# Recursive Querying of Neural Networks via Weighted Structures

MARTIN GROHE, RWTH Aachen University, Germany
CHRISTOPH STANDKE, RWTH Aachen University, Germany
JUNO STEEGMANS, University of Antwerp, Belgium and Hasselt University, Belgium
JAN VAN DEN BUSSCHE, Hasselt University, Belgium

Expressive querying of machine learning models—viewed as a form of intensional data—enables their verification and interpretation using declarative languages, thus making learned representations of data more accessible. Motivated by the querying of feedforward neural networks, we investigate logics for weighted structures. In the absence of a bound on neural network depth, such logics must incorporate recursion; thereto we revisit the functional fixpoint mechanism proposed by Grädel and Gurevich. We adopt it in a Datalog-like syntax; we extend normal forms for fixpoint logics to weighted structures; and show an equivalent "loose" fixpoint mechanism that allows values of inductively defined weight functions to be overwritten. We propose a "scalar" restriction of functional fixpoint logic, of polynomial-time data complexity, and show it can express all PTIME model-agnostic queries over reduced networks with polynomially bounded weights. In contrast, we show that very simple model-agnostic queries are already NP-complete. Finally, we consider transformations of weighted structures by iterated transductions.

CCS Concepts: • **Theory of computation → Database query languages (principles)**.

Additional Key Words and Phrases: First-order logic over the reals, Expressive power, Complexity, Model interpretability

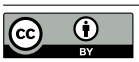
**ACM Reference Format:**
Martin Grohe, Christoph Standke, Juno Steegmans, and Jan Van den Bussche. 2026. Recursive Querying of Neural Networks via Weighted Structures. *Proc. ACM Manag. Data* 4, 2 (PODS), Article 115 (May 2026), 20 pages. https://doi.org/10.1145/3801911

## 1 Introduction

A case can be made, from several perspectives, for the *querying* of machine learning models:

- Data science projects generate a large amount of model artefacts, which should be managed using database technology, just like any other kind of data. In particular, we should be able to query this data. Platforms like MLflow or W&B offer administrative filtering and search based on experimental metadata, but no deep querying of the models themselves.
- In machine learning terminology, "querying" a model often just means to apply it to a new input. However, we can be much more ambitious. Consider a typical Boolean classifier on tuples (vectors) of numeric features. Such a model represents the potentially infinite relation consisting of all possible tuples that are classified as true. We would like to be able to query such relations just like ordinary relations in a relational database.

Authors' Contact Information: Martin Grohe, grohe@informatik.rwth-aachen.de, RWTH Aachen University, Aachen, Germany; Christoph Standke, standke@informatik.rwth-aachen.de, RWTH Aachen University, Aachen, Germany; Juno Steegmans, juno.steegmans@uantwerpen.be, University of Antwerp, Antwerp, Belgium and Hasselt University, Hasselt, Belgium; Jan Van den Bussche, jan.vandenbussche@uhasselt.be, Hasselt University, Hasselt, Belgium.




ACM 2836-6573/2026/5-ART115
https://doi.org/10.1145/3801911

- Querying infinite relations that are finitely presented by constraints was already intensively investigated in database theory under the heading of *constraint query languages* [32, 34, 37].
- The multitude of methods for model interpretability or explainable AI [40, 46] can be viewed as many different queries on models and data (e.g., finding a counterfactual, computing the Shapley value), but outside the framework of an encompassing structured query language.
- In the verification of neural networks [6, 10], models represent functions over the reals, and properties to be verified are expressed as universally quantified constraint expressions about such functions. These expressions can already be thought of as a minimal query language.

The above considerations can motivate us to investigate the theoretical foundations of query languages for neural networks. Indeed, research in this direction has already been started. Arenas et al. [7, 8] consider boolean decision trees and first-order logics over arbitrary-length boolean vectors. They combine these logics with logics that quantify over the nodes of the decision tree, enabling query evaluation via SAT solving. Grohe et al. [27] consider feedforward neural networks (FNNs) and the first-order logic FO(SUM) for weighted structures, with query evaluation via SQL [21].

Focusing on model-agnostic queries,[1] Grohe et al. show a remarkable dichotomy. Without a bound on the depth of the network, FO(SUM), due to its lack of recursion, can only express trivial model-agnostic queries. On fixed-depth FNNs, however, FO(SUM) has substantial expressive power and can express all queries in the linear-arithmetic fragment of the constraint query language $\mathsf{FO}(\mathbf{R}, f)$. Here, $\mathsf{FO}(\mathbf{R}, f)$ stands for first-order logic over the reals with an extra function symbol $f$. This language serves as a natural yardstick for expressing model-agnostic queries: the model is accessed as a black box via the symbol $f$.

In this paper, we investigate methods for, as well as obstacles to, lifting the fixed-depth assumption made in Grohe et al.'s work. We begin by developing the necessary theory for extending FO(SUM) with recursion. At its core, FO(SUM) is a logic for defining weight functions. Fixpoint logics for inductive definitions of *relations* are well known from logic, database theory and finite model theory. In contrast, for weight functions, we are aware of only one proposal, the functional fixpoint [23], which we revisit and develop more systematically. We examine alternative semantics, and show that they are equally expressive. We define the recursive extension of FO(SUM), called IFP(SUM), both in a Datalog-like and in a fixpoint-logic-like syntax, and establish a normal form that extends the known normal form for fixpoint logic and inflationary datalog with negation [3, 14, 36].

Recursion in query languages, including Datalog-like languages, has been a central topic in database theory from the beginning [1]. Our treatment of IFP(SUM) contributes to this and lays the foundations for querying neural networks without a fixed-depth assumption. Indeed, depth (number of layers) is an important hyperparameter in neural network architecture [22]. Query formulations that work across depths are more useful for assessing different models under development, or for analyzing ensembles of models with different architectures [39, 56].

Towards applying IFP(SUM) to FNNs, we define a restricted fragment, called *scalar* IFP(SUM), disallowing multiplication in combination with recursion. We prove that this fragment has PTIME data complexity. Due to the combination of recursion with arithmetic on weights, this is a nontrivial exercise. We then establish that scalar IFP(SUM) can express *all* PTIME model-agnostic queries, on FNNs that have polynomially bounded "reduced" weights. We make the latter condition precise and explain why it is necessary.

Next, we relate back to the black-box logic $\mathsf{FO}(\mathbf{R}, f)$. The relevant question is whether IFP(SUM) can now express all of linear $\mathsf{FO}(\mathbf{R}, f)$, without the fixed-depth assumption. An affirmative answer

[1] A query is model-agnostic if it does not distinguish between two models that may be structurally different but that happen to represent the same function or classifier [45].

is unlikely, however; we will show that very simple FO($\mathbf{R}, f$) queries are already NP-hard. This provides a negative answer at least for scalar IFP(SUM), unless P = NP. The question for unrestricted IFP(SUM) remains open.

Finally, we consider a highly expressive extension of IFP(SUM) that allows transductions, mapping structures to structures, to be defined iteratively. Reviewing the proof of the expressiveness result of Grohe et al., we conclude that every linear FO($\mathbf{R}, f$) query can indeed be computed by an iteration of IFP(SUM) transductions.

In summary, our main contributions are the following:

(1) Develop semantics and normal forms for IFP(SUM): recursive extensions of FO(SUM).
(2) Design a tractable fragment of IFP(SUM): the scalar fragment.
(3) Point out the limitations of IFP(SUM) for arbitrary (polynomial-time) weight computations.
(4) Identify the class of neural networks with polynomial reduced weights, on which scalar IFP(SUM) is complete for the PTIME model-agnostic queries.
(5) Warn that very simple model-agnostic queries on unbounded-depth FNNs are already NP-hard.
(6) Lay theoretical foundations and open up new research directions towards querying machine-learning models.

This paper is organized as follows. Section 2 discusses related work. Section 3 presents preliminaries. Section 4 contains the results on IFP(SUM), and Section 5 those on the scalar fragment. Section 6 presents the NP-completeness result. Section 7 discusses iterative transductions. Section 8 concludes. The omitted proofs can be found in the extended version of this paper [26].

## 2 Related work

The logic FO(SUM) for weighted structures is an instantiation of the logics for metafinite structures defined by Grädel and Gurevich [23]. Apparently unaware of that work, Torunczyk made a very similar proposal and studied the combined complexity of query evaluation and enumeration over general classes of numerical domains (semirings) [49]. In a statistical-learning context, Van Bergerem and Schweikardt [50, 51] study a closely related extension of first-order logic by "weight aggregation". Of course, FO(SUM) is also quite similar to the relational calculus with aggregates and arithmetic which formalizes basic SQL [35]. The difference is that in SQL, relations can have multiple numerical columns, while metafinite structures have a separate abstract domain on which relations and weight functions (taking values in a separate numerical domain) are defined.

Weight functions are also very similar to semiring-annotated relations [5, 24]. However, in that space, the focus is typically on languages where the weights are *implicitly* added, multiplied, or summed via corresponding relational operators. In contrast, FO(SUM) deals *explicitly* with the numerics.[2] Explicit tensor logics were also proposed by Geerts and Reutter [17, 19] for the purpose of characterizing indistinguishability of weight functions by graph neural networks. In contrast, our focus here is on model-agnostic querying of feedforward neural networks.

To this aim, through the logic IFP(SUM), we add recursion to FO(SUM). When mixing recursion and numerical computation, termination becomes a point of attention. We use and investigate variations of the inflationary fixpoint that are guaranteed to terminate. In contrast, in very interesting related work, Abo Khamis et al. consider recursive datalog over semiring-annotated relations (datalog-o) and investigate conditions on the semiring that guarantee convergence [4]. In the stages of a datalog-o program execution, weight functions (annotated relations) can be updated an unrestricted number of times, which may or may not converge. In the functional fixpoint semantics of IFP(SUM), a weight function (tuple annotation) can be changed from undefined to defined only

[2]See also work on implicit versus explicit handling of nonnumeric annotations [11, 20].

once on every tuple of elements, which guarantees termination. Our "loose" fixpoint semantics (Section 4.2) sits in between the two approaches.

We also mention work on the semantics for logic programs with aggregates [42]. There, the focus is on providing natural semantics for general sets of recursive rules involving negation and aggregation, and finding characterizations for when such rule sets are unambiguous (have unique wellfounded models).

Work related to our NP-hardness result in Section 6 has considered the complexity of deciding various properties of FNNs, including injectivity and surjectivity of the represented function [16]. Also various other forms of neural network verification have been shown to be NP-complete [55]. However, existing results crucially depend on the assumption that the *width* (input dimension) of the network to be verified contributes to the input size of the verification problem. In contrast, our NP-hardness result is stronger in that it already pertains to networks of width one.

## 3 Preliminaries

### 3.1 FNNs

The structure of a feedforward neural network [48], abbreviated FNN, is that of a directed acyclic graph with weights on nodes and edges. The source nodes are called *inputs* and are numbered from 1 to $m$; the sink nodes are called *outputs* and are numbered from 1 to $p$. All other nodes are called *hidden* nodes. We always assume input and output nodes to be distinct, that is, we disallow a graph of only isolated nodes. The weight of a node is also called its *bias*; an exception are the input nodes which do not have a bias.

Let $\mathcal{N}$ be an FNN as described. The function $f^{\mathcal{N}} : \mathbb{R}^m \to \mathbb{R}^p$ represented by $\mathcal{N}$, using ReLU activations and linear outputs, is defined as follows. We begin by defining, for every node $u$, a function $f_u^{\mathcal{N}} : \mathbb{R}^m \to \mathbb{R}$ by induction on the *depth* of $u$ (the maximum length of a path from an input node to $u$.) If $u$ is the $i$th input node then $f_u^{\mathcal{N}}(x_1, \ldots, x_m) = x_i$. If $u$ is a hidden node with bias $b$, and incoming edges $(v_1, u)$, …, $(v_k, u)$ with weights $w_1$, …, $w_k$, respectively, then $f_u^{\mathcal{N}}(\boldsymbol{x}) = \mathrm{ReLU}(b + \sum_j w_j f_{v_j}^{\mathcal{N}}(\boldsymbol{x}))$. Here, $\mathrm{ReLU} : \mathbb{R} \to \mathbb{R} : z \mapsto \max(0, z)$. If $u$ is an output node, $f_u^{\mathcal{N}}(\boldsymbol{x})$ is defined similarly as for hidden nodes, but the application of ReLU is omitted. We can finally define $f^{\mathcal{N}}(\boldsymbol{x})$ as $(f_{\mathrm{out}_1}^{\mathcal{N}}(\boldsymbol{x}), \ldots, f_{\mathrm{out}_p}^{\mathcal{N}}(\boldsymbol{x}))$, where $\mathrm{out}_1, \ldots, \mathrm{out}_p$ are the output nodes.

$\mathbf{K}(m, p)$ denotes the class of FNNs with $m$ inputs and $p$ outputs. We also write $\mathbf{K}(*, p)$, $\mathbf{K}(m, *)$, and $\mathbf{K}(*, *)$ when the numbers of inputs, or outputs, or both, are not fixed.

*Discussion.* A complementary view of FNNs is as a sequence of linear operators (matrices) interleaved with nonlinear activation functions. While that view is certainly natural and conforms to ML programming practice, the graph-based view taken here is equally natural and can be motivated from several angles. First, it is a well-established method to describe computational models in terms of their computation graphs; a classical example are Boolean circuits (which are more general than the specific circuits found in hardware). It allows us to describe the flow of information and analyse the role of individual network units. Second, formally, there is no great difference between a matrix and a weighted graph; both are number-valued functions on a finite domain. Third, our graph view allows logics based on a general notion of weighted structures, which are easily extended to other neural architectures, not just FNNs, although in this paper we do focus on the latter and develop the theory to some depth.

As a further remark, in taking the graph-based view, we could have equally defined an FFN more restrictively as a more rigid layered structure, with edges running only between nodes of successive layers. Under this restriction, our positive results that pertain to neural networks (Theorem 5.9)

will obviously still apply, but also, the proofs of our negative results (Theorem 5.5, Theorem 6.2) remain valid.

### 3.2 Model-agnostic queries and $\mathsf{FO}(\mathbf{R}, f)$

In general, we may define an *r-ary query on* $\mathbf{K}(m,p)$ *with* $k$ *parameters* to be a relation $Q \subseteq \mathbf{K}(m,p) \times \mathbb{R}^k \times \mathbb{R}^r$. If $Q(\mathcal{N}, \boldsymbol{z}, \boldsymbol{y})$ holds, we say $\boldsymbol{y}$ is a possible result of $Q$ on $\mathcal{N}$ and $\boldsymbol{z}$. In the special case $r = 0$ we obtain a *boolean query* (true if $Q(\mathcal{N}, \boldsymbol{z})$ holds, false otherwise).

*Example 3.1.* The simplest example of a query is inference, i.e., evaluating a model on a given input. Formally, for inference, we have $k = m$ and $r = p$ and $Q(\mathcal{N}, \boldsymbol{z}, \boldsymbol{y})$ holds iff $\boldsymbol{y} = f^{\mathcal{N}}(\boldsymbol{z})$. However, many more tasks in interpretable machine learning [40] fit the above notion of query. For example, returning a counterfactual explanation or an adversarial example, checking robustness, computing the Shapley value in a point, computing the gradient in a point, or checking differentiability between certain ranges given by the parameters.

The case $k = 0$ captures analyzing or verifying the global behavior of neural networks, instead of their behavior on given input vectors. For example, recall that the function represented by an FNN with ReLU and linear outputs is always piecewise linear. A possible 2-ary query for $m = 1 = p$ on an FNN $\mathcal{N}$ may ask for results $(b, s)$ such that $b$ is a breakpoint of $f^{\mathcal{N}}$ and $s$ is the right-hand slope at $b$. For higher input dimensions we could, for example, ask for the coefficients of supporting hyperplanes of the partitioning of input space induced by $f^{\mathcal{N}}$. □

All examples of queries just mentioned are *model-agnostic*, meaning that the query has the same results on two networks $\mathcal{N}$ and $\mathcal{N}'$ representing the same function, i.e., $f^{\mathcal{N}} = f^{\mathcal{N}'}$. Model-agnostic methods form an established category in interpretable machine learning and explainable AI [40].[3]

As a yardstick for model-agnostic queries on $\mathbf{K}(m,p)$, we can use first-order logic over the reals with $p$ function symbols $f_1, \ldots, f_p$ of arity $m$, together representing a function $f : \mathbb{R}^m \to \mathbb{R}^p$. For simplicity, in this paper, we will look mainly at the case $p = 1$, which in itself is already typical in machine learning, with tasks such as regression and binary classification.

Formally, let $\mathbf{R} = (\mathbb{R}, +, \cdot, (q)_{q\in\mathbb{Q}}, <)$ be the structure of the reals with constants for all rational numbers. By $\mathsf{FO}(\mathbf{R}, f/m)$, we mean first-order logic over the vocabulary of $\mathbf{R}$ with an extra $m$-ary function symbol $f$. When multiplication is restricted to be only between a term and a constant (scalar multiplication), we denote this by $\mathbf{R}_{\text{lin}}$ and $\mathsf{FO}(\mathbf{R}_{\text{lin}}, f/m)$. When $m$ is understood, we omit it from the notation.

A formula $\varphi$ of $\mathsf{FO}(\mathbf{R}, f/m)$ with $k + r$ free variables $z_1, \ldots, z_k, y_1, \ldots, y_r$ now naturally expresses an $r$-ary query $Q_\varphi$ on $\mathbf{K}(m, 1)$ with $k$ parameters. Specifically, $Q_\varphi(\mathcal{N}, \boldsymbol{z}, \boldsymbol{y})$ holds iff $\varphi(\boldsymbol{z}, \boldsymbol{y})$ is satisfied in $(\mathbf{R}, f^{\mathcal{N}})$. This query is model-agnostic by definition.

*Example 3.2.* We give two formulas to illustrate the syntax of $\mathsf{FO}(\mathbf{R}, f)$. The inference query (Example 3.1) is expressed by the formula $y = f(\boldsymbol{z})$. The query on $\mathbf{K}(1,1)$ that asks whether $\lim_{x\to+\infty} f^{\mathcal{N}}(x) = -\infty$, is expressed by $\forall u < 0\, \exists x_0 > 0\, \forall x > x_0\, f(x) < u$. Both formulas are in $\mathsf{FO}(\mathbf{R}_{\text{lin}}, f)$ since they do not use multiplication. We note that all queries from Example 3.1, with the exception of Shapley value, are expressible in $\mathsf{FO}(\mathbf{R}, f)$ [27].

### 3.3 FO(SUM)

FO(SUM) is a logic for querying weighted structures, which are standard relational structures additionally equipped with weight functions. These functions map tuples of elements to numeric values.

[3]To give an example of a method that is not model-agnostic, we can mention the pruning of a neural network: finding neurons that have only a negligible influence on the function that is represented.

In our case, the numeric values are taken from the "lifted" rationals $\mathbb{Q}_\perp := \mathbb{Q} \cup \{\perp\}$ or reals $\mathbb{R}_\perp = \mathbb{R} \cup \{\perp\}$. Here, $\perp$ is an extra element representing an undefined value. We extend the usual order $\leq$ on $\mathbb{R}$ to $\mathbb{R}_\perp$ by letting $\perp \leq x$ for all $x \in \mathbb{R}_\perp$. We extend addition, subtraction, and multiplication by letting $x + y := \perp$, $x - y := \perp$, and $x \cdot y := \perp$ if $x = \perp$ or $y = \perp$. Similarly, we extend division by letting $x/y := \perp$ if $x = \perp$ or $y = \perp$ or $y = 0$.

A *weighted vocabulary* $\Upsilon$ is a finite set of relation symbols and weight-function symbols. Each symbol $S$ has an arity $\mathrm{ar}(S)$, a natural number. A *weighted $\Upsilon$-structure* $\mathcal{A}$ consists of a finite set $A$ called the universe of $\mathcal{A}$, for each $k$-ary relation symbol $R \in \Upsilon$ a $k$-ary relation $R^{\mathcal{A}} \subseteq A^k$, and for each $k$-ary weight-function symbol $F \in \Upsilon$ a function $F^{\mathcal{A}} : A^k \to \mathbb{R}_\perp$. We will often refer to weighted structures simply as structures for short.

We define the sets of *formulas* $\varphi$ and *weight terms* $\theta$ of the logic FO(SUM) by the following grammar:

$$\varphi ::= x = y \mid R(x_1, \ldots, x_{\mathrm{ar}(R)}) \mid \theta \leq \theta \mid \neg\varphi \mid \varphi * \varphi \mid Qx\,\varphi \tag{3.A}$$

$$\theta ::= r \mid F(x_1, \ldots, x_{\mathrm{ar}(F)}) \mid \theta \circ \theta \mid \text{if } \varphi \text{ then } \theta \text{ else } \theta \mid \sum_{(x_1,\ldots,x_k):\varphi} \theta. \tag{3.B}$$

Here $x, y, x_i$ are variables, $R$ is a relation symbol, $* \in \{\vee, \wedge, \to\}$ is a Boolean connective, $Q \in \{\exists, \forall\}$ is a quantifier, $r \in \mathbb{Q}_\perp$ is a numeric constant,[4] $F$ is a weight-function symbol, and $\circ \in \{+, -, \cdot, /\}$ is an arithmetic operator. The semantics is defined with respect to pairs $(\mathcal{A}, \nu)$, where $\mathcal{A}$ is a structure and $\nu$ an assignment of elements from the universe $A$ to the variables. Formulas $\varphi$ take a Boolean value $[\![\varphi]\!]^{(\mathcal{A},\nu)} \in \{0, 1\}$ and weight terms $\theta$ take a value $[\![\theta]\!]^{(\mathcal{A},\nu)} \in \mathbb{R}_\perp$. These values are defined inductively; we omit most definitions as they are obvious or follow the familiar semantics of first-order logic. The semantics of the summation operator is as follows:

$$[\![ \sum_{(x_1,\ldots,x_k):\varphi} \theta ]\!]^{(\mathcal{A},\nu)} := \sum_{(a_1,\ldots,a_k)\in A^k} [\![\varphi]\!]^{(\mathcal{A},\nu\frac{a_1,\ldots,a_k}{x_1,\ldots,x_k})} \cdot [\![\theta]\!]^{(\mathcal{A},\nu\frac{a_1,\ldots,a_k}{x_1,\ldots,x_k})},$$

where $\nu\frac{a_1,\ldots,a_k}{x_1,\ldots,x_k}$ denotes the updated assignment obtained from $\nu$ by assigning $a_i$ to $x_i$ for $i = 1, \ldots, k$.

An FO(SUM) *expression* is either a formula or a weight term. The set $\mathrm{free}(\xi)$ of *free variables* of an expression $\xi$ is defined in a straightforward way, where a summation $\sum_{(x_1,\ldots,x_k):\varphi}$ binds the variables $x_1, \ldots, x_k$. A *closed expression* is an expression without free variables. A *closed formula* is also called a *sentence*.

For an expression $\xi$, the notation $\xi(x_1, \ldots, x_k)$ stipulates that all free variables of $\xi$ are in $\{x_1, \ldots, x_k\}$. It is easy to see that the value $[\![\xi]\!]^{(\mathcal{A},\nu)}$ only depends on the values $a_i := \nu(x_i)$ of the free variables. Thus we may avoid explicit reference to the assignment $\nu$ and write $[\![\xi]\!]^{\mathcal{A}}(a_1, \ldots, a_k)$ instead of $[\![\xi]\!]^{(\mathcal{A},\nu)}$. If $\xi$ is a closed expression, we just write $[\![\xi]\!]^{\mathcal{A}}$. For formulas $\varphi(x_1, \ldots, x_k)$, we also write $\mathcal{A} \models \varphi(a_1, \ldots, a_k)$ instead of $[\![\varphi]\!]^{\mathcal{A}}(a_1, \ldots, a_k) = 1$, and for sentences $\varphi$ we write $\mathcal{A} \models \varphi$.

Observe that we can express averages in FO(SUM) using summation and division. It will be useful to introduce a notation for averages: for a term $\theta$ and formula $\varphi$ we write $\mathrm{avg}_{\boldsymbol{x}:\varphi}\,\theta$ to abbreviate $(\sum_{\boldsymbol{x}:\varphi} \theta)/\sum_{\boldsymbol{x}:\varphi} 1$. Note that $\mathrm{avg}_{\boldsymbol{x}:\varphi}\,\theta$ takes value $\perp$ if there are no tuples $\boldsymbol{x}$ satisfying $\varphi$.

### 3.4 FNNs as weighted structures

Any FNN $\mathcal{N} \in \mathbf{K}(*, *)$ can be naturally regarded as a weighted structure $\mathcal{N} = (V, E^{\mathcal{N}}, \mathrm{In}^{\mathcal{N}}, \mathrm{Out}^{\mathcal{N}}, b^{\mathcal{N}}, w^{\mathcal{N}})$, where $V$, the universe, is the set of nodes; $E^{\mathcal{N}}$ is the binary edge relation; $\mathrm{In}^{\mathcal{N}}$ is a binary

[4] One can also allow arbitrary real constants, but in this paper we are concerned with algorithms evaluating expressions, and therefore reasonably restrict our attention to rational constants.

relation that is a linear order of the input nodes of $\mathcal{N}$ (and undefined on the remaining nodes); $\mathrm{Out}^{\mathcal{N}}$ similarly is a linear order of the output nodes; $b^{\mathcal{N}}$ is the unary bias weight function on nodes; and $w^{\mathcal{N}}$ is the binary weight function on edges.[5]

We can expand the corresponding vocabulary $(E, \mathrm{In}, \mathrm{Out}, b, w)$ with a weight function *val* giving input values to the network. Over the resulting vocabulary, we can now give a few examples of formulas and weight terms in FO(SUM).

*Example 3.3.* For any natural number $\ell$, there is a first-order logic formula $depth_{\leq \ell}(u)$ defining the nodes of depth at most $\ell$ in any network. We can then define the function $f_u^{\mathcal{N}}$ (cf. Section 3.1) for such nodes by the weight term $eval_{\leq \ell}(u)$ defined as follows:

$eval_{\leq 0} :=$ if $\mathrm{In}(u,u)$ then $val(u)$ else $\bot$
$eval_{\leq \ell+1} :=$ if $depth_{\leq \ell}(u)$ then $eval_{\leq \ell}(u)$ else if $depth_{\leq \ell+1}(u)$ then
if $\mathrm{Out}(u,u)$ then $b(u) + \sum_{v:E(v,u)} w(v,u) \cdot eval_{\leq \ell}(v)$
else $\mathrm{ReLU}(b(u) + \sum_{v:E(v,u)} w(v,u) \cdot eval_{\leq \ell}(v))$
else $\bot$

Here, $\mathrm{ReLU}(\theta)$, for an arbitrary weight term $\theta$, is an abbreviation for if $\theta \geq 0$ then $\theta$ else 0. □

The above example essentially shows that FO(SUM) can express the inference query on fixed-depth networks (cf. Section 3.2). It turns out that FO(SUM) can do much more than that and actually measures up to the yardstick set by $\mathrm{FO}(\mathbf{R}_{\mathrm{lin}}, f)$:

Theorem 3.4 ([27]). *Let $m$, $k$ and $\ell$ be natural numbers, and let $Q$ be a boolean query on $\mathbf{K}(m,1)$ with $k$ parameters, expressible in $\mathrm{FO}(\mathbf{R}_{\mathrm{lin}}, f)$. There exists an* FO(SUM) *sentence $\psi$ over vocabulary $(E, \mathrm{In}, \mathrm{Out}, b, w, val)$ that expresses $Q$ on all networks in $\mathbf{K}(m,1)$ of depth $\ell$.*

This result shows that FO(SUM) can simulate arbitrary quantification over the real numbers, a feature central to $\mathrm{FO}(\mathbf{R}_{\mathrm{lin}}, f)$. This is remarkable since FO(SUM) itself only offers quantification over the finite set of nodes of the network.

## 4 Extending FO(SUM) with recursion

This paper investigates methods for, as well as obstacles to, lifting the fixed-depth restriction of Theorem 3.4. A necessary development, which we investigate in this section, is to extend FO(SUM) with a recursion mechanism, allowing for inductive definitions.

Recall that FO(SUM) expressions can be formulas, which define relations, or weight terms, which define weight functions. For inductive definitions of relations, logical mechanisms are well understood; in this paper, we consider a Datalog-like syntax with stratification and the inflationary fixpoint semantics for strata [1, 3, 33]. We recall this semantics through an example.

*Example 4.1.* Consider finite directed graphs given by a binary edge relation $E$ with an extra unary relation $S$. The following program checks that the graph is acyclic when restricted to the nodes reachable from nodes in $S$:

$Reach(x) \leftarrow S(x) \vee \exists y(Reach(y) \wedge E(y,x))$;
$Ans \leftarrow \neg \exists x\, Reach(x,x)$.

Relation *Reach* is initialized empty. The defining formula is evaluated repeatedly and the result *added* to *Reach* until no change occurs.[6] Nullary relation *Ans* provides the boolean answer to the query and is defined non-recursively. Its defining rule is in a subsequent stratum, meaning that it is evaluated after the recursion for *Reach* has terminated. □

[5]To be precise, $b^{\mathcal{N}}(u) = \bot$ for every input node $u$, and $w^{\mathcal{N}}(u,v) = \bot$ for every pair $(u,v)$ not in $E$.
[6]The adding gives the inflationary aspect; the defining formula need not be positive or monotone.

To our knowledge, inductive definitions of *weight functions* has been much less studied.[7] We can adopt the only existing proposal, which is nicely compatible with inflationary fixpoints, called *functional fixpoints* [23]. The idea is that an inductively defined weight function $F$ is initialized to be undefined ($\perp$) everywhere. A defining weight term is then evaluated repeatedly and used to update $F$, but only on tuples where $F$ was not yet defined. This is repeated until no change occurs.

*Example 4.2.* The following inductively defined function *eval* uniformly expresses, given any FNN $\mathcal{N}$, the function $f_u^{\mathcal{N}}$ for all nodes. (Compare the weight term $eval_{\leq \ell}$ from Example 3.3 which does the same but only for nodes $u$ up to depth $\ell$.)

$$\begin{aligned} eval(u) \leftarrow\ & \text{if } \mathrm{In}(u,u) \text{ then } val(u) \\ & \text{else if } \mathrm{Out}(u,u) \text{ then } b(u) + \textstyle\sum_{v:E(v,u)} w(v,u) \cdot eval(v) \\ & \text{else } \mathrm{ReLU}(b(u) + \textstyle\sum_{v:E(v,u)} w(v,u) \cdot eval(v)). \end{aligned}$$

*Example 4.3.* Many more interesting queries on FFNs can be expressed. We can perform sensitivity analysis by comparing the output with the output we would get by setting a designated input node to zero. Thereto we define a variant of *eval* with two arguments:

$$\begin{aligned} eval_2(u,u_0) \leftarrow\ & \text{if } \mathrm{In}(u,u) \text{ then if } u = u_0 \text{ then } 0 \text{ else } val(u) \\ & \text{else if } \mathrm{Out}(u,u) \text{ then } b(u) + \textstyle\sum_{v:E(v,u)} w(v,u) \cdot eval_2(v,u_0) \\ & \text{else } \mathrm{ReLU}(b(u) + \textstyle\sum_{v:E(v,u)} w(v,u) \cdot eval_2(v,u_0)); \end{aligned}$$

In a subsequent stratum, given a threshold $\epsilon$, we can now retrieve the input nodes that are "unimportant" as follows: (using sum of squares as a comparison metric)

$$UnImp(u_0) \leftarrow \textstyle\sum_{v:\mathrm{Out}(v,v)} (eval(v) - eval_2(v,u_0))^2 < \epsilon.$$

It is also possible to evaluate the network on a collection of input valuations (rather than a single one) by extending the network structure with input identifiers. Instead of a unary weight function $val(u)$ defined only on input nodes $u$, we then provide a binary weight function $val(i,u)$ where $i$ ranges over the input identifiers. Using a similar approach as the input sensitivity example above, we can now retrieve inconsequential network edges given a collection of inputs, which is a step in neural network pruning [28].

Next to an input valuation *val* as above, we can also provide a valuation on the output nodes as input. We can then compute the gradient of the $L_2$ loss function, over the given input–output pair, by back-propagation. Indeed, this is again a similar computation, but with the induction proceeding backwards from output nodes to input nodes [47, Section 21.4.1].

As a final example, we recursively compute the depth of each network node as an inductively defined weight function:

$$depth(u) \leftarrow \text{if } \mathrm{In}(u,u) \text{ then } 0 \text{ else } 1 + \max_{v:E(v,u)} depth(v)$$

Here, we use max-aggregation, while weight terms in FO(SUM) only have sum-aggregation. Yet, max can be considered as a built-in that does not change the expressive power, since it is expressible in terms of sum as

$$\frac{\sum_{v:M(u,v)} depth(v)}{\sum_{v:M(u,v)} 1},$$

where $M(u,v)$ abbreviates

$$E(v,u) \wedge \forall v'(E(v',u) \rightarrow depth(v') \leq depth(v)).$$

The global depth of the network can now be expressed as $\max_{v:\mathrm{Out}(v,v)} depth(v)$.

[7] In standard first-order logic, whose syntax lacks the powerful interaction between formulas and weight terms we have in FO(SUM), one is limited to defining functions inductively as relations (graph of a function) using formulas, and somehow declaring the definition to be wrong if the resulting relation is not the graph of a function.

### 4.1 The logic IFP(SUM)

After the above informal introduction, we now formally define IFP(SUM): the extension of FO(SUM) with inflationary and functional fixpoints. Our syntax follows the pattern of stratified Datalog [1]. Later in this section we will also consider a syntax that is more in line with the way fixed-point extensions of first-order logic are defined in finite model theory [14, 36].

*Rules and strata.* A *relational rule* over a weighted vocabulary $\Upsilon$ is of the form $R(\boldsymbol{x}) \leftarrow \varphi$, where $R \in \Upsilon$ is a relation name, $\boldsymbol{x}$ is a tuple of $\mathrm{ar}(R)$ distinct variables, and $\varphi(\boldsymbol{x})$ is an FO(SUM) formula over $\Upsilon$.

A *weight function rule* over $\Upsilon$ is of the form $F(\boldsymbol{x}) \leftarrow \theta$, where $F \in \Upsilon$ is a weight function name, $\boldsymbol{x}$ is a tuple of $\mathrm{ar}(R)$ distinct variables, and $\theta(\boldsymbol{x})$ is a weight term over $\Upsilon$.

Let $\Upsilon$ and $\Gamma$ be two disjoint weighted vocabularies. An IFP(SUM) *stratum of type* $\Upsilon \to \Gamma$ is a set $\Sigma$ of rules over $\Upsilon \cup \Gamma$, with one rule for each symbol in $\Gamma$. In the spirit of stratified Datalog terminology, in $\Sigma$, the symbols from $\Upsilon$ are called *extensional* and those from $\Gamma$ *intensional*.

To define the semantics of an IFP(SUM) stratum $\Sigma$ as above we first introduce the *immediate consequence* $T_\Sigma$ on $\Upsilon \cup \Gamma$-structures.

*Definition 4.4.* Given structure $\mathcal{B}$, we define $T_\Sigma(\mathcal{B}) := \mathcal{C}$, where the universe of $\mathcal{C}$ equals $B$, the universe of $\mathcal{B}$, and $\mathcal{C}$ agrees with $\mathcal{B}$ on the extensional symbols; the intensional symbols in $\mathcal{C}$ are then defined as follows.

- Let $R(\boldsymbol{x}) \leftarrow \varphi$ be a relational rule in $\Sigma$. Then $R^{\mathcal{C}} := R^{\mathcal{B}} \cup \{\boldsymbol{a} \in B^{\mathrm{ar}(R)} \mid \mathcal{B} \models \varphi(\boldsymbol{a})\}$.
- Let $F(\boldsymbol{x}) \leftarrow \theta$ be a weight function rule in $\Sigma$. Then

$$F^{\mathcal{C}}(\boldsymbol{a}) := \begin{cases} [\![\theta]\!]^{\mathcal{B}}(\boldsymbol{a}) & \text{if } F^{\mathcal{B}}(\boldsymbol{a}) = \bot; \\ F^{\mathcal{B}}(\boldsymbol{a}) & \text{otherwise.} \end{cases} \tag{4.A}$$

Since $\mathcal{B}$ is finite and intensional relations and weight functions only grow under application of $T_\Sigma$, there exists a natural number $n$ such that $T_\Sigma^n(\mathcal{B})$ (i.e., the result of $n$ successive applications of $T_\Sigma$ starting from $\mathcal{B}$) equals $T_\Sigma^{n+1}(\mathcal{B})$. We denote this result by $T_\Sigma^\infty(\mathcal{B})$.

We are now ready the define the semantics of $\Sigma$ as a mapping from $\Upsilon$-structures to $\Upsilon \cup \Gamma$-structures. Let $\mathcal{A}$ be an $\Upsilon$-structure and let $\mathcal{A}'$ be its expansion to an $\Upsilon \cup \Gamma$-structure by setting all intensional relations to empty and all intensional weight functions to be undefined everywhere. Then $\Sigma(\mathcal{A}) := T_\Sigma^\infty(\mathcal{A}')$.

*Programs and queries.* An IFP(SUM) program is just a finite sequence of strata. Formally, every stratum, of type $\Upsilon \to \Gamma$, is also a program of that type; moreover, if $\Pi$ is a program of type $\Upsilon \to \Gamma_1$ and $\Sigma$ is a stratum of type $\Upsilon \cup \Gamma_1 \to \Gamma_2$, then $\Pi; \Sigma$ is a program of type $\Upsilon \to \Gamma_1 \cup \Gamma_2$. The semantics is given simply by sequential composition.

As illustrated in Example 4.1, it is customary to designate an *answer symbol* (relation name or weight function name) from among the intensional symbols of a program. In this way we can use programs to express *queries*, i.e., mappings from structures to relations or weight functions.

### 4.2 A loose semantics

The immediate consequence operator just defined only allows to set a new value for a weight function $F$ on a tuple $\boldsymbol{a}$ if $F$ was not yet defined on $\boldsymbol{a}$. The advantage of the resulting functional fixpoint semantics is that it is guaranteed to terminate on finite structures. In practice, however, it may be convenient to be able to perform updates on weight functions.

*Example 4.5.* Consider a distance matrix $W$, represented as a structure with a strict total order relation *ord* and a binary weight function $W$. An almost literal transcription of the Floyd-Warshall algorithm for all-pairs shortest-path distances in IFP(SUM) presents itself below.

$$
\begin{aligned}
&\mathit{chosen}(k) \leftarrow \mathit{next}(k).\\
&D(i,j) \leftarrow \text{if } \mathit{chosen} = \emptyset \text{ then}\\
&\qquad \text{if } \exists k(\mathit{next}(k) \wedge W(i,j) > W(i,k) + W(k,j)) \text{ then } \textstyle\sum_{k:\mathit{next}(k)} W(i,k) + W(k,j)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{else } W(i,j)\\
&\qquad \text{else if } \exists k(\mathit{next}(k) \wedge D(i,j) > D(i,k) + D(k,j)) \text{ then } \textstyle\sum_{k:\mathit{next}(k)} D(i,k) + D(k,j)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{else } D(i,j).
\end{aligned}
$$

Here, $i$, $j$ and $k$ are variables, and $\mathit{next}(k)$ is an abbreviation for the formula $\forall x(\mathit{ord}(x,k) \leftrightarrow \mathit{chosen}(x))$. The auxiliary intensional relation *chosen* is used to iterate over all nodes in the graph; $\mathit{next}(k)$ selects the next node.

Under the functional fixpoint semantics we are using so far, however, this program does not work as intended. After the first iteration, $D$ is already everywhere defined and further updates to $D$ will not be made. The program would work as intended under a more permissive semantics that allows weight functions to be updated. □

The above example suggests an alternative *loose fixpoint* semantics for strata, where updates to weight functions are possible. Care must be taken, however, since termination is then no longer guaranteed. A simple solution we propose is to stop when a fixpoint is reached on the intensional *relations* only.

To define the loose semantics formally for a stratum $\Sigma$ of type $\Upsilon \to \Gamma$, we introduce an alternative immediate consequence operator $L_\Sigma$ (using $L$ for 'loose'). It is defined like $T_\Sigma$ (Definition 4.4), except that Equation (4.A) is replaced simply by $F^C(\boldsymbol{a}) := [\![\theta]\!]^{\mathcal{B}}(\boldsymbol{a})$. The definition for intensional relations is not changed. Since we work with finite structures $\mathcal{B}$, and intensional relations can only grow, there exists a natural number $n$ such that $L_\Sigma^n(\mathcal{B})$ and $L_\Sigma^{n+1}(\mathcal{B})$ agree on the intensional relations. Taking $n$ the smallest such number, we define $L_\Sigma^\infty(\mathcal{B}) := L_\Sigma^n(\mathcal{B})$ and call $n$ the *loose termination index*. Note that the loose semantics is only useful if there are some intensional relations, for otherwise this index is zero.

The result of applying a stratum $\Sigma$ to an $\Upsilon$-structure $\mathcal{A}$, now under the loose semantics, is defined as before, but using $L_\Sigma^\infty$ instead of $T_\Sigma^\infty$, and denoted by $\Sigma^L(\mathcal{A})$. The result of a program (sequence of strata) $\Pi$ on $\mathcal{A}$, using loose semantics for the strata, is denoted by $\Pi^L(\mathcal{A})$.

### 4.3 Comparing the two semantics

It is not difficult to see that the functional fixpoint semantics can be simulated in the loose semantics:

Proposition 4.6. *For every stratum $\Sigma$ of type $\Upsilon \to \Gamma$ there exists a stratum $\Sigma'$ of type $\Upsilon \to \Gamma'$, with $\Gamma \subseteq \Gamma'$, such that $\Sigma'^L(\mathcal{A})$ and $\Sigma(\mathcal{A})$ agree on $\Gamma$.*

Proof. For the relation symbols in $\Gamma$, we have the same rule in $\Sigma'$ as in $\Sigma$. Each weight function rule $F(\boldsymbol{x}) \leftarrow \theta$ in $\Sigma$ is replaced in $\Sigma'$ by $F(\boldsymbol{x}) \leftarrow \text{if } F(\boldsymbol{x}) = \bot \text{ then } \theta \text{ else } F(\boldsymbol{x})$. Finally, to capture the functional fixpoint, for each $F$ as above we include an extra relation symbol $R_F$ in $\Gamma'$. The rule for $R_F$ is $R_F(\boldsymbol{x}) \leftarrow \theta \neq \bot$, thus keeping track of the tuples on which $F$ is defined. □

Conversely, the functional fixpoint semantics can simulate the loose semantics. We can prove this applying *timestamping* and *delayed evaluation* techniques developed for inflationary datalog with negation [54]. Here, a timestamp of a stage in the loose fixpoint is a concatenation of tuples, one for each intensional relation, so that at least one of the tuples is newly added to its intensional relation. Such timestamps become extra arguments of the intensional weight functions. We can

then simulate function updates by defining the function on new timestamps. We can maintain a weak order on the timestamps, or use an extra set of "delayed" intensional relation names, so we can identify the timestamps from the previous iteration as well as the current one. Upon termination of the simulating stratum, a subsequent stratum can be used to project on the last timestamps to obtain the final result of the loose fixpoint.

We conclude:

Theorem 4.7. IFP(SUM) *programs under the functional fixpoint semantics and* IFP(SUM) *programs under the loose fixpoint semantics are equivalent query languages for weighted structures.*

*Example 4.8.* The program from Example 4.5 under the loose fixpoint semantics is simulated by the following program under the functional fixpoint semantics:

$chosen(k') \leftarrow next(k').$
$D'(k', i, j) \leftarrow$ if $\neg next(k')$ then $\bot$ else
  if $chosen = \emptyset$ then
    if $W(i,j) > W(i,k') + W(k',j))$ then $W(i,k') + W(k',j)$ else $W(i,j)$
  else if $\exists k(last(k) \wedge D'(k,i,j) > D'(k,i,k') + D'(k,k',j))$ then $\sum_{k:last(k)} D'(k,i,k') + D'(k,k',j)$
    else $\sum_{k:last(k)} D'(k,i,j)$;
$D(i,j) \leftarrow \sum_{k:last(k)} D'(k,i,j).$

Here, $last(k)$ is an abbreviation for $chosen(x) \wedge \forall x((chosen(x) \wedge x \neq k) \rightarrow ord(x,k))$ which selects the node chosen in the previous iteration. The final rule defining $D$ is in a separate stratum. □

We remark that the above example is much simpler than the general construction in the proof of Theorem 4.7. Indeed, in the example, we may assume a total order, getting timestamps for free. The theorem, however, holds in general. It is an open question whether timestamping is necessary for going from loose to functional fixpoints. Specifically, does there exist a query from weighted structures to weight functions that is expressible in IFP(SUM) using only unary intensional weight functions under the loose semantics, but not under the functional fixpoint semantics?

### 4.4 A Normal Form

In this section, we consider IFP(SUM) in the framework and language of classical fixed-point logics, as studied in recursion theory and finite model theory. We will prove a normal form essentially stating that every IFP(SUM)-program is equivalent to a program consisting of a single stratum with a single intensional symbol followed by a selection and projection.

The redefinition of IFP(SUM) as a logic extending FO(SUM) is standard [14, 23, 41], so we will be brief. We add a new weight term formation rule to the grammar (3.A)–(3.B):

$$\theta ::= \mathsf{ifp}\left(F(x_1,\ldots,x_{\mathrm{ar}(F)}) \leftarrow \theta\right)(x'_1,\ldots,x'_{\mathrm{ar}(F)}), \tag{4.B}$$

where $F$ is a weight function symbol; note that $\theta$ can contain inner ifp operators.

The free variables of such an ifp term are $(\mathrm{free}(\theta) - \{x_1,\ldots,x_{\mathrm{ar}(F)}\} \cup \{x'_1,\ldots,x'_{\mathrm{ar}(F)}\}$. The free, or *extensional*, relation and function symbols in an IFP(SUM)-expression $\xi$, denoted by $\mathrm{ext}(\xi)$, are defined in a straightforward way, letting the ext of ifp-term (4.B) be $\mathrm{ext}(\theta) - \{F\}$. A symbol $F$ is *intensional* in $\xi$ if it appears in a subterm $\mathsf{ifp}\left(F(\boldsymbol{x}) \leftarrow \theta\right)(\boldsymbol{x}')$ of $\xi$. We denote the set of all intensional symbols of $\xi$ by $\mathrm{int}(\xi)$. *In the following, for all* IFP(SUM) *expressions* $\xi$ *we assume that* $\mathrm{int}(\xi) \cap \mathrm{ext}\,\xi = \emptyset$ *and that every intentional symbol is only bound by a single* ifp *operator.* This is without loss of generality by renaming intensional symbols.

The semantics of IFP(SUM) extends the semantics of FO(SUM). To define the semantics of $\eta \coloneqq \mathsf{ifp}\left(F(\boldsymbol{x}) \leftarrow \theta\right)(\boldsymbol{x}')$ on $(\mathcal{A}, \nu)$, with $\mathcal{A}$ an $\Upsilon$-structure with $\mathrm{ext}(\eta) \subseteq \Upsilon$, we define a sequence $(F^{(t)})_{t\in\mathbb{N}\cup\{\infty\}}$ of functions $F^{(t)} : A^{\mathrm{ar}(F)} \rightarrow \mathbb{R}_\bot$ as follows. We set $F^{(0)}(\boldsymbol{a}) \coloneqq \bot$ for all $\boldsymbol{a} \in A^{\mathrm{ar}(F)}$,

and $F^{(t+1)}(\boldsymbol{a}) \coloneqq [\![\theta]\!]^{(\mathcal{A}\frac{F^{(t)}}{F}, \nu\frac{\boldsymbol{a}}{\boldsymbol{x}})}$ if $F^{(t)}(\boldsymbol{a}) = \bot$, and $F^{(t)}(\boldsymbol{a})$ otherwise. Here, $\mathcal{A}\frac{F^{(t)}}{F}$ denotes the $\Upsilon \cup \{F\}$-structure that coincides with $\mathcal{A}$ on all symbols in $\Upsilon \setminus \{F\}$ and interprets $F$ by $F^{(t)}$. For $t = \infty$ we observe that for every $\boldsymbol{a} \in A^{\mathrm{ar}(F)}$, if there is some $t$ such that $F^{(t)}(\boldsymbol{a}) \neq \bot$ then $F^{(t')}(\boldsymbol{a}) = F^{(t)}(\boldsymbol{a})$ for all $t' \geq t$, and in this case we let $F^{(\infty)}(\boldsymbol{a}) \coloneqq F^{(t)}(\boldsymbol{a})$. Otherwise, we let $F^{(\infty)}(\boldsymbol{a}) \coloneqq \bot$. Finally, we let $[\![\eta]\!]^{(\mathcal{A},\nu)} \coloneqq F^{(\infty)}\big(\nu(\boldsymbol{x}')\big)$.

We use the name IFP(SUM) both for the logic in the previous section as well as the variant defined here. As we will show next, the logics are indeed essentially the same. If we explicitly need to distinguish between them, we speak of IFP(SUM) *strata* and *programs* for the syntax defined in Section 4.1 and of IFP(SUM) *(weight) terms* and *formulas* for the syntax defined here.

We begin by observing that the semantics of the ifp operator is compatible with the semantics of strata from Section 4.1. Specifically, let $\theta(\boldsymbol{x})$ be an FO(SUM)-term and consider the IFP(SUM) stratum $\Sigma \coloneqq F(\boldsymbol{x}) \leftarrow \theta$, where $\mathrm{ext}(\theta) \setminus \{F\} \subseteq \Upsilon$. Then for all $\Upsilon$-structures $\mathcal{A}$ we have $[\![\mathsf{ifp}(F(\boldsymbol{x}) \leftarrow \theta)(\boldsymbol{x}')]\!]^{\mathcal{A})} = F^{\Sigma(\mathcal{A})}$.

An apparent difference, however, between programs and ifp-terms, is that each ifp-term defines a single intensional weight function, while programs can inductively define multiple intensional relations and weight functions. This does not imply a higher expressivity for programs, however. By adapting the proof of the Simultaneous Induction Lemma [14, 41] to the IFP(SUM) setting, we can show:

Lemma 4.9. *For every* IFP(SUM) *program* $\Pi$ *with answer symbol* $S$ *there is a closed* IFP(SUM) *expression* $\xi$ *such that* $[\![\xi]\!]^{\mathcal{A}} = S^{\Pi(\mathcal{A})}$ *for all* $\Upsilon$*-structures* $\mathcal{A}$.

The remaining difference between ifp-terms and programs is that ifp-operators can be nested, but strata can only be composed sequentially. We can, however, show the following normal form. Let us say that a *selection condition* is a conjunction of equalities and inequalities.

Lemma 4.10. *Every* IFP(SUM) *formula* $\varphi(\boldsymbol{x})$*, as well as every* IFP(SUM) *program with a relation symbol as answer symbol, is equivalent to a formula of the form* $\exists \boldsymbol{y}\big(\chi(\boldsymbol{y}) \wedge \mathsf{ifp}\,\big(F(\boldsymbol{x},\boldsymbol{y}) \leftarrow \theta\big)(\boldsymbol{x},\boldsymbol{y})\big)$*, where* $\chi$ *is a selection condition and* $\theta(\boldsymbol{x},\boldsymbol{y})$ *is an* FO(SUM) *term. Also, every* IFP(SUM) *term* $\eta(\boldsymbol{x})$*, as well as every* IFP(SUM) *program with a weight function symbol as answer symbol, is similarly equivalent to a term of the form* $\mathrm{avg}_{\boldsymbol{y}:\chi(\boldsymbol{y})}\,\mathsf{ifp}\,\big(F(\boldsymbol{x},\boldsymbol{y}) \leftarrow \theta\big)(\boldsymbol{x},\boldsymbol{y})$.

This lemma can be proved along the lines of the analogous normal-form result for inflationary fixed-point logic [14, Chapter 8]. The idea of the proof is to first simulate nested ifp-operations by a simultaneous induction, as it is defined by an IFP(SUM) stratum. In the simultaneous induction, we first step through the inner induction until it reaches a fixed point. Then we take a single step of the outer induction, again run through the inner induction till it reaches a fixed-point, take a step of the outer induction, et cetera, until the outer induction reaches a fixed point. For this to work it is crucial that we can detect if an induction has reached a fixed point in FO(SUM) and then set a flag to indicate that to the outer induction.

*Remark 4.1.* Since our motivation is neural networks, we fixed the numerical domain to the reals. However, the results presented in this section generalize to any numerical domain with aggregates [23, 35]. The only requirement is that the logic, without recursion, can express the construct $\mathrm{uniq}_{x:\varphi}\,\theta$, with $\varphi$ a formula and $\theta$ a weight term, having the following semantics. Suppose there exists $a \in A$ such that $\mathcal{A} \models \varphi(a)$, and moreover, for all such $a$, the value $[\![\theta]\!]^{\mathcal{A}}(a)$ is the same, say $r \in \mathbb{R}_\bot$. Then we define $[\![\mathrm{uniq}_{x:\varphi}\,\theta]\!]^{\mathcal{A}}$ to be this $r$. Otherwise, we define it to be $\bot$.

In FO(SUM), we can indeed express this as if $\forall x \forall x'((\varphi(x) \wedge \varphi(x')) \rightarrow \theta(x) = \theta(x'))$ then $\mathrm{avg}_{x:\varphi}\,\theta$ else $\bot$, where by $\varphi(x')$ and $\theta(x')$ we mean that $x'$ (a fresh variable) is substituted for the free occurrences of $x$.

## 5 A Polynomial Time Fragment

In finite model theory, fixed-point logics are typically used to capture polynomial time computations, and it would be nice if we could use IFP(SUM) to capture the polynomial properties of neural networks. We have to restrict our attention to weighted structures with rational weights if we want to do this, at least if we want to work in a traditional computation model. But even then it turns out that IFP(SUM) terms cannot be evaluated in polynomial time, as their values may get too large to even be represented in space polynomial in the size of the input structure.

*Example 5.1 ([23]).* Consider the following IFP(SUM) term $\eta(x)$:

$$\eta(x) \coloneqq \mathsf{ifp}\,\big(F(x) \leftarrow \mathsf{if}\ \exists y\, E(y,x)\ \mathsf{then}\ (\sum_{y:E(y,x)} F(y)) \cdot (\sum_{y:E(y,x)} F(y))\ \mathsf{else}\ 2\big)(x).$$

Then if $\mathcal{A}$ is a path of length $n$ and $a$ is last vertex of this path, then $[\![\eta]\!]^{\mathcal{A}}(a) = 2^{2^n}$.

To avoid repeated squaring as illustrated above, we will define a fragment sIFP(SUM), called *scalar* IFP(SUM), that limits the way multiplication and division can be used. It forbids multiplication between two terms that both contain intensional function symbols. In other words, we only allow scalar multiplication when intensional weight functions are involved, where "scalar" is interpreted liberally as "defined nonrecursively". We also forbid division by recursively defined terms.

Formally, for a set $\mathcal{F}$ of function symbols, the syntax of *$\mathcal{F}$-scalar formulas and terms* follows exactly the grammar (3.A), (3.B), (4.B) of IFP(SUM) formulas and terms, except that the rules for multiplication, division, and ifp are changed as follows:

- if $\theta_1$ is $\mathcal{F}$-scalar and $\theta_2$ is a term with $\mathrm{ext}(\theta_2) \cap \mathcal{F} = \emptyset$, then $\theta_1 \cdot \theta_2$, $\theta_2 \cdot \theta_1$, and $\theta_1/\theta_2$ are $\mathcal{F}$-scalar;
- if $\theta$ is $\mathcal{F} \setminus \{F\}$-scalar, then $\mathsf{ifp}\,\big(F(\boldsymbol{x}) \leftarrow \theta\big)(\boldsymbol{x}')$ is $\mathcal{F}$-scalar.

Now an IFP(SUM) expression $\xi$ is called *scalar* if all its subexpressions are $\mathrm{int}(\xi)$-scalar. We denote the scalar fragment of IFP(SUM) by sIFP(SUM). Similarly, an IFP(SUM) stratum $\Sigma$ is scalar if all subexpressions in rules are $\mathrm{int}(\Sigma)$-scalar, and a program is scalar if all its strata are.

*Example 5.2.* The term from Example 5.1 is not scalar as it contains multiplication of two terms involving the intensional symbol $F$. The programs in Examples 4.2 and 4.5 are scalar. □

When we think about the complexity of evaluating IFP(SUM) expressions, we restrict our attention to structures with rational weights, which for simplicity we call *rational structures*. For a rational $\Upsilon$-structure $\mathcal{A}$, by $\|\mathcal{A}\|$ we denote the bitsize of an encoding of $\mathcal{A}$. Then

$$\|\mathcal{A}\| = O\big(|A| + \sum_{R \in \Upsilon \text{ relation symbol}} |R^{\mathcal{A}}| + \sum_{\substack{F \in \Upsilon \text{ weight-} \\ \text{function symbol}}} \sum_{\boldsymbol{a} \in A^{\mathrm{ar}(F)}} \|F^{\mathcal{A}}(\boldsymbol{a})\|\big),$$

where for an integer $n \in \mathbb{N}$, we let $\|n\|$ be the length of the binary encoding of $n$ and for a rational $p/q \in \mathbb{Q}$ in reduced form we let $\|p/q\| = \|p\| + \|q\|$.

We can show that scalar IFP(SUM) has polynomial data complexity:

Theorem 5.3. *There is an algorithm that, given an* sIFP(SUM) *expression $\xi$, a rational structure $\mathcal{A}$, and a tuple $\boldsymbol{a} \in A^{|\boldsymbol{x}|}$, computes $[\![\xi]\!]^{\mathcal{A}}(\boldsymbol{a})$ in time polynomial in $\|\mathcal{A}\|$.*

To prove this theorem (the proof is given in the extended version [26]), we can start from the normal form of Lemma 4.10 which can be seen to preserve scalarness. The proof amounts to a careful verification that, in a recursive rule $F(\boldsymbol{x}) \leftarrow \theta$, the numerator and denominator of $[\![\theta]\!]^{\mathcal{A}}(\boldsymbol{a})$ only depend linearly on $F^{\mathcal{A}}$, in a sense that can be made precise. The difficulty lies in controlling the growth of denominators under the arithmetic operators.

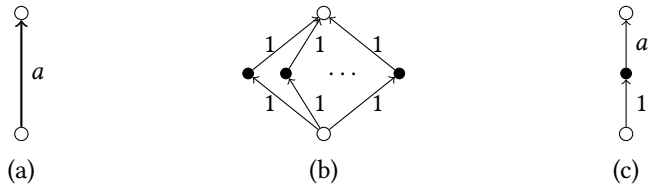



Fig. 1. Splitting (a) an edge with large weight $a \in \mathbb{N}$ into (b) $a$ internal nodes connected by edges of weight 1. Reducing the net in (b) yields the net (c)

*Remark 5.1.* In the language of parameterized complexity, the IFP(SUM)-evaluation problem is in the complexity class XP if we parameterized by the length of the expression. As IFP(SUM) contains the standard fixed-point logic UFP, the problem is actually hard for (uniform) XP [15]. □

*Remark 5.2.* We state Theorem 5.3 for expressions, i.e., weight terms and formulas, under the functional fixpoint semantics, but the theorem extends both to strata and programs, and to the loose fixpoint semantics.

Even on unweighted relational structures, sIFP(SUM) is more expressive than plain fixed point logic IFP, because it can count the number of elements in definable sets using the summation operator, then compare such counts to express, e.g., the majority query. Yet, unsurprisingly, we have the following even for full IFP(SUM):

Theorem 5.4. *There is a boolean query on graphs that is decidable in polynomial time, but not expressible in* IFP(SUM).

This result can be proved by standard techniques; we give a self-contained proof in the extended version [26]. We show that the bijective pebble game [29], which is the standard tool for proving inexpressibility in fixed-point logic with counting, can also be used to prove inexpressibility in IFP(SUM). This was already known for logics with aggregates quite similar to FO(SUM) [30]. Then the usual Cai-Fürer-Immerman construction [12] provides an example of a query inexpressible in IFP(SUM).

We are mainly interested, however, in model-agnostic queries on neural networks. Nevertheless, we still cannot express all such polynomial-time queries in IFP(SUM). An intuition for this is that weights in an FNN can be arbitrary large and IFP(SUM) is too weak for general (polynomial-time) computations with numbers. Actually, already the very simple query $Q_0$ where $Q_0(\mathcal{N})$ is true iff $f^{\mathcal{N}}(1)$ is a natural number, can be shown to be not expressible. But it gets worse: large weights can be avoided, because we can split large-weight edges into many small-weight edges, as Figure 1 illustrates. Using the variation of the above query $Q_0$ that asks if $f^{\mathcal{N}}(1)$ is an even natural number, we can show:

Theorem 5.5. *There is a model-agnostic Boolean query on the class* $\mathbf{K}(1,1)$ *that is decidable in polynomial time, but not expressible in* IFP(SUM) *even on FNNs where all weights are* 1 *or* 0.

The dependence on large weights indeed requires a more subtle analysis. Thereto, for every FNN $\mathcal{N}$ we shall define an equivalent *reduced* FNN $\widetilde{\mathcal{N}}$. Recall that $b(u)$ and $w(v,u)$ denote the bias of nodes $u$ and weight of edges $(v,u)$ in a network. For a set $V$ of nodes we let $w(V,u) \coloneqq \sum_{v\in V} w(v,u)$.

We define an equivalence relation $\sim$ on the set of nodes of $\mathcal{N}$ by induction on the depth of the nodes in $\mathcal{N}$. Only nodes of the same depth can be equivalent. For input nodes and output nodes $u, u'$ we let $u \sim u' \Leftrightarrow u = u'$, so input and output nodes are only equivalent to themselves. Now consider two hidden nodes $u$, $u'$ of the same depth, and suppose that we have already defined

$\sim$ on all nodes of smaller depth. Then $u \sim u'$ if $b(u) = b(u')$ and $w(V, u) = w(V, u')$ for every equivalence class $V$ of nodes of smaller depth.

By $\tilde{u}$ we denote the equivalence class of node $u$. We now define:

*Definition 5.6.* The reduced network $\widetilde{\mathcal{N}}$ has nodes $\tilde{u}$ and edges $(\tilde{v}, \tilde{u})$ for all edges $(v, u)$ of $\mathcal{N}$. We define $b(\tilde{u}) \coloneqq b(u)$ and $w(\tilde{v}, \tilde{u}) \coloneqq \sum_{v' \sim v} w(v', u)$. The input nodes of $\widetilde{\mathcal{N}}$ are the singleton classes of the input nodes of $\mathcal{N}$, and similarly for the output nodes.

A straightforward induction shows that if $u \sim u'$ then $f_u^{\mathcal{N}} = f_{u'}^{\mathcal{N}}$, so $f^{\mathcal{N}} = f^{\widetilde{\mathcal{N}}}$, i.e., an FNN and its reduction represent the same function.

*Example 5.7.* The reduction of the network in Figure 1(b) is shown in Figure 1(c). □

Let $P(X)$ be a polynomial, and let $\mathcal{N}$ be an FNN; we will use the notation $|\mathcal{N}|$ for the number of nodes of a network. With $n = |\mathcal{N}|$, we say that $\mathcal{N}$ has *$P$-bounded weights* if all node and edge weights are rational and of the form $r/q$ for integers $r \in \mathbb{Z}$, $q \in \mathbb{N}$ such that $|r|, q \leq P(n)$. Furthermore, we say that $\mathcal{N}$ has *$P$-bounded reduced weights* if $\widetilde{\mathcal{N}}$ has $P$-bounded weights. A class $\mathbf{K}$ of networks has *polynomially bounded (reduced) weights* if there exist a polynomial $P$ so that every $\mathcal{N} \in \mathbf{K}$ has $P$-bounded (reduced) weights.

*Example 5.8.* The class of networks depicted in Figure 1(b) obviously has polynomially bounded weights (they are all 1), but not polynomially bounded reduced weights. Indeed, the class depicted in Figure 1(c) does not have polynomially bounded weights (since $a$ can be arbitrary but $|\mathcal{N}| = 3$). □

We establish the following completeness result for scalar IFP(SUM).

Theorem 5.9. *Let $Q$ be a polynomial time computable query on $\mathbf{K}(*, *)$. Then $Q$ is expressible in* sIFP(SUM) *on every $\mathbf{K}' \subseteq \mathbf{K}(*, *)$ with polynomially bounded reduced weights.*

The proof (see the extended version [26]) observes that a quasi-order on the nodes of a network $\mathcal{N}$ can be defined uniformly in sIFP(SUM) such that it yields a linear order on $\widetilde{\mathcal{N}}$. Moreover, since $\widetilde{\mathcal{N}}$ has polynomially bounded weights, all numerators and denominators can be represented, using sIFP(SUM) formulas, by lexicographically ordered tuples of nodes. This yields a copy of $\widetilde{\mathcal{N}}$ as an ordered finite structure defined in $\mathcal{N}$. By the Immerman-Vardi theorem we can then also define, in $\mathcal{N}$, the answer of $Q$ on $\widetilde{\mathcal{N}}$. As $Q$ is model-agnostic, this is also the answer of $Q$ on $\mathcal{N}$.

*Remark 5.3.* For simplicity, Theorem 5.9 is stated and proved for boolean queries without parameters. A version for $r$-ary queries with parameters can be formulated and proved, where we then also need to restrict to polynomially bounded reduced rational numbers for the parameters and result tuples. The candidates for parameters and result tuples are passed to the sIFP(SUM) formula as extra weight functions.

## 6 Complexity of model-agnostic queries

With IFP(SUM) in place, it is now tempting to revisit Theorem 3.4 and wonder if the fixed-depth restriction can be lifted simply by replacing FO(SUM) by IFP(SUM). We conjecture, however, that the answer is negative:

Conjecture 6.1. *Let $m > 0$ be a natural number. There exist a boolean query on $\mathbf{K}(m, 1)$ expressible in $\mathsf{FO}(\mathbf{R}_{\text{lin}}, f)$ but not in* IFP(SUM).

The intuition behind this conjecture is that, although general (i.e., not necessarily scalar) IFP(SUM) expressions have the facility to generate very large number and calculate with these numbers,

it is hard to see how this facility helps to express properties about FNNs, or even about abstract structures. Both a proof and a disproof would be equally compelling.

For *scalar* IFP(SUM), we can prove the corresponding conjecture, assuming P ≠ NP. Indeed, whereas sIFP(SUM) queries are computable in polynomial time, there are very simple FO($\mathbf{R}_{\text{lin}}, f$) sentences that already express an NP-hard query, even on $\mathbf{K}(1,1)$.

Theorem 6.2. *It is NP-hard to decide if an FNN $\mathcal{N} \in \mathbf{K}(1,1)$ computes a non-zero function, that is, if $f^{\mathcal{N}}(x) \neq 0$ for some $x \in \mathbb{R}$.*

Note that testing non-zeroness is a boolean model-agnostic query, easily expressed by the FO($\mathbf{R}_{\text{lin}}, f$) sentence $\exists x\, f(x) \neq 0$. In fact, the query also belongs to NP [55, Proposition 1]. In the cited work, Wurm also proves coNP-hardness of deciding whether an FNN $\mathcal{N} \in \mathbf{K}(*,*)$ is zero (phrased as an equivalence problem). Our contribution here is that hardness already holds when the input and output dimensions are fixed to 1. The proof (given in the extended version [26]) reduces from 3-SAT. We interpret rational numbers between 0 and 1, with with binary representation $(0.a_1a_2\ldots a_n)_2$, as assignments on $n$ boolean variables. We then construct a network $\mathcal{N}$ simulating, on these numbers, a given 3-CNF formula over these variables.

*Discussion.* The above result indicates that, on unbounded-depth neural networks, model-agnostic querying is hard (in the sense of computational complexity). Indeed, given the simplicity of the query $\exists x\, f(x) \neq 0$, the result suggests that many of the common model-agnostic methods (cf. Example 3.1) will be hard as well in the absence of a bound on depth, although this needs be investigated in detail.

It remains a worthwhile goal to capture those model-agnostic queries that *are* polynomial-time computable; our Theorem 5.9 is a step in this direction. On the other hand, IFP(SUM) remains interesting as a language also for expressing queries that are not model-agnostic (Example 4.3).

## 7 Iterated transductions

In view of Conjecture 6.1, how can we go beyond IFP(SUM) to define a logic over weighted structures that can express all FO($\mathbf{R}, f$) queries without fixing the network depth? We can get inspiration from how Theorem 3.4 was proved [27]. The first step of that proof is to map any given (fixed-depth) FNN $\mathcal{N} \in \mathbf{K}(m,1)$ to a structure that represents the geometry of the piecewise linear function $f^{\mathcal{N}} : \mathbb{R}^m \to \mathbb{R}$. This geometry consists of a hyperplane arrangement that partitions $\mathbb{R}^m$ in polytopes, plus an affine function on each of the polytopes. Later steps embed the resulting geometry in a higher dimensional space as dictated by the number of variables of the FO($\mathbf{R}_{\text{lin}}, f$) query $\psi$ that we want to express, and construct a cylindrical cell decomposition of the space. The resulting cell decomposition is compatible both with $f^{\mathcal{N}}$ and with the constraints imposed by $\psi$, which allows us to express $\psi$ in FO(SUM).

The steps just described map weighted structures to weighted structures; these transductions are expressed in FO(SUM) by adapting the classical model-theoretic method of interpreting one logical theory in another [31]. Such interpretations (also called transductions [13, 25] or translations [27]) map structures from one vocabulary to structures from another vocabulary by defining the elements of the output structure as equivalence classes of tuples from the input structure, and defining the relations and weight functions by formulas and weight terms.

The important observation is that, in showing that the steps of the proof described above can be expressed as FO(SUM) transductions, the assumption of a fixed depth $\ell$ of $\mathcal{N}$ is only important for the first step. That construction is performed layer by layer, with a transduction that is iterated a fixed number $\ell$ times. We can thus lift the fixed-depth restriction if we can iterate a transduction an unbounded number of times; actually, $O(\ell)$ iterations suffice.

Formally, we can define an *iterated* FO(SUM) *transduction* from vocabulary $\Upsilon$ to vocabulary $\Gamma$ as a pair $(\tau, \varphi)$ where $\tau$ is an FO(SUM) transduction from $\Upsilon \cup \Gamma$ to itself, and $\varphi$ is a closed FO(SUM) formula over $\Upsilon \cup \Gamma$. The semantics, given an $\Upsilon$-structure $\mathcal{A}$, is to first expand $\mathcal{A}$ with the symbols of $\Gamma$ (initializing them to be empty or undefined everywhere). Then, $\tau$ is repeatedly applied until $\varphi$ becomes true.

The difference with IFP(SUM) programs, even under loose semantics, is that a transduction can grow the universe, and can arbitrarily change (also shrink) relations and functions. This can then happen in each step of an iterated transduction.

We can show that iterated transductions are closed under sequential composition. To express a query, we can designate an answer symbol, just like we did for IFP(SUM) programs. We conclude:

Proposition 7.1. *Let $m$ be a natural number. Every boolean* FO($\mathbf{R}_{\text{lin}}, f$) *query on* $\mathbf{K}(m, 1)$ *is expressible by an iterated* FO(SUM) *transduction that, on any* $\mathcal{N} \in \mathbf{K}(m, 1)$*, iterates only* $O(\ell)$ *times, where $\ell$ is the depth of $\mathcal{N}$.*

*Remark 7.1.* Iterated transductions are something of a "nuclear option," in that they are reminiscent of the extension of first-order logic with while-loops and object creation, investigated in the 1990s [2, 3, 52, 53]. Such an extension typically yields computationally complete query languages over finite unweighted structures. We can show (proof omitted) that, similarly, iterated FO(SUM) transductions (without a depth bound on the number of iterations as in the above proposition) are computationally complete over rational weighted structures.

## 8 Conclusion

We have explored approaches, and obstacles, to model-agnostic querying of deep neural networks. In the fixed-depth case, FO(SUM) is already quite expressive and the main challenges now lie in finding good implementation strategies. Without a bound on the depth, there are also challenges in expressivity, theoretical complexity, and query language design. We have introduced a language IFP(SUM) that enables us to evaluate neural networks of unbounded depth and, more generally, express a rich set of queries on such networks.

Some interesting questions remain open. A clear goal is to devise a logic capturing the PTIME model-agnostic queries without the assumption (Theorem 5.9) of polynomial reduced weights. A very concrete question, even independent of the application to neural networks, is to characterize the data complexity of IFP(SUM) (without the scalar restriction), even on unweighted structures. We also encountered the question whether the loose fixpoint semantics can keep arities lower than possible with the functional fixpoint semantics.

Another interesting question is how the yardstick logic FO($\mathbf{R}, f$) can be adapted to work over arbitrary functions $f : \mathbb{R}^m \to \mathbb{R}^p$ where $m$ and $p$ are not fixed in advance. Over boolean models, there are very elegant languages for this [7, 8].

The querying of feedforward neural networks (or other architectures) could also be approached from the perspective of matrix query languages, by viewing a neural network as a sequence of matrices. The expressive power of matrix query languages such as MATLANG [9] or $\sum$-MATLANG [18] is subsumed by FO(SUM) (extending the numerical domain with additional functions). However, to handle neural networks of arbitrary depth, these languages should be extended to handle arbitrary sequences of matrices.

Finally, and of course, we should also investigate the querying of other machine-learning (ML) models, such as numerical decision trees, Transformer models, and graph neural networks, as well as investigate what can be done better when focusing on specific classes of architectures. A large body of work has accumulated, in the ML, logic, and database theory communities, on

understanding the logical expressiveness of ML models. Nevertheless, it is quite a distinct subject to understand the querying of these models by logical methods [38].

## Acknowledgments

We thank the institutions that supported this work.

*Martin Grohe*: Supported by the European Union (ERC, SymSim, 101054974).

*Christoph Standke*: Supported by the German Research Foundation (DFG) under grants GR 1492/16-1 and GRK 2236 (UnRAVeL) and European Union (ERC, SymSim, 101054974).

*Juno Steegmans*: Supported by the Special Research Fund (BOF) of Hasselt University under Grant No. BOF21D11VDBJ.

*Jan Van den Bussche*: Partially supported by the Flanders AI Program (FAIR).

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

## References

[1] S. Abiteboul, R. Hull, and V. Vianu. 1995. *Foundations of Databases.* Addison-Wesley.

[2] S. Abiteboul and P.C. Kanellakis. 1998. Object Identity as a Query Language primitive. *J. ACM* 45, 5 (1998), 798–842.

[3] S. Abiteboul and V. Vianu. 1991. Datalog extensions for database queries and updates. *J. Comput. System Sci.* 43, 1 (1991), 62–124.

[4] M. Abo Khamis, H.Q. Ngo, R. Pichler, D. Suciu, and Y.R. Wang. 2024. Convergence of datalog over (pre-)semirings. *J. ACM* 71, 2 (2024), 8:1–8:55.

[5] M. Abo Khamis, H.Q. Ngo, and A. Rudra. 2017. Juggling functions inside a database. *SIGMOD Record* 46, 1 (2017), 6–13.

[6] A. Albarghouthi. 2021. Introduction to neural network verification. *Foundations and Trends in Programming Languages* 7, 1–2 (2021), 1–157. https://verifieddeeplearning.com

[7] M. Arenas, D. Báez, P. Barceló, J. Pérez, and B. Subercaseaux. 2021. Foundations of symbolic languages for model interpretability, See [44], 11690–11701.

[8] M. Arenas, P. Barceló, D. Bustamente, J. Caraball, and B. Subercaseaux. 2024. A uniform language to explain decision trees. In *Proceedings 21st International Conference on Principles of Knowledge Representation and Reasoning*, P. Marquis, M. Ortiz, and M. Pagnucco (Eds.). IJCAI Organization, 60–70.

[9] R. Brijder, F. Geerts, J. Van den Bussche, and T. Weerwag. 2019. On the expressive power of query languages for matrices. *ACM Transactions on Database Systems* 44, 4 (2019), 15:1–15:31.

[10] Ch. Brix et al. 2024. The fifth international verification of neural networks competition (VNN-COMP): Summary and results. arXiv:2412.19985.

[11] P. Buneman, J. Cheney, and S. Vansummeren. 2008. On the expressiveness of implicit provenance in query and update languages. *ACM Transactions on Database Systems* 33, 4 (2008), 28:1–28:47.

[12] J.-Y. Cai, M. Fürer, and N. Immerman. 1992. An optimal lower bound on the number of variables for graph identification. *Combinatorica* 12, 4 (1992), 389–410.

[13] B. Courcelle and J. Engelfriet. 2012. *Graph Structure and Monadic Second-Order Logic.* Cambridge University Press.

[14] H.-D. Ebbinghaus and J. Flum. 1999. *Finite Model Theory* (second ed.). Springer.

[15] Jörg Flum and Martin Grohe. 2006. *Parameterized Complexity Theory.* Springer.

[16] Vincent Froese, Moritz Grillo, and Martin Skutella. 2025. Complexity of Injectivity and Verification of ReLU Neural Networks (Extended Abstract). In *The Thirty Eighth Annual Conference on Learning Theory, 30-4 July 2025, Lyon, France (Proceedings of Machine Learning Research, Vol. 291)*, Nika Haghtalab and Ankur Moitra (Eds.). PMLR, 2188–2189. https://proceedings.mlr.press/v291/froese25a.html

[17] F. Geerts. 2023. A query language perspective on graph learning. In *Proceedings 42nd ACM Symposium on Principles of Database Systems.* ACM, 373–379.

[18] F. Geerts, T. Muñoz, C. Riveros, and D. Vrgoc. 2021. Expressive power of linear algebra query languages, See [43], 342–354.

[19] F. Geerts and J.L. Reutter. 2022. Expressiveness and approximation properties of graph neural networks. In *10th International Conference on Learning Representations.* OpenReview.net.

[20] F. Geerts and J. Van den Bussche. 2011. Relational completeness of query languages for annotated databases. *J. Comput. System Sci.* 77, 3 (2011), 491–504.

[21] M. Gerarts, J. Steegmans, and J. Van den Bussche. 2025. SQL4NN: Validation and expressive querying of models as data. In *Proceedings 9th Workshop on Data Management for End-to-End Machine Learning*. ACM, 10:1–10:5.

[22] I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep Learning*. MIT Press.

[23] E. Grädel and Y. Gurevich. 1998. Metafinite model theory. *Information and Computation* 140, 1 (1998), 26–81.

[24] T.J. Green, G. Karvounarakis, and V. Tannen. 2007. Provenance semirings. In *Proceedings 26th ACM Symposium on Principles of Database Systems*. ACM, 31–40.

[25] M. Grohe. 2017. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic, Vol. 47. Cambridge University Press.

[26] Martin Grohe, Christoph Standke, Juno Steegmans, and Jan Van den Bussche. 2026. Recursive querying of neural networks via weighted structures. arXiv:2601.03201 [cs.LO] https://arxiv.org/abs/2601.03201

[27] M. Grohe, C. Standke, J. Steegmans, and J. Van den Bussche. 2025. Query languages for neural networks. In *Proceedings 28th International Conference on Database Theory (Leibniz International Proceedings in Informatics, Vol. 328)*, S. Roy and A. Kara (Eds.). Schloss Dagstuhl–Leibniz Center for Informatics, 9:1–9:18.

[28] S. Hand, J. Pool, J. Tran, et al. 2015. Learning both weights and connections for efficient neural network. In *Proceedings Annual Conference on Neural Information Processing Systems*, C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, et al. (Eds.). 1135–1143.

[29] Lauri Hella. 1996. Logical hierarchies in PTIME. *Information and Computation* 129 (1996), 1–19.

[30] L. Hella, L. Libkin, J. Nurmonen, and L. Wong. 2001. Logics with aggregate operators. *J. ACM* 48, 4 (2001), 880–907.

[31] W. Hodges. 1993. *Model Theory*. Cambridge University Press.

[32] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. 1995. Constraint query languages. *J. Comput. System Sci.* 51, 1 (Aug. 1995), 26–52.

[33] Ph.G. Kolaitis and Ch.H. Papadimitriou. 1991. Why not negation by fixpoint? *J. Comput. System Sci.* 43, 1 (1991), 125–144.

[34] G. Kuper, L. Libkin, and J. Paredaens (Eds.). 2000. *Constraint Databases*. Springer.

[35] L. Libkin. 2003. Expressive power of SQL. *Theoretical Computer Science* 296 (2003), 379–404.

[36] L. Libkin. 2004. *Elements of Finite Model Theory*. Springer.

[37] L. Libkin. 2007. Embedded finite models and constraint databases. In *Finite Model Theory and Its Applications*. Springer, Chapter 5.

[38] J. Marques-Silva. 2023. Logic-based explainability in machine learning. In *Reasoning Web: Causality, Explanations and Declarative Knowledge (Lecture Notes in Computer Science, Vol. 13759)*, L.E. Bertossi and G. Xiao (Eds.). Springer, 24–104.

[39] C.T. Marx, F. du Pin Calmon, and B. Ustin. 2020. Predictive multiplicity in classification. In *Proceedings 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*. 6766–6774.

[40] Ch. Molnar. 2022. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable* (second ed.). https://christophm.github.io/interpretable-ml-book

[41] Y.N. Moschovakis. 1974. *Elementary induction on abstract structures*. North-Holland.

[42] N. Pelov, M. Denecker, and M. Bruynooghe. 2007. Well-founded and stable semantcs of logic programs with aggregates. *Theory and Practice of Logic Programming* 7, 3 (2007), 301–353.

[43] PODS 2021. *Proceedings 40th ACM Symposium on Principles of Database Systems*. ACM.

[44] M. Ranzato et al. (Eds.). 2021. .

[45] M.L. Ribeiro, S. Singh, and C. Guestrin. 2016. "Why should I trust you?": Explaining the predications of any classifier. In *Proceedings 22nd SIGKDD International Conference on Knowledge Discovery and Data Mining*, B. Krishnapuram, M. Shah, A.J. Smola, et al. (Eds.). ACM, 1135–1144.

[46] C. Rudin. 2019. Stop explaining black box maching learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1 (2019), 206–215.

[47] S.J. Russell and P. Norvig. 2022. *Artificial Intelligence: A Modern Approach* (fourth ed.). Pearson.

[48] S. Shalev-Shwartz and S. Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

[49] S. Torunczyk. 2020. Aggregate queries on sparse databases. In *Proceedings 39h ACM Symposium on Principles of Database Systems*. ACM, 427–443.

[50] Steffen van Bergerem and Nicole Schweikardt. 2021. Learning Concepts Described By Weight Aggregation Logic. In *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, Ljubljana, Slovenia (Virtual Conference), January 25-28, 2021 (LIPIcs, Vol. 183)*, Christel Baier and Jean Goubault-Larrecq (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 10:1–10:18. doi:10.4230/LIPICS.CSL.2021.10

[51] Steffen van Bergerem and Nicole Schweikardt. 2025. On the VC Dimension of First-Order Logic with Counting and Weight Aggregation. In *33rd EACSL Annual Conference on Computer Science Logic, CSL 2025, Amsterdam, The Netherlands, February 10-14, 2025 (LIPIcs, Vol. 326)*, Jörg Endrullis and Sylvain Schmitz (Eds.). Schloss Dagstuhl -

Leibniz-Zentrum für Informatik, 15:1–15:17. doi:10.4230/LIPICS.CSL.2025.15

[52] J. Van den Bussche and J. Paredaens. 1995. The expressive power of complex values in object-based data models. *Information and Computation* 120 (1995), 220–236.

[53] J. Van den Bussche, D. Van Gucht, M. Andries, and M. Gyssens. 1997. On the completeness of object-creating database transformation languages. *J. ACM* 44, 2 (1997), 272–319.

[54] V. Vianu. 2021. Datalog unchained, See [43], 57–69.

[55] Adrian Wurm. 2024. Robustness Verification in Neural Networks. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 21st International Conference, CPAIOR 2024, Uppsala, Sweden, May 28-31, 2024, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 14743)*, Bistra Dilkina (Ed.). Springer, 263–278. doi:10.1007/978-3-031-60599-4_18

[56] S. Zaidi, A. Zela, et al. 2021. Neural Ensemble Search for Uncertainty Estimation and Dataset Shift, See [44].

Received December 2025; accepted February 2026