On the Correspondence Between Monotonic Max-Sum GNNs and Datalog

David Tena Cucala[1] , **Bernardo Cuenca Grau**[1] , **Boris Motik**[1] , **Egor V. Kostylev**[2]
[1] Department of Computer Science, University of Oxford, UK
[2] Department of Informatics, University of Oslo, Norway
{david.tena.cucala, bernardo.cuenca.grau, boris.motik}@cs.ox.ac.uk, egork@uio.no

Abstract

Although there has been significant interest in applying machine learning techniques to structured data, the *expressivity* (i.e., a description of what can be learned) of such techniques is still poorly understood. In this paper, we study data transformations based on *graph neural networks* (GNNs). First, we note that the choice of how a dataset is encoded into a numeric form processable by a GNN can obscure the characterisation of a model's expressivity, and we argue that a *canonical* encoding provides an appropriate basis. Second, we study the expressivity of *monotonic max-sum* GNNs, which cover a subclass of GNNs with max and sum aggregation functions. We show that, for each such GNN, one can compute a Datalog program such that applying the GNN to any dataset produces the same facts as a single round of application of the program's rules to the dataset. Monotonic max-sum GNNs can sum an unbounded number of feature vectors which can result in arbitrarily large feature values, whereas rule application requires only a bounded number of constants. Hence, our result shows that the unbounded summation of monotonic max-sum GNNs does not increase their expressive power. Third, we sharpen our result to the subclass of *monotonic max* GNNs, which use only the max aggregation function, and identify a corresponding class of Datalog programs.

1 Introduction

Data management tasks such as query answering or logical reasoning can be abstractly seen as transforming an input dataset into an output dataset. A key aspect of such transformations is their *expressivity*, which is often established by identifying a logic-based language that realises the same class of transformations. For example, core aspects of the SQL and SPARQL query languages have been characterised using fragments of first-order logic (Abiteboul, Hull, and Vianu 1995; Pérez, Arenas, and Gutierrez 2009), and logical deduction over RDF datasets has been described using the rule-based language *Datalog* (Motik et al. 2012). Such correspondences enable rigorous understanding and comparison of different data management languages.

Recently, there has been an increasing interest in applying machine learning techniques to data management tasks. A key benefit is that the desired transformation between datasets can be induced from examples, rather than specified explicitly. Many models have been proposed for this purpose, such as recurrent (Hölldobler, Kalinke, and Störr 1999), fibring (Bader, d'Avila Garcez, and Hitzler 2005), and feed-forward networks (Bader et al. 2007), architectures that simulate forward (Dong et al. 2019; Campero et al. 2018) and backward chaining (Rocktäschel and Riedel 2017), and architectures for rule learning (Yang, Yang, and Cohen 2017; Sadeghian et al. 2019). *Graph neural networks* (GNNs) have proved particularly popular since they can express graph transformations and have been widely applied to link prediction and node classification tasks in structured datasets (Schlichtkrull et al. 2018; Pflueger, Tena Cucala, and Kostylev 2022; Liu et al. 2021; Ioannidis, Marques, and Giannakis 2019; Qu, Bengio, and Tang 2019; Yang, Cohen, and Salakhutdinov 2016; Kipf and Welling 2017; Zhang and Chen 2018; Teru, Denis, and Hamilton 2020).

Characterising the expressivity of ML models for data management has thus steadily gained importance, and computational logic provides a well-established methodology: we can describe conditions under which ML-induced models become equivalent to logical formalisms in the sense that applying the ML model to an arbitrary dataset produces the same result as applying a specific logical formula. In a pioneering study, Barceló et al. (2020) showed that each GNN-induced transformation expressible in first-order logic is equivalent to a concept query of the $\mathcal{ALCQ}$ *description logic* (Baader et al. 2007)—a popular KR formalism. Huang et al. (2023) proved an analogous result for a class of GNNs with a dedicated vertex and colour. Morris et al. (2019) showed that GNNs can express certain types of graph isomorphism tests. Sourek, Zelezný, and Kuzelka (2021) characterised the expressivity of GNNs using a hybrid language where each Datalog rule is annotated with a tensor. Tena Cucala et al. (2022) characterised the expressivity of *monotonic GNNs* (MGNNs), which use the max aggregation function and require all weights in the matrices to be nonnegative, in terms of a class of Datalog programs. Finally, Tena Cucala, Cuenca Grau, and Motik (2022) characterised the expressivity of the Neural-LP model of rule learning.

In this paper, we take a next step in the study of the expressivity of GNN-based transformations of structured data. A key technical challenge can be summarised as follows. GNNs typically use summation to aggregate feature vectors of all vertices adjacent to a given vertex in the input graph. The number of adjacent vertices in the input is unbounded (i.e., there is no a priori limit on the number of neighbours

a vertex can have), and so the summation result can be unbounded as well; hence, it appears that arbitrarily many vertices can influence whether a fact is derived. This seems fundamentally different to reasoning in fragments of first-order logic such as Datalog: the number of constants that need to be jointly considered in an application of a Datalog rule is determined by the number of rule variables, and *not* by the structure of the input dataset. Thus, at first glance, one might expect GNNs with summation to be fundamentally different from Datalog rules. To shed light on this issue, we present several novel contributions.

In Section 3 we focus on a key obstacle: to apply a GNN to a dataset, the latter must be encoded as a graph where each vertex is assigned a numeric feature vector; but then, the expressivity of the transformation inevitably depends on the details of the encoding, which obscures the contribution of the GNN itself. To overcome this, we adopt a *canonical* encoding, variants of which have already been considered by Schlichtkrull et al. (2018), Barceló et al. (2020), and Pflueger, Tena Cucala, and Kostylev (2022). We define a GNN to be *equivalent* to a Datalog program if applying the GNN to any dataset while using the canonical encoding produces the same facts as applying the program's rules to the dataset *once* (i.e., without fixpoint iteration). Finally, we observe that noncanonical encodings by Tena Cucala et al. (2022), Morris et al. (2019), or Liu et al. (2021) can be described using well-known extensions of Datalog, and so the expressivity of transformations based on such encodings can be characterised by composing all relevant programs.

In Section 4 we present our main technical contribution. First, we introduce a class of *monotonic max-sum* GNNs. Similarly to the MGNNs by Tena Cucala et al. (2022), monotonic max-sum GNNs require matrix weights to be be nonnegative; however, they allow for the max or sum aggregation functions in each network layer, and they place certain restrictions on the activation and classification functions (ReLU and threshold functions are allowed). Tena Cucala et al. (2022) showed that the performance of such GNNs with just max aggregation on tasks such as knowledge graph completion is on a par with that of other recent approaches. Hence, monotonic max-sum GNNs are practically relevant, but they also allow their predictions to be explained using logical proofs. Second, we prove that each monotonic max-sum GNN is equivalent to a Datalog program of a certain shape possibly containing inequalities in rule bodies. Strictly speaking, such a program can be recursive in the sense that the same predicate can occur in both rule bodies and heads; however, our notion of equivalence does not involve fixpoint iteration (i.e., the program's rules are applied just once). Thus, monotonic max-sum GNNs can derive facts with predicates from the input, but they cannot express true recursive properties such as reachability; moreover, the ability to produce unbounded feature values does not lead to a fundamental increase in expressivity. Our equivalence proof is quite different from the analogous result for MGNNs: when aggregation is limited to just max, the value of each feature of a vertex clearly depends on only a fixed number of neighbours of the vertex. Third, we prove that the equivalent Datalog program can be computed from the GNN itself. This result is interesting because it requires enumerating potentially infinite sets of real-valued candidate feature values in a way that guarantees termination. This provides a starting point for future development of practical techniques for extracting Datalog programs from monotonic max-sum GNNs.

Finally, in Section 5 we sharpen our results to *monotonic max* GNNs, which allow only for max aggregation. We show that, analogously to MGNNs, each monotonic max GNN is equivalent to a positive Datalog program; however, we also present a converse result: we identify a class Datalog programs such that, for each program in the class, there exists an equivalent monotonic max GNN. In this way, we obtain an exact characterisation of an interesting class of GNN-based transformations using logical formalisms.

The proofs of all theorems are given in full in Appendices A and B.

2 Preliminaries

We next recapitulate the basics of Datalog and GNNs.

Datasets and Datalog. We fix a signature consisting of countably infinite, disjoint sets of *predicates* and *constants*. Each predicate is associated with a nonnegative integer arity. We also consider a countably infinite set of *variables* that is disjoint with the sets of predicates and constants.

A *term* is a variable or a constant. An *atom* is of the form $P(t_1, \ldots, t_n)$ where P is a predicate of arity n and $t_1, \cdots, t_n$ are terms. An *inequality* is an expression of the form $t_1 \not\approx t_2$ where t_1 and t_2 are terms. A *literal* is an atom or an inequality. A term or a literal is *ground* if it is variable-free. A *fact* is a ground atom and a *dataset* is a finite set of facts; thus, datasets cannot contain inequalities. A conjunction α of facts is true in a dataset D, written $D \models \alpha$, if $A \in D$ for each fact A in α. A ground inequality $s \not\approx t$ is true if $s \neq t$; for uniformity with facts, we often write $D \models s \not\approx t$ even though the truth of $s \not\approx t$ does not depend on D. A (Datalog) *rule* is of the form (1) where $n \geq 0$, $B_1, \ldots, B_n$ are *body* literals, and H is the *head* atom:

$$B_1 \wedge \cdots \wedge B_n \rightarrow H. \tag{1}$$

A (Datalog) *program* is a finite set of rules. A *substitution* ν is a mapping of finitely many variables to ground terms; for α a literal, $\alpha\nu$ is the result of replacing in α each variable x with $\nu(x)$ provided the latter is defined. Each rule r of form (1) defines an *immediate consequence* operator T_r on datasets: for D a dataset, $T_r(D)$ is the dataset that contains the fact $H\nu$ for each substitution ν mapping all variables of r to terms occurring in D such that $D \models B_i\nu$ for each $1 \leq i \leq n$. For $\mathcal{P}$ a program, $T_{\mathcal{P}}(D) = \bigcup_{r \in \mathcal{P}} T_r(D)$.

To simplify the formal treatment, we do not make the usual *safety* requirement where each variable in a rule must occur in a body atom; in fact, the body can be empty, which we denote by $\top$. For example, rule $r = \top \rightarrow R(x, y)$ is syntactically valid; moreover, the definition of T_r ensures that $T_r(D)$ contains exactly each fact $R(s, t)$ for all (not necessarily distinct) terms s and t occurring in D.

Conjunctions α and β of literals are *equal up to variable renaming* if there exists a bijective mapping ν from the set

of all variables of α to the set of all variables of β such that $\alpha\nu$ and β contain exactly the same conjuncts; this notion is extended to rules in the obvious way. A set S *contains* a conjunction α of literals *up to variable renaming* if there exists $\beta \in S$ such that α and β are equal up to variable renaming.

Graph Neural Networks. We use $\mathbb{R}$ and $\mathbb{R}_0^+$ for the sets of real and nonnegative real numbers, respectively. Also, we use $\mathbb{N}$ for the set of natural numbers, and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

A function $\sigma : \mathbb{R} \to \mathbb{R}$ is *monotonically increasing* if $x < y$ implies $\sigma(x) \leq \sigma(y)$. Function σ is *Boolean* if its range is $\{0, 1\}$. Finally, σ is *unbounded* if, for each $y \in \mathbb{R}$, there exists $x \in \mathbb{R}$ such that $\sigma(x) > y$.

A real *multiset* is a function $S : \mathbb{R} \to \mathbb{N}_0$ that assigns to each $x \in \mathbb{R}$ the number of occurrences $S(x)$. Such S is *finite* if $S(x) > 0$ for finitely many $x \in \mathbb{R}$; the *cardinality* of such S is $|S| = \sum_{x \in \mathbb{R}} S(x)$; and $\mathcal{F}(\mathbb{R})$ is the set of all finite real multisets. We often write a finite S as a list of possibly repeated real numbers in double-braces $\{\!\{ \ldots \}\!\}$. Finally, we treat a set as a multiset where each element occurs just once.

We consider vectors and matrices over $\mathbb{R}$ and $\mathbb{R}_0^+$. For $\mathbf{v}$ a vector and i a natural number, $(\mathbf{v})_i$ is the i-th element of $\mathbf{v}$. We apply scalar functions to vectors element-wise; for example, given n vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ of equal dimension, $\max\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ is the vector whose i-th element is equal to $\max\{(\mathbf{v}_1)_i, \ldots, (\mathbf{v}_n)_i\}$.

For Col a finite set of *colours* and $\delta \in \mathbb{N}$ a *dimension*, a (Col, δ)-graph is a tuple $\mathcal{G} = \langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda \rangle$ where $\mathcal{V}$ is a finite set of *vertices*; for each $c \in \mathsf{Col}$, $\mathcal{E}^c \subseteq \mathcal{V} \times \mathcal{V}$ is a set of directed *edges*; and *labelling* λ assigns to each $v \in \mathcal{V}$ a *feature* vector $\mathbf{v}_\lambda$ of dimension δ. Graph $\mathcal{G}$ is *symmetric* if $\langle v, u \rangle \in \mathcal{E}^c$ implies $\langle u, v \rangle \in \mathcal{E}^c$ for each $c \in \mathsf{Col}$, and it is *Boolean* if $(\mathbf{v}_\lambda)_i \in \{0, 1\}$ for each $v \in \mathcal{V}$ and $i \in \{1, \ldots, \delta\}$. To improve readability, we abbreviate $\mathbf{v}_\lambda$ to just $\mathbf{v}$ when the labelling function is clear from the context; analogously, we abbreviate $\mathbf{v}_{\lambda_\ell}$ to $\mathbf{v}_\ell$.

A (Col, δ)-*graph neural network* (*GNN*) $\mathcal{N}$ with $L \geq 1$ *layers* is a tuple

$$\langle \{\mathbf{A}_\ell\}_{1 \leq \ell \leq L}, \{\mathbf{B}_\ell^c\}_{c \in \mathsf{Col} \text{ and } 1 \leq \ell \leq L}, \{\mathbf{b}_\ell\}_{1 \leq \ell \leq L}, \{\mathsf{agg}_\ell\}_{1 \leq \ell \leq L}, \sigma, \mathsf{cls} \rangle, \tag{2}$$

where, for each $\ell \in \{1, \ldots, L\}$ and $c \in \mathsf{Col}$, $\mathbf{A}_\ell$ and $\mathbf{B}_\ell^c$ are matrices over $\mathbb{R}$ of dimension $\delta_\ell \times \delta_{\ell-1}$ with $\delta_0 = \delta_L = \delta$, $\mathbf{b}_\ell$ is a vector over $\mathbb{R}$ of dimension δ_ℓ, $\mathsf{agg}_\ell : \mathcal{F}(\mathbb{R}) \to \mathbb{R}$ is an *aggregation* function, $\sigma : \mathbb{R} \to \mathbb{R}$ is an *activation* function, and $\mathsf{cls} : \mathbb{R} \to \{0, 1\}$ is a *classification* function.

Applying (Col, δ)-GNN $\mathcal{N}$ to (Col, δ)-graph $\mathcal{G}$ induces the sequence $\lambda_0, \ldots, \lambda_L$ of vertex labelling functions such that $\lambda_0 = \lambda$ and, for each $\ell \in \{1, \ldots, L\}$ and $v \in V$, the value of $\mathbf{v}_\ell$ is given by

$$\mathbf{v}_\ell = \sigma\Big(\mathbf{A}_\ell \mathbf{v}_{\ell-1} + \sum_{c \in \mathsf{Col}} \mathbf{B}_\ell^c \, \mathsf{agg}_\ell\big(\{\!\{ \mathbf{u}_{\ell-1} \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}\big) + \mathbf{b}_\ell\Big). \tag{3}$$

The result $\mathcal{N}(\mathcal{G})$ of applying $\mathcal{N}$ to $\mathcal{G}$ is the Boolean (Col, δ)-graph with the same vertices and edges as $\mathcal{G}$, but where each vertex $v \in \mathcal{V}$ is labelled by $\mathsf{cls}(\mathbf{v}_L)$.

3 Choosing an Encoding/Decoding Scheme

To realise a dataset transformation using a GNN, we must first encode the input dataset into a graph that can be processed by a GNN, and subsequently decode the GNN's output back into a dataset. Several encoding/decoding schemes have been proposed in the literature, and their details differ considerably. As a result, when characterising GNN-based transformations of datasets using logic, it can be hard to understand which properties of the characterisation are due to the chosen encoding/decoding scheme, and which are immanent to the GNN used to realise the transformation. In this paper we consider primarily the encoding scheme that straightforwardly converts a dataset into a graph, but we also discuss how to take other encoding schemes into account.

3.1 Canonical Encoding/Decoding Scheme

A straightforward way to encode a dataset containing only unary and binary facts into a Boolean (Col, δ)-graph is to transform terms into vertices, use vertex connectivity to describe binary facts, and encode presence of unary facts in feature vectors. Such encoding/decoding schemes, which we call *canonical*, have already been widely used in the literature with minor variations (Schlichtkrull et al. 2018; Pflueger, Tena Cucala, and Kostylev 2022; Barceló et al. 2020). They establish a direct syntactic correspondence between datasets and coloured graphs and are thus a natural starting point for studying the expressivity of GNNs.

We next describe one such scheme. In particular, we introduce (Col, δ)-datasets, which naturally correspond to a large class of (Col, δ)-graphs. Our definitions provide the foundation necessary to formulate our expressivity results in Section 4. In Section 3.2 we discuss how to combine our expressivity results with more complex encoding schemes.

Definition 1. *Let* Col *be a set of colours and let* $\delta \in \mathbb{N}$ *be a dimension. A* (Col, δ)-signature *contains*

- *a binary predicate* E^c *for each colour* $c \in \mathsf{Col}$*, and*
- *a unary predicate* U_i *for each* $i \in \{1, \ldots, \delta\}$.

A (Col, δ)-fact *has a predicate from the* (Col, δ)*-signature, and a* (Col, δ)-dataset *contains only* (Col, δ)*-facts.*

We assume that terms occurring in datasets correspond one-to-one to vertices of coloured graphs—that is, each term t is paired with a unique vertex v_t. This is again without loss of generality since the result of applying a GNN to a coloured graph does not depend on the identity of vertices, but only on the graph structure and the feature vectors.

We are now ready to define the canonical GNN-based transformations of (Col, δ)-datasets.

Definition 2. *The* canonical encoding $\mathsf{enc}(D)$ *of a* (Col, δ)*-dataset* D *is the Boolean* (Col, δ)*-graph* $\langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda \rangle$ *defined as follows:*

- $\mathcal{V}$ *contains the vertex* v_t *for each term* t *occurring in* D*;*
- $\langle v_t, v_s \rangle \in \mathcal{E}^c$ *if* $E^c(t, s) \in D$ *for each* $c \in \mathsf{Col}$*; and*
- $(\mathbf{v}_t)_i = 1$ *if* $U_i(t) \in D$*, and* $(\mathbf{v}_t)_i = 0$ *otherwise.*

The canonical decoding $\mathsf{dec}(\mathcal{G})$ *of a Boolean* (Col, δ)*-graph* $\mathcal{G} = \langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda \rangle$ *is the dataset that contains*

- *the fact* $E^c(t, s)$ *for each* $\langle v_t, v_s \rangle \in \mathcal{E}^c$ *and* $c \in \mathsf{Col}$*, and*

- *the fact* $U_i(t)$ *for each* $v_t \in \mathcal{V}$ *and* $i \in \{1, \dots, \delta\}$ *such that* $(\mathbf{v}_t)_i = 1$.

Each (Col, δ)*-GNN* $\mathcal{N}$ *induces the* canonical *transformation* $T_\mathcal{N}$ *on* (Col, δ)*-datasets where* $T_\mathcal{N}(D) = \mathsf{dec}(\mathcal{N}(\mathsf{enc}(D)))$ *for each* (Col, δ)*-dataset* D.

This encoding neither introduces nor omits any information from the input dataset, so a (Col, δ)-dataset D and its canonical encoding $\mathsf{enc}(D)$ straightforwardly correspond to one another. Since datasets are directional, (Col, δ)-graphs must be directed as well to minimise the discrepancy between the two representations. The canonical decoding is analogous to the encoding, and the two are inverse operations on graphs that are regular as per Definition 3.

Definition 3. *A* (Col, δ)*-graph* $\mathcal{G} = \langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda \rangle$ *is* regular *if* $\mathcal{G}$ *is Boolean and each vertex* $v \in \mathcal{V}$ *either occurs in* $\mathcal{E}^c$ *for some* $c \in \mathsf{Col}$*, or* $(\mathbf{v})_i = 1$ *for some* $i \in \{1, \dots, \delta\}$.

Our canonical encoding produces only regular graphs, and there is a one-to-one correspondence between (Col, δ)-datasets and regular (Col, δ)-graphs. Our results from the following sections can be equivalently framed as characterising expressivity of GNN transformations of regular graphs in terms of Datalog programs. Graphs that are not Boolean do not correspond to encodings of datasets, so we do not see a natural way to view GNN transformations over such graphs in terms of logical formalisms. Finally, a (Col, δ)-graph $\mathcal{G}$ that is Boolean but not regular contains 'isolated' vertices that are not connected to any other vertex and are labelled by zeros only. When such $\mathcal{G}$ is decoded into a (Col, δ)-dataset, such 'isolated' vertices do not produce any facts in $\mathsf{dec}(\mathcal{G})$ and thus several non-regular Boolean graphs can produce the same (Col, δ)-dataset. Note, however, that each 'isolated' zero-labelled vertex is transformed by a GNN in the same way—that is, the vector labelling the vertex in the GNN's output does not depend on any other vertices but only on the matrices of the GNN. Consequently, such vertices are not interesting for our study of GNN expressivity.

We are now ready to formalise our central notion of equivalence between a GNN and a Datalog program.

Definition 4. *A* (Col, δ)*-GNN* $\mathcal{N}$ captures *a rule or a Datalog program* α *if* $T_\alpha(D) \subseteq T_\mathcal{N}(D)$ *for each* (Col, δ)*-dataset* D*. Moreover,* $\mathcal{N}$ *and* α *are* equivalent *if* $T_\mathcal{N}(D) = T_\alpha(D)$ *for each* (Col, δ)*-dataset* D.

The key question we address in Sections 4 and 5 is the following: under what conditions is a given (Col, δ)-GNN $\mathcal{N}$ equivalent to a Datalog program, and can this program (at least in principle) be computed from $\mathcal{N}$?

3.2 Noncanonical Encoding/Decoding Schemes

For each (Col, δ)-dataset D, the binary facts of D and $T_\mathcal{N}(D)$ coincide, and so applying $T_\mathcal{N}$ to D cannot derive any binary facts. To overcome this limitation, more complex, noncanonical encodings have been proposed (Tena Cucala et al. 2022; Morris et al. 2019; Liu et al. 2021). These introduce vertices representing combinations of several constants so that facts of higher arity can be encoded in appropriate feature vectors, but there is no obvious canonical way to achieve this. Expressivity results based on such encodings are less transparent because it is not obvious which aspects of expressivity are due to the encoding/decoding scheme and which are immanent to the GNN itself.

We argue that noncanonical encoding/decoding schemes can often be described by a pair of programs $\mathcal{P}_\mathsf{enc}$ and $\mathcal{P}_\mathsf{dec}$, possibly expressed in a well-known extension of Datalog, which convert an input dataset into a (Col, δ)-dataset and vice versa. Thus, given an arbitrary dataset D, the result of applying the end-to-end transformation that uses a GNN $\mathcal{N}$ and the respective encoding/decoding scheme is $T_{\mathcal{P}_\mathsf{dec}}(T_\mathcal{N}(T_{\mathcal{P}_\mathsf{enc}}(D)))$. Furthermore, if $\mathcal{N}$ is equivalent to a Datalog program $\mathcal{P}_\mathcal{N}$, then the composition of $\mathcal{P}_\mathsf{enc}$, $\mathcal{P}_\mathcal{N}$, and $\mathcal{P}_\mathsf{dec}$ characterises the end-to-end transformation. This allows us to clearly separate the contribution of the GNN from the contributions of the encoding and decoding.

Tena Cucala et al. (2022) recently presented a dataset transformation based on a class of *monotonic* GNNs (MGNNs). Their approach is applicable to a dataset D that uses unary predicates $A_1, \dots, A_\epsilon$ and binary predicates $R_{\epsilon+1}, \dots, R_\delta$, and D is encoded into a symmetric (Col, δ)-graph over the set of colours $\mathsf{Col} = \{c_1, c_2, c_3, c_4\}$. The encoding introduces a vertex v_a for each constant a in D as well as vertices $v_{a,b}$ and $v_{b,a}$ for each pair of constants a, b occurring together in a binary fact in D. Predicates are assigned fixed positions in vectors so that the value of a component of a vector labelling a vertex indicates the presence or absence of a specific fact in D. For example, if $A_i(a) \in D$, then $(\mathbf{v}_a)_i$ is set to 1; analogously, if $R_j(a, b) \notin D$ but a and b occur in D in a binary fact, then $(\mathbf{v}_{a,b})_j$ is set to 0. Moreover, the edges of the coloured graph indicate different types of 'connections' between constants; for example, vertices v_a and $v_{a,b}$ are connected by an edge of colour c_1 to indicate that constant a occurs first in the constant pair (a, b). A variant of this approach was also proposed by Liu et al. (2021) in the context of knowledge graph completion.

We next show how to capture this encoding using rules. Note that the encoder introduces vertices of the form $v_{a,b}$ for pairs of constants a and b, so the encoding program $\mathcal{P}_\mathsf{enc}$ requires value invention. This can be conveniently realised using functional terms. For example, we can represent vertex $v_{a,b}$ using term $g(a, b)$, and we can represent each vertex of the form v_a using a term $f(a)$ for uniformity. Applying the encoding program $\mathcal{P}_\mathsf{enc}$ to a dataset thus produces a (Col, δ)-dataset with functional terms, which should be processed by the GNN as if they were constants; for example, the canonical encoding should transform $g(a, b)$ into vertex $v_{g(a,b)}$. Based on this idea, the encoding program $\mathcal{P}_\mathsf{enc}$ contains rule (4) instantiated for each $i \in \{1, \dots, \epsilon\}$, and rules (5)–(13) instantiated for each $j \in \{\epsilon + 1, \dots, \delta\}$.

$$A_i(x) \to U_i(f(x)) \tag{4}$$
$$R_j(x, y) \to U_j(g(x, y)) \tag{5}$$
$$R_j(x, y) \to E^{c_1}(f(x), g(x, y)) \tag{6}$$
$$R_j(x, y) \to E^{c_1}(g(x, y), f(x)) \tag{7}$$
$$R_j(x, y) \to E^{c_2}(f(y), g(x, y)) \tag{8}$$
$$R_j(x, y) \to E^{c_2}(g(x, y), f(y)) \tag{9}$$
$$R_j(x, y) \to E^{c_3}(g(x, y), g(y, x)) \tag{10}$$

$$R_j(x,y) \rightarrow E^{c_3}(g(y,x), g(x,y)) \tag{11}$$
$$R_j(x,y) \rightarrow E^{c_4}(f(x), f(y)) \tag{12}$$
$$R_j(x,y) \rightarrow E^{c_4}(f(y), f(x)) \tag{13}$$

Rules (4) and (5) ensure that all unary and binary facts in the input dataset are encoded as facts of the form $U_i(f(a))$ and $U_j(g(a,b))$; thus, when these are further transformed into a (Col, δ)-graph, the vectors labelling vertices $v_{f(a)}$ and $v_{g(a,b)}$ encode all input facts of the form $A_i(a)$ and $R_j(a,b)$ for $i \in \{1,\dots,\epsilon\}$ and $j \in \{\epsilon+1,\dots,\delta\}$. In addition, rules (6)–(13) encode the adjacency relationships between terms: colour c_1 connects terms $g(a,b)$ and $f(a)$, colour c_2 connects $g(a,b)$ and $f(b)$, colour c_3 connects $g(a,b)$ and $g(b,a)$, and colour c_4 connects terms $f(a)$ and $f(b)$ provided that a and b occur jointly in a binary fact.

Program $\mathcal{P}_{\mathsf{dec}}$ capturing the decoder contains rule (14) instantiated for each $i \in \{1,\dots,\epsilon\}$, as well as rule (15) instantiated for each $j \in \{\epsilon+1,\dots,\delta\}$.

$$U_i(f(x)) \rightarrow A_i(x) \tag{14}$$
$$U_j(g(x,y)) \rightarrow R_j(x,y) \tag{15}$$

Intuitively, these rules just 'read off' the facts from the labels of vertices such as $v_{f(a)}$ and $v_{g(a,b)}$. The composition of these three programs is a (function-free) Datalog program.

It is straightforward to show that, for each dataset D, the graph obtained by applying the encoder by Tena Cucala et al. (2022) is isomorphic to the graph obtained by applying the canonical encoding from Definition 2 to $T_{\mathcal{P}_{\mathsf{enc}}}(D)$ and thus program $\mathcal{P}_{\mathsf{enc}}$ correctly captures their encoder.

A limitation of this encoding is that the transformation's output can contain a fact of the form $R(a,b)$ only if the input dataset contains a fact of the form $S(a,b)$ or $S(b,a)$. Intuitively, the presence of $S(a,b)$ or $S(b,a)$ in the input ensures that the resulting (Col, δ)-graph contains a vertex $v_{g(a,b)}$ for representing binary facts of the form $R(a,b)$. An obvious way to overcome this limitation is to introduce terms $g(a,b)$ for all constants a and b occurring in the input, without requiring a and b to occur jointly in a binary fact. While this increases the expressivity of the end-to-end transformation, the increase is due to the encoding step, rather than the GNN. Our framework makes this point clear. For example, we can extend $\mathcal{P}_{\mathsf{enc}}$ with rules such as (16)–(19) and so on for all other combinations of unary and binary predicates and colours. The chaining of $\mathcal{P}_{\mathsf{enc}}$, $\mathcal{P}_{\mathcal{N}}$, and $\mathcal{P}_{\mathsf{dec}}$ can now capture different transformations even if $\mathcal{P}_{\mathcal{N}}$ remains the same.

$$A_i(x) \wedge A_j(y) \rightarrow E^{c_1}(f(x), g(x,y)) \tag{16}$$
$$A_i(x) \wedge A_j(y) \rightarrow E^{c_1}(g(x,y), f(x)) \tag{17}$$
$$R_i(x,z) \wedge A_j(y) \rightarrow E^{c_1}(g(x,y), f(x)) \tag{18}$$
$$R_i(z,x) \wedge A_j(y) \rightarrow E^{c_1}(g(x,y), f(x)) \tag{19}$$

Morris et al. (2019) introduced k-GNNs and showed them to be more expressive than standard GNNs. The input to a k-GNN is a symmetric (Col, δ_1)-graph $\mathcal{G}_1$ without self-loops where Col contains a single colour c and, for each vertex v of $\mathcal{G}_1$, $(\mathbf{v})_i = 1$ for exactly one $1 \leq i \leq \delta_1$. To apply a k-GNN to $\mathcal{G}_1$, the latter is transformed into another (Col, δ_2)-graph $\mathcal{G}_2$ that contains one vertex for each set of k distinct vertices of $\mathcal{G}_1$, and then a standard (Col, δ_2)-GNN is applied to $\mathcal{G}_2$.

We next show that the transformation of $\mathcal{G}_1$ into $\mathcal{G}_2$ can be captured by a program $\mathcal{P}_{\mathsf{enc}}$ that transforms a (Col, δ_1)-dataset over unary predicates $A_1,\dots,A_{\delta_1}$ and a binary predicate R into a (Col, δ_2)-dataset. Thus, the increase in expressivity of k-GNNs does not come from the GNN model itself, but rather from the encoding implicit in their approach. For readability, we make several simplifying assumptions. First, while Morris et al. (2019) consider sets of k distinct vertices in order to ensure practical scalability, we consider k-tuples instead and limit our presentation to just $k = 2$. Second, we consider just the *local neighbourhood* approach to connecting vertices in $\mathcal{G}_2$. Finally, our encoding requires extending Datalog not only with function symbols, but also with stratified negation-as-failure not (Dantsin et al. 2001).

Program $\mathcal{P}_{\mathsf{enc}}$ consists of rules (20)–(23) instantiated for all $i,j,k,\ell \in \{1,\dots,\delta_1\}$.

$$\begin{aligned} &A_i(x) \wedge A_j(y) \wedge x \not\approx y \wedge \\ &A_k(x) \wedge A_\ell(z) \wedge x \not\approx z \wedge \\ &\qquad R(y,z) \wedge y \not\approx z \rightarrow E^c(g(x,y), g(x,z)) \end{aligned} \tag{20}$$

$$\begin{aligned} &A_i(y) \wedge A_j(x) \wedge y \not\approx x \wedge \\ &A_k(z) \wedge A_\ell(x) \wedge z \not\approx x \wedge \\ &\qquad R(y,z) \wedge y \not\approx z \rightarrow E^c(g(y,x), g(z,x)) \end{aligned} \tag{21}$$

$$\begin{aligned} &A_i(x) \wedge A_j(y) \wedge x \not\approx y \wedge \mathsf{not}\, R(x,y) \\ &\qquad \rightarrow U_{i,j,0}(g(x,y)) \end{aligned} \tag{22}$$

$$\begin{aligned} &A_i(x) \wedge A_j(y) \wedge x \not\approx y \wedge R(x,y) \\ &\qquad \rightarrow U_{i,j,1}(g(x,y)) \end{aligned} \tag{23}$$

Conjunctions of the form $A_i(x) \wedge A_j(y) \wedge x \not\approx y$ in these rules identify pairs of distinct constants a and b (corresponding to the vertices of $\mathcal{G}_1$) in the input dataset, and, for each such pair, $g(x,y)$ introduces a term $g(a,b)$ (corresponding to a vertex of $\mathcal{G}_2$). Rules (20) and (21) encode the *local neighbourhood* approach: terms $g(a,b)$ and $g(d,e)$ are connected in $\mathcal{G}_2$ if either $a = b$ and $d \neq e$, or $a \neq b$ and $d = e$, and additionally the two constants in the inequality are connected in $\mathcal{G}_1$. Finally, rules (22) and (23) identify the type of the subgraph of $\mathcal{G}_1$ that a and b participate in. Specifically, a fact of the form $U_{i,j,0}(g(a,b))$ says that a and b are labelled in $\mathcal{G}_1$ by A_i and A_j respectively, but they are not connected in $\mathcal{G}_1$. A fact of the form $U_{i,j,1}(g(a,b))$ is analogous, but with the difference that a and b are connected in $\mathcal{G}_1$.

4 GNNs with Max-Sum Aggregation

In this section, we introduce monotonic max-sum GNNs and prove that each such GNN corresponds to a Datalog program (possibly with inequalities in the rule bodies) that can be computed from the GNN's definition. Monotonic max-sum GNNs can use the following aggregation function in all layers, which generalises both max and sum.

Definition 5. *For $k \in \mathbb{N}_0 \cup \{\infty\}$, a finite real multiset $S \in \mathcal{F}(\mathbb{R})$, and $\ell = \min(k, |S|)$, let*

$$\text{max-}k\text{-sum}(S) = \begin{cases} 0 & \text{if } \ell = 0, \\ \sum_{i=1}^{\ell} s_i & \text{where } s_1,\dots,s_\ell \text{ are the } \ell \text{ largest numbers of } S. \end{cases}$$

Each occurrence of a number is counted separately; for example, $\text{max-3-sum}(\{\!\{0,1,1,2,2,5\}\!\}) = 9$ because the

three largest numbers in S are 5 and the two occurrences of 2. Also, $\max$-1-sum is equivalent to $\max$, and $\max$-∞-sum is equivalent to sum; hence, $\max$-k-sum generalises both the $\max$ and sum aggregation functions. While the ability to sum just the k maximal elements may not be relevant in practice, it will allow us to formalise a key technical result. We next introduce monotonic max-sum GNNs.

Definition 6. *A* monotonic max-sum (Col, δ)*-GNN is a GNN of form* (2) *satisfying the following conditions:*

- *for each* $\ell \in \{1, \dots, L\}$ *and each* $c \in \mathsf{Col}$*, all elements of matrices* $\mathbf{A}_\ell$ *and* $\mathbf{B}_\ell^c$ *are nonnegative;*
- *for each* $\ell \in \{1, \dots, L\}$*, the aggregation function* agg_ℓ *is* $\max$-k_ℓ-sum *for some* $k_\ell \in \mathbb{N}_0 \cup \{\infty\}$*;*
- *the activation function* σ *is monotonically increasing and unbounded, and the range of* σ *is* $\mathbb{R}_0^+$*; and*
- *the classification function* cls *is a step function—that is, there exists a* threshold $t \in \mathbb{R}$ *such that* $\mathsf{cls}(t') = 0$ *for each* $t' < t$*, and* $\mathsf{cls}(t') = 1$ *for each* $t' \geq t$*.*

Monotonic max-sum GNNs are closely related to, but incomparable with MGNNs by Tena Cucala et al. (2022): MGNNs do not require the activation function to be unbounded, but they support only the $\max$ aggregation function in all layers. While ReLU satisfies Definition 6, neither ELU nor the sigmoid function is compatible.

In Section 4.1, we show that, in each monotonic max-sum GNN $\mathcal{N}$, one can replace each function $\max$-k_ℓ-sum where $k_\ell = \infty$ with $\max$-C_ℓ-sum for some $C_\ell \in \mathbb{N}_0$ without changing the canonical transformation induced by $\mathcal{N}$—that is, to apply a GNN to a dataset, we need to consider only a bounded number of vertices for aggregation. Number C_ℓ depends solely on $\mathcal{N}$ (i.e., it is independent of any dataset to which $\mathcal{N}$ is applied) and is called the *capacity* of layer ℓ. In Section 4.2, we use this result to show that $T_\mathcal{N}$ is equivalent to the immediate consequence operator of a Datalog program $\mathcal{P}_\mathcal{N}$ that depends only on $\mathcal{N}$. Finally, in Section 4.3, we show that the numbers C_ℓ can be computed from $\mathcal{N}$, and hence program $\mathcal{P}_\mathcal{N}$ is computable. Our objective is to show that extracting $\mathcal{P}_\mathcal{N}$ from $\mathcal{N}$ is possible in principle, but further work is needed to devise a practical procedure.

4.1 Limiting Neighbour Aggregation

Throughout the rest of Section 4, we fix a monotonic max-sum (Col, δ)-GNN $\mathcal{N}$ of form (2) and dimensions $\delta_0, \dots, \delta_L$ as specified in Section 2, and we fix $k_1, \dots, k_L$ as the numbers defining the aggregation functions of $\mathcal{N}$. We next show that each $k_\ell = \infty$ can be replaced with a natural number C_ℓ. We first introduce several auxiliary definitions.

Definition 7. *A* (Col, ℓ)-multiset family*, where* $0 \leq \ell \leq L$*, is a mapping* $\mathbf{Y}$ *that assigns to each colour* $c \in \mathsf{Col}$ *a finite multiset* $\mathbf{Y}^c$ *of vectors of dimension* δ_ℓ*.*

For each $1 \leq \ell \leq L$*, each* $1 \leq i \leq \delta_\ell$*, each vector* $\mathbf{x}$ *of dimension* $\delta_{\ell-1}$*, and each* $(\mathsf{Col}, \ell-1)$*-multiset family* $\mathbf{Y}$*, let*

$$\mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y}) = (\mathbf{A}_\ell \mathbf{x} + \sum_{c \in \mathsf{Col}} \mathbf{B}_\ell^c \max\text{-}k_\ell\text{-sum}(\mathbf{Y}^c) + \mathbf{b}_\ell)_i.$$

Sets $\mathcal{X}_{\ell,i}$ *with* $0 \leq \ell \leq L$ *and* $1 \leq i \leq \delta_\ell$ *are defined by induction on* ℓ *as follows.*

- *For each* $1 \leq i \leq \delta_0$*, let* $\mathcal{X}_{0,i} = \{0, 1\}$*.*
- *For each* $\ell \geq 1$ *and each* $1 \leq i \leq \delta_\ell$*, set* $\mathcal{X}_{\ell,i}$ *is the least set that contains* $\sigma(\mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y}))$ *for each vector* $\mathbf{x}$ *of dimension* $\delta_{\ell-1}$ *such that* $(\mathbf{x})_j \in \mathcal{X}_{\ell-1,j}$ *for each* j*, and each* $(\mathsf{Col}, \ell-1)$*-multiset family* $\mathbf{Y}$ *such that* $(\mathbf{y})_j \in \mathcal{X}_{\ell-1,j}$ *for all* $c \in \mathsf{Col}$*,* $\mathbf{y} \in \mathbf{Y}^c$*, and* j*.*

Algorithm 1 CAPACITY$(\mathcal{N})$

```
let α_L be the threshold of cls
for ℓ from L down to 1 do
    w_ℓ := the least non-zero element of A_ℓ and all B_ℓ^c
    ε_ℓ := the least non-zero number in ⋃_i X_{ℓ-1,i}
    if either w_ℓ or ε_ℓ does not exist then
        C_ℓ := C_{ℓ-1} := C_1 := 0
        return
    β_ℓ := the least natural number such that σ(β_ℓ) ≥ α_ℓ
    b_ℓ := the least element of b_ℓ
    C_ℓ := min(k_ℓ, ⌈(β_ℓ - b_ℓ)/(w_ℓ · ε_ℓ)⌉)
    α_{ℓ-1} := (β_ℓ - b_ℓ)/w_ℓ
```

Intuitively, sets $\mathcal{X}_{\ell,i}$ contain all real numbers that can occur in the i-th position of a vector labelling a vertex at layer ℓ when $\mathcal{N}$ is applied to a canonical encoding of some (Col, ℓ)-dataset. Indeed, by the base case of the definition, $\mathcal{X}_{0,i}$ contains all values that can be produced by the canonical encoding, and the inductive step considers all possible ways in which a vector in layer ℓ can be computed from vectors in layer $\ell - 1$ using propagation equation (3). In the latter case, a (Col, ℓ)-multiset family $\mathbf{Y}$ represents a collection of possible neighbour vectors, and $\mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y})$ is the argument of the activation function used to compute some $(\mathbf{v}_\ell)_i$.

Note that sets $\mathcal{X}_{\ell,i}$ are nonempty, and they can be infinite. However, Theorem 8 shows that $\mathcal{X}_{\ell,i}$ can be enumerated as a countable, monotonically increasing sequence of numbers. This is important because it shows that the notion of a least nonzero element of $\mathcal{X}_{\ell,i}$ is correctly defined. In the following, for each $\alpha \in \mathbb{R}$, let $\mathcal{X}_{\ell,i}^{>\alpha} = \{\alpha' \in \mathcal{X}_{\ell,i} \mid \alpha' > \alpha\}$.

Theorem 8. *Each set* $\mathcal{X}_{\ell,i}$ *satisfies* $\mathcal{X}_{\ell,i} \subseteq \mathbb{R}_0^+$*, and, for each* $\alpha \in \mathbb{R}$*, set* $\mathcal{X}_{\ell,i} \setminus \mathcal{X}_{\ell,i}^{>\alpha}$ *is finite.*

Theorem 8 ensures that, for each $\alpha \in \mathbb{R}$, set $\mathcal{X}_{\ell,i}^{>\alpha}$ is either empty or it contains a smallest number strictly larger than α. The proof uses the fact that the activation function σ is unbounded. We are now ready to define the capacity of $\mathcal{N}$.

Definition 9. *The* capacity *of each layer* ℓ *of* $\mathcal{N}$ *is defined in Algorithm 1. Moreover, the* capacity *of* $\mathcal{N}$ *is defined as* $C_\mathcal{N} = \max\{C_1, \dots, C_L\}$*.*

Sets $\mathcal{X}_{\ell,i}$ can be infinite, so Algorithm 1 can perhaps be better understood as inductively defining sequences of numbers α_ℓ, β_ℓ, C_ℓ and so on. However, in Section 4.3 we show that the smallest positive elements of $\mathcal{X}_{\ell,i}$ can in fact be computed, which justifies our usage of the term 'algorithm'.

Theorem 10 shows that, in each layer of ℓ, every k_ℓ that is larger than C_ℓ can be replaced by C_ℓ without affecting the result of applying $\mathcal{N}$ to any dataset.

Theorem 10. *Let $\mathcal{N}'$ be the (Col, δ)-GNN obtained from $\mathcal{N}$ by replacing k_ℓ with C_ℓ for each $1 \leq \ell \leq L$. Then, $T_{\mathcal{N}}(D) = T_{\mathcal{N}'}(D)$ for each (Col, δ)-dataset D.*

Theorem 10 can be intuitively understood as follows. Let $\mathbf{v}_{\lambda_\ell}$ and $\mathbf{v}_{\lambda'_\ell}$ be vectors labelling a vertex v in layer ℓ when $T_{\mathcal{N}}$ and $T_{\mathcal{N}'}$ are applied to some D. We prove the theorem by showing that either $(\mathbf{v}_{\lambda_\ell})_i = (\mathbf{v}_{\lambda'_\ell})_i$ or $(\mathbf{v}_{\lambda_\ell})_i > (\mathbf{v}_{\lambda'_\ell})_i \geq \alpha_\ell$ for each layer $\ell \geq \ell_{\mathsf{st}}$, where ℓ_{st} is either the layer where Algorithm 1 performs an early return (via line 7) or 0 if this does not happen. Indeed, assume that $\mathsf{cls}((\mathbf{v}_{\lambda_L})_i) = 1$ for some v. If $\mathbf{A}_L$ and all $\mathbf{B}^c_L$ contain only zeros, or if all $\mathcal{X}_{L,i}$ contain only zeros, then $L = \ell_{\mathsf{st}}$; no neighbours of v are needed so we can set all C_ℓ to 0 and the equality above holds. Otherwise, cls is a threshold function, so $(\mathbf{v}_{\lambda_L})_i \geq \alpha_L$ holds for α_L the threshold of cls, and so the argument to the activation function when computing $(\mathbf{v}_{\lambda_L})_i$ is at least β_L. Moreover, $(\mathbf{v}_{\lambda_L})_i$ is produced from $(\mathbf{v}_{\lambda_{L-1}})_i$ and the values of $(\mathbf{u}_{\lambda_{L-1}})_j$ where u ranges over the neighbours of v. If we assume that $(\mathbf{v}_{\lambda_{L-1}})_i = 0$ and that ϵ_ℓ is the least nonzero value that each u can contribute to $(\mathbf{v}_{\lambda_L})_i$, it suffices to have at least $\lceil \frac{\beta_\ell - b_\ell}{w_\ell \cdot \epsilon_\ell} \rceil$ nonzero neighbours to reach β_L. Thus, we can replace k_ℓ with this number whenever this number is smaller than k_ℓ; in contrast, if k_ℓ is smaller, we need to keep k_ℓ so that $\mathcal{N}'$ does not derive any new consequences. Finally, α_{L-1} is the value of $(\mathbf{v}_{\lambda_{L-1}})_i$ in layer $L-1$ to which we can apply analogous reasoning.

4.2 Equivalence with Datalog Programs

We next show that there exists a Datalog program $\mathcal{P}_{\mathcal{N}}$ that is equivalent to $\mathcal{N}$ in the sense described in Definition 4. Towards this goal, in Definition 11 we capture the syntactic structure of the rules in $\mathcal{P}_{\mathcal{N}}$ as rules of form (25) where φ is a *tree-like* formula for x. To understand the intuition, assume that we construct from φ a graph whose vertices are the variables in φ, and where a directed edge from x to y is introduced for each $E^c(x, y)$ in φ; then, such graph must be a directed tree. Moreover, if variable x has children y_1 and y_2 in this graph, then φ is allowed to contain inequalities of the form $y_1 \not\approx y_2$, which provide φ with a limited capability for counting; for example, formula $E^c(x, y_1) \wedge E^c(x, y_2) \wedge y_1 \not\approx y_2$ is true precisely for those values of x that are connected via the E^c predicate to at least two distinct constants. We also introduce intuitive notions of a *fan-out* (i.e., the number of children) and *depth* of a variable. Tree-like formulas contain all concepts of the $\mathcal{ALCQ}$ description logic (Baader et al. 2007) constructed from $\top$, atomic concepts, and concepts of the form $\geq nR.C$ and $C_1 \sqcap C_2$; however, our definition also allows for formulas such as $E^c(x, y_1) \wedge E^c(x, y_2) \wedge U(y_1) \wedge y_1 \not\approx y_2$, which do not correspond to the translation of $\mathcal{ALCQ}$ concepts.

Definition 11. *A* tree-like formula for a variable *is defined inductively as follows.*

- *For each variable x, formula $\top$ is tree-like for x.*
- *For each variable x and each unary predicate U, atom $U(x)$ is tree-like for x.*
- *For each variable x and all tree-like formulas φ_1 and φ_2 for x that share no variables other than x, formula $\varphi_1 \wedge \varphi_2$ is tree-like for x.*
- *For each variable x, each binary predicate E^c, and all tree-like formulas $\varphi_1, \ldots, \varphi_n$ for distinct variables $y_1, \ldots, y_n$ where no φ_i contains x and no φ_i and φ_j with $i \neq j$ share a variable, formula* (24) *is tree-like for x.*

$$\bigwedge_{i=1}^{n} \Big(E^c(x, y_i) \wedge \varphi_i \Big) \wedge \bigwedge_{1 \leq i < j \leq n} y_i \not\approx y_j \tag{24}$$

Let φ be a tree-like formula and let x be a variable in φ. The fan-out *of x in φ is the number of distinct variables y_i for which $E^c(x, y_i)$ is a conjunct of φ. The* depth *of x is the maximal n for which there exist variables $x_0, \ldots, x_n$ and predicates $E^{c_1}, \ldots, E^{c_n}$ such that $x_n = x$ and $E^{c_i}(x_{i-1}, x_i)$ is a conjunct of φ for each $1 \leq i \leq n$. The* depth *of φ is the maximum depth of a variable in φ.*

For d and f natural numbers, a tree-like formula φ is (d, f)-tree-like *if, for each variable x in φ, the depth i of x is at most d and the fan-out of x is at most $f(d - i)$. Moreover, a Datalog rule is* (d, f)-tree-like *if it is of form* (25)*, where φ is a (d, f)-tree-like formula for x.*

$$\varphi \rightarrow U(x) \tag{25}$$

Note that φ is allowed to be $\top$ in a rule of form (25); for example, $\top \rightarrow U(x)$ is a valid $(0, 0)$-tree-like rule. As explained in Section 2, when applied to a dataset D, such a rule derives $U(t)$ for each term t occurring in D.

Now let $\delta_{\mathcal{N}} = \max(\delta_0, \ldots, \delta_L)$. To construct $\mathcal{P}_{\mathcal{N}}$, we proceed as follows: we compute $f = |\mathsf{Col}| \cdot \delta_{\mathcal{N}} \cdot C_{\mathcal{N}}$, we enumerate all (L, f)-tree-like rules (up to variable renaming), and we add to $\mathcal{P}_{\mathcal{N}}$ each such rule that is captured by $\mathcal{N}$. Lemma 12 shows that this latter test can, at least in principle, be operationalised. In particular, to test whether a rule $\varphi \rightarrow U(x)$ with n variables is captured by $\mathcal{N}$, we consider each possible dataset D obtained from the atoms of φ by substituting the variables with up to n distinct constants, and we check whether applying $\mathcal{N}$ to D derives the analogously instantiated rule head; if this is the case for all such D, then the rule is captured by $\mathcal{N}$. Tena Cucala et al. (2022) used a similar test for MGNNs, but their approach was simpler since it did not need to support inequalities. Theorem 13 then shows that program $\mathcal{P}_{\mathcal{N}}$ is indeed equivalent to $\mathcal{N}$.

Lemma 12. *Let r be a constant-free Datalog rule with head H, let V be the set of variables in r, and let A be the set of body atoms of r. Then, $\mathcal{N}$ captures r if and only if $H\nu \in T_{\mathcal{N}}(A\nu)$ for each substitution $\nu : V \rightarrow S$ such that $H\nu \in T_r(A\nu)$, where S is a set of $|V|$ distinct constants.*

Theorem 13. *Let $\mathcal{P}_{\mathcal{N}}$ be the Datalog program containing, up to variable renaming, each $(L, |\mathsf{Col}| \cdot \delta_{\mathcal{N}} \cdot C_{\mathcal{N}})$-tree-like rule captured by $\mathcal{N}$, where $\delta_{\mathcal{N}} = \max(\delta_0, \ldots, \delta_L)$. Then, $\mathcal{N}$ and $\mathcal{P}_{\mathcal{N}}$ are equivalent.*

To understand this result intuitively, assume that $\mathcal{N}$ is applied to a dataset D. The fact that all rules of $\mathcal{P}_{\mathcal{N}}$ are captured by $\mathcal{N}$ clearly implies $T_{\mathcal{P}_{\mathcal{N}}}(D) \subseteq T_{\mathcal{N}}(D)$. Furthermore, by equation (3), the value of $(\mathbf{v}_L)_i$ for some i is computed from the values of $(\mathbf{v}_{L-1})_i$ and $(\mathbf{u}_{L-1})_j$ for $k \leq C_L$ distinct neighbours u of v per colour and position; but then, if t and s are terms represented by v and u, respectively, the canonical encoding ensures $E^c(t, s) \in D$ for some $c \in \mathsf{Col}$.

Also, $(\mathbf{u}_{L-1})_j$ are computed using the neighbours of u and so on. Hence, each term w in D that can possibly influence $\mathbf{v}_L$ must be connected in D to t by at most L such facts, so all relevant neighbours of t can be selected by a (d, f)-tree-like formula. The inequalities can be used to check for the existence of at least k distinct neighbours of t in D. Now let D' be the subset of D containing precisely the facts that contribute to the value of $(\mathbf{v}_L)_i$. We can unfold D' into another tree-like dataset D'' that corresponds to the body of an instantiated tree-like rule r. Since the elements of all $\mathbf{A}_\ell$ and $\mathbf{B}^c_\ell$ are nonnegative, applying $\mathcal{N}$ to D and D'' derives the same value for $\mathsf{cls}((\mathbf{v}_L)_i)$. If this value is 1, then applying the rule r to D produces the same fact as $\mathcal{N}$. Furthermore, by definition, $\mathcal{N}$ captures r and so $r \in \mathcal{P}_\mathcal{N}$. Thus, $T_{\mathcal{P}_\mathcal{N}}(D)$ contains all facts derived by $\mathcal{N}$ on D.

4.3 Enumerating Sets $\mathcal{X}_{\ell,i}$

The results we presented thus far show that program $\mathcal{P}_\mathcal{N}$ exists, but it is not yet clear that $\mathcal{P}_\mathcal{N}$ is computable: the definition of C_ℓ in Algorithm 1 uses sets $\mathcal{X}_{\ell,i}$, which can be infinite. We next show that each $\mathcal{X}_{\ell,i}$ can be enumerated algorithmically using function $\mathsf{Next}(\ell, i, \alpha)$ from Algorithm 2 as follows: for α a special symbol $\triangleright$, function $\mathsf{Next}(\ell, i, \triangleright)$ returns the smallest element of $\mathcal{X}_{\ell,i}$; moreover, for $\alpha \in \mathbb{R}$, function $\mathsf{Next}(\ell, i, \alpha)$ returns the smallest element of $\mathcal{X}^{>\alpha}_{\ell,i}$ if $\mathcal{X}^{>\alpha}_{\ell,i} \neq \emptyset$, or $\triangleleft$ otherwise. For example, $\mathsf{Next}(\ell, i, 0)$ returns the smallest nonzero element of $\mathcal{X}_{\ell,i}$, if one exists.

In the presentation of Algorithm 2, we use the following notation: for $\mathbf{x}$ a vector, j an index, and v a real number, $\mathbf{x}[j \leftarrow v]$ is the vector obtained from $\mathbf{x}$ by replacing its j-th component with v. The algorithm is based on the observation that, since $\mathbf{A}_\ell$ and $\mathbf{B}^c_\ell$ contain only nonnegative elements, and the activation function is monotonically increasing, we can enumerate the values computed by equation (3) in some $\mathbf{v}_\ell$ in a monotonically increasing fashion. To achieve this, the algorithm maintains a *frontier* F of triples $\langle \mathbf{x}, \mathbf{Y}, z\rangle$, each describing one way to compute a value of $(\mathbf{v}_\ell)_i$: vector $\mathbf{x}$ reflects the values of $(\mathbf{v}_{\ell-1})_i$, the $(\mathsf{Col}, \ell-1)$-multiset family $\mathbf{Y}$ describes multisets $\mathbf{Y}^c$ reflecting the values of $(\mathbf{u}_{\ell-1})_i$, and z is $\mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y})$—that is, the argument to the activation function when computing $(\mathbf{v}_\ell)_i$. The starting point for the exploration (line 8) is provided by $\mathsf{Start}(\ell)$, which returns $\mathbf{v}_\ell$ for a vertex v with no neighbours. To enumerate all candidate values for $(\mathbf{v}_\ell)_i$ in an increasing order, the algorithm selects a triple in the frontier with the smallest z (line 10), and considers ways to modify $\mathbf{x}$ or $\mathbf{Y}$ that increase z; each such combination is added to the frontier (lines 14, 19, and 27). Modifications involve replacing some component of $\mathbf{x}$ with the next component (lines 12–14), choosing some $\mathbf{y} \in \mathbf{Y}^c$ for some $c \in \mathsf{Col}$ and replacing some component of $\mathbf{y}$ with the next component (lines 16–19), or expanding some $\mathbf{Y}^c$ with an additional vector (lines 20–27). In the latter case, if $\mathsf{Start}(\ell)$ contains just zeros, then adding $\mathsf{Start}(\ell)$ to $\mathbf{Y}^c$ is not going to change the computed value of z so the algorithm considers vectors obtained by expanding $\mathsf{Start}(\ell)$ in order to allow z to increase. This process produces values of z in an increasing order and it guarantees that $\sigma(z) \in \mathcal{X}_{\ell,i}$. If $\alpha = \triangleright$, the algorithm stops when the first such value is produced (line 7). For $\alpha \in \mathbb{R}$, Theorem 8 guarantees that set $\mathcal{X}_{\ell,i} \setminus \mathcal{X}^{>\alpha}_{\ell,i}$ is finite; since F is extended only if the value of z increases, either F eventually becomes empty or $\sigma(z)$ exceeds α so the algorithm terminates (line 11 or 28). Theorem 14 captures the formal properties of the algorithm.

Algorithm 2 $\mathsf{Next}(\ell, i, \alpha)$

```
 1: if ℓ = 0 then
 2:     if α = ▷ or α < 0 then return 0
 3:     else if α < 1 then return 1
 4:     else return ◁
 5: let Y_∅ be such that Y_∅^c = ∅ for each c ∈ Col
 6: z := Val(ℓ, i, Start(ℓ), Y_∅)
 7: if α = ▷ then return σ(z)
 8: F := {⟨Start(ℓ), Y_∅, z⟩}
 9: while F ≠ ∅ do
10:     choose and remove ⟨x, Y, z⟩ in F with least z
11:     if σ(z) > α then return σ(z)
12:     for x' ∈ Expand(ℓ, x) do
13:         z' := Val(ℓ, i, x', Y)
14:         if z' > z then add ⟨x', Y, z'⟩ to F
15:     for c ∈ Col do
16:         for y ∈ Y^c and y' ∈ Expand(ℓ, y) do
17:             Y' := Y and Y'^c := (Y'^c \ {y}) ∪ {y'}
18:             z' := Val(ℓ, i, x, Y')
19:             if z' > z then add ⟨x, Y', z'⟩ to F
20:         if Start(ℓ) contains a nonzero then
21:             V := {Start(ℓ)}
22:         else
23:             V := Expand(ℓ, Start(ℓ))
24:         for y' ∈ V do
25:             Y' := Y and Y'^c := Y'^c ∪ {y'}
26:             z' := Val(ℓ, i, x, Y')
27:             if z' > z then add ⟨x, Y', z'⟩ to F
28: return ◁

29: function Start(ℓ)
30:     return the vector x of dimension δ_{ℓ-1} where
            (x)_j = Next(ℓ − 1, j, ▷) for 1 ≤ j ≤ δ_{ℓ-1}

31: function Expand(ℓ, v)
32:     V := ∅
33:     for 1 ≤ j ≤ δ_{ℓ-1} do
34:         v' := Next(ℓ − 1, j, (v)_j)
35:         if v' ≠ ◁ then V := V ∪ {v[j ← v']}
36:     return V
```

Theorem 14. *Algorithm 2 terminates on all inputs. Moreover, for* $0 \leq \ell \leq L$ *and* $1 \leq i \leq \delta_\ell$*,*

- $\mathsf{Next}(\ell, i, \triangleright)$ *returns the smallest element of* $\mathcal{X}_{\ell,i}$*, and*
- *for each* $\alpha \in \mathbb{R}$*,* $\mathsf{Next}(\ell, i, \alpha)$ *returns* $\triangleleft$ *if* $\mathcal{X}^{>\alpha}_{\ell,i} = \emptyset$*, and otherwise it returns the smallest element of* $\mathcal{X}^{>\alpha}_{\ell,i}$*.*

The complexity of Algorithm 14 depends on the number of recursive calls to Next, which in turn depends on the matrices of $\mathcal{N}$. We leave investigating this issue to future work.

5 Limiting Aggregation to Max

In this section we study the expressivity of *monotonic max GNNs*, which follow the same restrictions as monotonic max-sum GNNs but additionally allow only for the max aggregation function. Theorem 16 shows that each such GNN corresponds to a Datalog program without inequalities. Consequently, monotonic max GNNs cannot count the connections of a constant in a dataset.

Definition 15. *A* monotonic max (Col, δ)-GNN *is a monotonic max-sum GNN that uses the* max-1-sum *aggregation function in all layers.*

Theorem 16. *For each monotonic max (Col, δ)-GNN $\mathcal{N}$ with L layers, let $\delta_{\mathcal{N}} = \max(\delta_0, \dots, \delta_L)$, and let $\mathcal{P}_{\mathcal{N}}$ be the Datalog program containing up to variable renaming each $(L, |\mathsf{Col}| \cdot \delta_{\mathcal{N}})$-tree-like rule without inequalities captured by $\mathcal{N}$. Then, $\mathcal{N}$ and $\mathcal{P}_{\mathcal{N}}$ are equivalent.*

Tena Cucala et al. (2022) presented a closely related characterisation for MGNNs, and the main difference is that we use the canonical encoding. The latter allows us to describe the target Datalog class more precisely, which in turn allows us to prove the converse: each Datalog program with only tree-like rules and without inequalities is equivalent to a monotonic max GNN.

In what follows, we fix a program $\mathcal{P}$ consisting of (d, f)-tree-like rules without inequalities. Recall that the signature of $\mathcal{P}$ consists of unary predicates $U_1, \dots, U_\delta$ and binary predicates E^c for $c \in \mathsf{Col}$. Now let $\tau_1, \dots, \tau_n$ be a sequence containing up to variable renaming each (d, f)-tree-like formula for variable x without inequalities ordered by increasing depth—that is, for all $i < j$, the depth of τ_i is less than or equal to the depth of τ_j. Each τ_i can be written as

$$\tau_i = \varphi_{i,0} \wedge \bigwedge_{k=1}^{m_i} \Big(E^{c_k}(x, y_k) \wedge \varphi_{i,k} \Big), \tag{26}$$

where $\varphi_{i,0}$ is a conjunction of unary atoms using only variable x, each $\varphi_{i,k}$ with $1 \leq k \leq m_i$ is a $(d-1, f)$-tree-like formula for y_k, and, for all $1 \leq k < k' \leq m_i$, formulas $\varphi_{i,k}$ and $\varphi_{i,k'}$ do not have variables in common. Note that formulas $\varphi_{i,k}$ can be $\top$, and that colours c_k need not be distinct.

We define $\mathcal{N}_{\mathcal{P}}$ as the monotonic max (Col, δ)-GNN of form (2) satisfying the following conditions. The number of layers is $L = d + 2$, the activation function is ReLU, and the classification function cls is the step function with threshold 1. For $1 \leq \ell < L$, dimension δ_ℓ is defined as the number of formulas in the above sequence of depth at most $\ell - 1$. The elements of $\mathbf{A}_\ell$, $\mathbf{B}^c_\ell$, and $\mathbf{b}_\ell$ are defined as follows, for $c \in \mathsf{Col}$, $1 \leq \ell \leq L$, $1 \leq i \leq \delta_\ell$, and $1 \leq j \leq \delta_{\ell-1}$.

$$(\mathbf{A}_\ell)_{i,j} = \begin{cases} 1 \text{ if} \\ \quad \bullet\ \ell = 1 \text{ and } \tau_i \text{ contains } U_j(x); \text{ or} \\ \quad \bullet\ 2 \leq \ell < L \text{ and} \\ \quad\quad - 1 \leq i \leq \delta_{\ell-1} \text{ and } i = j, \text{ or} \\ \quad\quad - \delta_{\ell-1} < i \leq \delta_\ell \text{ and } \varphi_{i,0} = \tau_j; \text{ or} \\ \quad \bullet\ \ell = L \text{ and } \mathcal{P} \text{ contains rule} \\ \quad\quad \tau_j \to U_i(x) \text{ up to variable renaming;} \\ 0 \text{ otherwise.} \end{cases}$$

$$(\mathbf{B}^c_\ell)_{i,j} = \begin{cases} 1 \text{ if } 2 \leq \ell < L \text{ and there exists } 1 \leq k \leq m_i \\ \quad \text{such that } c = c_k \text{ and } \varphi_{i,k} \text{ and } \tau_j \\ \quad \text{are equal up to variable renaming;} \\ 0 \text{ otherwise.} \end{cases}$$

$$(\mathbf{b}_\ell)_i = \begin{cases} 1 - \sum_{j=1}^{\delta_{\ell-1}} ((\mathbf{A}_\ell)_{i,j} + \sum_{c \in \mathsf{Col}} (\mathbf{B}^c_\ell)_{i,j}) & \text{if } \ell = 1, \text{ or} \\ & 1 \leq \ell < L \text{ and} \\ & \delta_{\ell-1} < i \leq \delta_\ell; \\ 0 & \text{otherwise.} \end{cases}$$

To understand the intuition behind the construction of $\mathcal{N}_{\mathcal{P}}$, assume that $\mathcal{N}_{\mathcal{P}}$ is applied to a dataset D, and consider a vector $\mathbf{v}_\ell$ labelling in layer ℓ a vertex corresponding to some term t of D. Then, the i-th component of $\mathbf{v}_\ell$ is paired with formula τ_i from the above enumeration, and it indicates whether it is possible to evaluate τ_i over D by mapping variable x to t. This is formally captured by Lemma 17. To ensure that $\mathcal{N}_{\mathcal{P}}$ and $\mathcal{P}$ are equivalent, layer L of $\mathcal{N}_{\mathcal{P}}$ simply realises a disjunction over all rules in the program.

Lemma 17. *For each (Col, δ)-dataset D, layer $1 \leq \ell < L$ of $\mathcal{N}_{\mathcal{P}}$, position $1 \leq i \leq \delta_\ell$, and term t in D, and for $\mathbf{v}_\ell$ the labelling of the vertex corresponding to t when $\mathcal{N}_{\mathcal{P}}$ is applied to the canonical encoding of D,*

- $(\mathbf{v}_\ell)_i = 1$ *if there exists a substitution ν mapping x to t such that $D \models \tau_i \nu$, and*
- $(\mathbf{v}_\ell)_i = 0$ *otherwise.*

Note that each δ_ℓ with $1 \leq \ell < L$ is determined by the number of (d, f)-tree-like formulas of depth $\ell - 1$, and that δ_{L-1} is the largest such number. We next determine an upper bound on δ_{L-1}. By Definition 11, the fan-out of a variable of depth i is at most $f(d - i)$. The number of variables of depth i is at most the number of variables of depth $i-1$ times the fan-out of each variable, which is $f^i \cdot d \dots (d - i + 1)$ and is bounded by $f^i \cdot d!$. By adding up the contribution for each depth, there are at most $f^d \cdot (d+1)!$ variables. Each variable is labelled by one of the 2^δ conjunctions of depth zero, and each non-root variable is connected by one of the $|\mathsf{Col}|$ predicates to its parent. Hence, there are at most $(|\mathsf{Col}| \cdot 2^\delta)^{f^d \cdot (d+1)!}$ tree-like formulas.

Theorem 18. *Program $\mathcal{P}$ and GNN $\mathcal{N}_{\mathcal{P}}$ are equivalent, and moreover $\delta_{L-1} \leq (|\mathsf{Col}| \cdot 2^\delta)^{f^d \cdot (d+1)!}$.*

6 Conclusion

We have shown that each monotonic max-sum GNN (i.e., a GNN that uses max and sum aggregation functions and satisfies certain properties) is equivalent to a Datalog program with inequalities in the sense that applying the GNN or a single round of the rules of the program to any dataset produces the same result. We have also sharpened this result to monotonic max GNNs and shown the converse: each tree-like Datalog program without inequalities is equivalent to a monotonic max GNN. We see many avenues for future work. First, we aim to completely characterise monotonic max-sum GNNs. Second, we intend to implement rule extraction. Third, we shall investigate the empirical performance of monotonic max-sum GNNs on tasks other than link prediction, such as node classification.

Acknowledgements

This work was supported by the SIRIUS Centre for Scalable Data Access (Research Council of Norway, project number 237889), and the EPSRC projects ConCur (EP/V050869/1), UK FIRES (EP/S019111/1), and AnaLOG (EP/P025943/1). For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison Wesley.

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. F., eds. 2007. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2nd edition.

Bader, S.; Hitzler, P.; Hölldobler, S.; and Witzel, A. 2007. A Fully Connectionist Model Generator for Covered First-Order Logic Programs. In *Proc. IJCAI*, 666–671.

Bader, S.; d'Avila Garcez, A. S.; and Hitzler, P. 2005. Computing First-Order Logic Programs by Fibring Artificial Neural Networks. In *Proc. FLAIRS*, 314–319. AAAI Press.

Barceló, P.; Kostylev, E. V.; Monet, M.; Pérez, J.; Reutter, J. L.; and Silva, J. P. 2020. The Logical Expressiveness of Graph Neural Networks. In *Proc. ICLR*.

Campero, A.; Pareja, A.; Klinger, T.; Tenenbaum, J.; and Riedel, S. 2018. Logical rule induction and theory learning using neural theorem proving. *CoRR* abs/1809.02193.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Comput. Surv.* 33(3):374–425.

Dong, H.; Mao, J.; Lin, T.; Wang, C.; Li, L.; and Zhou, D. 2019. Neural Logic Machines (Poster). In *Proc. ICLR*.

Hölldobler, S.; Kalinke, Y.; and Störr, H.-P. 1999. Approximating the Semantics of Logic Programs by Recurrent Neural Networks. *Applied Intelligence* 11(1):45–58.

Huang, X.; Orth, M. A. R.; Ceylan, İ. İ.; and Barceló, P. 2023. A theory of link prediction via relational weisfeiler-leman. *CoRR* abs/2302.02209.

Ioannidis, V. N.; Marques, A. G.; and Giannakis, G. B. 2019. A recurrent graph neural network for multi-relational data. In *Proc. ICASSP*, 8157–8161. IEEE.

Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *Proc. ICLR*.

Liu, S.; Cuenca Grau, B.; Horrocks, I.; and Kostylev, E. V. 2021. INDIGO: gnn-based inductive knowledge graph completion using pair-wise encoding. In *Proc. NeurIPS*, 2034–2045.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *Proc. AAAI*, 4602–4609. AAAI Press.

Motik, B.; Cuenca Grau, B.; Horrocks, I.; Wu, Z.; Fokoue, A.; and Lutz, C. 2012. *OWL 2 Web Ontology Language: Profiles (2nd Edition)*. World Wide Web Consortium.

Pérez, J.; Arenas, M.; and Gutierrez, C. 2009. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.* 34(3):16:1–16:45.

Pflueger, M.; Tena Cucala, D. J.; and Kostylev, E. V. 2022. GNNQ: A neuro-symbolic approach to query answering over incomplete knowledge graphs. In *Proc. ISWC*, volume 13489 of *Lecture Notes in Computer Science*, 481–497. Springer.

Qu, M.; Bengio, Y.; and Tang, J. 2019. GMNN: graph markov neural networks. In *Proc. ICML*, volume 97 of *Proceedings of Machine Learning Research*, 5241–5250.

Rocktäschel, T., and Riedel, S. 2017. End-to-end Differentiable Proving. In *Proc. NeurIPS*, 3788–3800.

Sadeghian, A.; Armandpour, M.; Ding, P.; and Wang, D. Z. 2019. DRUM: End-To-End Differentiable Rule Mining On Knowledge Graphs. In *Proc. NeurIPS*, 15321–15331.

Schlichtkrull, M. S.; Kipf, T. N.; Bloem, P.; van den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *Proc. ESWC*, volume 10843 of *LNCS*, 593–607. Springer.

Sourek, G.; Zelezný, F.; and Kuzelka, O. 2021. Beyond graph neural networks with lifted relational neural networks. *Mach. Learn.* 110(7):1695–1738.

Tena Cucala, D.; Cuenca Grau, B.; Kostylev, E. V.; and Motik, B. 2022. Explainable GNN-Based Models over Knowledge Graphs. In *Proc. ICLR*.

Tena Cucala, D.; Cuenca Grau, B.; and Motik, B. 2022. Faithful Approaches to Rule Learning. In *Proc. KR*, 484–493.

Teru, K. K.; Denis, E. G.; and Hamilton, W. L. 2020. Inductive relation prediction by subgraph reasoning. In *Proc. ICML*, volume 119 of *Proceedings of Machine Learning Research*, 9448–9457. PMLR.

Yang, Z.; Cohen, W. W.; and Salakhutdinov, R. 2016. Revisiting semi-supervised learning with graph embeddings. In *Proc. ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, 40–48.

Yang, F.; Yang, Z.; and Cohen, W. W. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In *Proc. NeurIPS*, 2319–2328.

Zhang, M., and Chen, Y. 2018. Link prediction based on graph neural networks. In *Proc. NeurIPS*, 5171–5181.

A Proofs for Section 4

Throughout this appendix, we fix a max-sum GNN $\mathcal{N}$, dimensions $\delta_0, \ldots, \delta_L$, and aggregation functions $k_1, \ldots, k_L$ as specified in Section 4.1. As in Section 4.3, for $\mathbf{x}$ a vector, j an index, and v a real number, $\mathbf{x}[j \leftarrow v]$ is the vector obtained from $\mathbf{x}$ by replacing its jth component with v.

To prove our results, we shall define a nonempty sequence $\mathcal{S}_{\ell,i}$ for each $0 \leq \ell \leq L$ and $1 \leq i \leq \delta_\ell$; intuitively, each $\mathcal{S}_{\ell,i}$ enumerates $\mathcal{X}_{\ell,i}$ in ascending order. Our definition is inductive and uses two auxiliary notions that we define next. In particular, consider an arbitrary ℓ with $0 < \ell \leq L$ and assume that $\mathcal{S}_{\ell-1,i}$ have been defined for all $1 \leq i \leq \delta_\ell$. Then, $\mathbf{s}_{\ell-1}$ is the vector of dimension $\delta_{\ell-1}$ such that $(\mathbf{s}_{\ell-1})_i$ is the first element of $\mathcal{S}_{\ell,i}$ for each $1 \leq i \leq \delta_{\ell-1}$. Moreover, a (ℓ, i)-*triple* is a triple of the form $\langle \mathbf{x}, \mathbf{Y}, z \rangle$ whose components satisfy the following conditions:

- $\mathbf{x}$ is a vector of dimension $\delta_{\ell-1}$ such that $(\mathbf{x})_j \in \mathcal{S}_{\ell-1,j}$ holds for all $1 \leq j \leq \delta_{\ell-1}$;
- $\mathbf{Y}$ is a $(\mathsf{Col}, \ell-1)$-multiset family such that $(\mathbf{y})_j \in \mathcal{S}_{\ell-1,j}$ holds for all $c \in \mathsf{Col}$, $\mathbf{y} \in \mathbf{Y}^c$, and $1 \leq j \leq \delta_{\ell-1}$; and
- $z = \mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y})$.

An (ℓ, i)-triple $\langle \mathbf{x}_2, \mathbf{Y}_2, z_2 \rangle$ is a successor of an (ℓ, i)-triple $\langle \mathbf{x}_1, \mathbf{Y}_1, z_1 \rangle$ if exactly one of the following conditions holds:

- $\mathbf{Y}_1 = \mathbf{Y}_2$ and $\mathbf{x}_2 = \mathbf{x}_1[j \leftarrow x']$ for some $1 \leq j \leq \delta_{\ell-1}$ and x' the element that succeeds $(\mathbf{x})_j$ in $\mathcal{S}_{\ell-1,j}$; or
- $\mathbf{x}_2 = \mathbf{x}_1$ and there exist a colour $c \in \mathsf{Col}$, vector $\mathbf{y} \in \mathbf{Y}_1^c$, and index $1 \leq j \leq \delta_{\ell-1}$ such that $\mathbf{Y}_2^{c'} = \mathbf{Y}_1^{c'}$ for each colour $c' \in \mathsf{Col} \setminus \{c\}$, and $\mathbf{Y}_2^c = (\mathbf{Y}_1^c \setminus \{\mathbf{y}\}) \cup \{\mathbf{y}[j \leftarrow y']\}$ where y' is the element that succeeds $(\mathbf{y})_j$ in $\mathcal{S}_{\ell-1,j}$; or
- $\mathbf{x}_2 = \mathbf{x}_1$ and there exist a colour $c \in \mathsf{Col}$ and index $1 \leq j \leq \delta_{\ell-1}$ such that $\mathbf{Y}_2^{c'} = \mathbf{Y}_1^{c'}$ for each colur $c' \in \mathsf{Col} \setminus \{c\}$, and $\mathbf{Y}_2^c = \mathbf{Y}_1^c \cup \{\mathbf{s}_{\ell-1}[j \leftarrow y']\}$ where y' is the first positive element of $\mathcal{S}_{\ell-1,j}$.

We are now ready to define sequences $\mathcal{S}_{\ell,i}$ for all $0 \leq \ell \leq L$ and $1 \leq i \leq \delta_\ell$.

- For the base case $\ell = 0$, let $\mathcal{S}_{0,i} = (0, 1)$ for each $1 \leq i \leq \delta_0$.
- For the inductive step, assume that $\mathcal{S}_{\ell-1,i}$ has been defined for each $1 \leq i \leq \delta_{\ell-1}$, and consider arbitrary $1 \leq i \leq \delta_\ell$. To define $\mathcal{S}_{\ell,i}$, we first define an auxiliary sequence $\mathcal{F}_{\ell,i}$ of finite sets of (ℓ, i)-triples as follows.
  - For the base case, the first element of $\mathcal{F}_{\ell,i}$ is $f_0 = \{\langle \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset, z_0 \rangle\}$, where $\mathbf{Y}_\emptyset$ is such that $\mathbf{Y}_\emptyset^c = \emptyset$ for each $c \in \mathsf{Col}$ and $z_0 = \sigma(\mathsf{Val}(\ell, i, \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset))$.
  - For the inductive step, assuming that f_{n-1} with $n > 0$ has been defined and is not empty, let

$$\begin{aligned} f_n = \; & \{\langle \mathbf{x}, \mathbf{Y}, z \rangle \in f_{n-1} \mid z > \min(f_{n-1})\} \cup \\ & \{\langle \mathbf{x}, \mathbf{Y}, z \rangle \mid z > \min(f_{n-1}) \text{ and } \langle \mathbf{x}, \mathbf{Y}, z \rangle \text{ is a successor of some } \langle \mathbf{x}', \mathbf{Y}', \min(f_{n-1}) \rangle \in f_{n-1}\}, \end{aligned}$$

    where $\min(f_n)$ is the minimum number appearing in the third position of an (ℓ, i)-triple in f_{n-1}; such a number always exists since f_{n-1} is never empty and it contains a finite number of triples. Then, $\mathcal{S}_{\ell,i}$ is the sequence of real numbers whose n-th element is $\sigma(\min(f_n))$ if f_n is defined and $f_n \neq \emptyset$. Since f_0 is always defined and not empty, $\mathcal{S}_{\ell,i}$ is not empty.

The following lemma shows that sequences $\mathcal{S}_{\ell,i}$ capture our intuition mentioned above.

Lemma A.1. *For each $1 \leq \ell \leq L$ and each $1 \leq i \leq \delta_\ell$, sequence $\mathcal{S}_{\ell,i}$ satisfies the following conditions:*

(S1) each element of $\mathcal{S}_{\ell,i}$ is nonnegative;

(S2) $\mathcal{S}_{\ell,i}$ is strictly monotonically increasing;

(S3) $\mathcal{S}_{\ell,i}$ is either finite or it converges to ∞; and

(S4) the set of elements in $\mathcal{S}_{\ell,i}$ is $\mathcal{X}_{\ell,i}$.

Proof. We prove all four conditions by induction over ℓ. For the base case $\ell = 0$, sequence $\mathcal{S}_{0,i}$ is by definition is equal to $(0, 1)$ for each $1 \leq i \leq \delta_\ell$, so conditions (S1)–(S4) hold trivially. Now consider arbitrary $1 \leq \ell \leq L$ such that each $\mathcal{S}_{\ell-1,j}$ with $1 \leq j \leq \delta_{\ell-1}$ satisfies conditions (S1)–(S4), and consider arbitrary $1 \leq i \leq \delta_\ell$.

Condition (S1) follows straightforwardly from the fact that each element of $\mathcal{S}_{\ell,i}$ for $\ell \geq 1$ is the image of σ for some real number z, and $\sigma(z) \geq 0$ for all $z \in \mathbb{R}$, since the range of σ is $\mathbb{R}_0^+$. Condition (S2) follows from the fact that, for each $n \in \mathbb{N}$, each triple $\langle \mathbf{x}, \mathbf{Y}, z \rangle \in f_n \setminus f_{n-1}$ satisfies $z > \min(f_{n-1})$, and so $\min(f_n) > \min(f_{n-1})$ holds.

We prove Condition (S3) by contradiction—that is, we assume that $\mathcal{S}_{\ell,i}$ is infinite, and that there exists some $\bar{\alpha} \in \mathbb{R}$ such that each element of $\mathcal{S}_{\ell,i}$ is smaller than $\bar{\alpha}$.

Consider an arbitrary element α in $\mathcal{S}_{\ell,i}$. By the definition of $\mathcal{S}_{\ell,i}$, there exists a triple $\langle \mathbf{x}, \mathbf{Y}, z \rangle$ such that $z = \mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y})$, $\sigma(z) = \alpha$, and, for each $c \in \mathsf{Col}$ and $\mathbf{y} \in \mathbf{Y}^c$, vector $\mathbf{y}$ contains a nonzero element. Let $\bar{\beta}$ be the smallest natural number such that $\sigma(\bar{\beta}) \geq \bar{\alpha}$; such $\bar{\beta}$ exists since σ is unbounded. For each $1 \leq j \leq \delta_{\ell-1}$ and each $c \in \mathsf{Col}$, let w_j, α_j, and $n_{j,c}$ be as follows:

$$w_j = \min\{(\mathbf{A}_\ell)_{i,j}\} \cup \{(\mathbf{B}_\ell^c)_{i,j} \mid c \in \mathsf{Col}\}; \tag{27}$$

$$\alpha_j = \begin{cases} \frac{\bar{\beta} - (\mathbf{b}_\ell)_i}{w_\ell} & \text{if } w_\ell \neq 0, \\ \text{undefined} & \text{otherwise;} \end{cases} \tag{28}$$

$$\epsilon_j = \begin{cases} \text{the first positive value of } \mathcal{S}_{\ell-1,j} & \text{if such a value exists,} \\ \text{undefined} & \text{otherwise;} \end{cases} \tag{29}$$

$$n_{j,c} = \begin{cases} \lceil \frac{\bar{\beta} - (\mathbf{b}_\ell)_i}{(\mathbf{B}^c_\ell)_{i,j} \cdot \epsilon_j} \rceil & \text{if } (\mathbf{B}^c_\ell)_{i,j} \neq 0 \text{ and } \epsilon_j \text{ is defined;} \\ 0 & \text{otherwise.} \end{cases} \tag{30}$$

We next show the following properties:

1. if $(\mathbf{A}_\ell)_{i,j} > 0$, then $(\mathbf{x})_j < \alpha_j$;
2. for each $c \in \mathsf{Col}$, if $(\mathbf{B}^c_\ell)_{i,j} > 0$, then $(\mathbf{y})_j < \alpha_j$ for each $\mathbf{y} \in \mathbf{Y}^c$;
3. for each $c \in \mathsf{Col}$, if $(\mathbf{B}^c_\ell)_{i,j} > 0$, then there exist fewer than $n_{j,c}$ elements in $\mathbf{Y}^c$ whose j-th element is not zero.

To see the first property, note that if $(\mathbf{x})_j \geq \alpha_j$, then condition (S1) of the inductive hypothesis ensures that all elements of $\mathbf{x}$ and vectors in $\mathbf{Y}$ are nonnegative; since the weights of $\mathbf{A}_\ell$ and $\mathbf{B}^c_\ell$ are also nonnegative, we have

$$\alpha = \sigma\Big(\mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y})\Big) \geq \sigma\Big(w_j \alpha_j + (\mathbf{b}_\ell)_i\Big) = \sigma(\bar{\beta}) \geq \bar{\alpha},$$

which contradicts our assumption that all elements of $\mathcal{S}_{\ell,i}$ are smaller than $\bar{\alpha}$. The second property follows analogously. To see the third property, assume that there exist at least $n_{j,c}$ vectors $\mathbf{y}$ in $\mathbf{Y}^c$ such that $(\mathbf{y})_j > 0$. Then,

$$\alpha = \sigma\Big(\mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y})\Big) \geq \sigma\Big((\mathbf{B}^c_\ell)_{i,j} \sum_{\mathbf{y} \in \mathbf{Y}^c} (\mathbf{y})_j + (\mathbf{b}_\ell)_i\Big) \geq \sigma\Big((\mathbf{B}^c_\ell)_{i,j} \cdot n_j \cdot \epsilon_j + (\mathbf{b}_\ell)_i\Big) \geq \sigma(\bar{\beta}) \geq \bar{\alpha},$$

which again contradicts our assumption that all elements of $\mathcal{S}_{\ell,i}$ are smaller than $\bar{\alpha}$.

By conditions (S2) and (S3) of the inductive hypothesis, each $\mathcal{S}_{\ell-1,j}$ is countable, monotonically increasing, and either finite or converges to infinity; hence, set $\{s \in \mathcal{S}_{\ell-1,j} \mid s \leq \bar{\alpha}\}$ is finite. Thus, by the three properties shown above, if $(\mathbf{A}_\ell)_{i,j} > 0$, then $(\mathbf{x})_j$ can only take finitely many values; similarly, for all $c \in \mathsf{Col}$ and $\mathbf{y} \in \mathbf{Y}^c$, if $(\mathbf{B}^c_\ell)_{i,j} > 0$, then, $(\mathbf{y})_j$ can only take finitely many values. Notice also that each $\mathbf{Y}^c$ cannot have infinitely many elements, since it does not contain any vector where all elements are 0, and, for each $1 \leq j \leq \delta_{\ell-1}$, there exist fewer than $n_{j,c}$ elements in $\mathbf{Y}^c$ whose j-th component's value is positive, and none for which it is negative. Hence, there are only finitely many values that $\sigma(z)$ can take. Thus, $\mathcal{S}_{\ell,i}$ is finite, which contradicts our initial assumption.

Finally, we show condition (S4). To this end, we first show that all elements of $\mathcal{S}_{\ell,i}$ are in $\mathcal{X}_{\ell,i}$. Consider an arbitrary element $\alpha \in \mathcal{S}_{\ell,i}$; by definition, there exists an (ℓ, i)-triple of the form $\langle \mathbf{x}, \mathbf{Y}, z \rangle$ such that $\sigma(z) = \alpha$. Now, for all $1 \leq j \leq \delta_{\ell-1}$, $c \in \mathsf{Col}$, and $\mathbf{y} \in \mathbf{Y}^c$, the definition of an (ℓ, i)-triple ensures $(\mathbf{x})_j \in \mathcal{S}_{\ell-1,j}$ and $(\mathbf{y})_j \in \mathcal{S}_{\ell-1,j}$; thus, our inductive hypothesis implies $(\mathbf{x})_j \in \mathcal{X}_{\ell-1,j}$ and $(\mathbf{y})_j \in \mathcal{X}_{\ell-1,j}$. But then, the definition of an (ℓ, i)-triple ensures $z = \mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y})$, and the definition of $\mathcal{X}_{\ell,i}$ ensures $\sigma(z) \in \mathcal{X}_{\ell,i}$, as required.

To prove that each element of $\mathcal{X}_{\ell,i}$ appears in $\mathcal{S}_{\ell,i}$, consider arbitrary $\alpha \in \mathcal{X}_{\ell,i}$. By Definition 7, there exists a vector $\mathbf{x}_\alpha$ of dimension $\delta_{\ell-1}$ where $(\mathbf{x}_\alpha)_j \in \mathcal{X}_{\ell-1,j}$ for each $1 \leq j \leq \delta_{\ell-1}$, and there also exists a $(\mathsf{Col}, \ell-1)$-multiset family $\mathbf{Y}_\alpha$ such that $(\mathbf{y})_j \in \mathcal{X}_{\ell-1,j}$ holds for each $c \in \mathsf{Col}$, each $\mathbf{y} \in \mathbf{Y}^c_\alpha$, and each $1 \leq j \leq \delta_{\ell-1}$; moreover, $\sigma(z_\alpha) = \alpha$ for $z_\alpha = \mathsf{Val}(\ell, i, \mathbf{x}_\alpha, \mathbf{Y}_\alpha)$. By induction hypothesis, all elements in $\mathcal{X}_{\ell-1,j}$ are in $\mathcal{S}_{\ell-1,j}$. Hence, there exists at least one finite sequence $\langle \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset, \mathsf{Val}(\ell, i, \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset) \rangle = t_0, \ldots, t_K = \langle \mathbf{x}_\alpha, \mathbf{Y}_\alpha, z_\alpha \rangle$ such that t_n is a successor of t_{n-1} for each $1 \leq n \leq K$. Indeed, each multiset $\mathbf{Y}^c_\alpha$ is finite and, starting from t_0, we can reach t_K by, in each step, changing some vector component to the next element in $\mathcal{S}_{\ell-1,j}$ or adding a new vector to some multiset of the $(\mathsf{Col}, \ell-1)$-multiset family. We now show by induction over $0 \leq n \leq K$ the following statement ($*$): for each $t_n = \langle \mathbf{x}_n, \mathbf{Y}_n, z_n \rangle$ in this sequence, some element of $\mathcal{F}_{\ell,i}$ contains a (ℓ, i)-triple $\langle \mathbf{x}, \mathbf{Y}, z \rangle$, called a *witness* of t_n, such that

- if $(\mathbf{A}_\ell)_{i,j} > 0$, then $(\mathbf{x}_n)_j = (\mathbf{x})_j$,
- for each colour $c \in \mathsf{Col}$ and each index $1 \leq j \leq \delta_{\ell-1}$, if $(\mathbf{B}^c_\ell)_{i,j} > 0$, then multisets $\{\!\{ (\mathbf{y})_j \mid \mathbf{y} \in \mathbf{Y}^c_n \text{ and } (\mathbf{y})_j > 0 \}\!\}$ and $\{\!\{ (\mathbf{y})_j \mid \mathbf{y} \in \mathbf{Y}^c \text{ and } (\mathbf{y})_j > 0 \}\!\}$ are equal.

Observe that these properties imply $z = z_n$. For the base case, $\langle \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset, \mathsf{Val}(\ell, i, \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset) \rangle \in f_0$ holds by definition, so t_0 is its own witness in $\mathcal{F}_{\ell,i}$. For the induction step, we assume that $t_{n-1} = \langle \mathbf{x}_{n-1}, \mathbf{Y}_{n-1}, z_{n-1} \rangle$ with $0 < n \leq K$ has a witness in $\mathcal{F}_{\ell,i}$, and we show that let $t_n = \langle \mathbf{x}_n, \mathbf{Y}_n, z_n \rangle$ then has a witness in $\mathcal{F}_{\ell,i}$ as well. Let $t = \langle \mathbf{x}, \mathbf{Y}, z \rangle$ be a witness of t_{n-1} in $\mathcal{F}_{\ell,i}$. We first show that there exists $m \in \mathbb{N}_0$ such that f_m is defined, f_{m-1} contains t, but f_m does not contain t. If $\mathcal{F}_{\ell,i}$ is finite, the last element of $\mathcal{F}_{\ell,i}$ is empty so the claim clearly holds. Thus, assume that $\mathcal{F}_{\ell,i}$ is infinite. For the sake of a contradiction, assume that there exists some $m' \in \mathbb{N}_0$ such that t appears in all elements of $\mathcal{F}_{\ell,i}$ after $f_{m'}$. By the definition of $\mathcal{S}_{\ell,i}$, this implies that the elements of $\mathcal{S}_{\ell,i}$ after $s_{m'}$ form an infinite sequence that is strictly monotonic and whose values are always smaller than $\sigma(z)$; however, this contradicts condition (S3). Thus, there exists $m \in \mathbb{N}_0$ such that f_{m-1} contains t, but f_m does not. The definition of $\mathcal{S}_{\ell,i}$ ensures that the $m-1$-th element of $\mathcal{S}_{\ell,i}$ is precisely $\sigma(z_{n-1})$. If $z_n = z_{n-1}$, then the change from t_{n-1} to t_n can only take place in either the j-th component of $\mathbf{x}_{n-1}$ for j such that $(\mathbf{A}_\ell)_{i,j} = 0$, or in the j-th component of some $\mathbf{y} \in \mathbf{Y}^c_{n-1}$ for some $c \in \mathsf{Col}$ with $(\mathbf{B}^c_\ell)_{i,j} = 0$, so the statement holds since $\langle \mathbf{x}, \mathbf{Y}, z \rangle$ is a witness for t_n. If $z_n > z_{n-1}$,

then the change from t_{n-1} to t_n can only take place in either the j-th component of $\mathbf{x}_{n-1}$ for j such that $(\mathbf{A}_\ell)_{i,j} > 0$, or in the j-th component of some $\mathbf{y} \in \mathbf{Y}^c_{n-1}$ for some $c \in \mathsf{Col}$ with $(\mathbf{B}^c_\ell)_{i,j} > 0$, or by adding a new vector to some $\mathbf{Y}^c_{n-1}$ with the smallest positive value from $\mathcal{S}_{\ell-1,j}$ in the j-th component, for some j such that $(\mathbf{B}^c_\ell)_{i,j} > 0$. By the definition of a witness, both t_{n-1} and t agree on the components of vectors where the change from t_{n-1} to t_n takes place, and so the same change can be applied to the witness $\langle \mathbf{x}, \mathbf{Y}, z \rangle$, leading to a triple $t' = \langle \mathbf{x}', \mathbf{Y}', z' \rangle$ such that $z' = z_n$ and by definition of $\mathcal{F}_{\ell,i}$, t' must appear in f_{m+1}. Thus, t' is clearly a witness of t_n in $\mathcal{F}_{\ell,i}$. This concludes the proof of $(*)$.

Now, $(*)$ ensures that $\langle \mathbf{x}_\alpha, \mathbf{Y}_\alpha, z_\alpha \rangle$ has a witness in $\mathcal{F}_{\ell,i}$, and as we have already shown, there exists some element f_m of $\mathcal{F}_{\ell,i}$ such that this triple appears in f_m but not in f_{m+1}. But then, the definition of $\mathcal{S}_{\ell,i}$ ensures that $\sigma(z_\alpha)$ is the m-th element of $\mathcal{S}_{\ell,i}$; since $\sigma(z_\alpha) = \alpha$, numner α appears in $\mathcal{S}_{\ell,i}$, as desired. □

Theorem 8. *Each set $\mathcal{X}_{\ell,i}$ satisfies $\mathcal{X}_{\ell,i} \subseteq \mathbb{R}^+_0$, and, for each $\alpha \in \mathbb{R}$, set $\mathcal{X}_{\ell,i} \setminus \mathcal{X}^{>\alpha}_{\ell,i}$ is finite.*

Proof. By condition (S4) of Lemma A.1, for each $1 \leq \ell \leq L$ and each $1 \leq i \leq \delta_\ell$, the elements of $\mathcal{X}_{\ell,i}$ are precisely the elements of the sequence $\mathcal{S}_{\ell,i}$. By condition (S1), all elements in $\mathcal{S}_{\ell,i}$ are nonnegative, so $\mathcal{X}_{\ell,i} \subseteq \mathbb{R}^+_0$ holds. Moreover, assume that set $\mathcal{X}_{\ell,i} \setminus \mathcal{X}^{>\alpha}_{\ell,i}$ is infinite; then, by condition (S4) of Lemma A.1, set $\mathcal{S}_{\ell,i}$ contains infinitely many numbers that are smaller or equal than α. However, by condition (S2) ensures that $\mathcal{S}_{\ell,i}$ is strictly monotonically increasing, and so α is an upper bound of the sequence. This, in turn, contradicts condition (S3). Consequently, set $\mathcal{X}_{\ell,i} \setminus \mathcal{X}^{>\alpha}_{\ell,i}$ is finite. □

Lemma A.2. *For each (Col, δ)-dataset D, each $0 \leq \ell \leq L$, each vector $\mathbf{v}_\ell$ labelling a vertex when $\mathcal{N}$ is applied to D, and each $1 \leq i \leq \delta_\ell$, it holds that $(\mathbf{v}_\ell)_i \in \mathcal{X}_{\ell,i}$.*

Proof. The proof is by a straightforward induction on $0 \leq \ell \leq L$. For the base case, Definition 2 ensures $(\mathbf{v}_0)_i \in \{0, 1\} = \mathcal{X}_{0,i}$ for each i. For the induction step, consider some $1 \leq \ell \leq L$ and $1 \leq i \leq \delta_\ell$ and notice that the value of $(\mathbf{v}_\ell)_i$ is given by expression (31). Consider the triple $\langle \mathbf{x}, \mathbf{Y}, z \rangle$ where $\mathbf{x} = \mathbf{v}_{\ell-1}$, $\mathbf{Y}$ is the multiset family such that, for each $c \in \mathsf{Col}$, $\mathbf{Y}^c$ is the multiset $\{\!\{ \mathbf{u}_\ell \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}$, and $z = \mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y})$. By the inductive hypothesis, $(\mathbf{x})_j \in \mathcal{X}_{\ell-1,j}$, and $(\mathbf{y})_j \in \mathcal{X}_{\ell-1,j}$ for each $c \in \mathsf{Col}$ and each $\mathbf{y} \in \mathbf{Y}^c$. Finally, by comparing (31) with the definition of $\mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y})$ in Definition 7, we can see that $z = (\mathbf{v}_\ell)_i$. By the definition of $\mathcal{X}_{\ell,i}$, then $z \in \mathcal{X}_{\ell,i}$, and thus $(\mathbf{v}_\ell)_i \in \mathcal{X}_{\ell,i}$, as desired. □

Theorem 10. *Let $\mathcal{N}'$ be the (Col, δ)-GNN obtained from $\mathcal{N}$ by replacing k_ℓ with C_ℓ for each $1 \leq \ell \leq L$. Then, $T_{\mathcal{N}}(D) = T_{\mathcal{N}'}(D)$ for each (Col, δ)-dataset D.*

Proof. Consider an arbitrary (Col, δ)-dataset D and let $\mathcal{G} = \langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda \rangle$ be the canonical encoding of D. Let $\lambda_0, \ldots, \lambda_L$ and $\lambda'_0, \ldots, \lambda'_L$ be the functions labelling the vertices of $\mathcal{G}$ induced by applying $\mathcal{N}$ and $\mathcal{N}'$ to D, respectively.

We first prove by induction on $0 \leq \ell \leq L$ that for all $v \in \mathcal{V}$ and $1 \leq i \leq \delta_\ell$, it holds that $(\mathbf{v}_{\lambda_\ell})_i \geq (\mathbf{v}_{\lambda'_\ell})_i$. The base case holds trivially since $\lambda_0 = \lambda'_0$. For the induction step, consider $1 \leq \ell \leq L$, $v \in \mathcal{V}$, and $1 \leq i \leq \delta_\ell$, and suppose that both claims hold for $\ell - 1$. The formulas for $(\mathbf{v}_{\lambda_\ell})_i$ and $(\mathbf{v}_{\lambda'_\ell})_i$ are given by equations (31) and (32).

$$(\mathbf{v}_{\lambda_\ell})_i = \sigma\left(\sum_{j=1}^{\delta_{\ell-1}} (\mathbf{A}_\ell)_{i,j} (\mathbf{v}_{\lambda_{\ell-1}})_j + \sum_{c \in \mathsf{Col}} \sum_{j=1}^{\delta_{\ell-1}} (\mathbf{B}^\ell_c)_{i,j} \max\text{-}k_\ell\text{-sum}(\{\!\{ (\mathbf{u}_{\lambda_\ell})_j \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}) + (\mathbf{b}_\ell)_i\right) \tag{31}$$

$$(\mathbf{v}_{\lambda'_\ell})_i = \sigma\left(\sum_{j=1}^{\delta_{\ell-1}} (\mathbf{A}_\ell)_{i,j} (\mathbf{v}_{\lambda'_{\ell-1}})_j + \sum_{c \in \mathsf{Col}} \sum_{j=1}^{\delta_{\ell-1}} (\mathbf{B}^\ell_c)_{i,j} \max\text{-}C_\ell\text{-sum}(\{\!\{ (\mathbf{u}_{\lambda'_\ell})_j \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}) + (\mathbf{b}_\ell)_i\right) \tag{32}$$

In both equations, all summands except $(\mathbf{b}_\ell)_i$ are nonnegative: the weights of $\mathcal{N}$ and $\mathcal{N}'$ are nonnegative by Definition 6, and the feature vectors labelling vertices of $\mathcal{V}$ are also nonnegative by Theorem 8 and Lemma A.2. Note that the inductive hypothesis ensures $(\mathbf{u}_{\lambda_{\ell-1}})_j \geq (\mathbf{u}_{\lambda'_{\ell-1}})_j$ for all $u \in \mathcal{V}$ and $1 \leq j \leq \delta_{\ell-1}$. Furthermore, Algorithm 1 ensures that $C_\ell \leq k_\ell$. Since the weights of $\mathbf{A}_\ell$ and each $\mathbf{B}^c_\ell$ are nonnegative, subtracting (32) from (31) yields a positive value, and so $(\mathbf{v}_{\lambda_\ell})_i \geq (\mathbf{v}_{\lambda'_\ell})_i$. This concludes the proof by induction.

Now let ℓ_{st} be the largest $1 \leq \ell \leq L$ such that either all elements of $\mathbf{A}_\ell$ and $\mathbf{B}^c_\ell$ for each $c \in \mathsf{Col}$ are 0, or $\mathcal{X}_{\ell-1,j} = \{0\}$ for each $1 \leq j \leq \delta_{\ell-1}$; if such ℓ does not exist, let $\ell_{\mathsf{st}} = 0$. To complete the proof of the theorem, we prove by induction on $\ell_{\mathsf{st}} \leq \ell \leq L$ that, for all $v \in \mathcal{V}$ and $1 \leq i \leq \delta_\ell$, exactly one of the following two properties holds:

- $(\mathbf{v}_{\lambda_\ell})_i = (\mathbf{v}_{\lambda'_\ell})_i$ or
- $(\mathbf{v}_{\lambda_\ell})_i > (\mathbf{v}_{\lambda'_\ell})_i \geq \alpha_\ell$.

For the base case, if $\ell_{\mathsf{st}} = 0$, then the first property holds trivially since $\lambda_0 = \lambda'_0$. If $\ell_{\mathsf{st}} > 0$, consider an arbitrary $1 \leq i \leq \delta_\ell$. We have two possibilities. First, if all elements of $\mathbf{A}_{\ell_{\mathsf{st}}}$ and $\mathbf{B}^c_{\ell_{\mathsf{st}}}$ for each $c \in \mathsf{Col}$ are all zero, equations (31) and (32) and the fact that the matrices of $\mathcal{N}$ and $\mathcal{N}'$ are the same ensure that $(\mathbf{v}_{\lambda_{\ell_{\mathsf{st}}}})_i = (\mathbf{v}_{\lambda'_{\ell_{\mathsf{st}}}})_i = \sigma((\mathbf{b}_{\ell_{\mathsf{st}}})_i)$. Second, if $\mathcal{X}_{\ell-1,j} = \{0\}$ for each $1 \leq j \leq \delta_{\ell-1}$; equation (31) ensure $(\mathbf{v}_{\lambda_{\ell_{\mathsf{st}}}})_i = \sigma((\mathbf{b}_{\ell_{\mathsf{st}}})_i)$; moreover, we have shown that $(\mathbf{v}_{\lambda_{\ell_{\mathsf{st}}}})_i \geq (\mathbf{v}_{\lambda_{\ell'_{\mathsf{st}}}})_i$, and since the

elements in the sum in (32) other than $(\mathbf{b}_\ell)_i$ are not negative, we again have $(\mathbf{v}_{\lambda'_{\ell_{\mathsf{st}}}})_i = \sigma((\mathbf{b}_{\ell_{\mathsf{st}}})_i)$. Hence, $(\mathbf{v}_{\lambda_\ell})_i = (\mathbf{v}_{\lambda'_\ell})_i$ and the first property holds.

For the induction step, we consider arbitrary layer $\ell_{\mathsf{st}} < \ell \leq L$, vertex $v \in \mathcal{V}$, and position $1 \leq i \leq \delta_\ell$. We assume that $(\mathbf{v}_{\lambda_\ell})_i \neq (\mathbf{v}_{\lambda'_\ell})_i$; together with $(\mathbf{v}_{\lambda_\ell})_i \geq (\mathbf{v}_{\lambda'_\ell})_i$, this implies $(\mathbf{v}_{\lambda_\ell})_i > (\mathbf{v}_{\lambda'_\ell})_i$, so we next show $(\mathbf{v}_{\lambda'_\ell})_i \geq \alpha_\ell$. Since $\ell > \ell_{\mathsf{st}}$, Algorithm 1 defines ϵ_ℓ, β_ℓ, w_ℓ, b_ℓ, C_ℓ, α_ℓ, and $\alpha_{\ell-1}$. Furthermore, let $k'_\ell = \lceil \frac{\beta_\ell - b_\ell}{w_\ell \cdot \epsilon_\ell} \rceil$, so $C_\ell = \min(k_\ell, k'_\ell)$. We have already shown that $(\mathbf{u}_{\lambda_{\ell-1}})_j \geq (\mathbf{u}_{\lambda'_{\ell-1}})_j$ for all $u \in \mathcal{V}$ and $1 \leq j \leq \delta_{\ell-1}$. We next consider the following four possibilities.

Case 1. There exists $1 \leq j \leq \delta_{\ell-1}$ such that $(\mathbf{A}_\ell)_{i,j} > 0$ and $(\mathbf{v}_{\lambda'_{\ell-1}})_j \geq \alpha_{\ell-1}$. Since all summands in (32) except $(\mathbf{b}_\ell)_i$ are nonnegative, the argument of σ in (32) is greater or equal to $(\mathbf{A}_\ell)_{i,j}(\mathbf{v}_{\lambda'_{\ell-1}})_j + (\mathbf{b}_\ell)_i \geq w_\ell \alpha_{\ell-1} + b_\ell = \beta_\ell$; since $\sigma(\beta_\ell) \geq \alpha_\ell$ and σ is monotonically increasing, we have $(\mathbf{v}_{\lambda'_\ell})_i \geq \alpha_\ell$, as desired.

Case 2. Case 1 does not hold and $C_\ell = 0$. If $C_\ell = k_\ell = 0$, the sum over $c \in \mathsf{Col}$ in both (31) and (32) is always equal to 0. Furthermore, since case 1 does not hold, the induction hypothesis ensures that for any $1 \leq j \leq \delta_{\ell-1}$ such that $A^\ell_{i,j} > 0$, we have $(\mathbf{v}_{\lambda'_{\ell-1}})_j = (\mathbf{v}_{\lambda_{\ell-1}})_j$. Thus, it follows that $(\mathbf{v}_{\lambda_\ell})_i = (\mathbf{v}_{\lambda'_\ell})_i$. If $C_\ell = k'_\ell = 0$, Algorithm 1 ensures that $\beta_\ell = b_\ell$. Then, since all summands in the argument of σ other than $(\mathbf{b}_\ell)_i$ are nonnegative, we have that the argument of σ in (32) is greater or equal than $(\mathbf{b}_\ell)_i \geq b_\ell = \beta_\ell$, and since σ is monotonically increasing and $\sigma(\beta_\ell) \geq \alpha_\ell$, we have that $(\mathbf{v}_{\lambda'_\ell})_i \geq \alpha_\ell$, as desired.

Case 3. $C_\ell > 0$ and there exist $c \in \mathsf{Col}$, $1 \leq j \leq \delta_{\ell-1}$, and $\langle v, u \rangle \in \mathcal{E}^c$ such that $(\mathbf{B}^\ell_c)_{i,j} > 0$ and $(\mathbf{u}_{\lambda'_{\ell-1}})_j \geq \alpha_{\ell-1}$. All summands in the argument of σ in (32) except $(\mathbf{b}_\ell)_i$ are nonnegative and $\max\text{-}C_\ell\text{-sum}(\{\!\{ (\mathbf{u}_{\lambda'_{\ell-1}})_j \mid \langle v, w \rangle \in \mathcal{E}^c \}\!\}) \geq (\mathbf{u}_{\lambda'_{\ell-1}})_j$ due to $C_\ell > 0$, so the argument of σ in (32) is greater or equal to $(\mathbf{B}^\ell_c)_{i,j}(\mathbf{u}_{\lambda'_{\ell-1}})_j + (\mathbf{b}_\ell)_i \geq w_\ell \alpha_{\ell-1} + (\mathbf{b}_\ell)_i = \beta_\ell$. Since $\sigma(\beta_\ell) \geq \alpha_\ell$ and σ is monotonically increasing, we have $(\mathbf{v}_{\lambda'_\ell})_i \geq \alpha_\ell$.

Case 4. None of cases 1–3 hold. Since case 1 does not hold, the induction hypothesis ensures that for any $1 \leq j \leq \delta_{\ell-1}$ such that $A^\ell_{i,j} > 0$, we have $(\mathbf{v}_{\lambda'_{\ell-1}})_j = (\mathbf{v}_{\lambda_{\ell-1}})_j$. Furthermore, since case 2 does not hold, we have $C_\ell > 0$. Finally, case 3 does not hold, so, for each $c \in \mathsf{Col}$ and $1 \leq j \leq \delta_{\ell-1}$ such that $(\mathbf{B}^c_\ell)_{i,j} > 0$, we have $(\mathbf{u}_{\lambda'_{\ell-1}})_j = (\mathbf{u}_{\lambda_{\ell-1}})_j$ for each u such that $\langle v, u \rangle \in \mathcal{E}^c$, and so $\{\!\{ (\mathbf{u}_{\lambda_\ell})_j \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\} = \{\!\{ (\mathbf{u}_{\lambda'_\ell})_j \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}$ holds. By these observations, our assumption that $(\mathbf{v}_{\lambda_\ell})_i \neq (\mathbf{v}_{\lambda'_\ell})_i$, and equations (31) and (32), then $C_\ell = k'_\ell < k_\ell$ and there must exist at least one $1 \leq j \leq \delta_{\ell-1}$ such that $(\mathbf{B}^c_\ell)_{i,j} > 0$ and the number of distinct u such that $\langle v, u \rangle \in \mathcal{E}^c$ and $(\mathbf{u}_{\lambda_{\ell-1}})_j > 0$ is greater than C_ℓ. For such j, and since all summands in the argument of σ in (32) except $(\mathbf{b}_\ell)_i$ are nonnegative, it holds that the argument is greater or equal than

$$(\mathbf{B}^c_\ell)_{i,j} \max\text{-}C_\ell\text{-sum}(\{\!\{ (\mathbf{u}_{\lambda'_\ell})_j \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}) + (\mathbf{b}_\ell)_i. \tag{33}$$

However, as we have already observed, we know that there exist at least C_ℓ elements different from zero in the multiset in (33), and Lemma A.2 and the definitions of ϵ_ℓ amd $\mathcal{X}_{\ell-1,j}$ ensure that each of these elements is greater or equal than ϵ_ℓ. Thus, the value in (33) is greater or equal than $w_\ell C_\ell \epsilon_\ell + b_\ell \geq \beta_\ell$. However, $\sigma(\beta_\ell) \geq \alpha_\ell$ since σ is monotonic, we have $(\mathbf{v}_{\lambda_\ell})_i \geq \alpha_\ell$, which concludes the proof.

To complete the proof of the theorem, we consider an arbitrary term t in D and an arbitrary unary predicate U_i in the (Col, δ)-signature, where $1 \leq i \leq \delta$, and we show that $U_i(t) \in T_{\mathcal{N}}(D)$ if and only if $U_i(t) \in T_{\mathcal{N}'}(D)$; this implies the theorem since $T_{\mathcal{N}}(D)$ and $T_{\mathcal{N}'}(D)$ can only contain atoms of this form. By definition of the canonical encoder/decoder scheme and the definitions of both $\mathcal{N}$ and $\mathcal{N}'$, it suffices to show that

$$\mathsf{cls}((\mathbf{v}_{\lambda_L})_i) = 1 \text{ if and only if } \mathsf{cls}((\mathbf{v}_{\lambda'_L})_i) = 1, \tag{34}$$

for v the vertex of the form v_t in $\mathcal{V}$. By the first result shown above by induction, we have that $(\mathbf{v}_{\lambda_L})_i \geq (\mathbf{v}_{\lambda'_L})_i$, and the second result ensures that either $(\mathbf{v}_{\lambda_L})_i = (\mathbf{v}_{\lambda'_L})_i$ or $(\mathbf{v}_{\lambda'_L})_i \geq \alpha_L$, since $L \geq \ell_{\mathsf{st}}$. If $(\mathbf{v}_{\lambda_L})_i = (\mathbf{v}_{\lambda'_L})_i$, (34) holds trivially. If $(\mathbf{v}_{\lambda'_L})_i \geq \alpha_L$, then the definition of α_L in Algorithm 1 ensures that $\mathsf{cls}((\mathbf{v}_{\lambda'_L})_i) = 1$, and since $(\mathbf{v}_{\lambda_L})_i \geq (\mathbf{v}_{\lambda'_L})_i$, then $\mathsf{cls}((\mathbf{v}_{\lambda_L})_i) = 1$, so (34) holds. □

For a dataset D, let $\mathsf{tms}(D)$ be the set containing each term t such that D contains an atom of the form $U(t)$, $E^c(t, s)$, or $E^c(s, t)$, for U and E^c arbitrary unary and binary predicates, respectively, and s an arbitrary term. An *isomorphism* from a (Col, δ)-dataset D to a (Col, δ)-dataset D' is an injective mapping h of terms to terms that is defined (at least) on all $\mathsf{tms}(D)$ and satisfies $h(D) = D'$, where $h(D)$ is the dataset obtained by replacing each fact of the form $U(t)$ in D with $U(h(t))$, and each fact of the form $E^c(t, s) \in D$ with $E^c(h(t), h(s))$.

Lemma A.3. *For all (Col, δ) datasets D and D', the following properties holds:*

(M1) each isomorphism from D to D' is also an isomorphism from $T_{\mathcal{N}}(D)$ to $T_{\mathcal{N}}(D')$; and
(M2) $D \subseteq D'$ implies $T_{\mathcal{N}}(D) \subseteq T_{\mathcal{N}}(D')$.

Proof. It is straightforward to see that property (M1) holds: for any two (Col, δ)-datasets, an isomorphism h from D to D' induces a bijective mapping between the vertices of $\mathsf{enc}D$ and $\mathsf{enc}D'$; moreover, the result of applying $\mathcal{N}$ to a (Col, δ)-graph

depends only on the graph structure and not on the vertex names, so it is straightforward to show that the vectors labelling the corresponding vertices are identical.

To see that property (M2) holds, consider arbitrary datasets D and D' such that $D \subseteq D'$. Let $\mathcal{G} = \langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda \rangle$ and $\mathcal{G}' = \langle \mathcal{V}', \{\mathcal{E}'^c\}_{c \in \mathsf{Col}}, \lambda' \rangle$ be the canonical encodings of D and D', respectively, and $\lambda_0, \ldots, \lambda_L$ and $\lambda'_0, \ldots, \lambda'_L$ be the functions labelling the vertices of $\mathcal{G}$ and $\mathcal{G}'$ when $\mathcal{N}$ is applied to these graphs. By a straightforward induction on $0 \leq \ell \leq L$ we show that $(\mathbf{v}_{\lambda_\ell})_i \leq (\mathbf{v}_{\lambda'_\ell})_i$ holds for each vertex $v \in \mathcal{V}$ and each $1 \leq i \leq \delta_\ell$. The base case for $\ell = 0$ follows immediately from the canonical encoding and the fact that $D \subseteq D'$. For the induction step, the canonical encoding and $D \subseteq D'$ imply $\mathcal{E}^c \subseteq \mathcal{E}'^c$. The values of $(\mathbf{v}_{\lambda_\ell})_i$ and $(\mathbf{v}_{\lambda'_\ell})_i$ are computed by equation (3). Now by the inductive hypothesis, $(\mathbf{u}_{\lambda_{\ell-1}})_j \leq (\mathbf{u}_{\lambda'_\ell})_j$ holds for each $u \in \mathcal{V}$ and $1 \leq j \leq \delta_{\ell-1}$, which ensures

$$\max\text{-}k_\ell\text{-sum}(\{\!\{ (\mathbf{u}_{\lambda_{\ell-1}})_j \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}) \leq \max\text{-}k_\ell\text{-sum}(\{\!\{ (\mathbf{u}_{\lambda'_{\ell-1}})_j \mid \langle v, u \rangle \in \mathcal{E}'^c \}\!\}).$$

All elements of $\mathbf{A}_\ell$ and all $\mathbf{B}^c_\ell$ with $c \in \mathsf{Col}$ are nonnegative, and σ is monotonically increasing, which implies $(\mathbf{v}_{\lambda_\ell})_i \leq (\mathbf{v}_{\lambda'_\ell})_i$. Finally, cls is a step function, so $\mathsf{cls}((\mathbf{v}_{\lambda_\ell})_i) \leq \mathsf{cls}((\mathbf{v}_{\lambda'_\ell})_i)$ holds as well, which ensures $T_{\mathcal{N}}(D) \subseteq T_{\mathcal{N}}(D')$. □

Lemma 12. *Let r be a constant-free Datalog rule with head H, let V be the set of variables in r, and let A be the set of body atoms of r. Then, $\mathcal{N}$ captures r if and only if $H\nu \in T_{\mathcal{N}}(A\nu)$ for each substitution $\nu : V \to S$ such that $H\nu \in T_r(A\nu)$, where S is a set of $|V|$ distinct constants.*

Proof. If there exists a substitution $\nu : V \to S$ such that $H\nu \in T_r(A\nu)$ but $H\nu \notin T_{\mathcal{N}}(A\nu)$, then by definition $T_{\mathcal{N}}$ does not capture r. To conclude the proof of the lemma, it only remains to show the converse implication: if $H\nu \in T_{\mathcal{N}}(A\nu)$ for each substitution $\nu : v \to S$ such that $H\nu \in T_r(A\nu)$, then T captures r. To this end, we consider an arbitrary (Col, δ)-dataset D, and we prove that $T_r(D) \subseteq T_{\mathcal{N}}(D)$. If $T_r(D)$ is empty, then the claim holds vacuously, so suppose $T_r(D) \neq \emptyset$. Consider an arbitrary element α in $T_r(D)$; clearly, α is of the form $H\mu$ for some substitution μ such that $A\mu \subseteq D$ and $H\mu \in T_r(A\mu)$. Let h be an injective mapping from $\mathsf{tms}(A\mu)$ to the constants in S; such a mapping exists because the body of r contains at most $|V|$ variables, and so $\mathsf{tms}(A\mu)$ contains at most $|V|$ terms. Then, $\nu = h \circ \mu$ is a substitution mapping all variables in r to constants in $|V|$. Mapping h is injective, rule r is constant-free, and $H\mu \in T_r(A\mu)$, so the semantics of Datalog rule application ensure that $h(H\mu) \in T_r(h(A\mu))$, and so $H\nu \in T_r(A\nu)$. The latter implies $H\nu \in T_{\mathcal{N}}(A\nu)$ by the lemma assumption. Moreover, h is an isomorphism from $A\mu$ to $A\nu$, so property (M1) of Lemma A.3 implies $H\mu \in T_{\mathcal{N}}(A\mu)$. Finally, property (M2) of Lemma A.3 and $A\mu \subseteq D$ imply $\alpha = H\mu \in T_{\mathcal{N}}(D)$, as required. □

Theorem 13. *Let $\mathcal{P}_{\mathcal{N}}$ be the Datalog program containing, up to variable renaming, each $(L, |\mathsf{Col}| \cdot \delta_{\mathcal{N}} \cdot C_{\mathcal{N}})$-tree-like rule captured by $\mathcal{N}$, where $\delta_{\mathcal{N}} = \max(\delta_0, \ldots, \delta_L)$. Then, $\mathcal{N}$ and $\mathcal{P}_{\mathcal{N}}$ are equivalent.*

Proof. We prove the theorem by showing that $T_{\mathcal{N}}(D) = T_{\mathcal{P}_{\mathcal{N}}}(D)$ holds for each (Col, δ)-dataset D. GNN $\mathcal{N}$ captures every rule in $\mathcal{P}_{\mathcal{N}}$ and thus $T_r(D) \subseteq T_{\mathcal{N}}(D)$ for each $r \in \mathcal{P}_{\mathcal{N}}$; since $T_{\mathcal{P}_{\mathcal{N}}}(D) = \bigcup_{r \in \mathcal{P}_{\mathcal{N}}} T_r(D)$, we have $T_{\mathcal{P}_{\mathcal{N}}}(D) \subseteq T_{\mathcal{N}}(D)$.

To prove $T_{\mathcal{N}}(D) \subseteq T_{\mathcal{P}_{\mathcal{N}}}(D)$, we consider an arbitrary fact $\alpha \in T_{\mathcal{N}}(D)$, and we construct a $(L, |\mathsf{Col}| \cdot \delta_{\mathcal{N}} \cdot C_{\mathcal{N}})$-tree-like rule r such that $\alpha \in T_r(D)$ and r is captured by $T_{\mathcal{N}}$, which together imply $\alpha \in T_{\mathcal{P}_{\mathcal{N}}}(D)$. To find r, we consider the GNN $\mathcal{N}'$ obtained from $\mathcal{N}$ by replacing k_ℓ with C_ℓ for each $1 \leq \ell \leq L$. Theorem 10 ensures $T_{\mathcal{N}}(D) = T_{\mathcal{N}'}(D)$, and so $\alpha \in T_{\mathcal{N}'}(D)$. Let $\mathcal{G} = \langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda \rangle$ be the canonical encoding of D, and let $\lambda_0, \ldots, \lambda_L$ be the functions labelling the vertices of $\mathcal{G}$ when $\mathcal{N}'$ is applied to it. We next construct an atom H, a conjunction Γ, a substitution ν from the variables in Γ to $\mathsf{tms}(D)$, a graph U (without vertex labels) with fresh vertices not occurring in $\mathcal{G}$ of the form u_x for x a variable and edges with colours in Col, and mappings $M_{c,\ell,j} : U \to 2^{\mathcal{V}}$ for each $c \in \mathsf{Col}$, $1 \leq \ell \leq L$, and $1 \leq j \leq \delta_{\ell-1}$. We also assign to each vertex in U a level between 0 and L, and we identify a single vertex from U as the *root* vertex. In the rest of this proof, we use letters t and s for terms in $\mathsf{tms}(D)$, letters x and y for variables, letters v, w for the vertices in $\mathcal{V}$, and (possibly indexed) letter u for the vertices in U. Our construction is by induction from level L down to level 1. The base case defines a vertex of level L. Then, for each $1 \leq \ell \leq L$, the induction step considers the vertices of level ℓ and defines new vertices of level $\ell - 1$.

We initialise Γ as the empty conjunction, and we initialise ν and each $M_{c,\ell,j}$ as the empty mappings. For the base case, we note that α must be of the form $U_i(t)$, and so $\mathcal{V}$ contains a vertex v_t. We introduce a fresh variable x, and define $\nu(x) = t$; we define $H = U_i(x)$; we introduce vertex u_x of level L; and we make u_x the root vertex. Finally, we extend Γ with atom $U(x)$ for each $U(t) \in D$. For the induction step, consider $1 \leq \ell \leq L$ and assume that all vertices of level greater than ℓ have been already defined. We then consider each vertex of the form u_x of level ℓ. Let $t = \nu(x)$. For each colour $c \in \mathsf{Col}$, each layer $1 \leq \ell' < \ell$, and each dimension $j \in \{1, \ldots, \delta_{\ell'-1}\}$, let

$$M_{c,\ell',j}(u_x) = \big\{ w \mid \langle v_t, w \rangle \in \mathcal{E}^c \text{ and } (\mathbf{w}_{\lambda_{\ell'}})_j \text{ contributes to the result of } \max\text{-}C_{\ell'}\text{-sum}\big(\{\!\{ (\mathbf{w}_{\lambda_{\ell'}})_j \mid \langle v_t, w \rangle \in \mathcal{E}^c \}\!\}\big) \big\}. \quad (35)$$

At least one such set exists, but it may not be unique; however, any set satisfying (35) can be chosen. Each vertex of $M_{c,\ell',j}(u_x)$ must be of the form v_{s_n} for some term $s_n \in \mathsf{tms}(D)$, where $s_n \neq s_m$ for all $1 \leq n < m \leq |M_{c,\ell',j}(u_x)|$. We then introduce

a fresh variable y_n and define $\nu(y_n) = s_n$; we introduce a vertex u_{y_n} of level $\ell - 1$ and an edge $E^c(u_x, u_{y_n})$ to U; and we append to Γ the conjunction

$$\bigwedge_{n=1}^{|W|} \Big(E^c(x, y_n) \wedge B_D(y_n) \Big) \wedge \bigwedge_{1 \leq n < m \leq |W|} y_n \not\approx y_m, \tag{36}$$

where $W = M_{c,\ell',j}(u_x)$ and $B_D(y_n)$ is the conjunction consisting of an atom $U(y_n)$ for each $U(s_n) \in D$. Since each $M_{c,\ell,j}(u_x)$ contains at most $C_{\ell'}$ elements, this step adds at most $|\mathsf{Col}| \cdot \delta_{\ell'-1} \cdot C_{\ell'} \cdot \ell'$ new successors of u_x. This completes our inductive construction. At this point, $H = U_i(x)$ and Γ is a $(L, |\mathsf{Col}| \cdot \delta_{\mathcal{N}} \cdot C_{\mathcal{N}})$-tree-like formula for x. Thus, rule $H \leftarrow \Gamma$ is a $(L, |\mathsf{Col}| \cdot \delta_{\mathcal{N}} \cdot C_{\mathcal{N}})$-tree-like rule. Furthermore, the construction of ν ensures $D \models \Gamma\nu$ so $H\nu \in T_r(D)$, but $H\nu = \alpha$, so $\alpha \in T_r(D)$, as required.

To complete the proof, we next show that r is captured by $T_{\mathcal{N}}$, which is equivalent to showing that r is captured by $T_{\mathcal{N}'}$. To do this, we consider an arbitrary dataset D' and an arbitrary ground atom α' such that $\alpha' \in T_r(D')$. This implies that there exists some substitution ν' such that $D' \models \Gamma\nu'$ and and $\alpha' = H\nu'$. Consider the encoding of D' into a (Col, δ)-graph $\mathcal{G} = \langle \mathcal{V}', \{\mathcal{E}'^c\}_{c \in \mathsf{Col}}, \lambda' \rangle$, and let $\lambda'_0, \ldots, \lambda'_L$ be the functions labelling the vertices of $\mathcal{G}'$ when $\mathcal{N}'$ is applied to it. We use letters p, q, and q' for the vertices of $\mathcal{V}'$.

We now prove the following statement by induction: for each $0 \leq \ell \leq L$ and each vertex u_x of U whose level is at least ℓ, we have $(\mathbf{v}_{\lambda_\ell})_i \leq (\mathbf{p}_{\lambda'_\ell})_i$ for each $i \in \{1, \ldots, \delta_\ell\}$, where $v = v_{\nu(x)}$ and $p = v_{\nu'(x)}$. For the base case, $\ell = 0$, consider an arbitrary $1 \leq i \leq \delta_0$ and $u_x \in U$, and let $v = v_{\nu(x)}$ and $p = v_{\nu'(x)}$. Note that $(\mathbf{v}_{\lambda_0})_i \in \{0, 1\}$ and $(\mathbf{p}_{\lambda'_0}) \in \{0, 1\}$, so we only need to prove that $(\mathbf{v}_{\lambda_0})_i = 1$ implies $(\mathbf{p}_{\lambda'_0})_i = 1$. By Definition 2, $(\mathbf{v}_{\lambda_0})_i = 1$ implies $U_i(x\nu) \in D$. The construction of Γ ensures that $U_i(x) \in \Gamma$, and $D' \models \Gamma\nu'$ implies $U_i(x\nu') \in D'$, and so $(\mathbf{p}_{\lambda_0})_i = 1$, as required.

For the induction step, assume that the property holds for some $\ell - 1$, and consider an arbitrary vertex $u_x \in U$ whose level is at least ℓ; consider an arbitrary $c \in \mathsf{Col}$, $i \in \{1, \ldots, \delta_i\}$, and let $v = v_{\nu(x)}$ and $p = v_{\nu'(x)}$. Note that the following holds.

$$(\mathbf{v}_{\lambda_\ell})_i = \sigma \left(\sum_{j=1}^{\delta_{\ell-1}} (\mathbf{A}_\ell)_{i,j} (\mathbf{v}_{\lambda_{\ell-1}})_j + \sum_{c \in \mathsf{Col}} \sum_{j=1}^{\delta_{\ell-1}} (\mathbf{B}^c_{\ell-1})_{i,j} \ \text{max-}C_\ell\text{-sum}\{\!\{ (\mathbf{w}_{\lambda_{\ell-1}})_j \mid \langle v, w \rangle \in \mathcal{E}^c \}\!\} + (\mathbf{b}_\ell)_i \right) \tag{37}$$

$$(\mathbf{p}_{\lambda'_\ell})_i = \sigma \left(\sum_{j=1}^{\delta_{\ell-1}} (\mathbf{A}_\ell)_{i,j} (\mathbf{p}_{\lambda'_{\ell-1}})_j + \sum_{c \in \mathsf{Col}} \sum_{j=1}^{\delta_{\ell-1}} (\mathbf{B}^c_{\ell-1})_{i,j} \ \text{max-}C_\ell\text{-sum}\{\!\{ (\mathbf{q}_{\lambda'_{\ell-1}})_j \mid \langle p, q \rangle \in \mathcal{E}'^c \}\!\} + (\mathbf{b}_\ell)_i \right) \tag{38}$$

The induction assumption ensures $(\mathbf{v}_{\lambda_{\ell-1}})_j \leq (\mathbf{p}_{\lambda'_{\ell-1}})_j$ for each $1 \leq j \leq \delta_{\ell-1}$. Also, for each colour $c \in \mathsf{Col}$ and each $1 \leq j \leq \delta_{\ell-1}$, we have that $\text{max-}C_\ell\text{-sum}\{\!\{ (\mathbf{w}_{\lambda_{\ell-1}})_j \mid \langle v, w \rangle \in \mathcal{E}^c \}\!\}$ is equal to $\sum_{w \in W} (\mathbf{w}_{\lambda_{\ell-1}})_j$, where $W = M_{c,\ell-1,j}(u_x)$. Recall that the elements of W are of the form $v_{s_1}, \cdots, v_{s_{|W|}}$ where $s_1, \cdots, s_{|W|}$ are terms in $\mathsf{tms}(D)$. Furthermore, by the construction of U, there are W distinct vertices $u_{y_1}, \cdots, u_{y_{|W|}}$ in U of level $\ell - 1$ such that $\nu(y_n) = s_n$ and $E^c(u_x, u_{y_n})$ is in U for each $1 \leq n \leq |W|$. Furthermore, Γ contains atoms $E^c(x, y_1), \ldots, E^c(x, y_{|W|})$ as well as inequalities $y_n \not\approx y_m$ for $1 \leq n < m \leq |W|$. We then have $E^c(\nu'(x), \nu'(y_n)) \in D'$ and $\nu'(y_n) \neq \nu'(y_m)$ for $1 \leq n < m \leq |W|$. Thus, $W' = \{v_{\nu'(y_1)}, \ldots, v_{\nu'(y_{|W|})}\}$ is a set of $|W|$ *distinct* c-neighbours of $v_{\nu(x)}$ in $\mathcal{G}'$. The induction assumption ensures that $w = v_{\nu(y_n)}$ and $q = v_{\nu'(y_n)}$ imply $(\mathbf{w}_{\lambda_{\ell-1}})_j \leq (\mathbf{q}_{\lambda'_{\ell-1}})_j$, and so $\sum_{w \in W} (\mathbf{w}_{\lambda_{\ell-1}})_j \leq \sum_{q \in W'} (\mathbf{q}_{\lambda'_{\ell-1}})_j$. Thus, by equations (37) and (38), the fact that the elements from $\mathbf{A}_\ell$ and all $\mathbf{B}^c_\ell$ are nonnegative, and σ is monotonically increasing, we have $(\mathbf{v}_{\lambda_\ell})_i \leq (\mathbf{p}_{\lambda'_\ell})_i$, as required.

Recall that $\alpha' = H\nu'$ is of the form $U_i(t')$ with $t' = \nu'(x)$; moreover, $U_i(t) \in T_{\mathcal{N}'}(D)$ with $t = \nu(x)$. Now let $v = v_t$ and $p = v_{t'}$. Now $U_i(t) \in T_{\mathcal{N}'}(D)$ implies $\mathsf{cls}((\mathbf{v}_{\lambda_L})_i) = 1$, and the above property ensures $(\mathbf{v}_{\lambda_L})_i \leq (\mathbf{p}_{\lambda'_L})_i$; since cls is a step function, we have $\mathsf{cls}((\mathbf{p}_{\lambda'_L})_i) = 1$. Hence, $U_i(t') \in T_{\mathcal{N}'}(D')$, as required. □

Theorem 14. *Algorithm 2 terminates on all inputs. Moreover, for $0 \leq \ell \leq L$ and $1 \leq i \leq \delta_\ell$,*

- $\mathsf{Next}(\ell, i, \triangleright)$ *returns the smallest element of $\mathcal{X}_{\ell,i}$, and*
- *for each $\alpha \in \mathbb{R}$, $\mathsf{Next}(\ell, i, \alpha)$ returns $\triangleleft$ if $\mathcal{X}^{>\alpha}_{\ell,i} = \emptyset$, and otherwise it returns the smallest element of $\mathcal{X}^{>\alpha}_{\ell,i}$.*

Proof. We first prove the two items of the theorem, and then we prove that Algorithm 1 terminates.

First, recall that by condition (S4) of Lemma A.1, for each $0 \leq \ell \leq L$ and each $1 \leq i \leq \delta_\ell$, set $\mathcal{X}_{\ell,i}$ is contains exactly all the elements of $\mathcal{S}_{\ell,i}$. Furthermore, since $\mathcal{S}_{\ell,i}$ is strictly monotonically increasing by condition (S2) of Lemma A.1, its smallest element is its first element. Hence, the smallest element of $\mathcal{X}_{\ell,i}$ is the first element of $\mathcal{S}_{\ell,i}$. Furthermore, for any $\alpha \in \mathbb{R}$, let $S^{>\alpha}_{\ell,i}$ be the subsequence of $\mathcal{S}_{\ell,i}$ which contains all elements in $S^{>\alpha}_{\ell,i}$ greater than α. Clearly, $\mathcal{X}^{>\alpha}_{\ell,i}$ is identical to the set of elements in $S^{>\alpha}_{\ell,i}$. Furthermore, since $\mathcal{S}_{\ell,i}$ is strictly monotonically increasing, then either $S^{>\alpha}_{\ell,i}$ is empty or it contains an element $s^{>\alpha}_{\ell,i}$ which appears in $\mathcal{S}_{\ell,i}$ exactly once and satisfies the following conditions:

(A1) all elements that precede $s^{>\alpha}_{\ell,i}$ in $\mathcal{S}_{\ell,i}$ are smaller or equal to α; and

(A2) all elements that follow $s^{>\alpha}_{\ell,i}$ in $\mathcal{S}_{\ell,i}$ are strictly greater than $s^{>\alpha}_{\ell,i}$.

In particular, condition (A2) ensures that if $S^{>\alpha}_{\ell,i}$ is not empty, then $s^{>\alpha}_{\ell,i}$ is its smallest element. Hence, to show the items of the theorem, it suffices to prove the following:

- $\mathsf{Next}(\ell, i, \triangleright)$ returns the first element of $\mathcal{S}_{\ell,i}$, and
- for each $\alpha \in \mathbb{R}$, $\mathsf{Next}(\ell, i, \alpha)$ returns $\triangleleft$ if $S^{>\alpha}_{\ell,i}$ is empty, and otherwise it returns $s^{>\alpha}_{\ell,i}$.

We show both items simultaneously via induction over $0 \leq \ell \leq L$.

For the base case $\ell = 0$, consider an arbitrary $1 \leq i \leq \delta_0$. To see that $\mathsf{Next}(0, i, \triangleright)$ returns the first element of $\mathcal{S}_{0,i}$, simply note that line 2 of Algorithm 2 ensures that $\mathsf{Next}(0, i, \triangleright) = 0$, which is precisely the smallest element of $\mathcal{S}_{0,i}$. To prove the second item, consider an arbitrary $\alpha \in \mathbb{R}$. If $S^{>\alpha}_{0,i}$ is empty then, $\alpha \geq 1$, in which case line 4 of Algorithm 2 ensures $\mathsf{Next}(0, i, \alpha) = \triangleleft$, as expected. If $S^{>\alpha}_{0,i}$ is not empty, we consider two possible cases: $\alpha < 0$ or $0 \leq \alpha < 1$. If $\alpha < 0$, then $S^{>\alpha}_{0,i} = (0, 1)$, but $\mathsf{Next}(0, i, \alpha) = 0$ by line 2 of Algorithm 2, so the claim holds. If $0 \leq \alpha < 1$, then $S^{>\alpha}_{0,i} = (1)$. But then, line 3 of Algorithm 2 ensures $\mathsf{Next}(0, i, \alpha) = 1$.

For the induction step, consider some arbitrary $1 \leq \ell \leq L$, and suppose that both items above hold for $\ell - 1$. Consider an arbitrary $1 \leq i \leq \delta_\ell$. We first show the first item. Observe that, the definition of $\mathcal{S}_{\ell,i}$ ensures that its first element is $\sigma(z)$ for $z = \mathsf{Val}(\ell, i, \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset)$. Recall that $\mathbf{s}_{\ell-1}$ is defined as the vector of dimension $\delta_{\ell-1}$ where $(\mathbf{s}_{\ell-1})_j$ is the first element of $\mathcal{S}_{\ell-1,j}$, for each $1 \leq j \leq \delta_{\ell-1}$. However, by induction hypothesis, $(\mathbf{s}_{\ell-1})_j = \mathsf{Next}(\ell - 1, j, \triangleright)$, and so $\mathbf{s}_{\ell-1} = \mathsf{Start}(\ell)$. Then, lines 6 and 7 of Algorithm 2 ensure that $\mathsf{Next}(\ell, i, \triangleright)$ is precisely $\sigma(z)$. We now show the second item. Consider an arbitrary $\alpha \in \mathbb{R}$. We study the execution of $\mathsf{Next}(\ell, i, \alpha)$. Since $\alpha \neq \triangleright$, F is initialised as stated in line 8 and so the loop starting in line 9 is executed. We consider now the outcome of the loop's execution. Let $\mathcal{S}_{\ell,i} = q_0, q_1, \ldots$. Let $N \geq 0$ be the smallest natural number such that either q_N is not defined or $q_N > \alpha$; such N must exist since $\mathcal{S}_{\ell,i}$ is either finite or it converges to infinity, and furthermore it is strictly monotonically increasing. We next show the following claim ($*$): for each $0 \leq n \leq N$, the algorithm's loop reaches a state where $F = f_n$ after a finite number of iterations. We prove this by induction on n.

The base case is straightforward since F initially contains only the triple $\langle \mathsf{Start}(\ell), \mathbf{Y}_\emptyset, z \rangle$, where $z = \mathsf{Val}(\ell, i, \mathsf{Start}(\ell), \mathbf{Y}_\emptyset)$. Furthermore, f_0 contains only the triple $\langle \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset, z' \rangle$, for $z' = \mathsf{Val}(\ell, i, \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset)$. However, we have already shown that $\mathsf{Start}(\ell) = \mathbf{s}_{\ell-1}$, so the initial state of F is identical to f_0. For the induction step, consider an arbitrary $0 \leq n < N$ and suppose that $F = f_n$ after a finite number of iterations of the algorithm's loop; we then show that $F = f_{n+1}$ holds after a finite number of additional iterations. By definition, f_n contains (at least) a triple of the form $\langle \mathbf{x}, \mathbf{Y}, z \rangle$ with $\sigma(z) = q_n$, and all other triples in f_n are of the form $\langle \mathbf{x}', \mathbf{Y}', z' \rangle$ with $z' \geq z$. The condition in line 10 then ensures that one of the triples of the form $\langle \mathbf{x}, \mathbf{Y}, z \rangle$ with $\sigma(z) = q_n$ will be selected; since $n < N$ and so $q_n \leq \alpha$, the condition in line 11 will not be satisfied, so the algorithm will not exit the loop and will afterwards start a new loop iteration. Then, the condition in line 14 ensures that no triple $\langle \mathbf{x}', \mathbf{Y}', z' \rangle$ with $z' \leq z$ is added to F. Let K be the number of triples in f_n of the form $\langle \mathbf{x}, \mathbf{Y}, z \rangle$ with $\sigma(z) = q_n$. We then have that after reaching the state where $F = f_n$, the algorithm's loop will run (at least) K additional times. Looking at lines 12 to 27, it is clear that each iteration removes from F one of the K triples and adds to F all of the triple's successors of the form $\langle \mathbf{x}', \mathbf{Y}', z' \rangle$ with $z' > z$. Thus, after those K additional steps, F will be exactly f_{n+1}. This concludes the proof of ($*$).

Suppose now that $S^{>\alpha}_{\ell,i}$ is empty, which means that all elements of $\mathcal{S}_{\ell,i}$ are smaller than α. Then, N is precisely the number of elements of $\mathcal{S}_{\ell,i}$ plus 1, that is, $N > 0$ and q_{N-1} is the last defined element of $\mathcal{S}_{\ell,i}$. By the claim ($*$), in the execution of $\mathsf{Next}(\ell, i, \alpha)$, F becomes equal to f_N after a finite number of steps. Since q_N is undefined, f_N must be empty. But then, since $F = f_N$, the condition in line 9 ensures that the loop is skipped, and line 28 ensures that the algorithm outputs $\triangleleft$, as expected. If $S^{>\alpha}_{\ell,i}$ is not empty, then there exists an element $s^{>\alpha}_{\ell,i}$ satisfying conditions (A1), and (A2). In particular, the definition of N, the condition (A1), and the fact that $s^{>\alpha}_{\ell,i} > \alpha$ together ensure that $s^{>\alpha}_{\ell,i}$ is precisely the Nth element of $\mathcal{S}_{\ell,i}$. Claim ($*$) ensures that in the execution of $\mathsf{Next}(\ell, i, \alpha)$, F becomes equal to f_N after a finite number of steps. Since the Nth element of $\mathcal{S}_{\ell,i}$ is defined, there exists a triple $\langle x, \mathbf{Y}, z \rangle \in f_n$ with $\sigma(z) = s^{>\alpha}_{\ell,i}$ and every other triple $\langle x', \mathbf{Y}', z' \rangle \in F$ is such that $z' \geq z$. But then, since $f_n = F$, the next iteration of the loop must select a triple with z as the third component (note that this triple may not be $\langle x, \mathbf{Y}, z \rangle$). But since $\sigma(z) = s^{>\alpha}_{\ell,i} > \alpha$, the test in line 11 succeeds and so the algorithm returns $s^{>\alpha}_{\ell,i}$, as expected. This completes the proof of the second item.

Finally, to see that Algorithm 1 is terminating, we simply observe that the smallest positive number in each $\mathcal{X}_{\ell,i}$ can be obtained by calling $\mathsf{Next}(\ell, i, \triangleright)$ and then, if this returns 0, calling $\mathsf{Next}(\ell, i, 0)$. We have already shown that such calls terminate and return the expected result. All other elements defined in the pseudocode of Algorithm 1 are easily computable from the parameters of $\mathcal{N}$, and so the algorithm terminates. □

B Proofs for Section 5

Theorem 16. *For each monotonic max (Col, δ)-GNN $\mathcal{N}$ with L layers, let $\delta_\mathcal{N} = \max(\delta_0, \ldots, \delta_L)$, and let $\mathcal{P}_\mathcal{N}$ be the Datalog program containing up to variable renaming each $(L, |\mathsf{Col}| \cdot \delta_\mathcal{N})$-tree-like rule without inequalities captured by $\mathcal{N}$. Then, $\mathcal{N}$ and $\mathcal{P}_\mathcal{N}$ are equivalent.*

Proof. We show that $\mathcal{N}$ and $\mathcal{P}_\mathcal{N}$ are equivalent by taking an arbitrary (Col, δ)-dataset D and showing $T_{\mathcal{P}_\mathcal{N}}(D) = T_\mathcal{N}(D)$. Inclusion $T_{\mathcal{P}_\mathcal{N}}(D) \subseteq T_\mathcal{N}(D)$ holds because, by definition, $T_\mathcal{N}$ captures each rule $r \in \mathcal{P}_\mathcal{N}$, which implies $T_r(D) \subseteq T_\mathcal{N}(D)$. Since $T_{\mathcal{P}_\mathcal{N}}(D) = \bigcup_{r \in \mathcal{P}_\mathcal{N}} T_r(D)$, we have $T_{\mathcal{P}_\mathcal{N}}(D) \subseteq T_\mathcal{N}(D)$.

For the converse inclusion, consider an arbitrary fact $\alpha \in T_\mathcal{N}(D)$. Since $\mathcal{N}$ is a max (Col, δ)-GNN, its capacity $C_\mathcal{N}$ is bounded by 1. The procedure in the proof of Theorem 18 therefore constructs a $(L, |\mathsf{Col}| \cdot \delta_\mathcal{N})$-tree-like rule r that is captured by $\mathcal{N}$ satisfying $\alpha \in T_r(D)$. Furthermore, since $C_\ell \leq 1$ for each $1 \leq \ell \leq L$, in equation (36) in the construction of r we have $|W| \leq C_{\ell'} \leq 1$, so the construction does not introduce any inequalities in the body of r, and so $r \in \mathcal{P}_\mathcal{N}$ holds. Hence, we have $\alpha \in T_{\mathcal{P}_\mathcal{N}}(D)$, and so $T_\mathcal{N}(D) \subseteq T_{\mathcal{P}_\mathcal{N}}(D)$, as required. □

Lemma 17. *For each (Col, δ)-dataset D, layer $1 \leq \ell < L$ of $\mathcal{N}_\mathcal{P}$, position $1 \leq i \leq \delta_\ell$, and term t in D, and for $\mathbf{v}_\ell$ the labelling of the vertex corresponding to t when $\mathcal{N}_\mathcal{P}$ is applied to the canonical encoding of D,*

- $(\mathbf{v}_\ell)_i = 1$ *if there exists a substitution ν mapping x to t such that $D \models \tau_i\nu$, and*
- $(\mathbf{v}_\ell)_i = 0$ *otherwise.*

Proof. For an arbitrary (Col, δ)-dataset D, let $\mathcal{G} = \langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda \rangle$ be the canonical encoding of D, and consider applying $\mathcal{N}$ to $\mathcal{G}$. We prove the claim by induction over $1 \leq \ell < L$. For the base case $\ell = 1$, consider an arbitrary term t, an arbitrary position $1 \leq i \leq \delta_1$, and let v be the vertex corresponding to t. Let J_1 and J_1' be the following sets of indices.

$$J_1 = \{j \mid 1 \leq j \leq \delta_0 \text{ and } (\mathbf{v}_0)_j = 1\} \tag{39}$$

$$J_1' = \{j \mid 1 \leq j \leq \delta_0 \text{ and } (\mathbf{A}_1)_{i,j} = 1\} \tag{40}$$

Recall that $(\mathbf{v}_0)_j \in \{0, 1\}$ and $(\mathbf{A}_1)_{i,j} \in \{0, 1\}$ for each $1 \leq j \leq \delta_0$; furthermore, $\mathbf{B}_1^c$ has all elements equal to 0 for each $c \in \mathsf{Col}$, and one can check that $(\mathbf{b}_1)_i = 1 - |J_1'|$. Thus, the argument of σ in the computation of $(\mathbf{v}_{\lambda_1})_i$ is equal to

$$|J_1 \cap J_1'| + 1 - |J_1'|, \tag{41}$$

which is equal to 1 if $J_1' \subseteq J_1$, and otherwise it less than or equal to 0. Hence, $(\mathbf{v}_0)_i = 1$ if $J_1' \subseteq J_1$, and otherwise $(\mathbf{v}_0)_i = 0$. Thus, to prove the claim, we show that $J_1' \subseteq J_1$ if and only if $D \models \tau_i\nu$ holds for $\nu = \{x \mapsto t\}$. For the $(\Leftarrow)$ direction, assume that $D \models \tau_i\nu$ holds for $\nu = \{x \mapsto t\}$, and consider an arbitrary $j \in J_j'$. The definition of J_1' implies $(\mathbf{A}_1)_{i,j} = 1$, so τ_i contains $U_j(x)$. But then, $D \models \tau_i\nu$ implies $U_j(t) \in D$, so our encoding ensures $(\mathbf{v}_0)_j = 1$; hence, $j \in J$ holds, as required. For the $(\Rightarrow)$ direction, assume that $J_1' \subseteq J_1$ holds. Then, for each $U_j(x) \in \tau_i$, we have $j \in J_1$ and so $U_j(t) \in D$. Hence, $D \models \tau_i\nu$ holds for $\nu = \{x \mapsto t\}$.

For the induction step, consider $1 < \ell < L$ such that the claim holds for $\ell - 1$, an arbitrary term t, an arbitrary position $1 \leq i \leq \delta_\ell$, and let v be the vertex corresponding to t. We consider two cases. The first case is $1 \leq i \leq \delta_{\ell-1}$; then, $(\mathbf{A}_\ell)_{i,j} = 1$ if and only if $j = i$, for each j we have $(\mathbf{B}_\ell^c)_{i,j} = 0$, and $(\mathbf{b}_\ell)_i = 0$; hence, we have $(\mathbf{v}_\ell)_i = (\mathbf{v}_{\ell-1})_i$, so both properties hold by the induction hypothesis. The second case is $\delta_{\ell-1} < i \leq \delta_\ell$. For each $c \in \mathsf{Col}$, let $J_{\ell,c}$ and $J_{\ell,c}'$ be defined as follows.

$$J_{\ell,c} = \{j \mid 1 \leq j \leq \delta_{\ell-1} \text{ and there exists a vertex } u \text{ such that } \langle v, u \rangle \in \mathcal{E}^c \text{ and } (\mathbf{u}_{\ell-1})_j = 1\} \tag{42}$$

$$J_{\ell,c}' = \{j \mid 1 \leq j \leq \delta_{\ell-1} \text{ and } (\mathbf{B}_\ell^c)_{i,j} = 1\} \tag{43}$$

Let τ_i be of the form (26). Since $\varphi_{i,0}$ is a conjunction of atoms of the form $U(x)$, there exists some $1 \leq j_0 \leq \delta_{\ell-1}$ such that $\varphi_{i,0} = \tau_{j_0}$. Furthermore, recall that $(\mathbf{A}_\ell)_{i,j} \in \{0, 1\}$ and $(\mathbf{v}_{\ell-1})_j \in \{0, 1\}$ for each $1 \leq j \leq \delta_{\ell-1}$, $(\mathbf{B}_\ell^c)_{i,j} \in \{0, 1\}$ for all $c \in \mathsf{Col}$, and $(\mathbf{b}_\ell)_i = -\sum_{c \in \mathsf{Col}} |J_{\ell,c}'|$. Thus, the argument of σ in the computation of $(\mathbf{v}_{\lambda_\ell})_i$ is equal to

$$(\mathbf{v}_{\ell-1})_{j_0} + \sum_{c \in \mathsf{Col}} \left(|J_{\ell,c} \cap J_{\ell,c}'| - |J_{\ell,c}'| \right), \tag{44}$$

which is equal to 1 if $(\mathbf{v}_{\ell-1})_{j_0} = 1$ and $J_{\ell,c}' \subseteq J_{\ell,c}$ for each $c \in \mathsf{Col}$, and otherwise it is less than or equal to 0. Consequently, $(\mathbf{v}_\ell)_i = 1$ if $(\mathbf{v}_{\ell-1})_{j_0} = 1$ and $J_{\ell,c}' \subseteq J_{\ell,c}$ for each $c \in \mathsf{Col}$, and otherwise $(\mathbf{v}_\ell)_i = 0$. Thus, to prove the claim, we show that $(\mathbf{v}_{\ell-1})_{j_0} = 1$ and $J_{\ell,c}' \subseteq J_{\ell,c}$ for each $c \in \mathsf{Col}$ if and only if there exists a substitution ν mapping x to t such that $D \models \tau_i\nu$.

For the $(\Leftarrow)$ direction, assume that such substitution ν exists. We then have $D \models \varphi_{i,0}\nu$; however, $\varphi_{i,0} = \tau_{j_0}$ and $j_0 \leq \delta_{\ell-1}$, so the induction hypothesis implies $(\mathbf{v}_{\ell-1})_{j_0} = 1$. To prove $J_{\ell,c}' \subseteq J_{\ell,c}$ for each $c \in \mathsf{Col}$, we consider arbitrary $c \in \mathsf{Col}$ and $j_k \in J_{\ell,c}'$, and we let $s = \nu(y_k)$. Note that $D \models E^c(x, y)\nu$ and so $\langle v, v_s \rangle \in \mathcal{E}^c$. Furthermore, $\varphi_{i,k}$ is a $(\ell - 2, c)$-tree-like formula for y_k equal to τ_{j_k} up to variable renaming. Also, $D \models \varphi_{i,k}\nu$ ensures that there exists a substitution ν_k mapping x to s such that $D \models \tau_{j_k}\nu_k$, so, by applying the induction hypothesis to the vertex u for term s, we have that $(\mathbf{u}_{\ell-1})_{j_k} = 1$. Consequently, $j \in J_{\ell,c}$ holds, as required.

For the $(\Rightarrow)$ direction, assume that $(\mathbf{v}_{\ell-1})_{j_0} = 1$ and $J_{\ell,c}' \subseteq J_{\ell,c}$ for each $c \in \mathsf{Col}$. Since $(\mathbf{v}_{\ell-1})_{j_0} = 1$, the induction hypothesis ensures $D \models \varphi_{i,0}\{x \mapsto t\}$. Furthermore, for each $1 \leq k \leq m_i$, $\varphi_{i,k}$ is a $(\ell - 2, |\mathsf{Col}|)$-tree-like formula, and so there exists $1 \leq j_k \leq \delta_{\ell-1}$ such that $\varphi_{i,k}$ is equal to τ_{j_k} up to variable renaming. Furthermore, $(\mathbf{B}_\ell^c)_{i,j_k} = 1$ and so $j_k \in J_{\ell,c}'$, which in turn implies $j_k \in J_{\ell,c}$. Thus, there exists vertex u for a term $s \in \mathsf{tms}(D)$ such that $\langle v, u \rangle \in \mathcal{E}^c$ and $(\mathbf{u}_{\ell-1})_{j_k} = 1$. By the

induction hypothesis, there exists a substitution ν_k mapping x to s such that $D \models \tau_{j_k}\nu_k$. Moreover, τ_{j_k} is equal to $\varphi_{i,k}$ up to variable renaming, so there exists a substitution ν'_k mapping y_k to s such that $D \models \varphi_{i,k}\nu'_k$. Note that $\varphi_{i,k}$ has no variables in common with $\varphi_{i,k'}$ for each $1 \leq k < k' \leq m_i$, and none of these formulas mention x, so substitution $\nu = \{x \mapsto t\} \cup \bigcup_{k=1}^{m_i} \nu'_k$ is correctly defined. Observe that $D \models \varphi_{i,0}\nu$, $D \models \varphi_{i,k}\nu$ for each $1 \leq k \leq m_i$, and $D \models E^c(x, y_k)\nu$ since $\langle v, u\rangle \in \mathcal{E}^c$. Thus, $D \models \tau_i\nu$ holds, as required. □

Theorem 18. *Program $\mathcal{P}$ and GNN $\mathcal{N}_\mathcal{P}$ are equivalent, and moreover $\delta_{L-1} \leq (|\mathsf{Col}| \cdot 2^\delta)^{f^d \cdot (d+1)!}$.*

Proof. For an arbitrary (Col, δ)-dataset D, let $\mathcal{G} = \langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda\rangle$ be the canonical encoding of D, and consider applying $\mathcal{N}$ to $\mathcal{G}$. Moreover, consider an arbitrary vertex $v \in \mathcal{V}$ for a term $t \in \mathsf{tms}(D)$, and an arbitrary position $1 \leq i \leq \delta_L$. We show that $U_i(t) \in T_{\mathcal{N}_\mathcal{P}}(D)$ if and only if $U_i(t) \in T_\mathcal{P}(D)$. Towards this goal, let J_L and J'_L be the following sets of indices.

$$J_L = \{j \mid 1 \leq j \leq \delta_{L-1} \text{ and } (\mathbf{v}_{L-1})_j = 1\} \tag{45}$$

$$J'_L = \{j \mid 1 \leq j \leq \delta_{L-1} \text{ and } (\mathbf{A}_L)_{i,j} = 1\} \tag{46}$$

For each $1 \leq j \leq \delta_{L-1}$, we have $(\mathbf{A}_L)_{i,j} \in \{0, 1\}$, matrices $\mathbf{B}^c_L$ and $\mathbf{b}_L$ have all elements equal to 0, and Lemma 17 ensures $(\mathbf{v}_{L-1})_j \in \{0, 1\}$. Thus, the argument of σ in the computation of $(\mathbf{v}_{\lambda_L})_i$ is equal to $|J_L \cap J'_L|$, which is greater than or equal to 1 if $J'_L \cap J_L \neq \emptyset$, and smaller than or equal to 0 otherwise. Hence, $(\mathbf{v}_L)_i = 1$ if $J'_L \cap J_L \neq \emptyset$, and $(\mathbf{v}_L)_i = 0$ otherwise.

Now assume that $U_i(t) \in T_{\mathcal{N}_\mathcal{P}}(D)$ holds. The latter implies $\mathsf{cls}((\mathbf{v}_L)_i) = 1$, which implies $(\mathbf{v}_L)_i \geq 1$; moreover, as shown in the previous paragraph, then $J'_L \cap J_L \neq \emptyset$. Consider an arbitrary $j \in J'_L \cap J_L$. Since $j \in J'_L$, there exists a rule of the form $U_i(x) \leftarrow \varphi \in \mathcal{P}$ where φ is equal to τ_j. Furthermore, $j \in J_L$ implies $(\mathbf{v}_{L-1})_j = 1$, and by Lemma 17 there exists a substitution mapping x to t such that $D \models \varphi\nu$. Hence, $U_i(x)\nu \in T_\mathcal{P}(D)$, and so $U_i(t) \in T_\mathcal{P}(D)$ holds, as required.

Conversely, assume that $U_i(t) \in T_\mathcal{P}(D)$ holds. Fact $U_i(t)$ is produced by a rule $\varphi \to U_i(x) \in \mathcal{P}$ and a substitution ν mapping x to t such that $D \models \varphi\nu$. Since φ is a $(L-2, f)$-tree-like formula for x, there exists $1 \leq j \leq \delta_{L-1}$ such that φ is equal to τ_j up to variable renaming, and τ_j is a $(L-2, f)$-tree-like formula for x. Hence, Lemma 17 ensures that $(\mathbf{v}_{L-1})_j = 1$ and so $j \in J_L$. Furthermore, the definition of $\mathbf{A}_L$ ensures that $(\mathbf{A}_L)_{i,j} = 1$, and so $j \in J'_L$. Thus, $J'_L \cap J_L \neq \emptyset$, which implies $(\mathbf{v}_L)_i = 1$; this, in turn, ensures $\mathsf{cls}((\mathbf{v}_L)_i) = 1$, so $U_i(t) \in T_{\mathcal{N}_\mathcal{P}}(D)$ holds, as required.

We next provide an upper bound on δ_{L-1}. By Definition 11, the fan-out of a variable of depth i is at most $f(d-i)$; moreover, the number of variables of depth i is at most the number of variables of depth $i-1$ times the fan-out of each variable, which is $f^i \cdot d \dots (d-i+1)$ and is bounded by $f^i \cdot d!$. By adding up the contribution of each depth, there are at most $f^d \cdot (d+1)!$ variables. Each variable is labelled by one of the 2^δ formulas of depth zero, and each non-root variable is connected by one of the $|\mathsf{Col}|$ predicates to its parent. Hence, there are at most $(|\mathsf{Col}| \cdot 2^\delta)^{f^d \cdot (d+1)!}$ tree-like formulas. □