

Query Languages for Machine-Learning Models

Martin Grohe  

RWTH Aachen University, Germany

Abstract

In my invited talk and this accompanying paper, I discuss two logics for weighted finite structures: first-order logic with summation (FO(SUM)) and its recursive extension IFP(SUM). These logics originate from foundational work by Grädel, Gurevich, and Meer in the 1990s. In recent joint work with Standke, Steegmans, and Van den Bussche, we have investigated these logics as query languages for machine learning models, specifically neural networks, which are naturally represented as weighted graphs. I present illustrative examples of queries to neural networks that can be expressed in these logics and discuss fundamental results on their expressiveness and computational complexity.

2012 ACM Subject Classification Theory of computation → Database query languages (principles); Theory of computation → Finite Model Theory; Theory of computation → Machine learning theory

Keywords and phrases Expressive power of query languages, fixed-point logics, weighted structures, neural networks, explainable AI

Digital Object Identifier 10.4230/LIPICs.STACS.2026.1

Category Invited Talk

Funding Funded by the European Union (ERC, SymSim, 101054974). 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.

Acknowledgements Thanks to Nicole Schweikardt, Erich Grädel, and Jan Van den Bussche for valuable comments on an earlier version of this paper.

1 Introduction

Background

In the 1980s, finite model theory [13, 29] emerged as a new subfield of mathematical logic motivated by the idea of applying tools from mathematical logic to the analysis of algorithms for and the complexity of computational problems. *Finite* model theory was needed because in many cases problem instances are finite structures such as graphs, Boolean formulas, or circuits, or, generically, finite strings over a finite alphabet. In principle, the input to any computational problem can be described as a finite string over the binary alphabet. Yet the mathematical models naturally describing computational problems often have a numerical component that goes beyond discrete finite structures.

An important showcase for finite model theory is the theory of relational databases [1]. As a first approximation, relational databases are finite relational structures, and (the core of) the query language SQL is the relational calculus, that is, first-order logic. Valuable insights, for example, about the expressivity of query languages can be obtained from this perspective. Yet real databases have datatypes with infinite domains, such as integers, and query languages like SQL can express arithmetical or other operations over these infinite domains. Constraint databases [27] even consider finitely defined relations in an infinite model, such as the ordered field of real numbers, and are analysed by combining methods from finite and infinite model theory. Hybrid dynamical systems [12] combine continuous dynamics with a discrete control. They are typically described by models such as timed



© Martin Grohe;

licensed under Creative Commons License CC-BY 4.0

43rd International Symposium on Theoretical Aspects of Computer Science (STACS 2026).

Editors: Meena Mahajan, Florin Manea, Annabelle McIver, and Nguyễn Kim Thăng

Article No. 1; pp. 1:1–1:18



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



or hybrid automata and analysed using logics with a discrete and continuous component. Weighted graphs, which appear as inputs to many optimisation problems, provide another typical example of *hybrid models* with a discrete and a numerical component. Perhaps the most important hybrid models in current computer-science research are machine-learning models, specifically neural networks, which can also be viewed as weighted graphs.

Despite the infinite, often continuous, parts that all these hybrid models involve, the methods of finite model theory, such as logics describing computation and capturing complexity classes, or Ehrenfeucht-Fraïssé games and locality techniques for analysing expressiveness, seem well-suited for analysing computational aspects of the hybrid models as well. Recognising this, Grädel and Gurevich [17] systematically extended the framework of finite model theory to such “meta-finite models”, as they called them. They extended important logics of finite model theory to the meta-finite realm and developed a basic descriptive complexity theory. Grädel and Meer [18] further developed this descriptive complexity theory for hybrid models over the reals, which are precisely the structures we are most interested in here. Over the years, there have been various other suggestions of logics for hybrid models, in different degrees of generality and with different applications in mind, for example, [14, 15, 28, 32, 33].¹ Let me also mention the very interesting work on logics with semiring semantics, for example, [8, 11, 16, 19], differing from the logics considered here in that weights and numerical values are not addressed explicitly in the language, but only appear as semantic values of formulas.

Querying ML Models

Machine learning models are typical hybrid models in the sense just discussed. Due to their ubiquitous presence, they are being studied from many different perspectives, and I would like to argue that the perspectives of finite (or meta-finite) model theory and database theory, in particular, the idea of querying machine learning models, have the potential to offer new insights and methods.

Modern data science projects generate vast numbers of model artefacts throughout their lifecycle, encompassing training runs, hyperparameter configurations, architectural variations, and performance metrics. While contemporary platforms like MLflow provide valuable metadata-based filtering and search capabilities for organising experiments, they offer little support for querying the internal behaviour or semantics of the models themselves. Here “querying” a model goes far beyond evaluating it on a specific input; it allows for an interaction with the inner workings of a model. The models themselves are complex structures. Specific queries may enable us to understand the functionality of individual components in a model. Or, on the semantical side, we may target the global behaviour of models, that is, the behaviour on all inputs and not just a specific one, for example, to understand the robustness of a model or to verify its correctness.

Crucially, we do not aim for a fixed, prescribed set of queries, but for a flexible language that allows us to formulate queries tailored towards specific properties of models, possibly in a specific application context, and to adaptively refine queries to interact with a model. This may allow for a much more refined understanding, or explanation, of machine learning models than current methods based on fixed, generic explanation modes, for example, counterfactual explanations of specific outcomes, offer. Querying a model is also the basis of verification (just phrased in a different language: querying is the database version of model checking).

¹ ...just to name a few. I do not aim for completeness, and this paper is not intended as a survey. View it as a “personal perspective”. Nevertheless, the area would deserve a more comprehensive treatment.

In [2, 3], the authors study Boolean languages that are well-suited for querying discrete ML models such as decision trees, and they analyse the expressiveness and complexity of these languages. However, most ML models are hybrid models (in the sense discussed above) with a discrete and a continuous component, and to fully access these models, the query language should reflect this. In Sections 3 and 4, we will see two languages with this ability, FO(SUM) and IFP(SUM). The logics go back to the work of Grädel and Meer [18] and Grädel and Gurevich [17]; the exact versions that we consider here have recently been introduced in [22, 23]. Both of these languages stand in the tradition of database query languages [1, 9] and finite-model theory, and they can be seen as instantiations of Grädel and Gurevich’s framework of meta-finite model theory [17].

Let me remark that this paper’s perspective on logics as query languages for ML models is fundamentally different from the work on logical characterisations of the expressiveness of ML models such as graph neural networks and transformers [4, 20, 21, 31].

2 Preliminaries

We denote the sets of real, rational, and natural numbers (including 0) by $\mathbb{R}, \mathbb{Q}, \mathbb{N}$ and let $\mathbb{R}_\perp := \mathbb{R} \cup \{\perp\}$ and $\mathbb{Q}_\perp := \mathbb{Q} \cup \{\perp\}$, where \perp is an additional element representing undefined values. We extend the usual order \leq on \mathbb{R} and \mathbb{Q} to \mathbb{R}_\perp and \mathbb{Q}_\perp by letting $\perp < x$ for all $x \in \mathbb{R}$. 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 $y = 0$.

We use bold-face letters to denote tuples, such as $\mathbf{x} = (x_1, \dots, x_k)$, and we denote the length k of this tuple by $|\mathbf{x}|$.

2.1 Weighted Structures

The hybrid models that we consider here take the form of *weighted (finite, relational) structures*. A *weighted vocabulary* is a finite set Υ of *relation symbols* and *weight-function symbols*. Each symbol S has an *arity* $\text{ar}(S) \in \mathbb{N}$. Note that we do admit relations and weight-functions of arity 0. While 0-ary relations are not particularly interesting, 0-ary weight functions can be useful, because we can view them as *weight constants*. An Υ -structure \mathcal{A} consists of a finite nonempty set A , the *universe* of \mathcal{A} , for each relation symbol $R \in \Upsilon$ a relation $R^{\mathcal{A}} \subseteq A^{\text{ar}(R)}$, and for each weight-function symbol $F \in \Upsilon$ a function $F^{\mathcal{A}} : A^{\text{ar}(F)} \rightarrow \mathbb{R}_\perp$. We always denote structures by calligraphic letters $\mathcal{A}, \mathcal{B}, \dots$ and their universes by the corresponding roman letters A, B, \dots

► **Example 2.1.** A weighted graph can be described as a $\{\text{wt}\}$ -structure \mathcal{G} for a binary weight-function symbol wt . The universe of \mathcal{G} is the vertex set of the graph, and for all vertices v, w , if there is an edge from v to w then $\text{wt}^{\mathcal{G}}(v, w) \in \mathbb{R}$ is the weight of this edge, and otherwise $\text{wt}^{\mathcal{G}}(v, w) = \perp$. For a weighted graph \mathcal{G} , we usually denote the vertex set (the universe) by $V^{\mathcal{G}}$ instead of G , and we let $E^{\mathcal{G}} := \{(v, w) \in (V^{\mathcal{G}})^2 \mid \text{wt}^{\mathcal{G}}(v, w) \neq \perp\}$.

The weighted graph \mathcal{G} is undirected if $\text{wt}^{\mathcal{G}}$ is symmetric, and it is loop-free if $\text{wt}^{\mathcal{G}}(v, v) = \perp$ for all vertices v . ◻

For vocabularies $\Upsilon \subseteq \Upsilon'$, an Υ' -*expansion* of an Υ -structure \mathcal{A} is an Υ' -structure \mathcal{A}' with $A' = A$ and $S^{\mathcal{A}'} = S^{\mathcal{A}}$ for all $S \in \Upsilon$. If \mathcal{A}' is an expansion of \mathcal{A} , then \mathcal{A} is a *reduct* of \mathcal{A}' .

Observe that standard finite relational structures are weighted structures whose vocabulary contains no weight-function symbols. We can think of weighted structures as 2-sorted structures consisting of a finite relational structure as the first sort and the extended real

numbers as the second sort. We typically assume additional structure on the reals, such as a linear order and arithmetic operations. Weight functions map elements from the first to the second sort.

A weighted structure \mathcal{A} is *rational* if the range of all its weight functions is in \mathbb{Q}_\perp . When we discuss algorithmic aspects of weighted structures, we usually restrict our attention to rational structures. We fix a reasonable encoding of rational weighted structures by binary strings and let $\|\mathcal{A}\|$ denote the length of the encoding of \mathcal{A} . Moreover, the *order* $|\mathcal{A}|$ of \mathcal{A} is the number of elements of its universe, that is, $|\mathcal{A}| := |A|$. Note that for a fixed vocabulary Υ , the coding length $\|\mathcal{A}\|$ is polynomial in the order $|\mathcal{A}|$ and the maximum bitlength of the weights.

2.2 Feedforward Neural Networks

We assume that the reader is, at least superficially, familiar with neural networks. We consider only the simplest type of neural network, the feedforward neural network, also known as a multilayer perceptron. We view neural networks as directed acyclic graphs with weighted edges and nodes. Computation takes place at the nodes, or *neurons*. They form an affine linear combination of the values at their in-neighbours, and then apply a (typically nonlinear) *activation function*. The resulting value is passed to the out-neighbours of the node. Nodes of in-degree 0 are input nodes; nodes of out-degree 0 are output nodes. The affine linear combination a node forms is determined by the weights of the incoming edges and the weight of the node itself (called the *bias* of the node). The only activation functions we consider here are the *rectified linear unit* defined by $\text{ReLU}(x) := \max\{0, x\}$ and the identity function $\text{id}(x) = x$ (only at output nodes).

As the reader will know, in practice, the weights and biases of an FNN are learned from data consisting of argument-value pairs of the function to be represented by the neural network. We are not concerned with the learning process here, but only with analysing the learned model.

Formally, a *feedforward neural network (FNN)* is a weighted structure \mathcal{N} of vocabulary $\Upsilon_{\text{FNN}} := \{\text{wt}, \text{bias}, \leq_{\text{in}}, \leq_{\text{out}}\}$ consisting of a binary weight-function symbol wt , a unary weight-function symbol bias , and two binary relation symbols $\leq_{\text{in}}, \leq_{\text{out}}$ satisfying the following conditions:

- $E^{\mathcal{N}} := \{(v, w) \in N^2 \mid \text{wt}^{\mathcal{N}}(v, w) \neq \perp\}$ is the edge set of a directed acyclic graph on $V^{\mathcal{N}} := N$;
- $\text{bias}^{\mathcal{N}}(v) = \perp$ if and only if v is an *input node*, that is, a node of in-degree 0 in the directed acyclic graph $(V^{\mathcal{N}}, E^{\mathcal{N}})$;
- $\leq_{\text{in}}^{\mathcal{N}}$ is a linear order on the set of all input nodes and undefined on all other nodes, and $\leq_{\text{out}}^{\mathcal{N}}$ is a linear order on the set of all *output nodes*, that is, nodes of out-degree 0, and undefined on all other nodes.

The *input dimension* of an FNN is the number of input nodes, and the *output dimension* is the number of output nodes.

An FNN \mathcal{N} of input dimension m and output dimension n defines a function $f^{\mathcal{N}}: \mathbb{R}^m \rightarrow \mathbb{R}^n$. To define this function, we inductively define a function $f_v^{\mathcal{N}}: \mathbb{R}^m \rightarrow \mathbb{R}$ for every node $v \in V^{\mathcal{N}}$. If v is the i th input node with respect to the linear order \leq_{in} , then we let

$f_v^{\mathcal{N}}(x_1, \dots, x_m) := x_i$. If v has in-neighbours u_1, \dots, u_k , we let²

$$f_v^{\mathcal{N}}(x_1, \dots, x_m) := \text{bias}^{\mathcal{N}}(v) + \sum_{i=1}^k \text{wt}(u_i, v) \cdot \text{ReLU}(f_{u_i}^{\mathcal{N}}(x_1, \dots, x_m)).$$

Finally, if v_1, \dots, v_n are the output nodes of \mathcal{N} listed in the order $\leq_{\text{out}}^{\mathcal{N}}$ then we let

$$f^{\mathcal{N}}(x_1, \dots, x_m) := (f_{v_1}^{\mathcal{N}}(x_1, \dots, x_m), \dots, f_{v_n}^{\mathcal{N}}(x_1, \dots, x_m)).$$

Let me remark that we can represent other, more complex neural networks as weighted structures in a similar way. But we restrict our attention to FNNs in this paper.

3 First-Order Logic with Weight Aggregation

Our base logic is a straightforward extension of first-order logic to weighted structures. Variables and quantification range over the (finite) universe of a structure, and the numerical sort only provides values for terms. *First-order logic FO over weighted structures* has two kinds of syntactical objects, *formulas* φ and *weight terms* θ (just called *terms* in the following), which are defined by the following grammar:

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

$$\theta ::= 0 \mid 1 \mid F(x_1, \dots, x_{\text{ar}(F)}) \mid (\theta \circ \theta). \quad (3.B)$$

Here x, y, x_i are variables, R is a relation symbol, $*$ $\in \{\vee, \wedge, \rightarrow\}$ is a Boolean connective, $Q \in \{\exists, \forall\}$ is a quantifier, F is a weight-function symbol, and $\circ \in \{+, -, \cdot, /\}$ is an arithmetic operator. An FO *expression* is either a formula or a term. The *vocabulary* of an expression ξ is the set of all relation and weight function symbols appearing in ξ . (Note that the relation \leq , the constants 0, 1, and the arithmetic operations $+, -, \cdot, /$ are not part of the vocabulary; we think of them as “built-in” relations, functions, constants.) We define the *free variables* of an expression in the obvious way. A *closed expression* is an expression without free variables. Closed formulas are also called *sentences*. We write $\xi(x_1, \dots, x_k)$ to denote that the free variables of an expression ξ are among x_1, \dots, x_k . (Not all of these variables actually have to appear in the expression.)

The semantics of FO over weighted structures is defined in the obvious way. We define a value $\llbracket \xi \rrbracket^{\mathcal{A}}(a_1, \dots, a_k)$ for every FO expression $\xi(x_1, \dots, x_k)$, every weighted structure \mathcal{A} , and all elements $a_1, \dots, a_k \in A$. If ξ is a formula then $\llbracket \xi \rrbracket^{\mathcal{A}}(a_1, \dots, a_k)$ is a Boolean value (0 or 1), and if ξ is a term then $\llbracket \xi \rrbracket^{\mathcal{A}}(a_1, \dots, a_k) \in \mathbb{R}_{\perp}$. If the vocabulary of ξ is not contained in the vocabulary of \mathcal{A} we let $\llbracket \xi \rrbracket^{\mathcal{A}}(a_1, \dots, a_k) := 0$ if ξ is a formula and $\llbracket \xi \rrbracket^{\mathcal{A}}(a_1, \dots, a_k) := \perp$ if ξ is a term. Otherwise, the value $\llbracket \xi \rrbracket^{\mathcal{A}}(a_1, \dots, a_k)$ is defined by a straightforward induction, using the usual semantics of the Boolean connectives and quantifiers and the rules of arithmetic over the extended reals \mathbb{R}_{\perp} explained at the beginning of Section 2.

For closed expressions ξ , we write $\llbracket \xi \rrbracket^{\mathcal{A}}$ instead of $\llbracket \xi \rrbracket^{\mathcal{A}}()$, and for formulas $\varphi(x_1, \dots, x_k)$ we write $\mathcal{A} \models \varphi(a_1, \dots, a_k)$ instead of $\llbracket \varphi \rrbracket^{\mathcal{A}}(a_1, \dots, a_k) = 1$, or just $\mathcal{A} \models \varphi$ if φ is a sentence.

² Our way of defining the function computed by a node is non-standard in that we do not yet apply the activation function, but only apply the ReLU-activation when we feed the value into the next node. This way, we ensure that at the output nodes we apply no activation, or equivalently, the identity activation. We want to do this because, if we apply ReLU to all output values, our neural networks can only compute nonnegative functions.

Observe that for every rational $q \in \mathbb{Q}_\perp$ there is a closed term θ_q with empty vocabulary such that $\llbracket \theta_q \rrbracket^{\mathcal{A}} = q$ for all structures \mathcal{A} (we let $\theta_\perp := 1/0$). This means that we can actually use all elements of \mathbb{Q}_\perp as constants in FO expressions. It is easy to see that the FO terms express precisely all rational functions (quotients of polynomials) with rational coefficients in the weights of a structure.

► **Example 3.1.** We can define the edge set $E^{\mathcal{G}}$ of a weighted graph \mathcal{G} (see Example 2.1) by the formula $\text{edge}(x, y) := \text{wt}(x, y) \neq \perp$, which is an abbreviation for $\neg(\text{wt}(x, y) \leq \perp \wedge \perp \leq \text{wt}(x, y))$.³ Then the formula $\text{triangle}(x, y, z) := \text{edge}(x, y) \wedge \text{edge}(y, z) \wedge \text{edge}(z, x)$ defines the set of all triangles in \mathcal{G} .

The following formula defines the set of all triangles of minimum weight.

$$\begin{aligned} \text{min-wt-triangle}(x, y, z) &:= \text{triangle}(x, y, z) \wedge \forall x' \forall y' \forall z' (\text{triangle}(x', y', z') \\ &\rightarrow \text{wt}(x, y) + \text{wt}(y, z) + \text{wt}(z, x) \leq \text{wt}(x', y') + \text{wt}(y', z') + \text{wt}(z', x')). \end{aligned}$$

⌋

The expressiveness of FO over weighted structures is very limited. For example, it is easy to prove that there is no FO formula expressing that the sum of the weights of the incoming edges to a node equals the sum of the weights of the outgoing edges. To prove such a result, consider weighted graphs where all edge weights are 1. Then local isomorphisms between tuples of vertices of such weighted graphs are just local isomorphisms between the tuples in the underlying unweighted graphs, and we can use standard Ehrenfeucht-Fraïssé-game arguments. Similarly, it can be shown that there is no term expressing the affine linear function needed to evaluate a neural network (see Section 2.2).

3.1 FO(SUM)

To obtain a more expressive logic, we add an aggregation operator that allows us to take sums over definable sets. We also add a conditional operation, which is convenient, but can be viewed as “syntactic sugar” (see Example 3.6). We define the formulas and terms of the logic FO(SUM) using the rules (3.A) and (3.B) and adding two new term-formation rules:

$$\theta ::= \text{if } \varphi \text{ then } \theta_1 \text{ else } \theta_2 \mid \sum_{(x_1, \dots, x_k):\varphi} \theta. \quad (3.C)$$

The summation operator $\sum_{(x_1, \dots, x_k):\varphi}$ binds the variables x_1, \dots, x_k . Hence, the free variables of $\sum_{(x_1, \dots, x_k):\varphi} \theta$ are those of φ and θ except x_1, \dots, x_k . The free variables of $\text{if } \varphi \text{ then } \theta_1 \text{ else } \theta_2$ are those of $\varphi, \theta_1, \theta_2$.

The semantics of the conditional term $\text{if } \varphi \text{ then } \theta_1 \text{ else } \theta_2$ is obvious: if φ holds it takes the value of θ_1 and otherwise the value of θ_2 . To define the semantics of the summation operator, suppose that $\varphi = \varphi(x_1, \dots, x_k, y_1, \dots, y_\ell)$ and $\theta = \theta(x_1, \dots, x_k, y_1, \dots, y_\ell)$. Then for the term $\left(\sum_{(x_1, \dots, x_k):\varphi} \theta \right)(y_1, \dots, y_\ell)$, a structure \mathcal{A} , and elements $b_1, \dots, b_\ell \in A$ we let

$$\left[\sum_{(x_1, \dots, x_k):\varphi} \theta \right]^{\mathcal{A}}(b_1, \dots, b_\ell) := \sum_{\substack{(a_1, \dots, a_k) \in A^k \\ \mathcal{A} \models \varphi(a_1, \dots, a_k, b_1, \dots, b_\ell)}} \llbracket \theta \rrbracket^{\mathcal{A}}(a_1, \dots, a_k, b_1, \dots, b_\ell).$$

³ From now on, we use similar abbreviations without commenting on it. We also omit unnecessary parentheses to improve readability.

We define the sum over the empty set to be 0. Furthermore, if one of the summands is undefined (\perp), then the whole sum is undefined.

► **Example 3.2.** Let us start by considering weighted graphs again. Of course the FO formulas $\text{edge}(x, y)$ and $\text{triangle}(x, y, z)$ defined in Example 3.1 are FO(SUM) formulas as well. Then the terms

$$\#\text{edges} := \sum_{(x,y):\text{edge}(x,y)} 1 \quad \text{and} \quad \#\text{triangles} := \sum_{(x,y,z):\text{triangle}(x,y,z)} 1$$

define the numbers of edges and triangles, respectively. \lrcorner

► **Example 3.3.** Next, we turn to FNNs. As in Example 3.1, we let $\text{edge}(x, y) := \text{wt}(x, y) \neq \perp$ be a formula defining the edge relation. Then the term

$$\#\text{weights} := \sum_{(x,y):\text{edge}(x,y)} 1 + \sum_{x:\text{bias}(x) \neq \perp} 1$$

defines the total number of weights of an FNN. \lrcorner

► **Example 3.4.** For every $d \in \mathbb{N}$, we shall construct an FO(SUM) term $\text{eval}_d(x)$ evaluating the function computed at a node x of depth at most d in an FNN. Here, the *depth* of a node v in an FNN \mathcal{N} is the length of the longest path from an input node to v in the directed acyclic graph $(V^{\mathcal{N}}, E^{\mathcal{N}})$. The notation we use in this example is explained in Section 2.2.

If we want to evaluate neural networks, we have to specify an input. We do this by adding a unary weight function inp to the vocabulary, letting $\Upsilon_{\text{FNN}^I} := \Upsilon_{\text{FNN}} \cup \{\text{inp}\}$. Then an *FNN with input* is an Υ_{FNN^I} -structure \mathcal{N}^I such that the Υ_{FNN} -reduct \mathcal{N} of \mathcal{N}^I is an FNN and $\text{inp}^{\mathcal{N}^I}(v) \neq \perp$ if and only if v is an input node of \mathcal{N} . For an FNN \mathcal{N} of input dimension m , say, with input nodes u_1, \dots, u_m listed in the order \leq_{in} , and an input $(r_1, \dots, r_m) \in \mathbb{R}^m$ we let $\mathcal{N}(r_1, \dots, r_m)$ be the FNN with input expanding \mathcal{N} with $\text{inp}^{\mathcal{N}(r_1, \dots, r_m)}(u_i) = r_i$.

By induction on $d \geq 0$, we shall define an FO(SUM) term $\text{eval}_d(x)$ such that for every $m \in \mathbb{N}$, every FNN \mathcal{N} of input dimension m , every node $v \in V^{\mathcal{N}}$, and every $(r_1, \dots, r_m) \in \mathbb{R}^m$ it holds that

$$\llbracket \text{eval}_d \rrbracket^{\mathcal{N}(r_1, \dots, r_m)}(v) = \begin{cases} f_v^{\mathcal{N}}(r_1, \dots, r_m) & \text{if the depth of } v \text{ is at most } d, \\ \perp & \text{otherwise.} \end{cases}$$

We let $\text{eval}_0(x) := \text{inp}(x)$ and

$$\text{eval}_{d+1}(x) := \text{if } \text{inp}(x) \neq \perp \text{ then } \text{inp}(x) \\ \text{else } \text{bias}(x) + \sum_{y:\text{edge}(y,x)} \text{wt}(y, x) \cdot \text{if } \text{eval}_d(y) \geq 0 \text{ then } \text{eval}_d(y) \text{ else } 0 \cdot \text{eval}_d(y).$$

To understand this term, note that the subterm $\text{if } \text{eval}_d(y) \geq 0 \text{ then } \text{eval}_d(y) \text{ else } 0$ computes ReLU of $\text{eval}_d(y)$. In the else clause, we write $0 \cdot \text{eval}_d(y)$ instead of just 0 to make sure the term is undefined if $\text{eval}_d(y) = \perp$.

Using the term $\text{eval}_d(x)$, for every $i \in \mathbb{N}$ we can define a closed term $\text{eval}_{d,i}$ such that for all $m, n \in \mathbb{N}$, all FNNs \mathcal{N} of input dimension m and output dimension n , and all inputs $(r_1, \dots, r_m) \in \mathbb{R}^m$, if the depth of \mathcal{N} is at most d and $i \leq n$ then $\llbracket \text{eval}_d \rrbracket^{\mathcal{N}(r_1, \dots, r_m)}$ is the i th coordinate of $f^{\mathcal{N}}(r_1, \dots, r_m) \in \mathbb{R}^n$, and otherwise $\llbracket \text{eval}_d \rrbracket^{\mathcal{N}(r_1, \dots, r_m)} = \perp$. \lrcorner

The previous example shows that we can evaluate FNNs of bounded depth in FO(SUM). It is not hard to prove that the bounded-depth assumption cannot be dropped. This follows from Theorem 3.14 below. Before we get there, let us consider a few more examples.

► **Example 3.5.** Summation is only one possible aggregation operator. Other common aggregations are arithmetic mean, count, minimum and maximum. All these are definable in FO(SUM) using just summation.

1. For tuples $\mathbf{x} = (x_1, \dots, x_k)$, $\mathbf{y} = (y_1, \dots, y_\ell)$ of variables and a formula $\varphi(\mathbf{x}, \mathbf{y})$ we define a term $\text{count}_{\mathbf{x}:\varphi}(\mathbf{y})$ by $\text{count}_{\mathbf{x}:\varphi} := \sum_{\mathbf{x}:\varphi} 1$.
Then for all structures \mathcal{A} and $\mathbf{b} \in A^\ell$ we have

$$\llbracket \text{count}_{\mathbf{x}:\varphi} \rrbracket^{\mathcal{A}}(\mathbf{b}) = |\{\mathbf{a} \in A^k \mid \mathcal{A} \models \varphi(\mathbf{a}, \mathbf{b})\}|.$$

2. For tuples $\mathbf{x} = (x_1, \dots, x_k)$, $\mathbf{y} = (y_1, \dots, y_\ell)$ of variables, a formula $\varphi(\mathbf{x}, \mathbf{y})$, and a term $\theta(\mathbf{x}, \mathbf{y})$ we define a term $(\text{avg}_{\mathbf{x}:\varphi}\theta)(\mathbf{y})$ by $\text{avg}_{\mathbf{x}:\varphi}\theta := (\sum_{\mathbf{x}:\varphi} \theta) / \text{count}_{\mathbf{x}:\varphi}$.

Then for all structures \mathcal{A} and elements $\mathbf{b} \in A^\ell$ the value $\llbracket \text{avg}_{\mathbf{x}:\varphi}\theta \rrbracket^{\mathcal{A}}(\mathbf{b})$ is the arithmetic mean of the values $\llbracket \theta \rrbracket^{\mathcal{A}}(\mathbf{a}, \mathbf{b})$ for all $\mathbf{a} \in A^k$ such that $\mathcal{A} \models \varphi(\mathbf{a}, \mathbf{b})$. The arithmetic mean over the empty set is undefined, and indeed, if there are no $\mathbf{a} \in A^k$ such that $\mathcal{A} \models \varphi(\mathbf{a}, \mathbf{b})$ then $\llbracket \text{avg}_{\mathbf{x}:\varphi}\theta \rrbracket^{\mathcal{A}}(\mathbf{b}) = \perp$.

3. For tuples $\mathbf{x} = (x_1, \dots, x_k)$, $\mathbf{y} = (y_1, \dots, y_\ell)$ of variables, a formula $\varphi(\mathbf{x}, \mathbf{y})$, and a term $\theta(\mathbf{x}, \mathbf{y})$ we let $\varphi'(\mathbf{x}, \mathbf{y}) := \varphi(\mathbf{x}, \mathbf{y}) \wedge \forall \mathbf{x}' (\varphi(\mathbf{x}', \mathbf{y}) \rightarrow \theta(\mathbf{x}', \mathbf{y}) \leq \theta(\mathbf{x}, \mathbf{y}))$.

We define a term $(\text{max}_{\mathbf{x}:\varphi}\theta)(\mathbf{y})$ by $\text{max}_{\mathbf{x}:\varphi}\theta := \text{avg}_{\mathbf{x}:\varphi'}\theta$.

Then for all structures \mathcal{A} and $\mathbf{b} \in A^\ell$ we have

$$\llbracket \text{max}_{\mathbf{x}:\varphi}\theta \rrbracket^{\mathcal{A}}(\mathbf{b}) = \max \left\{ \llbracket \theta \rrbracket^{\mathcal{A}}(\mathbf{a}, \mathbf{b}) \mid \mathbf{a} \in A^k \text{ such that } \mathcal{A} \models \varphi(\mathbf{a}, \mathbf{b}) \right\}.$$

Similarly, we can define a term $\text{min}_{\mathbf{x}:\varphi}\theta$. ┘

► **Example 3.6.** In this example, we will show that we can express the conditional if φ then θ_1 else θ_2 using only FO and the summation operation. For simplicity, we assume that $\varphi, \theta_1, \theta_2$ are closed expressions; the generalisation to arbitrary expressions is straightforward.

Note that we have not used the conditional in Example 3.5, so we can use the aggregation operators defined there. Observe that all structures \mathcal{A} we have

$$\llbracket \text{count}_{\mathbf{x}:\varphi} \rrbracket^{\mathcal{A}} = \begin{cases} |\mathcal{A}| & \text{if } \mathcal{A} \models \varphi, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\eta := (\text{count}_{\mathbf{x}:\varphi}) / (\text{count}_{\mathbf{x}:x=x})$. Then $\llbracket \eta \rrbracket^{\mathcal{A}} = 1$ if $\mathcal{A} \models \varphi$ and $\llbracket \eta \rrbracket^{\mathcal{A}} = 0$ otherwise. Thus the term $\eta \cdot \theta_1 + (1 - \eta) \cdot \theta_2$ is equivalent to if φ then θ_1 else θ_2 . ┘

► **Example 3.7.** Trying to understand the inner workings of neural networks, we can look for edges that have no effect on the result of a computation for a given input. We call such edges *useless*.

Let $\text{eval}_d(x)$ be the FO(SUM) term (from Example 3.4) defining the value of an FNN with input at a node x of depth at most d . Let $\text{edge}'(x, y, x_0, y_0) := \text{edge}(x, y) \wedge \neg(x = x_0 \wedge y = y_0)$. This formula defines the edge relation of the FNN obtained by removing edge (x_0, y_0) . Let $\text{eval}'_d(x, x_0, y_0)$ be the term obtained from $\text{eval}_d(x)$ by replacing each subformula $\text{edge}(x, y)$ by $\text{edge}'(x, y, x_0, y_0)$. Then for every FNN \mathcal{N} of input dimension m , every input $\mathbf{r} = (r_1, \dots, r_m) \in \mathbb{R}^m$, and all $v, u_0, v_0 \in V^{\mathcal{N}}$ such that the depth of v is at most d we have

$$\llbracket \text{eval}'_d \rrbracket^{\mathcal{N}(r_1, \dots, r_m)}(v, u_0, v_0) = \llbracket \text{eval}_d \rrbracket^{\mathcal{N} - (u_0, v_0)}(r_1, \dots, r_m)(v) = f_v^{\mathcal{N} - (u_0, v_0)}(r_1, \dots, r_m),$$

where $\mathcal{N} - (u_0, v_0)$ is the FNN obtained from \mathcal{N} by deleting the edge (u_0, v_0) , that is, setting $\text{wt}^{\mathcal{N} - (u_0, v_0)}(u_0, v_0) := \perp$.

Then the formula

$$\text{useless}(x_0, y_0) := \text{edge}(x_0, y_0) \wedge \forall x (x \leq_{\text{out}} x \rightarrow \text{eval}_d(x) = \text{eval}_d(x, x_0, y_0)).$$

defines the set of all useless edges. \lrcorner

Our last example shows that we can also define natural global properties of FNNs that do not depend on a particular input. This example is much more involved (it is actually a deep theorem proved in [23]), and we do not give any details.

► **Example 3.8** ([23]). In this example, we want to show that we can define integrals over functions computed by FNNs in FO(SUM). We need to fix the depth d and the input dimension m of our FNNs.

We specify the boundaries of the region we want to integrate over by $2m$ weight constants $\mathbf{a}_1, \dots, \mathbf{a}_m, \mathbf{b}_1, \dots, \mathbf{b}_m$ (recall that weight constants are 0-ary weight functions). In this example, for an FNN \mathcal{N} and $\mathbf{a} = (a_1, \dots, a_m), \mathbf{b} = (b_1, \dots, b_m) \in \mathbb{R}^m$ with $a_i \leq b_i$ for all $i \in [m]$, by $\mathcal{N}(\mathbf{a}, \mathbf{b})$ we denote the expansion of \mathcal{N} to the vocabulary $\Upsilon_{\text{FNN}} \cup \{\mathbf{a}_1, \dots, \mathbf{a}_m, \mathbf{b}_1, \dots, \mathbf{b}_m\}$ with $\mathbf{a}_i^{\mathcal{N}(\mathbf{a}, \mathbf{b})} = a_i$ and $\mathbf{b}_i^{\mathcal{N}(\mathbf{a}, \mathbf{b})} = b_i$.

Then there is a closed FO(SUM) term $\text{integrate}_{d,m}$ such that for all FNNs \mathcal{N} of input dimension m , output dimension 1, and depth at most d and all $\mathbf{a} = (a_1, \dots, a_m), \mathbf{b} = (b_1, \dots, b_m) \in \mathbb{R}^m$ with $a_i \leq b_i$ we have

$$\llbracket \text{integrate}_{d,m} \rrbracket^{\mathcal{N}(\mathbf{a}, \mathbf{b})} = \int_{a_1}^{b_1} \dots \int_{a_m}^{b_m} f^{\mathcal{N}}(x_1, \dots, x_m) dx_1 \dots dx_m.$$

The proof is difficult. It is based on the FO(SUM) definability of cylindrical cell decomposition of the (piecewise linear) functions computed by FNNs. To get at least an idea of how integrals can be defined, let us consider the special case $d = 2, m = 1$. That is, we look at FNNs \mathcal{N} with one input node, one hidden layer, and one output node. Such an FNN computes a piecewise linear function $f^{\mathcal{N}}: \mathbb{R} \rightarrow \mathbb{R}$. The breakpoints of this function are determined by the nodes on the hidden layer of \mathcal{N} , and using these nodes, we can define the coordinates of all the breakpoints. We can, moreover, define the slopes of the linear pieces connecting the breakpoints. From this, the integral is easy to compute. \lrcorner

3.2 Complexity

In this section, we assume all weighted structures to be rational (see Section 2.1). It is a well-known fact that the evaluation of formulas of first-order logic over (unweighted) finite relational structures is in the complexity class uniform AC^0 , the class of problems decided by dlogtime-uniform families of Boolean circuits over \neg, \wedge, \vee of unbounded fan-in, bounded depth, and polynomial size (see [26]).

To evaluate FO formulas and terms over rational weighted structures, we need to evaluate terms over the rationals. It is known that this is possible in the complexity class uniform TC^0 , defined like AC^0 but allowing threshold gates in the circuits in addition to the standard logic gates [5, 25]. As iterated addition (addition of a family of n numbers of bitlength n) is in uniform TC^0 , we can actually carry out all the arithmetic needed to evaluate FO(SUM) formulas and terms and hence obtain the following theorem.⁴

⁴ It is difficult to trace the results on arithmetic in uniform TC^0 . It was proved in the 1980s (going back to [10]) that arithmetic is in non-uniform TC^0 , but for addition, subtraction, iterated addition, and multiplication, it was known that the constructions could actually be made uniform. Certainly, this was known at the time of Barrington, Immerman and Straubing's work on logic and uniform TC^0 [5]. However, I am not aware of a reference where these results are proved. Iterated addition is critical; therefore, I added a proof that iterated addition is in uniform TC^0 to my recent paper [21]. The most difficult case is division; it was only proved by Hesse in 2000 [24, 25] that division is in uniform TC^0 .

► **Theorem 3.9.** *The data complexity of FO(SUM) is in uniform TC⁰.*

That is, for every closed FO(SUM) expression ξ the value $\llbracket \xi \rrbracket^{\mathcal{A}}$ of ξ in a given structure \mathcal{A} can be computed by a dlogtime uniform family of threshold circuits of bounded depth and polynomial size.

3.3 Model-Agnostic Queries

Often, we are only interested in the function a neural network computes and not in the structure of the network. Note that for every function computable by an FNN, there are infinitely many very different FNNs computing this function. We call queries whose outcome only depends on the function and not on the specific FNN *model agnostic*.

A closed FO(SUM) expression ξ is *model agnostic on a class \mathbf{N}* of FNNs if for all $\mathcal{N}, \mathcal{N}' \in \mathbf{N}$, if $f^{\mathcal{N}} = f^{\mathcal{N}'}$ then $\llbracket \xi \rrbracket^{\mathcal{N}} = \llbracket \xi \rrbracket^{\mathcal{N}'}$. We can extend this definition to other logics on weighted structures, in particular to the logic IFP(SUM) that will be discussed in Section 4. We can even generalise the definition to abstract *Boolean queries* on \mathbf{N} , that is, isomorphism-invariant functions $\Phi: \mathbf{N} \rightarrow \{0, 1\}$, and to isomorphism invariant functions $\Theta: \mathbf{N} \rightarrow \mathbb{R}_{\perp}$. In a different direction, we can also generalise it to classes of expansions of FNNs such as FNNs with input (see Example 3.4) and FNNs with additional weight constants (see Example 3.8). But note that the invariance condition $f^{\mathcal{N}} = f^{\mathcal{N}'}$ always refers to the plain neural networks. For example, for neural networks with input $\mathcal{N}(\mathbf{r}), \mathcal{N}'(\mathbf{r}')$ we still require $f^{\mathcal{N}} = f^{\mathcal{N}'}$ and not $f^{\mathcal{N}}(\mathbf{r}) = f^{\mathcal{N}'}(\mathbf{r}')$.

We denote the class of all FNNs of input dimension m and output dimension n by $\mathbf{N}(m, n)$. Moreover, we let $\mathbf{N}(m, *) := \bigcup_{n \geq 1} \mathbf{N}(m, n)$ and define $\mathbf{N}(*, n)$ and $\mathbf{N}(*, *)$ similarly. Note that $\mathbf{N}(*, *)$ is the class of all FNNs. We denote the class of all FNNs of depth at most d by $\mathbf{N}_d(*, *)$, and for $m, n \in \mathbb{N}_{>0} \cup \{*\}$ we let $\mathbf{N}_d(m, n) := \mathbf{N}(m, n) \cap \mathbf{N}_d(*, *)$. Moreover, we denote the corresponding classes of FNNs with input by $\mathbf{N}^I(m, n)$ and $\mathbf{N}_d^I(m, n)$.

We call an expression, query, or function *fully model agnostic* if it is model agnostic on the class of all FNNs or expansions of FNNs of the suitable form. We leave the “suitable form” and hence the class of structures vague on purpose to be able to phrase results in the most natural form. Typically, we consider the classes $\mathbf{N}(*, *)$ or $\mathbf{N}(*, 1)$, or $\mathbf{N}^I(*, *)$ or $\mathbf{N}^I(*, 1)$ for FNNs with input.

► **Example 3.10.** The FO(SUM) term $\text{eval}_{d,i}$ of Example 3.4 is model agnostic on $\mathbf{N}_d^I(*, *)$, but it is not model agnostic on $\mathbf{N}_{d+1}^I(1, 1)$ and hence not fully model agnostic.

However, for each $i \geq 1$ the evaluation function $\text{Eval}_i: \mathbf{N}^I(*, *) \rightarrow \mathbb{R}$, defined by letting $\text{Eval}_i(\mathcal{N}(\mathbf{r}))$ be the i th entry of the vector $f^{\mathcal{N}}(\mathbf{r})$, is fully model agnostic. In the following, most often we restrict our attention to FNNs with a single output, for which we denote the evaluation function by Eval instead of Eval_1 .

► **Example 3.11.** The term $\#\text{weights}$ of Example 3.3 is not model-agnostic, because FNNs computing the same function may have different weights. Similarly, the sentence $\#\text{weights} \leq 10.000$ is not model agnostic.

However, for every function $f: \mathbb{R} \rightarrow \mathbb{R}$ the query “Is there an FNN \mathcal{N}' with at most 10.000 weights that computes the same function?”, that is, $\Phi: \mathbf{N}(*, *) \rightarrow \{0, 1\}$ defined by

$$\Phi(\mathcal{N}) := \begin{cases} 1 & \text{there exists an } \mathcal{N}' \in \mathbf{N}(*, *) \text{ such that } \llbracket \#\text{weights} \rrbracket^{\mathcal{N}'} \leq 10.000 \\ & \text{and } f^{\mathcal{N}'} = f^{\mathcal{N}}, \\ 0 & \text{otherwise} \end{cases}$$

is fully model-agnostic. ┘

► **Example 3.12.** The FO(SUM) term $\text{integrate}_{d,m}$ of Example 3.8 is model agnostic on the class of suitable expansions of $\mathbf{N}_d(m, 1)$ by weight constants, but not fully model-agnostic.

And again, we can define a fully model-agnostic integration function Integrate on the class of all FNNs with output dimension 1 expanded by suitable weight constants.

We can also consider the special case $\text{Integrate}_{[0,1]}$ of the Integrate query where we set all integration boundaries a_i to 0 and all b_i to 1. Then there is no need to add explicit weight constants for the integration boundaries, and we can view $\text{Integrate}_{[0,1]}$ as a fully model agnostic function on $\mathbf{N}(*, 1)$. \lrcorner

► **Example 3.13.** The Boolean query Zero on $\mathbf{N}(*, *)$ defined by

$$\text{Zero}(\mathcal{N}) := \begin{cases} 1 & \text{if } f^{\mathcal{N}}(\mathbf{r}) = 0 \text{ for all } \mathbf{r} \in \mathbb{R}^m, \text{ where } m \text{ is the input dimension of } \mathcal{N}, \\ 0 & \text{otherwise} \end{cases}$$

is fully model-agnostic.

It was proved in [22] that it is NP-complete to decide if $\text{Zero}(\mathcal{N}) = 0$ for a rational FNN $\mathcal{N} \in \mathbf{N}(1, 1)$. (Wurm [34] had proved earlier that the same query on $\mathbf{N}(*, 1)$ is NP-complete.) Hence, the NonZero query is NP-complete, which is unsurprising, because it may be viewed as a form of satisfiability query for FNNs. Let me remark that the proof of [22] shows that the query $\text{NonZero}_{[0,1]}$ on $\mathbf{N}(1, 1)$, asking if there is an input $r \in [0, 1]$ such that $f^{\mathcal{N}}(r) \neq 0$ for an $\mathcal{N} \in \mathbf{N}(1, 1)$, is NP-hard already. This also implies that it is NP-hard to decide if the function $\text{Integrate}_{[0,1]}$ (see Example 3.12) is nonzero. \lrcorner

We would like to understand which fully model-agnostic queries are expressible in FO(SUM). The answer is disappointing: only the trivial constant queries are.

► **Theorem 3.14** ([23]). *Let $m, n \in \mathbb{N}_{>0}$, and let φ be an FO(SUM) sentence that is model agnostic on $\mathbf{N}(m, n)$. Then either $\mathcal{N} \models \varphi$ for all $\mathcal{N} \in \mathbf{N}(m, n)$ or $\mathcal{N} \not\models \varphi$ for all $\mathcal{N} \in \mathbf{N}(m, n)$.*

A similar result holds for the class $\mathbf{N}^I(m, n)$ of FNNs with input. Hence the evaluation function Eval (see Example 3.10) is not expressible in FO(SUM).

The proof of Theorem 3.14 is easy. Intuitively, it is based on the fact that FO(SUM) can only express local properties, but we can extend edges in an FNN by long paths of weight-1 edges without changing the function that is computed.

However, FO(SUM) can express interesting model-agnostic queries and functions on classes of FNNs of bounded depth (see Examples 3.10 and 3.12). As a benchmark for the expressivity of model-agnostic queries, we take a logic that only has “black box” access to the function computed by an FNN, but not to the FNN itself. The logic we consider is first-order logic over the ordered field of the reals expanded by the function computed by our FNN. For an m -ary function symbol f we let $\text{FO}(\mathcal{R}, f)$ be first-order logic in the language $\{+, \cdot, 0, 1, \leq, f\}$, where $+$, \cdot are binary function symbols, $0, 1$ are constant symbols, and \leq is a binary relation symbol. We interpret $\text{FO}(\mathcal{R}, f)$ formulas over expansions (\mathcal{R}, \hat{f}) of the ordered field of the reals, $\mathcal{R} = (\mathbb{R}, +, \cdot, 0, 1, \leq)$ by an m -ary function $\hat{f}: \mathbb{R}^m \rightarrow \mathbb{R}$. Every $\text{FO}(\mathcal{R}, f)$ sentence φ defines a query over $\mathbf{N}(m, 1)$ by

$$\varphi(\mathcal{N}) = 1 \iff (\mathcal{R}, f^{\mathcal{N}}) \models \varphi.$$

No confusion should arise from the fact that we use the same symbol φ for the query and the sentence that defines it. Clearly, this query is invariant over $\mathbf{N}(m, 1)$. Note that we need to fix the input dimension and the output dimension to work in this logic, because function

symbols have fixed arity. It is not crucial that the output dimension is 1. If we wanted to query FNNs with a (fixed) output dimension n , we could simply add n function symbols instead of just one.

We can also define functions from FNNs to the reals. Each $\text{FO}(\mathcal{R}, f)$ formula $\varphi(y)$ defines a function from $\mathbf{N}(m, 1)$ to \mathbb{R}_\perp by

$$\varphi(\mathcal{N}) = \begin{cases} s & \text{if } (\mathcal{R}, f^{\mathcal{N}}) \models \varphi(s) \text{ and there is no } s' \neq s \text{ such that } (\mathcal{R}, f^{\mathcal{N}}) \models \varphi(s'), \\ \perp & \text{otherwise.} \end{cases}$$

To define queries over FNNs with input, we use formulas $\varphi(x_1, \dots, x_m)$ with m free variables. Each such formula defines a query over $\mathbf{N}^I(m, 1)$ by

$$\varphi(\mathcal{N}, r_1, \dots, r_m) = 1 \quad :\iff \quad (\mathcal{R}, f^{\mathcal{N}}) \models \varphi(r_1, \dots, r_m).$$

Similarly, we can define queries defined over FNNs in $\mathbf{N}(m, 1)$ expanded by any fixed number ℓ of weight constants using $\text{FO}(\mathcal{R}, f)$ formulas with ℓ free variables, and of course, we can also define functions using one additional free variable.

We also consider the *linear fragment* $\text{FO}(\mathcal{R}_{\text{lin}}, f)$ of $\text{FO}(\mathcal{R}, f)$ consisting of all formulas in the language $\{+, (q)_{q \in \mathbb{Q}}, \leq\}$, in which we add a constant symbol for every rational. We interpret these formulas $\text{FO}(\mathcal{R}, f)$ formulas φ over expansions $(\mathcal{R}_{\text{lin}}, \hat{f})$ of $\mathcal{R}_{\text{lin}} = (\mathbb{R}, +, (q)_{q \in \mathbb{Q}}, \leq)$. It is useful to have constants for the rationals in this language. As each rational number is definable in \mathcal{R} , we do not need these additional constants in the logic $\text{FO}(\mathcal{R}, f)$. The formulas of $\text{FO}(\mathcal{R}_{\text{lin}}, f)$ define queries over FNNs in the same way as the formulas of $\text{FO}(\mathcal{R}, f)$.

► **Example 3.15.** The evaluation query **Eval** (see Example 3.10) on $\mathbf{N}^I(m, 1)$ is expressible in $\text{FO}(\mathcal{R}_{\text{lin}}, f)$ by the formula $\varphi(x_1, \dots, x_m, y) := f(x_1, \dots, x_m) = y$.

► **Example 3.16.** The query **Zero** (see Example 3.13) on $\mathbf{N}(m, 1)$ is expressible in $\text{FO}(\mathcal{R}_{\text{lin}}, f)$ by the sentence $\varphi := \forall x_1 \dots \forall x_m f(x_1, \dots, x_m) = 0$.

► **Example 3.17.** We may want to understand how much a function computed by an FNN depends on a particular input feature. For an $\varepsilon > 0$, we call a coordinate $i \in [m]$ ε -irrelevant for a function $f: \mathbb{R}^m \rightarrow \mathbb{R}$ if for all $r_1, \dots, r_m, r'_i \in \mathbb{R}$ it holds that

$$|f(r_1, \dots, r_m) - f(r_1, \dots, r_{i-1}, r'_i, r_{i+1}, \dots, r_m)| \leq \varepsilon.$$

For every (rational) $\varepsilon > 0$, we can easily define an $\text{FO}(\mathcal{R}_{\text{lin}}, f)$ -formula expressing that i is ε -irrelevant for f .

Hence the query $\text{lrr}_{\varepsilon, i}$ on $\mathbf{N}(m, 1)$ defined by $\text{lrr}_{\varepsilon, i}(\mathcal{N}) = 1$ if i is ε -irrelevant for $f^{\mathcal{N}}$ is expressible in $\text{FO}(\mathcal{R}_{\text{lin}}, f)$. \dashv

By Theorem 3.14, the queries **Zero** and $\text{lrr}_{\varepsilon, i}$ on $\mathbf{N}(1, 1)$ are not expressible in $\text{FO}(\text{SUM})$. It is not even obvious how to express the queries on FNNs of bounded depth. However, the following theorem shows that for all d, m they are $\text{FO}(\text{SUM})$ -expressible on $\mathbf{N}_d(m, 1)$.

► **Theorem 3.18** ([23]). *Let $d \in \mathbb{N}_{>0}$. Then every query on $\mathbf{N}_d(m, 1)$ that is expressible in $\text{FO}(\mathcal{R}_{\text{lin}}, f)$ is expressible in $\text{FO}(\text{SUM})$.*

In fact, the theorem also extends to FNNs with input and to arbitrary expansions by weight functions. The proof is difficult. The problem is that $\text{FO}(\mathcal{R}_{\text{lin}}, f)$ can quantify over arbitrary reals, whereas $\text{FO}(\text{SUM})$ formulas only have access to the finite set of weights in the FNN. To resolve this issue, in $\text{FO}(\text{SUM})$ we define a cylindrical cell decomposition of the piecewise-linear function computed by our FNN. Each of the cells can be dealt with uniformly, because the function is linear in each cell. Then we can simulate quantification over the reals by quantification over the cells.

► **Example 3.19.** Let us once more revisit the **Zero** query of Examples 3.13, 3.16. It follows from Theorem 3.18 that for each fixed m and d the restrictions of **Zero** and **NonZero** to $\mathbf{N}_d(m, 1)$ are expressible in $\text{FO}(\text{SUM})$. Hence by Theorem 3.9, the queries can be evaluated in uniform TC^0 . This is remarkable because, as pointed out, **NonZero** is a form of satisfiability, and satisfiability for Boolean circuits is already hard for circuits of depth 3. \lrcorner

We close this section with an example showing that there are model-agnostic queries on bounded-depth FNNs that are expressible in $\text{FO}(\text{SUM})$, but not in $\text{FO}(\mathcal{R}, f)$.

► **Example 3.20** ([23]). The query $\int_0^1 f^{\mathcal{N}}(x)dx = 0$ on $\mathbf{N}_2(1, 1)$ is expressible in $\text{FO}(\text{SUM})$ (see Example 3.8), but not in $\text{FO}(\mathcal{R}, f)$. This follows from the fact that every continuous piecewise linear function $f: \mathbb{R} \rightarrow \mathbb{R}$ can be computed by an FNN of depth 2 and [23, Theorem 6.1] stating that integration over continuous piecewise linear functions $f: \mathbb{R} \rightarrow \mathbb{R}$ is not expressible in $\text{FO}(\mathcal{R}, f)$. The proof is based on the (deep) fact that it is not expressible in $\text{FO}(\mathcal{R}, X)$ if a set $X \subseteq \mathbb{R}$ has even cardinality (see [27, 29]) and a (straightforward) reduction from even cardinality to integration. \lrcorner

4 Fixed-Point Logic with Weight Aggregation

While we may argue that in practice, we often have fixed neural network architectures and just adapt the weights, the fact that we cannot even express the evaluation function for FNNs of unbounded depth in $\text{FO}(\text{SUM})$ seriously limits its applicability as a query language for FNNs.

4.1 IFP(SUM)

To evaluate FNNs of unbounded depth, we need some form of recursion. *Fixed-point logic* is the classical mechanism for formalising inductive definability in a first-order logic framework (see [13, 30]). We define the logic $\text{IFP}(\text{SUM})$, *inflationary fixed-point logic with weight aggregation*, following [22]. The logic is closely related to the functional fixed-point logic for \mathbb{R} -algebras from [18]. The formulas and weight terms of $\text{IFP}(\text{SUM})$ are defined by the same rules as those for $\text{FO}(\text{SUM})$, (3.A), (3.B), and (3.C) adding one additional term-formation rule:

$$\theta ::= \text{ifp}(F(x_1, \dots, x_{\text{ar}(F)}) \leftarrow \theta)(x'_1, \dots, x'_{\text{ar}(F)}), \quad (4.A)$$

where the x_i, x'_i are variables and F is a weight function symbol. Note that the tuples $(x_1, \dots, x_{\text{ar}(F)})$ and $(x'_1, \dots, x'_{\text{ar}(F)})$ are not necessarily disjoint; in fact, we will typically let $x_i = x'_i$ for all i .

The ifp -operator in (4.A) binds the variables $x_1, \dots, x_{\text{ar}(F)}$. Thus the free variables of the fixed-point term $\text{ifp}(F(\mathbf{x}) \leftarrow \theta)(\mathbf{x}')$ are those of θ minus the variables in \mathbf{x} plus the variables in \mathbf{x}' . The ifp operator also binds the weight-function symbol F ; we say that F is an *intensional symbol* of the fixed-point term. All other relation and weight-function symbols appearing in θ are *extensional symbols* of the fixed-point term.

$\text{IFP}(\text{SUM})$ faithfully extends $\text{FO}(\text{SUM})$, so to define the semantics of $\text{IFP}(\text{SUM})$, we only need to define the semantics of the ifp -operator. Consider a term

$$\eta(\mathbf{x}', \mathbf{y}) := \text{ifp}(F(\mathbf{x}) \leftarrow \theta)(\mathbf{x}'), \quad (4.B)$$

where $\theta = \theta(\mathbf{x}, \mathbf{y})$ is a term of vocabulary $\Upsilon \cup \{F\}$. Thus, all extensional symbols of η are in Υ . Let $k := |\mathbf{x}| = |\mathbf{x}'| = \text{ar}(F)$ and $\ell := |\mathbf{y}|$. Let \mathcal{A} be an Υ -structure and $\mathbf{b} \in A^\ell$. For every function $\widehat{F}: A^k \rightarrow \mathbb{R}_\perp$, let $(\mathcal{A}, \widehat{F})$ be the $\Upsilon \cup \{F\}$ -expansion of \mathcal{A} with $F^{(\mathcal{A}, \widehat{F})} = \widehat{F}$. We define a sequence $(F^{(i)})_{i \geq 0}$ of functions $F^{(i)}: A^k \rightarrow \mathbb{R}_\perp$ as follows:

- $F^{(0)}(\mathbf{a}) := \perp$ for all $\mathbf{a} \in A^k$;
- $F^{(i+1)}(\mathbf{a}) := \begin{cases} \llbracket \theta \rrbracket^{(\mathcal{A}, F^{(i)})}(\mathbf{a}, \mathbf{b}) & \text{if } F^{(i)}(\mathbf{a}) = \perp, \\ F^{(i)}(\mathbf{a}) & \text{if } F^{(i)}(\mathbf{a}) \neq \perp. \end{cases}$

Since we never change the value of $F^{(i)}(\mathbf{a})$ once it is defined ($\neq \perp$), there is an $i < |A|^k$ such that $F^{(i)} = F^{(i+1)}$. Let i_∞ be the least such i . Then $F^{(i)} = F^{(i_\infty)}$ for all $i \geq i_\infty$.

Finally, the value of the term $\eta(\mathbf{x}', \mathbf{y})$ in (4.B) is defined by

$$\llbracket \eta \rrbracket^{\mathcal{A}}(\mathbf{a}', \mathbf{b}) := F^{(i_\infty)}(\mathbf{a}').$$

► **Example 4.1.** The evaluation function $\text{Eval}: \mathbf{N}^f(*, *) \rightarrow \mathbb{R}$ (see Example (3.10)) is expressible in IFP(SUM). We first define a term evaluating the function computed at every node of an FNN:

$$\begin{aligned} \text{eval-node}(x) := & \text{ifp}\left(F(x) \leftarrow \text{if } \text{inp}(x) \neq \perp \text{ then } \text{inp}(x) \right. \\ & \left. \text{else } \text{bias}(x) + \sum_{y: \text{edge}(y,x)} \text{wt}(y,x) \cdot \text{if } F(y) \geq 0 \text{ then } F(y) \text{ else } 0 \cdot F(y)\right)(x) \end{aligned}$$

Note how this term formalises the (meta-level) inductive definition of the terms $\text{eval}_d(x)$ in Example 3.4 within the logic. It is easy to see that for every $m \in \mathbb{N}_{>0}$, every FNN $\mathcal{N} \in \mathbf{N}(m, *)$, every input $\mathbf{r} \in \mathbb{R}^m$, and every node v we have $\llbracket \text{eval-node} \rrbracket^{\mathcal{N}(\mathbf{r})}(v) = f_v^{\mathcal{N}}(\mathbf{r})$.

Now we let $\text{eval} := \text{avg}_{x: x \leq_{\text{out}} x} \text{eval-node}(x)$. Then for every FNN $\mathcal{N} \in \mathbf{N}(m, 1)$ and every input $\mathbf{r} \in \mathbb{R}^m$ we have $\llbracket \text{eval} \rrbracket^{\mathcal{N}(\mathbf{r})} = f^{\mathcal{N}}(\mathbf{r})$. \dashv

The definition of IFP(SUM) allows nested fixed-point operators (the term θ in (4.A) may contain further fixed-point operators), but the following theorem shows that these are not necessary. A *selection condition* is a conjunction of equalities and inequalities between variables.

► **Theorem 4.2** ([22]).

1. Every IFP(SUM) formula $\varphi(\mathbf{x})$ is equivalent to a formula of the form $\exists \mathbf{y}(\chi(\mathbf{y}) \wedge \text{ifp}(F(\mathbf{x}, \mathbf{y}) \leftarrow \theta(\mathbf{x}, \mathbf{y})))$, where χ is a selection condition and $\theta(\mathbf{x}, \mathbf{y})$ is an FO(SUM) term.
2. Every IFP(SUM) term $\eta(\mathbf{x})$ is equivalent to a term of the form $\text{avg}_{\mathbf{y}: \chi(\mathbf{y})} \text{ifp}(F(\mathbf{x}, \mathbf{y}) \leftarrow \theta(\mathbf{x}, \mathbf{y}))$, where again χ is a selection condition and $\theta(\mathbf{x}, \mathbf{y})$ is an FO(SUM) term. Furthermore, $\text{avg}_{\mathbf{y}: \chi(\mathbf{y})}$ is the operator defined in Example 3.5.

The theorem is proved using standard techniques from finite model theory (see [13, Chapter 8]).

It is also possible to simulate simultaneous inductions with a single fixed-point operator. In [22], this was used to give a datalog-style syntax for IFP(SUM).

4.2 Complexity

An important property of (least or inflationary) fixed-point logic over finite structures is that queries can be evaluated in polynomial time and that, over ordered finite structures, all polynomial-time queries can be expressed in the logic. We say that fixed-point logic *captures* polynomial time over ordered finite structures. Similarly, it can be proved [18] that over ordered weighted structures, IFP(SUM) captures polynomial time in the Blum-Shub-Smale model of computation over the reals [7, 6]. However, here we are interested in traditional bit complexity, and the following example shows that in this model, IFP(SUM) terms cannot be evaluated in polynomial time, even over rational structures, because they can get too large.

► **Example 4.3** ([17]). Let

$$\sigma(x) := \text{ifp} \left(F(x) \leftarrow \text{if } \exists y \text{ edge}(y, x) \text{ then } \left(\sum_{y: \text{edge}(y, x)} F(y) \right) \cdot \left(\sum_{y: \text{edge}(y, x)} F(y) \right) \text{ else } 2 \right) (x).$$

If we evaluate the term over an FNN $\mathcal{N} \in \mathbf{N}(1, 1)$ that is just a path of length d , then, regardless of the weights, for the output node v of \mathcal{N} it holds that $\llbracket \sigma \rrbracket^{\mathcal{N}}(v) = 2^{2^d}$. \lrcorner

To resolve the issue with repeated squaring that the example shows, in [22] we introduced the *scalar fragment* $\text{sIFP}(\text{SUM})$ of $\text{IFP}(\text{SUM})$ by restricting the use of intensional weight function symbols in multiplications and divisions: in an $\text{sIFP}(\text{SUM})$ expression ξ , for every subterm of the form $\theta_1 \cdot \theta_2$ either θ_1 or θ_2 contains no intensional symbol, and for every subterm of the form θ_1/θ_2 the term θ_2 contains no intensional symbol. (The precise formal definition is a bit tricky, because we need to specify which symbols count as intensional in a specific context. I refer the reader to [22] for details.)

► **Example 4.4.** The term eval of Example 4.1 is in $\text{sIFP}(\text{SUM})$. The term $\sigma(x)$ in Example 4.3 is not. \lrcorner

The following theorem shows that $\text{sIFP}(\text{SUM})$ expressions can be evaluated in polynomial time over rational weighted structures.

► **Theorem 4.5** ([22]). *The data complexity of $\text{sIFP}(\text{SUM})$ is in P.*

Let me remark that the proof of this theorem is not as trivial as it may seem. The difficulty is to bound the size of the denominators of the fractions appearing in the evaluation of an expression.

As for $\text{FO}(\text{SUM})$, we can now ask which model-agnostic queries over FNNs we can express in $\text{IFP}(\text{SUM})$ or $\text{sIFP}(\text{SUM})$.

► **Example 4.6.** If $\text{P} \neq \text{NP}$, then the Zero query and the integration function $\text{Integrate}_{[0,1]}$ are not expressible in $\text{sIFP}(\text{SUM})$. The reason is that NonZero and $\text{Integrate}_{[0,1]} \neq 0$ are NP-hard (see Example 3.13). \lrcorner

While the reason for the inexpressibility in the previous example is high computational complexity, it turns out that we cannot even express all queries computable in polynomial time, not even in $\text{IFP}(\text{SUM})$.

► **Theorem 4.7** ([22]). *There is a model-agnostic query on $\mathbf{N}(1, 1)$ that is decidable in polynomial time, but not expressible in $\text{IFP}(\text{SUM})$ even on FNNs where all weights are 1 or 0.*

To prove this theorem, we exploit the fact that $\text{IFP}(\text{SUM})$ only has a limited ability to carry out computations on the numerical part of the weighted structure, because it has no variables ranging over numbers and hence no quantification over numbers.

So $\text{IFP}(\text{SUM})$ is limited in its ability to carry out computations with rationals of large bit size. This turns out to be the main limitation when it comes to polynomial-time computable queries. In [22], we define a normal form for FNNs, *reduced FNNs*, which are obtained by factoring through an equivalence relation similar to bisimilarity. Every FNN \mathcal{N} is equivalent to a unique reduced FNN $\widetilde{\mathcal{N}}$. For every $b \in \mathbb{N}$, we let $\mathbf{R}_b(*, *)$ be the class of all FNNs \mathcal{N} such that all weights of the reduced FNN $\widetilde{\mathcal{N}}$ have bitsize at most b .

► **Theorem 4.8** ([22]). *For every $b \in \mathbb{N}$, every polynomial-time decidable model-agnostic query on $\mathbf{R}_b(*, *)$ is expressible in $\text{sIFP}(\text{SUM})$.*

5 Concluding Remarks

We studied two logics over weighted structures, $\text{FO}(\text{SUM})$ and $\text{IFP}(\text{SUM})$, mainly with respect to their ability to express queries over feedforward neural networks. While quite a few nontrivial results about these logics have been proved in [22, 23], many interesting questions remain open.

A central result is Theorem 3.18, stating that $\text{FO}(\text{SUM})$ can simulate $\text{FO}(\mathcal{R}_{\text{lin}}, f)$ on bounded depth FNNs. Can we extend this theorem from \mathcal{R}_{lin} to \mathcal{R} ? Is there a way to lift the results to FNNs of arbitrary input dimension? For this, we would have to adapt the definition of $\text{FO}(\mathcal{R}_{\text{lin}}, f)$, because as it is, the input dimension of the FNNs we can consider is simply the arity of the fixed function symbol f .

Our original hope was that we might be able to drop the bounded depth assumption in Theorem 3.18 if we replace $\text{FO}(\text{SUM})$ by $\text{IFP}(\text{SUM})$. In view of the NP-completeness of simple $\text{FO}(\mathcal{R}_{\text{lin}}, f)$ -expressible queries like NonZero , this seems unlikely. But can we actually prove that NonZero is not expressible in $\text{sIFP}(\text{SUM})$ or $\text{IFP}(\text{SUM})$ without making a complexity-theoretic assumption?

We showed that other aggregation operators (counting, arithmetic mean, minimum and maximum) can be expressed in terms of summation alone. In general, aggregation functions can be defined as functions from multisets of reals to the reals (see [17]). It would be interesting to develop an understanding for the inter-definability between different aggregation operators. For example, it seems unlikely that summation can be defined in terms of arithmetic mean.

In a different direction, is there a query language that expresses precisely the model-agnostic queries on FNNs that are polynomial-time computable?

References

- 1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- 2 Marcelo Arenas, Daniel Báez, Pablo Barceló, Jorge Pérez, and Bernardo Subercaseaux. Foundations of symbolic languages for model interpretability. In Marc’Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 11690–11701, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/60cb558c40e4f18479664069d9642d5a-Abstract.html>.
- 3 Marcelo Arenas, Pablo Barceló, Diego Bustamante, Jose Caraball, and Bernardo Subercaseaux. A uniform language to explain decision trees. In Pierre Marquis, Magdalena Ortiz, and Maurice Pagnucco, editors, *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024*, 2024. doi:10.24963/KR.2024/6.
- 4 Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations (ICLR 2020)*. OpenReview.net, 2020. URL: <https://openreview.net/forum?id=r1127AEKvB>.
- 5 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1 . *Journal of Computer and System Sciences*, 41(3):274–306, 1990. doi:10.1016/0022-0000(90)90022-D.
- 6 L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer, 1998.

- 7 L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the AMS*, 21:1–46, 1989.
- 8 Sophie Brinke, Erich Grädel, Lovro Mrkonjic, and Matthias Naaf. Semiring provenance in the infinite. In Antoine Amarilli and Alin Deutsch, editors, *The Provenance of Elegance in Computation - Essays Dedicated to Val Tannen, Tannen's Festschrift, University of Pennsylvania, Philadelphia, PA, USA, May 24-25, 2024*, volume 119 of *OASICs*, pages 3:1–3:26. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/OASICs.TANNEN.3.
- 9 Ashok K. Chandra and David Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25(1):99–128, 1982. doi:10.1016/0022-0000(82)90012-5.
- 10 Ashok K. Chandra, Larry J. Stockmeyer, and Uzi Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, 1984. doi:10.1137/0213028.
- 11 Katrin M. Dannert, Erich Grädel, Matthias Naaf, and Val Tannen. Semiring provenance for fixed-point logic. In Christel Baier and Jean Goubault-Larrecq, editors, *Proceedings of the 29th EACSL Annual Conference on Computer Science Logic*, volume 183 of *LIPICs*, pages 17:1–17:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CSL.2021.17.
- 12 Laurent Doyen, Goran Frehse, George J. Pappas, and André Platzer. Verification of hybrid systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 1047–1110. Springer, 2018. doi:10.1007/978-3-319-10575-8_30.
- 13 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer, 2nd edition, 1999.
- 14 F. Geerts. A query language perspective on graph learning. In Floris Geerts, Hung Q. Ngo, and Stavros Sintos, editors, *Proceedings 42nd ACM Symposium on Principles of Database Systems*, pages 373–379, 2023. doi:10.1145/3584372.3589936.
- 15 Floris Geerts, Thomas Muñoz, Cristian Riveros, Jan Van den Bussche, and Domagoj Vrgoc. Matrix query languages. *SIGMOD Record*, 50(3):6–19, 2021. doi:10.1145/3503780.3503782.
- 16 E. Grädel and V. Tannen. Provenance analysis and semiring semantics for first-order logic. In K. Meer, A. Rabinovich, E. Ravve, and A. Villaveces, editors, *Model Theory, Computer Science and Graph Polynomials. Festschrift for Johann A. Makowsky*. Birkhäuser, 2025. doi:10.1007/978-3-031-86319-6_21.
- 17 Erich Grädel and Yuri Gurevich. Metafinite model theory. *Information and Computation*, 140(1):26–81, 1998. doi:10.1006/INCO.1997.2675.
- 18 Erich Grädel and Klaus Meer. Descriptive complexity theory over the real numbers. In Frank Thomson Leighton and Allan Borodin, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 315–324, 1995. doi:10.1145/225058.225151.
- 19 T.J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In Leonid Libkin, editor, *Proceedings 26th ACM Symposium on Principles of Database Systems*, pages 31–40, 2007. doi:10.1145/1265530.1265535.
- 20 Martin Grohe. The logic of graph neural networks. In *Proceedings of the 36th ACM-IEEE Symposium on Logic in Computer Science*, pages 1–17, 2021. doi:10.1109/LICS52264.2021.9470677.
- 21 Martin Grohe. The descriptive complexity of graph neural networks. *TheoretCS*, 3(25):1–93, 2024. doi:10.46298/theoretics.24.25.
- 22 Martin Grohe, Christoph Standke, Juno Steegmans, and Jan Van den Bussche. Recursive querying of neural networks via weighted structures. *ArXiv*, 2601.03201, 2026. doi:10.48550/arXiv.2601.03201.
- 23 Martin Grohe, Christoph Standke, Juno Steegmans, and Jan Van den Bussche. Query Languages for Neural Networks. In Sudeepa Roy and Ahmet Kara, editors, *28th International Conference on Database Theory (ICDT 2025)*, volume 328 of *Leibniz International Proceedings in Informatics (LIPICs)*, pages 9:1–9:18, 2025. doi:10.4230/LIPICs.ICDT.2025.9.

- 24 William Hesse. Division is in uniform tc^0 . In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 104–114. Springer, 2001. doi:10.1007/3-540-48224-5_9.
- 25 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002. doi:10.1016/S0022-0000(02)00025-9.
- 26 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.
- 27 Gabriel Kuper, Leonid Libkin, and Jan Paredaens, editors. *Constraint Databases*. Springer, 2000.
- 28 Dietrich Kuske and Nicole Schweikardt. First-order logic with counting. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005133.
- 29 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.
- 30 Yiannis N. Moschovakis. *Elementary Induction on Abstract Structures*. North Holland, 1974.
- 31 Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? A survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024. doi:10.1162/TACL_A_00663.
- 32 S. Torunczyk. Aggregate queries on sparse databases. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings 39th ACM Symposium on Principles of Database Systems*, pages 427–443, 2020. doi:10.1145/3375395.3387660.
- 33 Steffen van Bergerem and Nicole Schweikardt. Learning concepts described by weight aggregation logic. In Christel Baier and Jean Goubault-Larrecq, editors, *Proceedings of the 29th EACSL Annual Conference on Computer Science Logic*, volume 183 of *LIPICs*, pages 10:1–10:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CSL.2021.10.
- 34 Adrian Wurm. Robustness verification in neural networks. In Bistra Dilkina, editor, *Proceedings of the 21st International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Part II*, volume 14743 of *Lecture Notes in Computer Science*, pages 263–278. Springer, 2024. doi:10.1007/978-3-031-60599-4_18.