal{T} = \mathcal{D}$, $I_\theta(\mathsf{digit}) : \mathbb{R}^{28\times 28} \to \mathcal{D}(\{0,\dots,9\})$ *(softmax CNN), and* $I(+)$ *lifted via* η *to* $\eta_{\mathbb{N}} \circ I(+) : \{0,\dots,9\}^2 \to \mathcal{D}(\mathbb{N})$, *the Kleisli semantics of the addition axiom gives:*

$$[\![[x, y].\ \mathsf{add}(x, y) = \mathsf{digit}(x) + \mathsf{digit}(y)]\!]_{I_\theta} = \sum_{d_1+d_2=n} \rho_0(d_1) \cdot \rho_1(d_2),$$

where $\rho_0 = I_\theta(\mathsf{digit})(img_0)$ *and* $\rho_1 = I_\theta(\mathsf{digit})(img_1)$. *This is exactly the DeepProbLog training signal from Example 4 in the main text, now derived as a direct consequence of the Kleisli evaluation rules rather than stated as an external design choice.*

D Monad Instantiations

Section 3 introduced monads via Definition 5 and motivated them with the distribution monad $\mathcal{D}$. This appendix makes the three components of Definition 5 — the computation-space mapping $\mathcal{T}$, the unit η, and the Kleisli extension $(-)^*$ — explicit for each monad appearing in Table 3. The goal is to give the reader a concrete handle on what changes when the semantic choice changes: the *same* schema $(\mathcal{T}, \eta, (-)^*)$ is filled with different mathematical content, but the composition rule $g^* \circ f$ always has the same structural role.

Throughout, $f : X \to \mathcal{T}Y$ and $g : Y \to \mathcal{T}Z$ are Kleisli arrows, and all categories are as in Table 3.

Identity Monad Id *on* $\mathcal{C}$

The identity monad introduces *no* computational effects. It is used for classical Boolean semantics, fuzzy semantics, and the neural tensor row in Table 3.

- **Computation space.** $\mathrm{Id}(X) := X$ — no wrapping occurs; a computation over X is just an element of X itself.
- **Unit.** $\eta_X := \mathrm{id}_X : X \to X$ — the unit is theidentity function; a plain value is already a trivial computation.
- **Kleisli extension.** For $f : X \to Y$, the extension is $f^* := f$ — lifting does nothing because there is no uncertainty to propagate.

Kleisli composition therefore reduces to ordinary function composition:$g^* \circ f = g \circ f$.Every deterministic NeSy morphism — a symbolic rule, a crisp neuralclassifier after argmax, or a Boolean relation — lives here.

Distribution Monad $\mathcal{D}$ *on* Set

The distribution monad is the workhorse for *discrete probabilistic* semantics (e.g. DeepProbLog).

- **Computation space.** $\mathcal{D}(X) := \{\rho : X \to [0,1] \mid \sum_{x \in X} \rho(x) = 1,\ |\mathrm{supp}(\rho)| < \infty\}$ — the set of all finitely supported probability distributions over X. For $X = \{0, \ldots, 9\}$, $\mathcal{D}(X)$ is the 9-simplex, i.e. the space of all softmax outputs a CNN can produce.

- **Unit.** $\eta_X(x) := \delta_x$ — the Dirac distribution concentrated at x; a crisp value becomes a point mass. This is how deterministic morphisms such as $\mathcal{I}(+)$ are lifted into the probabilistic world: $\eta_{\mathbb{N}} \circ \mathcal{I}(+)$ wraps each sum $d_1 + d_2$ into $\delta_{d_1+d_2}$.

- **Kleisli extension.** For $f : X \to \mathcal{D}(Y)$ and $\rho \in \mathcal{D}(X)$:

$$f^*(\rho)(y) \;=\; \sum_{x \in X} \rho(x) \cdot f(x)(y).$$

  This is the *law of total probability*: it marginalises the input distribution ρ through f to produce an output distribution over Y.

Kleisli composition therefore gives: $(g^* \circ f)(x)(z) = \sum_y f(x)(y) \cdot g(y)(z)$, which, for the MNIST-Addition example, yields $\sum_{d_1+d_2=n} \rho_0(d_1) \cdot \rho_1(d_2)$.

Giry Monad $\mathcal{G}$ *on* **BorelMeas**

The Giry monad generalises $\mathcal{D}$ to *continuous* probability (e.g. DeepSeaProbLog). Objects in **BorelMeas** are measurable spaces (X, Σ_X), where Σ_X is a σ-algebra on X.

- **Computation space.** $\mathcal{G}(X) := \{\mu : \Sigma_X \to [0,1] \mid \mu \text{ is a probability measure on } (X, \Sigma_X)\}$, equipped with the smallest σ-algebra making all evaluation maps $\mu \mapsto \mu(A)$ measurable. Where $\mathcal{D}(X)$ can only place mass on finitely many points, $\mathcal{G}(X)$ includes continuous distributions such as $\mathcal{N}(\mu_\theta, \sigma_\theta^2)$.

- **Unit.** $\eta_X(x) := \delta_x$, where $\delta_x(A) = \mathbf{1}[x \in A]$ — the Dirac measure at x. The key difference from $\mathcal{D}$: here δ_x is a measure *on a σ-algebra*, not a function on a finite set.

- **Kleisli extension.** For a measurable $f : X \to \mathcal{G}(Y)$ (a Markov kernel) and $\mu \in \mathcal{G}(X)$:

$$f^*(\mu)(A) \;=\; \int_X f(x)(A)\, \mathrm{d}\mu(x), \quad A \in \Sigma_Y.$$

  This is a *Lebesgue integral*: it integrates the probability that $f(x)$ assigns to A over all x, weighted by μ.

Kleisli composition $(g^* \circ f)$ chains two Markov kernels into two nested Lebesgue integrals — exactly what appears in the DeepSeaProbLog formula in Sec. 3.2. In the discrete case (μ has finite support), the Lebesgue integral reduces to a sum and $\mathcal{G}$ collapses back to $\mathcal{D}$.

Powerset Monads $\mathcal{P}$ and $\mathcal{P}_{\neq\emptyset}$n Set

The powerset monads model *non-determinism*: a computation returns a *set* of possible outcomes rather than a single value or a distribution. $\mathcal{P}$ allows the empty set (failure); $\mathcal{P}_{\neq\emptyset}$ requires at least one result (non-failure non-determinism). Both underlie the three- and four-valued rows of Table 3 (Kleene / Belnap semantics), where a predicate can return {} (false), {0} (false with info), {1} (true), or {0, 1} (both / unknown).

- **Computation space.** $\mathcal{P}(X) := \{A \mid A \subseteq X\}$ — the powerset of X; $\mathcal{P}_{\neq\emptyset}(X)$ is the same but restricted to non-empty subsets.
- **Unit.** $\eta_X(x) := \{x\}$ — a crisp value becomes the singleton set containing it.
- **Kleisli extension.** For $f : X \to \mathcal{P}(Y)$ and $A \in \mathcal{P}(X)$:

$$f^*(A) \;=\; \bigcup_{x\in A} f(x).$$

  Apply f to every element of the non-deterministic input and collect all possible outputs into one set.

Kleisli composition is relational composition: $(g^* \circ f)(x) = \bigcup_{y\in f(x)} g(y)$ — all z reachable from x in two steps. In the Belnap case ($\mathcal{T} = \mathcal{P}$, $\Omega = \{0, 1\}$), a relation R is interpreted as a Kleisli arrow $\mathcal{I}(R) : X \to \mathcal{P}(\{0, 1\})$, so $\mathcal{T}\Omega = \mathcal{P}(\{0, 1\}) = \{\{\}, \{0\}, \{1\}, \{0, 1\}\}$, the four Belnap truth values $\{F, \text{None}, T, \text{Both}\}$.

Summary

Table 7 collects the four cases side by side. Reading across a row shows how the same Definition 5 is concretely filled in; reading down a column shows how the same structural slot (e.g. f^*) plays a different computational role depending on the effect being modelled.

Table 7. Concrete instantiations of Definition 5 for the monads used in Table 3. $f : X \to \mathcal{T}Y$ and $\rho/\mu/A$ denote a generic element of $\mathcal{T}X$.

Monad	$\mathcal{T}X$	$\eta_X(x)$	$f^*(\cdot)(y)$	**Kleisli composition**
Id	X	x	$f(x)$	function composition $g \circ f$
$\mathcal{D}$	fin. distributions on X	δ_x	$\sum_x \rho(x) \cdot f(x)(y)$	law of total probability
$\mathcal{G}$	prob. measures on X	δ_x	$\int_X f(x)(B)\,\mathrm{d}\mu(x)$	Markov kernel integration
$\mathcal{P}$	subsets of X	$\{x\}$	$\bigcup_{x\in A} f(x)$	relational composition

E Inference

Here, we provide more detail on the inference methods mentioned in Section 4, categorising them by paradigm, type, method, and faithfulness. Table 8 presents this categorization, whereas Appendicies E.1–E.3 give further information regarding the inference methods of the different paradigms.

Table 8. Taxonomy of inference methods across logical, probabilistic, and neural paradigms. Each row identifies a distinct class of algorithms characterized by the inference type, method, and faithfulness of the result, together with representative examples.

Paradigm	Type	Method	Faithfulness	Examples
Logical	Deduction	Proof-theoretic	Exact	SLD Resolution
		Model-theoretic	Exact	DPLL, CDCL, sharpSAT, Knowledge Compilation (KC)
	Induction	Proof-theoretic	Approx.	FOIL, Progol
		Model-theoretic	Approx.	ILASP, DL-Learner
	Abduction	Proof-theoretic	Approx.	IFF proof procedure
		Model-theoretic	Approx.	Reiter's diagnosis, ASP abduction
Probabilistic	Deduction	Proof-theoretic	Exact	PRISM inside-outside
		Model-theoretic	Exact	WMC, WMI, sum-product BP
		Model-theoretic	Approx.	MCMC, Variational Inference, Loopy BP
	Induction	Proof-theoretic	Approx.	PRISM inside-outside EM
		Model-theoretic	Approx.	LFI-ProbLog, WMC-EM
		Hybrid	Approx.	ProbFOIL+, SLIPCOVER
	Abduction	Proof-theoretic	Approx.	Viterbi proof procedure
		Model-theoretic	Exact	Junction tree algorithm
		Model-theoretic	Approx.	Max-product belief propagation
Neural	Deduction	Model-theoretic	Exact*	Forward propagation (FP) such as in CNNs or Transformers
	Induction	Model-theoretic	Approx.	Backpropagation, Adam, SGD
	Abduction	Model-theoretic	Approx.	Input optimization (FGSM, IG), latent space inference (VAE, GAN inversion, Flows), neural program synthesis (DreamCoder)

E.1 Logical Inference

Logical inference operates on discrete symbols and explicit logical relations. Within this paradigm, both proof-theoretic and model-theoretic methods are well-developed, and exact inference is often tractable.

Deduction. **Proof-theoretic** algorithms for deduction operate by syntactic symbol manipulation. The canonical examples are **resolution** (Robinson 1965), which derives new clauses by cancelling complementary literals, and **SLD resolution** (Selective Linear Definite), the backward-chaining algorithm underlying Prolog and most logic programming systems. As an illustration: given the rule $\forall x.Human(x) \rightarrow \exists y.mother(x) = y$ and the case $Human(Charlie)$, SLD resolution constructs a derivation tree concluding $\exists y.mother(Charlie) = y$ via modus ponens.
Model-theoretic algorithms evaluate truth by searching over possible interpretations, and are naturally organised by the decision or counting problem they solve. The satisfiability problem **SAT** (e.g., *Does any satisfying assignment exist?*) is solved by **DPLL** (Davis et al. 1962) and its descendant **CDCL** (Conflict-Driven Clause Learning) (Marques-Silva and Sakallah 1999), which adds non-chronological backtracking and clause learning. The threshold problem **MAJSAT** asks whether *more than half* of all assignments satisfy the formula, and can be decided by any model counter via comparison against 2^{n-1}. The counting problem #**SAT** (Model Counting) (e.g., *How many assignments satisfy the formula?*) is addressed by algorithms such as **sharpSAT** (Thurley 2006) and **knowledge compilation** into d-DNNF (Darwiche and Marquis 2002). These three form an ascending complexity hierarchy (NP $\subseteq$ PP $\subseteq$ #P), and all produce exact answers. Model Counting in particular serves as the computational backbone of Weighted Model Counting (WMC) introduced in Section E.2.

Induction. **Proof-theoretic** induction operates in the *learning from entailment* setting (De Raedt et al. 2016): we seek a hypothesis H such that $Background \wedge H \vdash Examples$, testing coverage via provability rather than semantic evaluation. **FOIL** (Quinlan 1990) induces Prolog-like rules top-down by greedily adding literals to clause bodies until positive examples are provably covered and negative ones excluded. **Progol** (Muggleton 1995) instead applies inverse entailment to construct the most specific clause consistent with each example, then searches upward toward more general hypotheses. Both are **approximate** because their hypothesis-space search is heuristic-guided and not guaranteed to find a globally optimal theory.

Model-theoretic induction operates in the *learning from interpretations* setting (De Raedt et al. 2016): examples are complete interpretations, and we seek H such that $Examples \models H$, without invoking provability. **ILASP** (Law et al. 2014) learns Answer Set Programs whose stable models cover the given examples, grounding evaluation directly in ASP model semantics. **DL-Learner** (Lehmann 2009) learns OWL class expressions by checking class membership against description logic interpretations. Both are likewise **approximate**, as exhaustive search over the hypothesis space is generally infeasible.

Abduction. **Proof-theoretic** abduction constructs hypotheses by extending a proof procedure to treat unresolvable goals as abducibles. The canonical algorithm is the **IFF proof procedure** (Fung and Kowalski 1997), which augments SLD resolution with integrity constraint checking to systematically generate minimal abductive explanations — finding Cases such that $Background \wedge$

Case $\vdash$ *Result* holds syntactically. It is approximate because when multiple explanations exist, selecting among them relies on minimality heuristics.

Model-theoretic abduction instead seeks hypotheses that make the observation semantically entailed, i.e. *Background* $\cup$ *Case* $\models$ *Result*, without constructing a proof. **Reiter's consistency-based diagnosis** (Reiter 1987) computes minimal sets of component abnormalities whose assumption renders the observation consistent with the background theory, evaluated directly against interpretations. **ASP-based abduction** (Sakama and Inoue 2000) uses answer set solvers to enumerate stable models that contain the abduced hypotheses, exploiting the model-theoretic semantics of ASP. Both are approximate when selecting among multiple minimal explanations.

E.2 Probabilistic Inference

Probabilistic inference extends logical inference to handle uncertainty. Here $\Omega = \{0, 1\}$ but truth values live in the space $\mathcal{T}\Omega \cong [0, 1]$, interpreted as probabilities. This paradigm inherits both proof-theoretic and model-theoretic methods from logic, but adds a richer notion of exact and approximate inference reflecting the need to aggregate over possible worlds.

Deduction. **Proof-theoretic** probabilistic deduction computes query probabilities by enumerating SLD derivations and aggregating over proof trees. The canonical algorithm is the **inside-outside algorithm** in PRISM (Sato 1995), which performs dynamic programming over a tabling of SLD derivations to compute exact marginal probabilities. It is exact under the assumption that all proof paths are mutually exclusive (the distribution semantics).
Model-theoretic probabilistic deduction aggregates probabilities over possible worlds and represents the dominant approach in practice. **Weighted Model Counting (WMC)** is the canonical exact algorithm: it sums weights over all satisfying assignments of a Boolean formula,

$$P(\varphi) = \sum_{[\![\varphi]\!]_{\mathcal{I}}=1} \text{weight}(\mathcal{I}),$$

where each assignment carries a weight reflecting probabilistic fact parameters; in practice WMC is implemented via knowledge compilation to d-DNNF (Darwiche and Marquis 2002) or DPLL-based counters such as sharpSAT (Thurley 2006). **Weighted Model Integration (WMI)** (Belle et al. 2015) generalises WMC to hybrid domains with both discrete and continuous variables, computing probability mass by integrating a weight function over the region defined by the models of an SMT formula. In practice, exact WMC/WMI is often replaced by approximate algorithms: **MCMC sampling** (Andrieu et al. 2003) draws samples from the posterior to estimate expectations, **variational inference** (Wainwright and Jordan 2007) optimises a tractable surrogate distribution to approximate the true posterior, and **loopy belief propagation** (Murphy et al. 1999) propagates messages on cyclic graphical models where exact inference is intractable. In all such cases the overall inference procedure is likewise approximate.

Induction. **Proof-theoretic** probabilistic induction learns parameters by counting derivations. The canonical algorithm is PRISM's **inside-outside EM** (Sato and Kameya 2001), which performs the E-step by dynamic programming over tabled SLD proof trees (computing expected derivation counts) and the M-step by normalising those counts — the direct probabilistic

analogue of the inside-outside algorithm for Probabilistic Context-Free Grammars (PCFGs). It is exact when proof paths are mutually exclusive.

Model-theoretic probabilistic induction instead optimises parameters over possible worlds. Given observed data, the objective is maximum likelihood estimation (MLE):

$$\theta^* = \arg\max_{\theta} P(\text{Data} \mid \theta).$$

LFI-ProbLog (Gutmann et al. 2011) realises this by gradient descent on the WMC-based log-likelihood, differentiating through the compiled d-DNNF circuit. When latent variables are present, **WMC-EM** alternates between a model-theoretic E-step (computing posteriors via WMC under current θ) and an M-step (updating weights by gradient ascent); ProbLog (Fierens et al. 2015) applies this schema to learn the probabilistic weights of axioms from partial observations. Both are approximate when the optimisation landscape is non-convex.

The algorithms above perform **parameter learning** — they assume a fixed model structure and optimise its weights. However, the structure itself (i.e., which rules or clauses to include) can also be learned from data in this context. This is **probabilistic structure learning**, realised by algorithms such as **ProbFOIL+** (De Raedt et al. 2015), which extends the FOIL top-down rule learner to learn weighted probabilistic rules under distribution semantics, and **SLIPCOVER** (Bellodi and Riguzzi 2015), which searches the clause space to learn the structure of a ProbLog (De Raedt et al. 2007) program while simultaneously estimating clause probabilities. Both combine the hypothesis-space search of logical ILP with the probabilistic parameter estimation methods described above.

Abduction. **Proof-theoretic** probabilistic abduction finds the most probable set of ground facts (abducibles) such that the observation is syntactically derivable. The canonical algorithm is PRISM's **Viterbi proof procedure** (Sato and Kameya 2001), which performs dynamic programming over tabled SLD proof trees to identify the most probable derivation — the proof-theoretic analogue of the Viterbi algorithm for Hidden Markov Models (HMMs).

Model-theoretic probabilistic abduction seeks the assignment of latent variables (Case) that maximises the posterior given the probabilistic model (Rule) and the observation (Result) — formally, Maximum A Posteriori (MAP) inference:

$$\text{Case}^* = \arg\max_{\text{Case}} P(\text{Case} \mid \text{Rule}, \text{Result}).$$

Exact MAP is computed by the **junction tree algorithm** (Koller and Friedman 2010) via variable elimination over the graphical model. Approximate MAP is realised by **max-product belief propagation** (Murphy et al. 1999), which propagates max-marginal messages on cyclic graphs where exact inference is intractable. Both are model-theoretic as they evaluate probability directly against the model structure rather than via proof construction.

A note on fuzzy vs. probabilistic semantics. Although both yield numbers in $[0, 1]$, the two semantics are fundamentally different. *Fuzzy semantics* takes $\mathcal{T} = \text{Id}$, $\Omega = [0, 1]$, so a formula directly evaluates to a degree via chosen t-norms — deduction is exact evaluation of these connectives. *Probabilistic semantics* takes $\mathcal{T} = \mathcal{D}$ or $\mathcal{G}$, $\Omega = \{0, 1\}$, so the truth basis remains Boolean and we reason about distributions over Boolean outcomes — WMC aggregates over

Boolean assignments. Fuzzy logic models vague concepts with graded truth; probabilistic logic models uncertainty over crisp facts.

E.3 Neural Inference

Neural inference instantiates the Rule–Case–Result schema using differentiable representations and gradient-based optimization. Following the neural semantics from Section 3 ($\mathcal{T} = \mathcal{M}$ or $\mathrm{Id}_{\mathbf{Diff}}$), selected symbols are interpreted by neural modules with parameters θ. All three inference tasks are carried out on a model-theoretic and approximate basis: neural networks evaluate formulas by forward computation rather than symbolic derivation, and they approximate rather than compute the true underlying function exactly.

Deduction (Forward Pass). Given a fixed rule (parameters θ) and a case (input x), the result is the predicted truth value $p_\theta := [\![\varphi]\!]_{\mathcal{I}_\theta}(x) \in [0, 1]$, obtained by **forward propagation** (Goodfellow et al. 2016): the input is passed through successive layers of linear transformations and non-linear activations until a scalar output is produced. The specific architecture instantiating this computation varies by domain — **Convolutional Neural Networks (CNNs)** (Lecun et al. 1998) are the standard choice for grid-structured inputs such as images, while **Transformers** (Vaswani et al. 2017) handle sequence and relational inputs via self-attention. In all cases, deduction is model-theoretic and exact*[4]: the network computes a real-valued output in an exact style; regardless whether it has learned the true predicate extension or not.

Induction (Backpropagation). Given observed data (Cases + Results) and optionally background knowledge $\mathcal{K}$, we update θ by minimising a loss function:

$$\theta^* = \arg\min_\theta \left[\mathcal{L}_{\text{task}}(\theta) + \lambda\, \mathcal{L}_{\text{know}}(\theta)\right],$$

where $\mathcal{L}_{\text{task}}$ measures prediction error on labelled data and $\mathcal{L}_{\text{know}}$ penalises violations of logical or probabilistic constraints from $\mathcal{K}$. The gradient $\nabla_\theta \mathcal{L}$ is computed by **backpropagation** (Rumelhart et al. 1986), which applies the chain rule recursively through the computation graph. The gradient is then consumed by an optimiser: **SGD** performs a fixed-step gradient descent, while **Adam** (Kingma and Ba 2015) adapts per-parameter learning rates using running estimates of the first and second gradient moments and is the standard choice in practice. Both are approximate as they operate on mini-batches rather than the full dataset, and model-theoretic as optimisation proceeds by evaluating the loss against data rather than constructing symbolic proofs.

Abduction (Input Optimization). Keeping θ fixed and a desired output y^* given, neural abduction finds an input x^* such that:

$$x^* = \arg\min_x \mathrm{loss}([\![\varphi]\!]_{\mathcal{I}_\theta}(x),\, y^*)\,.$$

[4]Here we have to distinguish between what we account for: The deduction procedure is of exact nature, while neural networks itself are known to be *universal approximators* meaning that they can approximate arbitrary functions.

Three canonical instantiations illustrate the range of this schema. **Adversarial examples** are computed by the **Fast Gradient Sign Method** (Goodfellow et al. 2014b), which perturbs a base input x_0 by a single signed gradient step to find $x^* \approx x_0$ that flips the network's prediction. **Feature attribution** methods such as **Integrated Gradients** (Sundararajan et al. 2017) accumulate gradients along a path from a baseline to the input, identifying which input dimensions most influence the output. **Neural program synthesis** algorithms such as **DreamCoder** (Ellis et al. 2020) search for latent programs that, when executed, produce the observed output. All three are model-theoretic and approximate: they optimise by evaluating the loss forward through θ rather than constructing a symbolic derivation.

Generative models fit naturally into this abductive schema. In a **Variational Autoencoder (VAE)** (Kingma and Welling 2014), sampling is deductive (Rule + Case $z \to$ Result x), while the encoder $q_\phi(z \mid x)$ is abductive: it recovers the latent explanation z for an observation x by approximate posterior inference — precisely the variational inference algorithm discussed under probabilistic abduction (Appendix E.2), establishing a direct bridge between neural and probabilistic abduction. **Normalizing flows** (Rezende and Mohamed 2015) are invertible by construction, so their abductive direction $z = f^{-1}(x)$ is exact rather than approximate. **GANs** (Goodfellow et al. 2014a) are primarily inductive (training the generator) and deductive (sampling), but *GAN inversion* — finding z such that $G(z) \approx x$ — is approximate abduction, typically realised by gradient-based optimisation in the latent space.