
ULLER: A UNIFIED LANGUAGE FOR LEARNING AND REASONING

A PREPRINT

Emile van Krieken*
University of Edinburgh

Samy Badreddine*
Sony AI
Fondazione Bruno Kessler
UniTrento

Robin Manhaeve
KU Leuven

Eleonora Giunchiglia
TU Wien

May 2, 2024

ABSTRACT

The field of neuro-symbolic artificial intelligence (NeSy), which combines learning and reasoning, has recently experienced significant growth. There now are a wide variety of NeSy frameworks, each with its own specific language for expressing background knowledge and how to relate it to neural networks. This heterogeneity hinders accessibility for newcomers and makes comparing different NeSy frameworks challenging. We propose a unified language for NeSy, which we call ULLER, a Unified Language for LEarning and Reasoning. ULLER encompasses a wide variety of settings, while ensuring that knowledge described in it can be used in existing NeSy systems. ULLER has a neuro-symbolic first-order syntax for which we provide example semantics including classical, fuzzy, and probabilistic logics. We believe ULLER is a first step towards making NeSy research more accessible and comparable, paving the way for libraries that streamline training and evaluation across a multitude of semantics, knowledge bases, and NeSy systems.

1 Introduction

Deep learning has driven innovation in many fields for the past decade. Among the many reasons behind its central role is the ease with which it can be applied to a multitude of problems. Recently, neuro-symbolic (NeSy) methods (see, e.g., [30, 4, 49, 21, 35, 56, 24]), which belong to the NeSy subfield *informed machine learning*, [20, 52] have overcome some well-known problems affecting deep learning models by exploiting *background knowledge* available for the problem at hand. For example, knowledge can help in training models with fewer data points and/or incomplete supervisions, creating models that are compliant by-design with a set of requirements, and being more robust in out-of-distribution prediction.

However, the presence of background knowledge makes it more challenging to obtain “frictionless reproducibility” [17] which characterises machine learning (ML). Indeed, in ML, shared datasets and clear evaluation metrics allow ML practitioners to quickly get started with evaluating new methods and comparing it to existing work. To achieve this goal for NeSy research, we also need *frictionless sharing of knowledge*. Current NeSy frameworks all have different approaches to encode the background knowledge: some use logical languages, like first-order [4, 35], propositional [2, 55, 21], logic programming [30] or answer set programming [56] - with a wide array of different syntaxes - while other methods use plain Python programs [14]. See Section 5 for an overview. To compare the performance of different NeSy systems, a researcher needs to specify the same knowledge in many languages. This is a significant barrier for researchers new to the field, and, even for experts, is a time-consuming and error-prone task.

*Equal contribution.

Correspondence to Emile.van.Krieken@ed.ac.uk and samy.badreddine@sony.com

1.0.1 ULLER, a Unified Language for LEarning and Reasoning

We take a first step towards frictionless sharing of knowledge in the NeSy field by proposing a *Unified Language for Learning and Reasoning* (ULLER, pronounced “OOH-ler” like the god of the Norse mythology). ULLER allows us to express the knowledge used in informed machine learning. The long-term goal is to create a Python library implementing ULLER to be shared among the significant NeSy systems. First, the user expresses the knowledge in ULLER. Then, they load the data, after which they call different NeSy systems with a single line of code to train neural networks, or to use the knowledge at prediction time. This allows the NeSy community (i) to define benchmarks that include both data and knowledge, (ii) to easily compare the available NeSy systems on such benchmarks, and (iii) to lower the barrier to entry into NeSy research for the broader machine learning community.

To achieve the above requires decoupling the *syntax* of the knowledge representation from the *semantics* given by the NeSy system. The syntax of ULLER, defined in Section 2, is syntactic sugar around first-order logic (FOL), with specifically designed *statement bindings*. Statements simplify the process of writing down function application and composition - and hence dealing with data sampling and processing pipelines. We opt for a FOL-like syntax. It generalises propositional logic, while being a common language for declaring general constraints. We adapt the FOL syntax to one that is more familiar to ML researchers, who are mostly used to writing procedural statements like in Python, while having a well-defined semantics for logicians. Furthermore, FOL is highly expressive: We believe that it can express all knowledge currently used in NeSy methods.

The semantics of ULLER (Section 3), depends on (i) an *interpretation*, often referred to as a “symbol grounding” [23], which maps symbols to meanings, and (ii) a “NeSy system”, which takes knowledge and its interpretation, and computes loss functions and outputs accordingly. We formalise the differences between NeSy systems by what they compute given a program in ULLER and an interpretation. We also provide examples for several common systems, such as classical logic, fuzzy logic (such as Logic Tensor Networks [4]), probabilistic logic (such as Semantic Loss [55] and DeepProbLog [30]). This highlights the flexibility of our language, as it can be used to express knowledge in many formalisms.

2 Syntax of ULLER

Let \mathcal{V} be a set of variable symbols, \mathcal{C} be a set of constant symbols, \mathcal{D} be a set of a set of domain symbols, \mathcal{P} be a set of predicate symbols, \mathcal{T} be a set of property symbols, and \mathcal{F} be a set of function symbols. We then define the syntax of ULLER $\mathcal{L}_{\text{ULLER}}$ as a context-free grammar:

$$\begin{aligned}
 F &::= \forall x \in D (F) \mid \exists x \in D (F) \\
 F &::= F \wedge F \mid F \vee F \mid F \Rightarrow F \mid \neg F \mid P(T, \dots, T) \mid (F) \\
 F &::= x := f(T, \dots, T) (F) \\
 T &::= x \mid c \mid T.\text{prop} \mid T + T \mid T - T \mid \dots
 \end{aligned} \tag{1}$$

where $D \in \mathcal{D}$, $x \in \mathcal{V}$, $c \in \mathcal{C}$, $f, +, - \in \mathcal{F}$, $P \in \mathcal{P}$ and $\text{prop} \in \mathcal{T}$. The nonterminal symbol F is a *formula* and T is a *term*. We call $x := f(T, \dots, T)(F)$ a statement binding, or just *statement*, which we discuss in Section 2.1. Notice that, except for basic arithmetic operations ($+$, $-$, \dots), functions only appear in statements.

The syntax of ULLER does not include a special syntactic construct for neural networks. Instead, we treat them as functions, where the intended meaning is given by the semantics specified by the NeSy system. We therefore hide how the NeSy system uses the neural networks to the user, so the focus is on specifying constraints rather than implementation details.

Syntactic Sugar. We use $\forall x_1 \in D_1, x_2 \in D_2 (F)$ as syntactic sugar for $\forall x_1 \in D_1 (\forall x_2 \in D_2 (F))$ for the quantifiers. We also use $x_1 := f_1(T, \dots, T), x_2 := f_2(T, \dots, T) (F)$ as syntactic sugar for $x_1 := f_1(T, \dots, T) (x_2 := f_2(T, \dots, T) (F))$. Finally, we also allow for binary predicates in infix notation, such as $T \leq T$.

Typing. ULLER is a dynamically typed language. We do not guarantee syntactically nor via type checker that functions and predicates only take arguments from the domain defined in their interpretations. This mimics the design of the type system of Python.

2.1 Statements

A key design choice of ULLER is the use of special statement bindings $x := f(T, \dots, T) (F)$ to declare (possibly random) variables obtained by applying (possibly non-deterministic) functions. The function symbols f appear only in statements, and not in the definition of terms T , like in standard FOL. Statements simplify the composition of

functions. They give a syntax that is both familiar to ML researchers who are used to writing Python, and gives a clear separation between the machine learning pipeline that processes data and the constraints on the data given by the logic. We will motivate statements with the two following examples.

Example 2.1 (Procedural composition of functions). Consider the MNISTAdd example from Appendix A. To emphasise the ease of composing functions in ULLER, consider a scenario where (i) the two MNIST digits always represent different digits, and (ii) the classifier f expects greyscale images while the data points in the dataset T are RGB images. We can easily apply transformations and formulate the new condition using ULLER statements:

$$\begin{aligned} \forall x \in T(\\ & x'_1 := \text{greyscale}(x.\text{im1}), x''_1 := \text{normalise}(x'_1), \\ & x'_2 := \text{greyscale}(x.\text{im2}), x''_2 := \text{normalise}(x'_2), \\ & n_1 := f(x''_1), n_2 := f(x''_2), \\ & ((n_1 + n_2 = x.\text{sum}) \wedge n_1 \neq n_2) \\) \end{aligned} \quad (2)$$

◁

Example 2.2 (Scoping independence). Another key feature of ULLER statements is that they explicitly delimit the scopes of variables, giving control over the memoisation and independence assumptions. Consider a non-deterministic function $\text{dice}()$ which associates a probability to each outcome of a six-sided dice throw. Consider the following program written in ULLER:

$$x := \text{dice}() \ (x = 6 \wedge \text{even}(x)). \quad (3)$$

The formula asks whether a die-throw outcome is both a six and even. For a fair dice, the probability of the formula is $\frac{1}{6}$ under probabilistic semantics.

Now consider the alternative ULLER program:

$$(x := \text{dice}() \ (x = 6)) \wedge (x := \text{dice}() \ (\text{even}(x))). \quad (4)$$

In this program, we throw two independent dice, and check if the first lands on six and the second is even. For fair dice, the probability of this formula is $\frac{1}{6} \cdot \frac{1}{2} = \frac{1}{12}$.

Consider a similar program in regular FOL (which is not allowed in ULLER):

$$(\text{dice}() = 6) \wedge \text{even}(\text{dice}()) \quad (5)$$

Here, it is ambiguous whether the outcomes of the dice are shared like in the ULLER program of (3) or not, like in (4). Many probabilistic NeSy frameworks choose the first option and memorise the outcome of the function. We argue that this behaviour should not be a default assumption from the NeSy system. Instead, dependence and memoisation scopes should be explicitly defined by the program. ULLER statements give researchers control over these scopes. ◁

3 Semantics of ULLER

In this Section, we define the semantics of ULLER. In Section 3.1 we discuss how ULLER interprets the symbols in the language, such as the function and domain symbols. Then, in Section 3.2, we discuss how different NeSy systems interpret the formulas in ULLER.

3.1 Interpretation of the Symbols

To assign meaning to ULLER programs, we need to *interpret* the non-logical symbols in $\mathcal{L}_{\text{ULLER}}$, that is, \mathcal{D} , \mathcal{P} , \mathcal{F} , and \mathcal{C} , using an interpretation function I .

Definition 3.1. An interpretation I is a function assigning a meaning to the symbols in $\mathcal{L}_{\text{ULLER}}$ under the following rules, where $\Omega_1, \dots, \Omega_n, \Omega_{n+1}$ are sets.

1. The interpretation of a domain $D \in \mathcal{D}$ is a set Ω .
2. The interpretation of a predicate P of arity n is a function of n domains to $\{0, 1\}$. That is, $I(P) : \Omega_1 \times \dots \times \Omega_n \rightarrow \{0, 1\}$.
3. The interpretation of the predicate $\text{true} \in \mathcal{P}$ is the identity function on $\{0, 1\}$, that is, $I(\text{true}) : \{0, 1\} \rightarrow \{0, 1\}$ such that $I(\text{true})(x) = x$.

4. The interpretation of a constant c is an element of a domain $I(c) \in \Omega_i$.
5. The interpretation of a function f of arity n is a *conditional probability distribution*¹ $I(f) : \Omega_1 \times \dots \times \Omega_n \rightarrow (\Omega_{n+1} \rightarrow [0, 1])$. That is, for any set of inputs $x_1 \in \Omega_1, \dots, x_n \in \Omega_n$, $I(f)(x_1, \dots, x_n)$ is a probability distribution on the domain Ω_{n+1} . If for all $x_1 \in \Omega_1, \dots, x_n \in \Omega_n$ the probability distribution $I(f)(x_1, \dots, x_n)$ is a deterministic distribution, we say that $I(f)$ is a *deterministic function*.

We give a probabilistic interpretation to both domains and functions. In particular, we treat functions, such as neural networks, as a *conditional distribution* given assignments $x_1 \in \Omega_1, \dots, x_n \in \Omega_n$ to input variables. This allows us to represent the uncertainty of the neural networks, which NeSy systems using, for example, probabilistic and fuzzy semantics can use to compute probabilities and fuzzy truth values. We will also frequently want to use regular (deterministic) functions $f : \Omega_1 \times \dots \times \Omega_n \rightarrow \Omega_{n+1}$. A regular function is a special case of a conditional distribution that we refer to as a deterministic function. We define deterministic functions with a conditional distribution using the Dirac delta distribution at $f(x_1, \dots, x_n)$ for continuous distributions, and a distribution that assigns 1 to the output value $f(x_1, \dots, x_n)$ for finite domains, and 0 to the other values.

3.2 Semantics of neuro-symbolic systems

We next define the meaning of a formula in $\mathcal{L}_{\text{ULLER}}$, which requires both an interpretation I and a *NeSy system* $\llbracket \cdot \rrbracket$. Here, $\llbracket \cdot \rrbracket$ is a function that interprets the semantics of the program statements in $\mathcal{L}_{\text{ULLER}}$. We also need a variable assignment $\eta : \mathcal{V} \rightarrow \mathcal{O}$ that maps variables $v \in \mathcal{V}$ to an element of a domain $\mathcal{O} = \cup_i \Omega_i$, where $\Omega_i = I(D_i)$ is a set associated to a domain $D_i \in \mathcal{D}$.

Definition 3.2. A *NeSy structure* is a tuple $(I, \eta, \mathcal{B}, \llbracket \cdot \rrbracket_{I, \eta})$ where I is an interpretation, $\eta : \mathcal{V} \rightarrow \mathcal{O}$ is a variable assignment, \mathcal{B} is a set of outputs and $\llbracket \cdot \rrbracket_{I, \eta} : \mathcal{L}_{\text{ULLER}} \rightarrow \mathcal{B} \cup \mathcal{O}$ is a *neuro-symbolic system* which is a function that assigns an output in \mathcal{B} to each formula in $\mathcal{L}_{\text{ULLER}}$ and a domain element in \mathcal{O} for terms T . If the interpretation and variable assignment are clear from the context, we write $\llbracket \cdot \rrbracket$ for $\llbracket \cdot \rrbracket_{I, \eta}$.

We discuss several NeSy systems and their semantics for the NeSy language in the following sections, and provide a visual overview in Figure 1. Each NeSy system is defined over some set of outputs \mathcal{B} . For example, classical logic is defined over the output $\{0, 1\}$, while fuzzy logics are defined over the interval $[0, 1]$. A neuro-symbolic system $\llbracket \cdot \rrbracket_{I, \eta}$ defines the semantics of a language expression. When a language expression is a term T , $\llbracket T \rrbracket_{I, \eta}$ returns an element of the universe \mathcal{O} . When the language expression is a formula F , it returns an element in \mathcal{B} .

Notation. We use $\eta[x \mapsto a]$ to update a variable assignment η with the assignment of a to x :

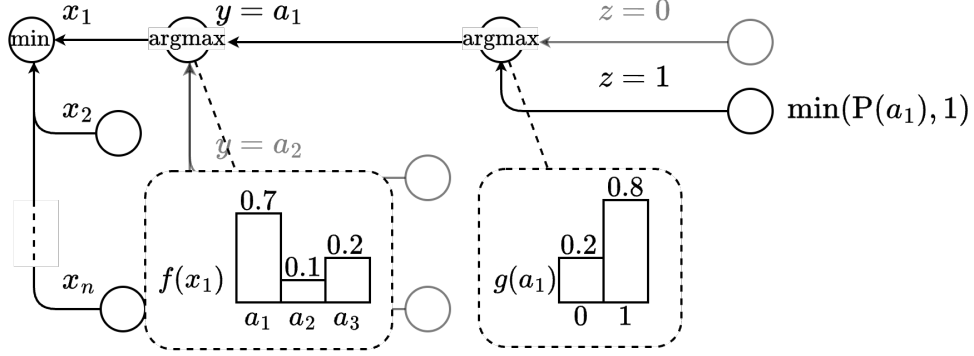
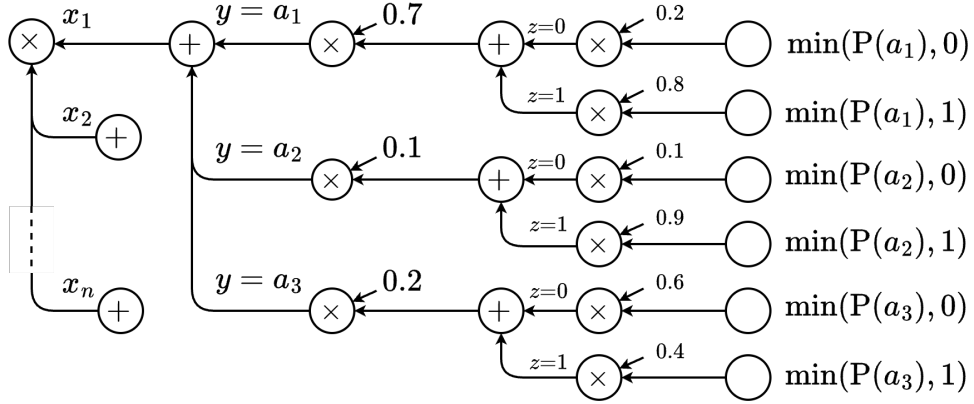
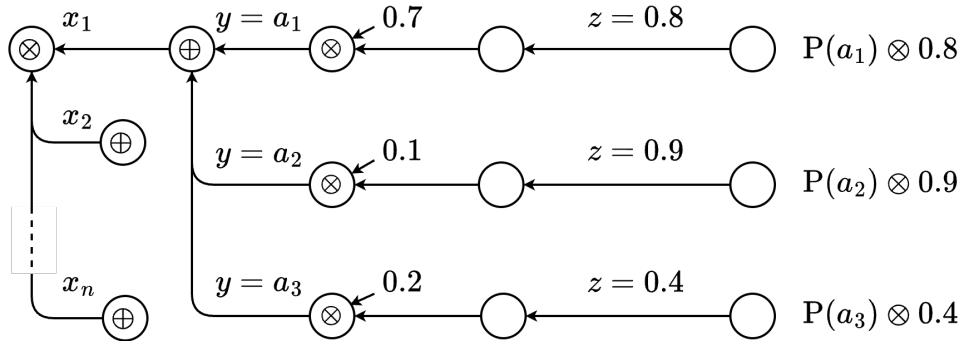
$$\begin{aligned} \eta[x \mapsto a](x) &= a \\ \eta[x \mapsto a](x') &= \eta(x') \quad \text{for } x' \neq x \end{aligned} \tag{6}$$

We also define $p_f(a|T_1, \dots, T_n) = I(f)(\llbracket T_1 \rrbracket, \dots, \llbracket T_n \rrbracket)(a)$, which computes the probability of the element $a \in \Omega_{n+1}$ under the distribution $I(f)$, conditioned on the interpretation of the terms T_1 to T_n . That is, under $\llbracket T_1 \rrbracket, \dots, \llbracket T_n \rrbracket$. In the coming sections, we will frequently use this shorthand to talk about the semantics of the different NeSy systems.

3.3 Classical semantics

We first define the semantics of the NeSy language if we “choose” an option deterministically from a conditional distribution. Then, under the classical semantics of the logical symbols, ULLER is a regular first-order logic. A common way to make a deterministic choice from a distribution is to take its mode, that is, the most likely output according to the neural network.

¹To be precise, our definition is equivalent to a *probability kernel* or *Markov kernel*, which is a formalisation of the concept of a conditional probability distributions in measure theory.

Classical**Probabilistic****Fuzzy**

$$\forall x \in X (y := f(x), z := g(y) (P(y) \wedge \text{true}(z)))$$

Figure 1: The meaning of an example ULLER formula under classical, probabilistic and fuzzy semantics. We interpret the function symbols as conditional distributions $f : \{x_1, \dots, x_n\} \rightarrow (\{a_1, a_2, a_3\} \rightarrow [0, 1])$ and $g : \{(a_1, a_2, a_3) \rightarrow (\{0, 1\} \rightarrow [0, 1])\}$. With abuse of notation, we ignore $I()$ and \mathbb{I} .

Definition 3.3. The classical structure $(I, \eta, \{0, 1\}, \llbracket \cdot \rrbracket_{I, \eta}^C)$ is defined on boolean outputs $\{0, 1\}$ as:

$$\llbracket \forall x \in D (F) \rrbracket_{I, \eta}^C = \min_{a \in I(D)} \llbracket F \rrbracket_{I, \eta[x \mapsto a]}^C \quad (7)$$

$$\llbracket \exists x \in D (F) \rrbracket^C = \llbracket \neg \forall x \in D (\neg F) \rrbracket^C \quad (8)$$

$$\llbracket F_1 \wedge F_2 \rrbracket^C = \min(\llbracket F_1 \rrbracket^C, \llbracket F_2 \rrbracket^C), \llbracket F_1 \vee F_2 \rrbracket^C = \llbracket \neg(\neg F_1 \wedge \neg F_2) \rrbracket^C \quad (9)$$

$$\llbracket \neg F \rrbracket^C = 1 - \llbracket F \rrbracket^C, \llbracket F_1 \Rightarrow F_2 \rrbracket^C = \llbracket \neg F_1 \vee F_2 \rrbracket^C \quad (10)$$

$$\llbracket P(T_1, \dots, T_n) \rrbracket_{I, \eta}^C = I(P)(\llbracket T_1 \rrbracket^C, \dots, \llbracket T_n \rrbracket^C) \quad (11)$$

$$\llbracket x \rrbracket_{I, \eta}^C = \eta(x), \llbracket c \rrbracket^C = I(c) \quad (12)$$

$$\llbracket T_1 + T_2 \rrbracket^C = \llbracket T_1 \rrbracket^C + \llbracket T_2 \rrbracket^C \quad (13)$$

$$\llbracket T.\text{prop} \rrbracket^C = \text{get}(\llbracket T \rrbracket^C, \text{prop}) \quad (14)$$

$$\llbracket x := f(T_1, \dots, T_n)(F) \rrbracket_{I, \eta}^C = \llbracket F \rrbracket_{I, \eta[x \mapsto \arg\max_{a \in \Omega_{n+1}} p_f(a|T_1, \dots, T_n)]}^C \quad (15)$$

In Equation 14, $\text{get}(\llbracket T \rrbracket^C, \text{prop})$ is a deterministic function that retrieves the value of an object property.

Equation 15 demands some explanation. The $\arg \max$ takes the probability distribution given by the interpretation of the function f and chooses a value from the codomain Ω_{n+1} . In the classical structure, this choice is made deterministically by picking the mode of the distribution: the most likely element a . Then we assign this element a to the variable x through the variable assignment $\eta[x \mapsto a]$, and evaluate the rest of the formula F under this new assignment.

Importantly, the classic semantics allows us to prove whether a neuro-symbolic system “is faithful” to classical logic when all functions are deterministic. We formally introduce this notion by noting we can transform any program into a deterministic program by choosing the mode of the distribution like in Equation 15.

Definition 3.4. For some interpretation I , the *mode interpretation* \hat{I} , is another interpretation such that for all function symbols $f \in \mathcal{F}$, $\hat{I}(f)$ returns the mode of p_f . That is, $\hat{p}_f(a|T_1, \dots, T_n) = \delta(a - \arg \max_{a'} p_f(a'|T_1, \dots, T_n))$, where δ is the Dirac delta distribution. Then a neuro-symbolic system $\llbracket \cdot \rrbracket$ is *classical in the limit* if for all language statements $L \in \mathcal{L}_{\text{ULLER}}$, $\llbracket L \rrbracket_{\hat{I}, \eta} = \llbracket L \rrbracket_{I, \eta}^C$.

3.4 Probabilistic Semantics

Probabilistic semantics, also known as weighted model counting or possible world semantics in the literature, computes the probability that a formula is true. This is done by iterating over all possible assignments to the variables.

In the next sections, we will not redefine semantics whenever it is equal to the classical semantics, up to domain differences. For instance, we will not repeat constants and variable semantics.

Definition 3.5. The probabilistic structure $(I, \eta, [0, 1], \llbracket \cdot \rrbracket^P)$ is defined on probabilities $[0, 1]$ as:

$$\llbracket \forall x \in D (F) \rrbracket^P = \prod_{a \in I(D)} \llbracket F \rrbracket_{I, \eta[x \mapsto a]}^P \quad (16)$$

$$\llbracket F_1 \wedge F_2 \rrbracket^P = \llbracket F_1 \rrbracket^P \cdot \llbracket F_2 \rrbracket^P \quad (17)$$

$$\llbracket x := f(T_1, \dots, T_n) (F) \rrbracket^P = \mathbb{E}_{a \sim p_f(\cdot|T_1, \dots, T_n)} \left[\llbracket F \rrbracket_{I, \eta[x \mapsto a]}^P \right] \quad (18)$$

In probabilistic semantics, a function $f(x)$ is interpreted as a conditional distribution conditioned on x . In this case, we require computing the *expectation* of the formulas being true under the interpreted functions. This happens in Equation 18. The computation of the expectation depends on whether the output domain Ω_{n+1} is discrete or continuous. For discrete domains, Equation 18 equals

$$\llbracket x := f(T_1, \dots, T_n) (F) \rrbracket^P = \sum_{a \in \Omega_{n+1}} p_f(a|T_1, \dots, T_n) \cdot \llbracket F \rrbracket_{I, \eta[x \mapsto a]}^P, \quad (19)$$

while for continuous domains it equals

$$\llbracket x := f(T_1, \dots, T_n) (F) \rrbracket^P = \int_{a \in \Omega_{n+1}} p_f(a|T_1, \dots, T_n) \cdot \llbracket F \rrbracket_{I, \eta[x \mapsto a]}^P da. \quad (20)$$

We should note that probabilistic semantics in most practical cases will be intractable because of the exponential recursion introduced in Equation 19, not to mention the usually intractable integral in Equation 20 [5]. We can speed this up with techniques that compile formulas into representations where computing the probability of the formula is tractable [8, 11]. The probabilistic semantics is classical in the limit (Appendix C.1), and is connected to the standard weighted model counting semantics used in, for example, Semantic Loss [55], SPL [2] and DeepProbLog [30]. See Appendix D for details.

We can generalise the probabilistic semantics to algebraic model counting [25, 15] by considering semirings \mathcal{B} together with a product and a sum operation. This, for example, allows us to compute the most likely assignment to the variables in a formula, or to compute the gradient of the probabilistic semantics using dual numbers.

3.5 Fuzzy Semantics

The setup for fuzzy semantics is very similar to that of the probabilistic semantics. The two differences are using t-norms and t-conorms to connect fuzzy truth values, and the interpretation of sampling from boolean distributions.

Definition 3.6. The fuzzy structure $(I_F, \eta, [0, 1], \llbracket \cdot \rrbracket^F)$, where I_F is an interpretation I except that the predicate symbol true is interpreted as the identity function on $[0, 1]$, is defined on fuzzy truth values $[0, 1]$ as:

$$\llbracket \forall x \in D (F) \rrbracket_{I_F, \eta}^F = \bigotimes_{a \in I(D)} \llbracket F \rrbracket_{I_F, \eta[x \mapsto a]}^F \quad (21)$$

$$\llbracket \exists x \in D (F) \rrbracket_{I_F, \eta}^F = \bigoplus_{a \in I(D)} \llbracket F \rrbracket_{I_F, \eta[x \mapsto a]}^F \quad (22)$$

$$\llbracket F_1 \wedge F_2 \rrbracket^F = \llbracket F_1 \rrbracket^F \otimes \llbracket F_2 \rrbracket^F, \quad \llbracket F_1 \vee F_2 \rrbracket^F = \llbracket F_1 \rrbracket^F \oplus \llbracket F_2 \rrbracket^F \quad (23)$$

$$\llbracket \text{true}(x) \rrbracket_{I, \eta}^F = \eta(x), \quad \text{if } \eta(x) \in [0, 1] \quad (24)$$

$$\llbracket x := f(T_1, \dots, T_n)(F) \rrbracket^F = \begin{cases} \llbracket F \rrbracket_{I_F, \eta[x \mapsto p_f(1|T_1, \dots, T_n)]}^F & \text{if } \Omega_{n+1} = \{0, 1\} \\ \bigoplus_{a \in \Omega_{n+1}} p_f(a|T_1, \dots, T_n) \otimes \llbracket F \rrbracket_{I_F, \eta[x \mapsto a]}^F & \text{if } \Omega_{n+1} \text{ is finite} \end{cases} \quad (25)$$

where $\otimes : [0, 1] \times [0, 1] \mapsto [0, 1]$ is a fuzzy t-norm and $\oplus : [0, 1] \times [0, 1] \mapsto [0, 1]$ is a fuzzy t-conorm [4, 48].

In the first case of Equation 25, fuzzy semantics manipulates distributions over boolean codomains $\Omega_{n+1} = \{0, 1\}$ as a single truth value $p_f(1|T_1, \dots, T_n)$. The second case is defined for discrete, non boolean codomains. Fuzzy semantics reasons disjointly over all possible outcomes $a \in \Omega_{n+1}$ by interpreting the probability $p_f(a|T_1, \dots, T_n) \in [0, 1]$ as truth degrees. This truth degree is then conjoined with the interpretation of the rest of the formula F . Intuitively, they ask if there “exists a such that $f(T_1, \dots, T_n)$ maps to a and that a verifies the rest of the formula F ”. We do not give a semantics for continuous or infinite domains in the fuzzy semantics, as we do not know of a standard definition in the neuro-symbolic literature. The fuzzy semantics is classical in the limit (see Appendix C.2), and is closely related to differentiable fuzzy logics such as Logic Tensor Networks [48, 4] (see Appendix E).

In addition to Fuzzy Logics with t-norms and t-conorms for conjunction and disjunction, other NeSy frameworks such as DL2 [18] and STL [51] can also be implemented with this semantics. While fuzzy logic acts on truth values in $[0, 1]$, DL2 acts on truth values in $[-\infty, 0]$ and STL in $[-\infty, \infty]$. They choose appropriate differentiable operators to implement the conjunction and disjunction. We refer the reader to [42] for details.

3.6 Sampling semantics

The sampling semantics $\llbracket \cdot \rrbracket^S$ is a simple modification to the classical semantics. It samples a value from each conditional distribution and uses that value to evaluate the formula. Therefore, the only difference in $\llbracket \cdot \rrbracket^S$ with classical semantics in Definition 3.3 is in Equation 15:

$$\llbracket x := f(T_1, \dots, T_n)(F) \rrbracket^S = \llbracket F \rrbracket_{I, \eta[x \mapsto \text{sample}(p_f(\cdot|T_1, \dots, T_n))]}^S \quad (26)$$

Here, `sample` is a (random) function that takes a probability distribution and samples a value from the codomain Ω_{n+1} under the distribution $p_f(\cdot|T_1, \dots, T_n)$. We can repeat the computation of the sampling semantics $\llbracket \cdot \rrbracket^S$ to reduce variance, like in standard Monte Carlo methods. This semantics can be combined with gradient estimation methods to learn the parameters of neural networks [39, 50]. A recent implementation of gradient estimation in the context of NeSy is the CatLog derivative trick [14], but any type of estimator based on the score function (commonly known as REINFORCE) can be used [26]. See Appendix B for a short discussion.

4 Learning and Reasoning

This section describes how to use ULLER for neuro-symbolic learning and reasoning. For a learning setting, we extend the definition of an interpretation (Definition 3.1) to a *parameterised interpretation*. A parameterised implementation allows us to implement neural networks with learnable parameters. For instance, a function `model()` can be interpreted as a neural network $I_\theta(\text{model}) = NN_\theta$.

Definition 4.1. A *parameterised interpretation* is an interpretation I_θ that is uniquely defined by a set of parameters $\theta \in \mathbb{R}^d$.

Let $F \in \mathcal{L}_{\text{ULLER}}$ denote a ULLER formula that has a quantifier ranging over a dataset symbol T (for instance Example 2.1). Learning a parameterised interpretation typically involves searching for an optimal set of parameters $\theta^* \in \mathbb{R}^d$ maximising the neuro-symbolic system on F over a dataset Ω_T . In most machine learning settings, we are interested in minimising a loss function over a random minibatch $x_1, \dots, x_n \sim \Omega_T$. We can define such a loss function and corresponding minimisation problem with

$$L(\theta) = -\llbracket F \rrbracket_{I_\theta \cup \{T \mapsto \{x_1, \dots, x_n\}\}, \{\}}, \quad \theta^* = \arg \max_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta). \quad (27)$$

To allow for minibatching, we interpret the domain symbol T as the minibatch $\{x_1, \dots, x_n\}$. We can easily implement variations of this loss. For instance, we can combine multiple formulas and give each different weights. Notice that, for probabilistic and fuzzy semantics, $L(\theta)$ is differentiable, allowing us to use common optimisers. However, not all NeSy structures can be optimised over: This loss only makes sense when a semantics returns a value in an ordered set \mathcal{B} , but we also allow NeSy structures to return other kinds of values.

A different pattern, more related to reasoning, is to find the input x that maximises or minimises the neuro-symbolic system:

$$x^* = \arg \max_{x' \in X} \llbracket F \rrbracket_{I_\theta \cup \{T \mapsto \{x\}\}, \{\}} \quad (28)$$

This strategy can be combined with adversarial learning to first find the input that most violates the background knowledge, and then corrects that input [34].

5 Related Work

The last decade has seen the rise of neuro-symbolic frameworks that allow for specifying knowledge about the behaviour of neural networks symbolically [31]. However, unlike ULLER they are restricted to a single semantics, usually variations of probabilistic (Section 3.4) or fuzzy semantics (Section 3.5). The majority of current frameworks use the syntactic *neural predicate* construct as discussed in Section 3.1. DeepProbLog [30] is a probabilistic logic programming language [12] with neural predicates. Variations of its syntax are used in multiple follow-up works [13, 53, 28]. Scallop [24] chooses to restrict its language to Datalog to improve scalability, among others [29]. For ULLER, we choose to use an expressive first-order language, leaving scalable inference to the implementation of the NeSy system. Other NeSy frameworks are based on Answer Set Programming [56, 41, 3], relational languages [36, 33, 9], temporal logics [45] and description logics [54, 43, 44], while Logic Tensor Networks [4] is also based on first-order logic, among others [32, 16]. Finally, many commonly used NeSy frameworks are restricted to propositional logic [55, 2, 27, 10, 21, 18].

Logic of Differentiable Logics (LDL) [42] defines a first-order language to compare formal properties of several NeSy frameworks. Compared to ULLER, LDL is strongly typed, while ULLER is weakly typed, and LDL does not model probabilistic semantics. In LDL, uncertainty comes from predicates, rather than functions, and does not have a syntactic construct like ULLERs statement blocks. Pylon [1] is a Python library similar in goal to ULLER. It also allows for expressing propositional logic (CNF) formulas, which can then get compiled into a Semantic Loss or fuzzy loss functions. However, by being restricted to a propositional language, Pylon is limited in expressiveness, and requires the user to manually ground out formulas.

ULLER is also heavily inspired by probabilistic programming languages [22] such as Stan [7] that specify probabilistic models in a high-level language. In particular, ULLER can be considered a first-order probabilistic programming language (FOPPL) [46] defined on boolean outputs. These boolean outputs represent the conditioning (observations) of the probabilistic model. By being first-order, the language is restricted to having a finite number of random variables. Other FOPPL languages centred on neural networks include Pyro [6] and ProbTorch [40]. These languages enforce a probabilistic semantics corresponding to that of ULLER defined in Section 3.4. However, ULLER does not enforce this semantics and also supports, for instance, fuzzy semantics. We leave an in-depth analysis of the relations between ULLER and aforementioned probabilistic programming languages for future work.

Other related work attempts to define building blocks for neuro-symbolic AI [47] or to categorise existing approaches [38]. We instead focus on a particular set of informed machine learning approaches, and develop a unifying language to allow communicating with them.

6 Conclusion

We introduced ULLER, a Unified Language for LEarning and Reasoning. ULLER is a first-order logic language designed for neuro-symbolic learning and reasoning, with a special statement syntax for constraining neural networks. We showed how to implement the common fuzzy and probabilistic semantics in ULLER, allowing for easy comparison between different NeSy systems. For future work, we want to implement ULLER as an easy-to-use Python library to increase the “frictionless reproducibility” of NeSy research. In this library, a researcher can easily write and share knowledge, and develop new NeSy benchmarks. We also believe such a library is a good avenue for reducing the barrier of entry into NeSy research.

Acknowledgements

We would like to thank Frank van Harmelen, Tarek Richard Besold, Luciano Serafini, Antonio Vergari, Pasquale Minervini, Thiviyan Thanapalasasingam, Guy van den Broeck, Connor Pryor, Patrick Koopmann, and Mihaela Stoian for fruitful discussions during the writing of this paper. This work was supported by the EU H2020 ICT48 project “TAI-LOR” under contract #952215. Emile van Krieken was funded by ELIAI (The Edinburgh Laboratory for Integrated Artificial Intelligence), EPSRC (grant no. EP/W002876/1).

References

- [1] Ahmed, K., Li, T., Ton, T., Guo, Q., Chang, K., Kordjamshidi, P., Srikumar, V., den Broeck, G.V., Singh, S.: PYLON: A pytorch framework for learning with constraints. In: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022. pp. 13152–13154. AAAI Press (2022). <https://doi.org/10.1609/AAAI.V36I11.21711>, <https://doi.org/10.1609/aaai.v36i11.21711>
- [2] Ahmed, K., Teso, S., Chang, K., den Broeck, G.V., Vergari, A.: Semantic probabilistic layers for neuro-symbolic learning. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022 (2022), http://papers.nips.cc/paper_files/paper/2022/hash/c182ec594f38926b7fcb827635b9a8f4-Abstract-Conference.html
- [3] Aspis, Y., Broda, K., Lobo, J., Russo, A.: Embed2Sym - Scalable Neuro-Symbolic Reasoning via Clustered Embeddings. In: Proceedings of the Nineteenth International Conference on Principles of Knowledge Representation and Reasoning. pp. 421–431. International Joint Conferences on Artificial Intelligence Organization, Haifa, Israel (Jul 2022). <https://doi.org/10.24963/kr.2022/44>
- [4] Badreddine, S., d’Avila Garcez, A., Serafini, L., Spranger, M.: Logic Tensor Networks. Artificial Intelligence **303**, 103649 (Feb 2022). <https://doi.org/10.1016/j.artint.2021.103649>
- [5] Belle, V., Passerini, A., Van den Broeck, G.: Probabilistic inference in hybrid domains by weighted model integration. In: Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI). vol. 2015, pp. 2770–2776. IJCAI-INT JOINT CONF ARTIF INTELL (2015)
- [6] Bingham, E., Chen, J.P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P.A., Horsfall, P., Goodman, N.D.: Pyro: Deep universal probabilistic programming. Journal of Machine Learning Research **20**, 28:1–28:6 (2019)
- [7] Carpenter, B., Gelman, A., Hoffman, M.D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M.A., Guo, J., Li, P., Riddell, A.: Stan: A probabilistic programming language. Journal of statistical software **76** (2017)
- [8] Chavira, M., Darwiche, A.: On probabilistic inference by weighted model counting. Artificial Intelligence **172**(6), 772–799 (Apr 2008). <https://doi.org/10.1016/j.artint.2007.11.002>
- [9] Cohen, W.W.: TensorLog: A Differentiable Deductive Database. arXiv:1605.06523 [cs] (Jul 2016)
- [10] Daniele, A., van Krieken, E., Serafini, L., van Harmelen, F.: Refining neural network predictions using background knowledge. Mach. Learn. **112**(9), 3293–3331 (2023). <https://doi.org/10.1007/S10994-023-06310-3>, <https://doi.org/10.1007/s10994-023-06310-3>

- [11] Darwiche, A.: SDD: A new canonical representation of propositional knowledge bases. IJCAI International Joint Conference on Artificial Intelligence pp. 819–826 (2011). <https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-143>
- [12] De Raedt, L., Kimmig, A.: Probabilistic (logic) programming concepts. Machine Learning **100**(1), 5–47 (Jul 2015). <https://doi.org/10.1007/s10994-015-5494-z>
- [13] De Smet, L., Martires, P.Z.D., Manhaeve, R., Marra, G., Kimmig, A., De Raedt, L.: Neural Probabilistic Logic Programming in Discrete-Continuous Domains (Mar 2023). <https://doi.org/10.48550/arXiv.2303.04660>
- [14] De Smet, L., Sansone, E., Zuidberg Dos Martires, P.: Differentiable sampling of categorical distributions using the catlog-derivative trick. Advances in Neural Information Processing Systems **36** (2024)
- [15] Derkinderen, V., Manhaeve, R., Dos Martires, P.Z., De Raedt, L.: Semirings for probabilistic and neuro-symbolic logic programming. International Journal of Approximate Reasoning p. 109130 (2024)
- [16] Diligenti, M., Gori, M., Sacca, C.: Semantic-based regularization for learning and inference. Artificial Intelligence **244**, 143–165 (2017)
- [17] Donoho, D.: Data science at the singularity. arXiv preprint arXiv:2310.00865 (2023)
- [18] Fischer, M., Balunovic, M., Drachsler-Cohen, D., Gehr, T., Zhang, C., Vechev, M.: DL2: training and querying neural networks with logic. In: International Conference on Machine Learning. pp. 1931–1941. PMLR (2019)
- [19] Foerster, J., Farquhar, G., Al-Shedivat, M., Rocktäschel, T., Xing, E., Whiteson, S.: DiCE: The infinitely differentiable monte carlo estimator. In: International Conference on Machine Learning. pp. 1529–1538 (2018)
- [20] Giunchiglia, E., Stoian, M.C., Lukasiewicz, T.: Deep Learning with Logical Constraints. In: Raedt, L.D. (ed.) Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022. pp. 5478–5485. ijcai.org (2022). <https://doi.org/10.24963/ijcai.2022/767>
- [21] Giunchiglia, E., Tatomir, A., Stoian, M.C., Lukasiewicz, T.: CCN+: A neuro-symbolic framework for deep learning with requirements. International Journal of Approximate Reasoning p. 109124 (2024). <https://doi.org/10.1016/j.ijar.2024.109124>
- [22] Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: Future of software engineering proceedings, pp. 167–181 (2014)
- [23] Harnad, S.: The symbol grounding problem. Physica D: Nonlinear Phenomena **42**(1-3), 335–346 (1990)
- [24] Huang, J., Li, Z., Chen, B., Samel, K., Naik, M., Song, L., Si, X.: Scallop: From Probabilistic Deductive Databases to Scalable Differentiable Reasoning. In: Advances in Neural Information Processing Systems (May 2021)
- [25] Kimmig, A., Van den Broeck, G., De Raedt, L.: Algebraic model counting. Journal of Applied Logic **22**, 46–62 (2017)
- [26] Kool, W., van Hoof, H., Welling, M.: Buy 4 REINFORCE samples, get a baseline for free! p. 14 (2019)
- [27] van Krieken, E., Thanapalasingam, T., Tomczak, J.M., van Harmelen, F., ten Teije, A.: A-nesi: A scalable approximate method for probabilistic neurosymbolic inference. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023 (2023), http://papers.nips.cc/paper_files/paper/2023/hash/4d9944ab3330fe6af8efb9260aa9f307-Abstract-Conference.html
- [28] Maene, J., Raedt, L.D.: Soft-Unification in Deep Probabilistic Logic. In: Thirty-Seventh Conference on Neural Information Processing Systems (Nov 2023)
- [29] Magnini, M., Ciatto, G., Omicini, A.: On the Design of PSyKI: A Platform for Symbolic Knowledge Injection into Sub-symbolic Predictors. In: Calvaresi, D., Najjar, A., Winikoff, M., Främling, K. (eds.) Explainable and Transparent AI and Multi-Agent Systems, vol. 13283, pp. 90–108. Springer International Publishing, Cham (2022). <https://doi.org/10.1007/978-3-031-15565-9-6>
- [30] Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., De Raedt, L.: DeepProbLog: Neural probabilistic logic programming. In: Proceedings of NeurIPS (2018)
- [31] Marra, G., Dumančić, S., Manhaeve, R., De Raedt, L.: From Statistical Relational to Neural Symbolic Artificial Intelligence: A Survey. arXiv:2108.11451 [cs] (Aug 2021)
- [32] Marra, G., Giannini, F., Diligenti, M., Gori, M.: Lyrics: A general interface layer to integrate logic inference and deep learning. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 283–298. Springer (2019)

- [33] Marra, G., Kuželka, O.: Neural markov logic networks. In: Uncertainty in Artificial Intelligence. pp. 908–917. PMLR (2021)
- [34] Minervini, P., Riedel, S.: Adversarially regularising neural NLI models to integrate logical background knowledge. In: Korhonen, A., Titov, I. (eds.) Proceedings of the 22nd Conference on Computational Natural Language Learning. pp. 65–74. Association for Computational Linguistics, Brussels, Belgium (Oct 2018). <https://doi.org/10.18653/v1/K18-1007>, <https://aclanthology.org/K18-1007>
- [35] Pryor, C., Dickens, C., Augustine, E., Albalak, A., Wang, W.Y., Getoor, L.: NeuPSL: Neural Probabilistic Soft Logic. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence. pp. 4145–4153. International Joint Conferences on Artificial Intelligence Organization, Macau, SAR China (Aug 2023). <https://doi.org/10.24963/ijcai.2023/461>
- [36] Pryor, C., Dickens, C., Augustine, E., Albalak, A., Wang, W.Y., Getoor, L.: Neupsl: Neural probabilistic soft logic. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China. pp. 4145–4153. ijcai.org (2023). <https://doi.org/10.24963/IJCAI.2023/461>, <https://doi.org/10.24963/ijcai.2023/461>
- [37] Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62**(1–2), 107–136 (Jan 2006). <https://doi.org/10.1007/s10994-006-5833-1>, <http://dx.doi.org/10.1007/s10994-006-5833-1>
- [38] Sarker, M.K., Zhou, L., Eberhart, A., Hitzler, P.: Neuro-symbolic artificial intelligence. AI Communications **34**(3), 197–209 (2021). <https://doi.org/10.3233/AIC-210084>
- [39] Schulman, J., Heess, N., Weber, T., Abbeel, P.: Gradient estimation using stochastic computation graphs. In: Advances in Neural Information Processing Systems (2015)
- [40] Siddharth, N., Paige, B., van de Meent, J.W., Desmaison, A., Goodman, N.D., Kohli, P., Wood, F., Torr, P.: Learning disentangled representations with semi-supervised deep generative models. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30. pp. 5927–5937. Curran Associates, Inc. (2017)
- [41] Skryagin, A., Stammer, W., Ochs, D., Dhimi, D.S., Kersting, K.: SLASH: Embracing Probabilistic Circuits into Neural Answer Set Programming. arXiv:2110.03395 [cs] (Oct 2021)
- [42] Slusarz, N., Komendantskaya, E., Daggitt, M.L., Stewart, R., Stark, K.: Logic of differentiable logics: Towards a uniform semantics of dl. In: Proceedings of 24th International Conference on Logic. vol. 94, pp. 473–493 (2023)
- [43] Tang, Z., Hinnerichs, T., Peng, X., Zhang, X., Hoehndorf, R.: Falcon: faithful neural semantic entailment over alc ontologies. arXiv preprint arXiv:2208.07628 (2022)
- [44] Tang, Z., Pei, S., Peng, X., Zhuang, F., Zhang, X., Hoehndorf, R.: TAR: Neural Logical Reasoning across TBox and ABox (Aug 2022)
- [45] Umili, E., Capobianco, R., De Giacomo, G.: Grounding Itlf specifications in image sequences. In: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning. vol. 19, pp. 668–678 (2023)
- [46] van de Meent, J.W., Paige, B., Yang, H., Wood, F.: An Introduction to Probabilistic Programming (Oct 2021). <https://doi.org/10.48550/arXiv.1809.10756>
- [47] van Harmelen, F., ten Teije, A.: A Boxology of Design Patterns for Hybrid Learning and Reasoning Systems. Journal of Web Engineering **18**(1), 97–124 (2019). <https://doi.org/10.13052/jwe1540-9589.18133>
- [48] van Krieken, E., Acar, E., van Harmelen, F.: Analyzing differentiable fuzzy logic operators. Artificial Intelligence **302**, 103602 (2022). <https://doi.org/10.1016/j.artint.2021.103602>
- [49] van Krieken, E., Thanapalasingam, T., Tomczak, J., van Harmelen, F., Ten Teije, A.: A-NeSI: A scalable approximate method for probabilistic neurosymbolic inference. In: Oh, A., Neumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems. vol. 36, pp. 24586–24609. Curran Associates, Inc. (2023)
- [50] van Krieken, E., Tomczak, J., Ten Teije, A.: Stochastic: A framework for general stochastic automatic differentiation. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems. vol. 34, pp. 7574–7587. Curran Associates, Inc. (2021)
- [51] Varnai, P., Dimarogonas, D.V.: On robustness metrics for learning stl tasks. In: 2020 American Control Conference (ACC). pp. 5394–5399. IEEE (2020)
- [52] 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., Schuecker, J.: 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 (Jan 2023). <https://doi.org/10.1109/TKDE.2021.3079836>
- [53] Winters, T., Marra, G., Manhaeve, R., De Raedt, L.: Deepstochlog: Neural stochastic logic programming. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. vol. 36, pp. 10090–10100 (2022)
- [54] Wu, X., Zhu, X., Zhao, Y., Dai, X.: Differentiable Fuzzy \mathcal{ALC} : A Neural-Symbolic Representation Language for Symbol Grounding (Dec 2022)
- [55] Xu, J., Zhang, Z., Friedman, T., Liang, Y., den Broeck, G.V.: A semantic loss function for deep learning with symbolic knowledge. In: Dy, J.G., Krause, A. (eds.) *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research*, vol. 80, pp. 5498–5507. PMLR (2018), <http://proceedings.mlr.press/v80/xu18h.html>
- [56] Yang, Z., Ishay, A., Lee, J.: NeurASP: Embracing neural networks into answer set programming. In: Bessiere, C. (ed.) *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. pp. 1755–1762. International Joint Conferences on Artificial Intelligence Organization (Jul 2020). <https://doi.org/10.24963/ijcai.2020/243>

A Practical Examples

Example A.1 (MNIST Addition). Suppose we want to express the standard (single-digit) MNIST addition program using ULLER. In this setting, we have a domain T that represents a training dataset $I(T)$. In Section 4, we discuss how this training dataset can also be a minibatch of examples.

Each data point x consists of a pair of images (which we access with the properties `im1` and `im2`) associated to a label representing the value of their sum (which we can intuitively access via the property `sum`). Finally, we have a function f that we interpret as a neural network classifying MNIST images. Then, if we want to write that for every input the outputs of the neural network should be equal to the sum of the inputs, we can write:

$$\begin{aligned} \forall x \in T \\ (n1 := f(x.im1), n2 := f(x.im2) \\ (n1 + n2 = x.sum)) \end{aligned}$$

Example A.2 (Smokes Friends Cancer). In this classical example of Statistical Relational Learning introduced by [37], uncertain facts in a population group are modeled using the neural predicates $\text{Friends}(x, y)$ for friendship, $\text{Smokes}(x)$ for smoking, and $\text{Cancer}(x)$ for cancer. As ULLER relies on functions rather than predicates to model uncertainty, we must use $\text{true}(a)$ to formalise the problem in our language as explained in Section 3.1. For simplicity, we use $(A \Leftrightarrow B) \equiv ((A \Rightarrow B) \wedge (B \Rightarrow A))$ to denote logical equivalences.

Here is an example of a knowledge base for this problem. Friends of friends are friends:

$$\begin{aligned} \forall x \in \text{People}, y \in \text{People}, z \in \text{People} \\ (a_1 := \text{Friends}(x, y), a_2 := \text{Friends}(y, z), a_3 := \text{Friends}(x, z) \\ ((\text{true}(a_1) \wedge \text{true}(a_2)) \Rightarrow \text{true}(a_3))) \end{aligned}$$

If two people are friends, either both smoke or neither does:

$$\begin{aligned} \forall x \in \text{People}, y \in \text{People} \\ (a_1 := \text{Friends}(x, y), a_2 := \text{Smokes}(x), a_3 := \text{Smokes}(y) \\ (\text{true}(a_1) \Rightarrow (\text{true}(a_2) \Leftrightarrow \text{true}(a_3)))) \end{aligned}$$

Friendless people smoke:

$$\begin{aligned} \forall x \in \text{People} \\ (\neg \exists y \in \text{People} (a_1 := \text{Friends}(x, y)(\text{true}(a_1))) \\ \Rightarrow a_2 := \text{Smokes}(x)(\text{true}(a_2))) \end{aligned}$$

Smoking causes cancer:

$$\begin{aligned} \forall x \in \text{People} \\ (a_1 := \text{Smokes}(x), a_2 := \text{Cancer}(x) \\ (\text{true}(a_1) \Rightarrow \text{true}(a_2))) \end{aligned}$$

Notice that, according to the definitions of Section 3.1, the probabilistic interpretation of the above formula will assume conditional independence between $a_1 \sim p_{\text{Smokes}}(\cdot|x)$ and $a_2 \sim p_{\text{Cancer}}(\cdot|x)$. To model a dependence of cancer on smoking, i.e. $a_2 \sim p_{\text{Cancer}}(\cdot|x, a_1)$, we can make the probability explicitly depend on the previous variable:

$$\begin{aligned} \forall x \in \text{People} \\ (a_1 := \text{Smokes}(x), a_2 := \text{Cancer}(x, a_1) \\ (\text{true}(a_1) \Rightarrow \text{true}(a_2))) \end{aligned}$$

Next, we have labelled examples for each relationship. For example, for $\text{Friends}()$, drawing examples from a dataset T_{Friends} :

$$\begin{aligned} \forall t \in T_{\text{Friends}} \\ (l := \text{Friends}(t.x, t.y) \\ (l = t.\text{label})) \end{aligned}$$

B Gradient estimation

The sampling semantics in Equation 26 is a simple way to estimate the truth value of a formula. However, since sampling is not a differentiable operation, it is not possible to use this semantics to train the neural networks. Instead, we can use the score function gradient estimation method [39] to estimate the gradient of the truth value of a formula with respect to the parameters of the neural networks. However, this requires adapting the evaluation of the formula to incorporate score function terms. We give a brief description of how one might go about doing that in Appendix B.

One way to implement gradient estimation methods for simple ULLER programs is to use the DiCE estimator [19] which introduces the *MagicBox* operator $\Box(x) = \exp(x - \perp(x))$, where \perp is the *StopGradient* operator used in deep learning frameworks. This operator allows us to add a term that only appears when we differentiate it, and equals 1 during the forward pass. To incorporate DiCE for Unified Language for Learning and Reasoning, we have to modify Equation 15

$$\llbracket x := f(T_1, \dots, T_n)(F) \rrbracket^S = \llbracket F \rrbracket_{I, \mathcal{A}(\eta, S)}^C \cdot \sum_{i=1}^n \Box(\log p_{f_i}(\mathcal{A}(\eta, S)[x_i])) \quad (29)$$

Extensions of the DiCE estimator can be used to implement a wide variety of gradient estimation methods [50].

C Classical in the limit

C.1 Probabilistic semantics

The probabilistic semantics is *classical in the limit*. To show this, we note that we require that the domain becomes $\{0, 1\}$ instead of probabilities $[0, 1]$. Under this domain, the product is equal to the min function. We can use this to rewrite all but the interpretation of statements into the classical semantics.

Next, take for a statement $x := f(T_1, \dots, T_n)(F)$ the induction assumption that $\llbracket F \rrbracket_{\hat{I}, \eta}^P = \llbracket F \rrbracket_{\hat{I}, \eta}^C$, where \hat{I} is defined as in Definition 3.4. Then the interpretation of a statement is:

$$\mathbb{E}_{a \sim p_f(\cdot|T_1, \dots, T_n)}[\llbracket F \rrbracket_{\hat{I}, \eta[x \mapsto a]}^P] = \llbracket F \rrbracket_{\hat{I}, \eta[x \mapsto \arg \max_{a \in \Omega_{n+1}} p_f(a|T_1, \dots, T_n)]}^P$$

Here, we reduce the expectation by noting that since $p_{\hat{f}}(a|T_1, \dots, T_n) = \delta(a - \arg \max_{a'} p_f(a'|T_1, \dots, T_n))$, exactly one element gets 1 probability (or a single element with non-zero probability, in the case of continuous distributions). This single element is chosen on the right side. Then, we use the induction assumption to find that this is equal to the classical semantics of statements given in Equation 15.

C.2 Fuzzy semantics

Using the axioms of t-norms, we find that the fuzzy semantics is also classical in the limit. This again can be proven by induction. For Equations 21 and 23, we use the boundary conditions of t-norms, which states that $x \otimes 1 = x$ for $x \in [0, 1]$. Therefore, if $x = 0$, $0 \otimes 1 = 0$ and if $x = 1$, $1 \otimes 1 = 1$, meaning t-norms act as the min operator under the domain $\{0, 1\}$.

Next, consider the program fragment $x := f(T_1, \dots, T_n) (F)$ and take the induction assumption $\llbracket F \rrbracket_{I, \eta}^F = \llbracket F \rrbracket_{I, \eta}^C$. First, assume the domain $\Omega_{n+1} = \{0, 1\}$ and assume $\arg \max_{a \in \{0, 1\}} p_f(a|T_1, \dots, T_n) = 1$. Then, the interpretation of the statement is $\llbracket F \rrbracket_{I, \eta[x \mapsto p_f(a=1|T_1, \dots, T_n)]}^F = \llbracket F \rrbracket_{I, \eta[x \mapsto 1]}^F$, since the Dirac distribution will put all its mass on the output 1. Similarly, if $\arg \max_{a \in \{0, 1\}} p_f(a|T_1, \dots, T_n) = 0$, then the interpretation is $\llbracket F \rrbracket_{I, \eta[x \mapsto 0]}^F$. Then we can simply use the induction assumption.

Finally, if $\Omega_{n+1} \neq \{0, 1\}$, then we know there is a unique output $a \in \Omega_{n+1}$ such that $p_f(a|T_1, \dots, T_n) = 1$, while for the other outputs $p_f(a|T_1, \dots, T_n) = 0$. Then, using associativity and commutativity of the t-conorm \oplus , the interpretation of the statement is

$$\begin{aligned} & \llbracket F \rrbracket_{I, \eta[x \mapsto a]}^F \otimes p_f(a|T_1, \dots, T_n) \oplus \bigoplus_{a' \in \Omega_{n+1} \setminus \{a\}} \llbracket F \rrbracket_{I, \eta[x \mapsto a']}^F \otimes p_f(a'|T_1, \dots, T_n) \\ & \llbracket F \rrbracket_{I, \eta[x \mapsto a]}^F \otimes 1 \oplus \bigoplus_{a' \in \Omega_{n+1} \setminus \{a\}} \llbracket F \rrbracket_{I, \eta[x \mapsto a']}^F \otimes 0 \\ & \llbracket F \rrbracket_{I, \eta[x \mapsto a]}^F \oplus \bigoplus_{a' \in \Omega_{n+1} \setminus \{a\}} 0 = \llbracket F \rrbracket_{I, \eta[x \mapsto a]}^F \end{aligned}$$

where we again use the boundary conditions of the t-norm \otimes ($1 \otimes x = x$) and t-conorm \oplus ($0 \oplus x = x$).

D Relation of Probabilistic semantics to the Semantic Loss

Here, we show why the probabilistic semantics is equivalent to the weighted model counting semantics used in, for instance, the Semantic Loss. Let F be a closed formula without any statements $x := f(T_1, \dots, T_n)(F')$ that only involves variables x_1, \dots, x_n over finite domains. The *weighted model count (WMC)* is the evaluation of the classical semantics weighted by probabilities of the assignments to variables. These probabilities are often assumed to be independent, although our framework also allows for the probabilities to depend on previous variables. This is illustrated in Example A.2. The definition of the WMC is

$$\begin{aligned} \text{WMC} &= \sum_{a_1 \in \Omega_1} \dots \sum_{a_n \in \Omega_n} \prod_{i=1}^n p_{f_i}(a_i) \llbracket F \rrbracket_{I, \{x_1 \mapsto a_1, \dots, x_n \mapsto a_n\}}^C \\ &= \sum_{a_1 \in \Omega_1} p_{f_1}(a_1) \dots \sum_{a_n \in \Omega_n} p_{f_n}(a_n) \llbracket F \rrbracket_{I, \{x_1 \mapsto a_1, \dots, x_n \mapsto a_n\}}^C. \end{aligned} \tag{30}$$

Next, we rewrite this into a program $x_1 := f_1(), \dots, x_n := f_n() (F)$ such that the probabilistic semantics in Definition 3.5 is equal to the weighted model count. For ease of notation, let us denote S_i each statement $x_i := f_i()$ for $i = 1, \dots, n$. Then, we find the probabilistic semantics of the program by sequentially expanding the interpretation of the statements:

$$\begin{aligned} \llbracket S_1, \dots, S_n(F) \rrbracket_{I, \{\}}^P &= \sum_{a_1 \in \Omega_1} p_{f_1}(a_1) \cdot \llbracket S_2, \dots, S_n(F) \rrbracket_{I, \{x_1 \mapsto a_1\}}^P \\ &\dots \\ &= \sum_{a_1 \in \Omega_1} p_{f_1}(a_1) \dots \sum_{a_n \in \Omega_n} p_{f_n}(a_n) \llbracket F \rrbracket_{I, \{x_1 \mapsto a_1, \dots, x_n \mapsto a_n\}}^C \\ &= \text{WMC} \end{aligned} \tag{31}$$

where in the last step we use that since the domains are finite and F does not contain statements, the probabilistic semantics of F is equal to the classic one.

E Relation of Fuzzy Semantics to Differentiable Fuzzy Logics

Fuzzy logics are actively used in NeSy [4, 48, 10, 21]. We show how existing NeSy systems using fuzzy logics arise from the fuzzy semantics of ULLER. Existing fuzzy logics systems align with our interpretations of terms and logical operators, but differ in their use of fuzzy predicates, which are interpreted as functions to $[0, 1]$, that is, $I_{\text{NeSy}}(P) : \Omega_1 \times \dots \times \Omega_n \rightarrow [0, 1]$. Then, the truth value of a formula is computed by evaluating the formula with the fuzzy semantics.

We can emulate this in our fuzzy semantics with the $\text{true}()$ predicate and proof by induction. For each neural predicate $I_{\text{NeSy}}(P_i) : \Omega_1^i \times \dots \times \Omega_{n_i}^i \rightarrow [0, 1]$, we define a ULLER function $I(f_i) : \Omega_1^i \times \dots \times \Omega_{n_i}^i \rightarrow (\{0, 1\} \rightarrow [0, 1])$ such that:

$$I_{\text{NeSy}}(P_i)(T_1^i, \dots, T_{n_i}^i) = I(f_i)(T_1^i, \dots, T_{n_i}^i)(1) \quad (32)$$

Let F be a first-order logic formula with no statements nor functions, and $\llbracket F \rrbracket^{\text{NeSy}}$ be its interpretation in a fuzzy NeSy system. Let F contain k neural atoms $P_i(T_1^i, \dots, T_{n_i}^i)$, $i = 1 \dots k$. Let S_1, \dots, S_k (F') be a ULLER program with k statements where S_i defines $x_i := f_i(T_1^i, \dots, T_{n_i}^i)$, $i = 1, \dots, k$, and F' is F where we replace every mention of $P_i(T_1^i, \dots, T_{n_i}^i)$ by $\text{true}(x_i)$. We have:

$$\llbracket S_1, \dots, S_k (F') \rrbracket^F = \llbracket F' \rrbracket_{I, \eta[x_1 \mapsto p_{f_1}(1|T_1^1, \dots, T_{n_1}^1), \dots, x_k \mapsto p_{f_k}(1|T_1^k, \dots, T_{n_k}^k)]}^F \quad (33)$$

$$= \llbracket F' \rrbracket_{I, \eta[x_1 \mapsto I_{\text{NeSy}}(P_1)(T_1^1, \dots, T_{n_1}^1), \dots, x_k \mapsto I_{\text{NeSy}}(P_k)(T_1^k, \dots, T_{n_k}^k)]}^F \quad (34)$$

$$= \llbracket F \rrbracket^{\text{NeSy}} \quad (35)$$

Equality (34) stems from definition (32). We derive equality (35) by induction. First, note that according to the definition of $I(\text{true})$ and the assignment in (34), we have:

$$\llbracket \text{true}(x_i) \rrbracket^F = I_{\text{NeSy}}(P_i)(T_1^i, \dots, T_{n_i}^i) = \llbracket P(T_1^i, \dots, T_{n_i}^i) \rrbracket^{\text{NeSy}} \quad \text{for } i = 1, \dots, k \quad (36)$$

If our semantics use the same t-norm operator \otimes as the NeSy system, then:

$$\llbracket F_1 \wedge F_2 \rrbracket^F = \llbracket F_1 \rrbracket^F \otimes \llbracket F_2 \rrbracket^F = \llbracket F_1' \rrbracket^{\text{NeSy}} \otimes \llbracket F_2' \rrbracket^{\text{NeSy}} = \llbracket F_1' \wedge F_2' \rrbracket^{\text{NeSy}} \quad (37)$$

where in the second equality we use the induction hypothesis $\llbracket F_1 \rrbracket^F = \llbracket F_1' \rrbracket^{\text{NeSy}}$ and $\llbracket F_2 \rrbracket^F = \llbracket F_2' \rrbracket^{\text{NeSy}}$. The same can naturally be derived for other logical connectives. It follows that we can emulate any formula F built with the neural predicates $P(T_1^i, \dots, T_{n_i}^i)$, by building formula F' with the equivalently interpreted $\text{true}(x_i)$ (see Equation (34)) and the same logical constructs, such that $\llbracket F \rrbracket^{\text{NeSy}} = \llbracket S_1, \dots, S_k (F') \rrbracket^F$.