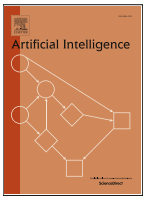
Artificial Intelligence 357 (2026) 104545

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



From monotonic graph neural networks to datalog and back: Expressive power and practical applications

David Tena Cucala [a,b,*], Bernardo Cuenca Grau [b], Boris Motik [b], Egor V. Kostylev [c]

[a] Department of Computer Science, University of London, Royal Holloway, United Kingdom
[b] Department of Computer Science, University of Oxford, United Kingdom
[c] Department of Informatics, University of Oslo, Norway

A B S T R A C T

Many tasks over knowledge graphs, such as link prediction, can be conceptualised as a problem of learning a transformation of sets of relational facts. Machine learning models such as graph neural networks (GNNs) can be used to realise this transformation, allowing the transformation to be learned from examples. However, it is often difficult to verify formally the properties of such a transformation, or understand why it derives a specific fact. Alternatively, such a transformation can be realised using a set of rules expressed in a knowledge representation language such as Datalog. Formal properties of such a transformation can be verified using symbolic means, and each derived fact can be justified by a rule; however, writing and curating the rules is costly and requires expertise in both the application domain and the formal language. To bridge the gap between these two approaches, in this paper we study the relationship between transformations realised by *monotonic max-sum GNNs*, a subclass of GNNs with nonnegative weights and max and sum aggregation functions, and transformations realised by Datalog rules. First, we provide an algorithm that can verify whether a given Datalog rule is *sound* for a network, in the sense that the GNN always derives all consequences of the rule on any input dataset. Second, we provide an algorithm that allows us to justify any fact derived by a GNN by computing a rule that is sound for the GNN and that derives the fact. Third, we study the expressive power of monotonic max-sum GNNs and show that, for each such GNN, one can compute a Datalog program where applying the GNN to any dataset produces the same facts as a single round of application of the program's rules to the dataset; we also 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. Finally, we carry out a practical evaluation and show that monotonic max-sum GNNs can be successfully trained in practice on common knowledge graph tasks, and that extracting rules from max-sum GNNs is practically feasible.

1. Introduction

Numerous tasks on knowledge graphs (KGs), such as KG completion [1], node classification [2], question answering [3], and recommendation [4], can be conceptualised as transformations of datasets consisting of relational facts. For example, in a recommender system, we can represent user–item interactions and the relevant background knowledge as a set of facts [5], and the system can then transform this dataset into another dataset containing the recommended user–item interactions. Machine learning (ML) models are increasingly being used to induce these transformations from examples: doing so is more cost-effective than specifying the transformations manually, and it does not require domain expertise. Many types of models have been proposed for this purpose, such as recurrent [6], fibring [7], and feed-forward networks [8], architectures that simulate forward [9,10] and backward chaining [11], and architectures for rule learning [12,13]. Recently, *graph neural networks* (GNNs) [14] were developed as ML models for processing graph data. Inferences made by GNNs are *permutation-invariant* (for graph-level outputs) and *permutation-equivariant* (for node-level

* Corresponding author.
E-mail addresses: david.tenacucala@rhul.ac.uk (D. Tena Cucala), bernardo.cuenca.grau@cs.ox.ac.uk (B. Cuenca Grau), boris.motik@cs.ox.ac.uk (B. Motik), egork@uio.no (E.V. Kostylev).

https://doi.org/10.1016/j.artint.2026.104545
Received 3 September 2024; Received in revised form 23 March 2026; Accepted 20 April 2026
Available online 24 April 2026

outputs): applying a GNN to two isomorphic graphs always produces the same output, regardless of how vertices are labelled. These properties are desirable whenever the objective is to analyse the graph structure independently from the identity of graph vertices. Dataset transformations are often expected to exhibit analogous properties. For example, if two distinct users interact with distinct sets of items in exactly the same way, one can reasonably expect recommendations to be produced using the same general rules. As a consequence, developing dataset transformations based on GNNs has received considerable attention recently [15–23].

A key question in such applications is understanding the *expressivity* of the ML models used—that is, describing the class of dataset transformations that the models can compute. In traditional data management tasks such as query answering and logical reasoning, expressivity is often established by identifying a logic-based language that realises the same class of transformations as the task being studied. For example, core aspects of relational and graph query languages have been characterised using fragments of first-order logic [5,24], and logical deduction over RDF datasets has been described using the rule-based language *Datalog* [25]. Such correspondences enable rigorous understanding and comparison of different data management languages. Characterising the expressivity of ML models for data management has thus steadily gained importance, and computational logic provides a well-established methodology: one can describe conditions under which an ML-based dataset transformation becomes equivalent to a logical formalism in the sense that applying the ML-based transformation to an arbitrary dataset produces the same result as applying a specific formula of the formalism in question. In a pioneering study, [26] showed that each GNN-induced transformation expressible in first-order logic is equivalent to a concept query of the $\mathcal{ALCQ}$ *description logic* [27]—a popular KR formalism. Huang et al. [28] proved an analogous result for a class of GNNs with a dedicated vertex and colour, and [29] generalised these results by showing that GNN-induced transformations can, under very mild restrictions, be captured by the fragment $\mathrm{GFO}+\mathrm{C}$ of guarded first-order logic with counting terms. [30] and [31] showed that GNNs can express certain types of graph isomorphism tests, such as the 1-dimensional Weisfeiler–Leman test. Sourek et al. [32] characterised the expressivity of GNNs using a hybrid language where each logical formula is annotated with a tensor. Finally, Benedikt et al. [33] characterised the expressivity of several GNN classes with rational coefficients using variants of logics with Presburger quantifiers.

Characterising a class of ML models using a logic-based formalism also has the potential to address the key practical challenge of explaining the models' predictions. Datalog seems especially suited to this purpose. In Datalog, one can capture domain-specific knowledge using 'if-then' rules, which can be applied to a set of facts to derive new conclusions. Furthermore, any Datalog inference can be explained using a proof describing precisely all steps necessary to derive a fact from the input dataset. Such rigorous, yet human-readable explanations can help foster trust in system predictions, ensure norm compliance, and enable verification of fairness standards. For example, in a recommender application, one could capture domain knowledge about literature using facts $Author(Dostoevsky, Crime\_and\_Punishment)$ and $Author(Dostoevsky, The\_Idiot)$, and one could further represent a user's past interactions with the system using facts such as $Likes(Crime\_and\_Punishment)$. The system could then learn an ML model that transforms these facts into recommendations expressed using facts such as $Recommend(The\_Idiot)$. A possible way to explain such recommendations is to extract Datalog rules that correspond to the model. For example, if one can prove that the model produces the same conclusions as rule (1), then the recommendation can explained by showing that it is derived by applying rule (1) to the background knowledge and past interactions.

$$Author(x, y_1) \wedge Author(x, y_2) \wedge Likes(y_1) \rightarrow Recommend(y_2) \tag{1}$$

In this paper we introduce a new family of GNN-based dataset transformations that can be trained from examples as usual, but whose expressive power can be characterised using Datalog. Moreover, we present practical algorithms that can extract an equivalent set of rules from any such transformation. GNNs are usually applied to a dataset in three stages. First, the input dataset is *encoded* into a graph whose vertices are labelled with numeric feature vectors. Second, this graph is processed by a GNN with several layers; in each layer, the feature vectors are updated based on the model parameters and the feature vectors of the previous layer. Finally, the vectors of the output layer are *decoded* to the output dataset. Our transformations are analogous, but the stages are carefully crafted to ensure that the overall transformation can always be described symbolically using rules.

Towards this goal, we encounter two key obstacles. First, the expressivity of the overall transformation clearly depends on the encoding and decoding steps, which can make isolating the contribution of the ML model difficult. As a solution, in Section 3 we adopt a *canonical* encoding that minimally affects the expressibility of the transformation: the structure of the graph produced by the canonical encoding closely reflects the structure of the input dataset. Variants of the canonical encoding have already been considered by Schlichtkrull et al. [15], Barceló et al. [26], and Pflueger et al. [16]. We also introduce natural notions of soundness and completeness: a Datalog program (i.e., a set of rules) is sound or complete for a GNN if, for each dataset, applying the program to the dataset *once* (i.e., without fixpoint iteration) produces a *subset* or a *superset*, respectively, of the facts obtained by applying the GNN to the same dataset while using the canonical encoding. We also say that a program is *equivalent* to a GNN if it is both sound and complete for the GNN. Finally, we observe that encodings proposed in the literature that are not canonical, such as the ones by Morris et al. [30] or Liu et al. [17], can be described using extensions of Datalog, and so the expressivity of the transformations based on such encodings can be characterised by composing the relevant Datalog programs.

Second, the expressivity of GNNs and Datalog seems fundamentally different. In contrast to GNNs, Datalog can only realise monotonic transformations, meaning that adding facts to the input always leads to additional facts in the output—that is, adding facts cannot invalidate existing rule consequences. Moreover, 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, arbitrarily many vertices may influence whether a fact is derived. This seems fundamentally at odds with 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. As a solution, in Section 4, we introduce a class of *monotonic max-sum* GNNs, which allow for max or sum aggregation functions in each layer; however, matrix weights are required to be nonnegative, and the activation and classification functions need to satisfy certain restrictions (ReLU and threshold functions are allowed, but sigmoid and ELU are not). The design of max-sum GNNs ensures that the resulting transformation is *monotonic under injective homomorphisms*, and it allows us to show that one can replace each sum aggregation function by an aggregation function that sums only a bounded number of elements without affecting the model's output on any dataset. These observations are sufficient to bring the expressivity of max-sum GNN-based dataset transformations in line with the expressivity of Datalog. We also introduce *monotonic max* GNNs, which allow only for max aggregation in all layers.

In Section 5, we characterise the soundness of Datalog rules for monotonic max-sum GNNs—that is, we give necessary and sufficient conditions for a Datalog rule to be sound for a monotonic max-sum GNN. Furthermore, we provide an analogous, specific characterisation for monotonic max GNNs that we later use to develop practical soundness checks.

In Section 6, we prove that for each fact derived by a monotonic max-sum GNN on an input dataset, we can explain the derivation by computing a Datalog rule of a certain tree-like shape, possibly with inequalities in the body, that is sound for the GNN and that derives the fact from the dataset in one round of rule application. Moreover, we present a practical algorithm that computes one such rule from an input dataset and a fact produced by the GNN on this dataset.

In Section 7, we show that each monotonic max-sum GNN is equivalent to a Datalog program that can be computed algorithmically from the GNN itself. 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). This result is interesting because it requires enumerating elements from potentially infinite sets of real-valued candidate feature values in a way that guarantees termination. We also show that for monotonic max GNN, it is sufficient to consider Datalog tree-like programs without inequalities, which in turn can simplify rule extraction.

In Section 8 we present several optimisations of the algorithm for extracting an equivalent Datalog program from a monotonic max GNN. Our optimisations are effective at reducing the search space of the relevant rules, and at the same time they tend to produce more concise rules—that is, rules with fewer body atoms.

In Section 9 we prove that monotonic max GNNs can be characterised exactly using tree-like Datalog programs: not only is each such GNN equivalent to a tree-like Datalog program as per Section 7, but the converse holds as well in the sense that for each tree-like Datalog program, there exists an equivalent monotonic max GNN.

In Section 10, we present the results of our evaluation of monotonic max-sum GNNs on several knowledge graph completion benchmarks. Our results show that the performance of max-sum GNNs on this task is comparable or superior to the state-of-the-art approaches. Furthermore, we show that our algorithm from Section 6 can quickly extract short and simple rules that explain facts derived by our models. Finally, for several of our models, we show that our optimised algorithm from Section 8 can extract a program that is equivalent to the GNN. Monotonic max-sum GNNs are thus competitive with existing techniques for inductive knowledge graph completion, but with the added benefit that a model's predictions can be explained fully using logical rules.

This paper extends our earlier work [34] published at the KR 2023 conference. New results include the algorithm for extracting a rule that explains the derivation of a single fact, an optimised algorithm for extracting an equivalent Datalog program, a discussion of several nonstandard encoding schemes, an empirical evaluation of max-sum GNNs and our rule extraction algorithms, and detailed and revised proofs of all technical results.

2. Preliminaries

In this section, we recapitulate the basic definitions of first-order rules, Datalog, and Graph Neural Networks (GNNs) that this paper builds upon. For f a function, $\mathsf{dom}(f)$ is the domain of f.

2.1. Datasets and Datalog

All logical formulas in this paper consist of symbols drawn from countably infinite, disjoint sets of *variables*, *constants*, *function symbols*, and *predicates*. Each function symbol and each predicate is associated with a nonnegative integer arity. A *term* is a variable, a constant, or an expression of the form $f(t_1, \dots, t_n)$ where f is a function symbol of arity n and $t_1, \dots, t_n$ are terms. An *atom* is an expression of the form $P(t_1, \dots, t_n)$ where P is a predicate of arity n and $t_1, \dots, t_n$ 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 can contain facts with functional terms, but they cannot contain inequalities. A fact α is true in a dataset D, written $D \models \alpha$, if $\alpha \in D$. A ground inequality $t_1 \not\approx t_2$ is true if $t_1 \neq t_2$; thus, we adopt the *Unique Name Assumption (UNA)*, which stipulates that any two distinct constants are always interpreted as distinct objects. For uniformity with atoms, we often write $D \models t_1 \not\approx t_2$ even though the truth of $t_1 \not\approx t_2$ does not depend on D. Furthermore, for α a conjunction of facts and ground inequalities, $D \models \alpha$ if $D \models \beta$ for each conjunct β of α. A *rule* is an expression of form (2), where n is a nonnegative integer, $B_1, \dots, B_n$ are *body* literals, H is the *head* atom, and each variable occurring in a body inequality also occurs in a body atom.

$$B_1 \wedge \cdots \wedge B_n \rightarrow H \tag{2}$$

A rule is *safe* if each variable occurring in the head also occurs in a body atom. Contrary to most of related literature, we do not require rules to be safe, and we discuss shortly the consequences of this change. A *program* is a finite set of rules. A *Datalog* rule is a

rule in which all terms are either variables or constants (i.e., functional terms are not allowed, but rule bodies are allowed to contain inequalities), and a *Datalog* program contains only Datalog rules.

A *substitution* μ is a mapping of finitely many variables to ground terms. For α a term or a literal, $\alpha\mu$ is the result of replacing in α each occurrence of a variable $x \in \mathsf{dom}(\mu)$ with $\mu(x)$. Conjunctions α and β of literals are *equal up to variable renaming* if there exists a bijective mapping μ from the set of all variables of α to the set of all variables of β such that $\alpha\mu$ and β contain exactly the same conjuncts; this notion is extended to rules in the obvious way. Moreover, a set S *contains* a conjunction or a rule α *up to variable renaming* if there exists $\beta \in S$ such that α and β are equal up to variable renaming. To simplify the presentation, we sometimes abuse the notation by treating conjunctions as sets of conjuncts, and we use them in expressions involving $\in$ and $\subseteq$; for example, $\alpha \subseteq \beta$ means that each conjunct of α is a conjunct of β.

Each rule r of form (2) defines an *immediate consequence* operator T_r on datasets: for D a dataset, $T_r(D)$ is the dataset that contains the fact $H\mu$ for each substitution μ mapping all variables of r to terms of D such that $D \models B_i\mu$ for each $i \in \{1, \dots, n\}$. For $\mathcal{P}$ a program, $T_{\mathcal{P}}(D) = \bigcup_{r \in \mathcal{P}} T_r(D)$. As mentioned earlier, we do not require rules to be safe—that is, the rule head can contain a variable not occurring in the rule body. In fact, the body can be empty, in which case it is written as $\top$. For example, rule $r = \top \rightarrow R(x, y)$ is well-formed, and the definition of T_r ensures that $T_r(D)$ contains exactly each fact $R(t_1, t_2)$ for all (not necessarily distinct) terms t_1 and t_2 occurring in D. Rule $r = B_1 \wedge \cdots \wedge B_n \rightarrow H$ *subsumes* rule $r' = B'_1 \wedge \cdots \wedge B'_m \rightarrow H'$ if there exists a substitution μ such that $H\mu = H'$ and $\{B_1\mu, \dots, B_n\mu\} \subseteq \{B'_1, \dots, B'_m\}$. If this is the case, then $T_{r'}(D) \subseteq T_r(D)$ for each dataset D. The immediate consequence operator is usually applied to a dataset iteratively until a fixpoint is reached, but this is not relevant to our work: we will develop GNN-based models that are equivalent to a *single* round of rule applications. We thus do not recapitulate the definition of the fixpoint computation as it is not necessary to present our results.

A *homomorphism* from a dataset D to a dataset D' is a mapping h of constants to constants that is defined at least on the set of all constants of D and that satisfies $h(D) \subseteq D'$, where $h(D)$ is the dataset obtained by replacing each constant a in D with $h(a)$ and removing all duplicate facts. Homomorphism h is *injective* if $h(a) \neq h(b)$ for all distinct pairs a, b of constants in $\mathsf{dom}(h)$. An operator T from datasets to datasets over the same constants is *monotonic under homomorphisms* if, for all datasets D and D' to which T is applicable, each homomorphism from D to D' is also a homomorphism from $T(D)$ to $T(D')$; moreover, T is *monotonic under injective homomorphisms* if the analogous property holds for injective homomorphisms only.

For each constant-free program $\mathcal{P}$, operator $T_{\mathcal{P}}$ is monotonic under injective homomorphisms; moreover, if $\mathcal{P}$ additionally does not contain inequalities in the rule bodies, then $T_{\mathcal{P}}$ is monotonic under homomorphisms. We illustrate these definitions by means of the dataset D and programs $\mathcal{P}$ and $\mathcal{P}'$ given by

$$D = \{A(a), R(a, b)\}, \qquad \mathcal{P} = \{A(x) \wedge R(x, y) \rightarrow S(x, y)\}, \qquad \text{and} \qquad \mathcal{P}' = \{A(x) \wedge R(x, y) \wedge x \not\approx y \rightarrow S(x, y)\}.$$

Clearly, $T_{\mathcal{P}}(D) = T_{\mathcal{P}'}(D) = \{S(a, b)\}$. First, operator $T_{\mathcal{P}}$ is monotonic in the usual set-theoretic sense: for each dataset D' with $D \subseteq D'$, the rule of $\mathcal{P}$ applies in D' in the same way as in D, so $S(a, b) \in T_{\mathcal{P}}(D')$. Operator $T_{\mathcal{P}'}$ is also monotonic—that is, the presence of inequalities in the body does not invalidate monotonicity. Second, operator $T_{\mathcal{P}}$ is monotonic under homomorphisms. For example, let $D' = \{A(c), R(c, c)\}$ be the dataset obtained from D by replacing a and b with c. Then, $T_{\mathcal{P}}(D') = \{S(c, c)\}$—that is, program $\mathcal{P}$ derives the fact obtained from $S(a, b)$ by the same constant replacements. In contrast, the body inequality ensures that $T_{\mathcal{P}'}(D') = \emptyset$—that is, operator $T_{\mathcal{P}'}$ is not monotonic under homomorphisms. Third, operators $T_{\mathcal{P}}$ and $T_{\mathcal{P}'}$ are both monotonic under injective homomorphisms. For example, let $D'' = \{A(c), R(c, d)\}$ be the dataset obtained from D by replacing a and b with c and d, respectively; in other words, D'' is an isomorphic copy of D. Now $T_{\mathcal{P}}(D'') = T_{\mathcal{P}'}(D'') = \{S(c, d)\}$—that is, both programs derive the fact obtained from $S(a, b)$ by the same replacements of constants.

2.2. Graph neural networks with Boolean outputs

As usual, $\mathbb{R}$ is the set of real numbers, $\mathbb{R}^+_0$ is its nonnegative subset, $\mathbb{N}_0$ is the set of nonnegative integers, and $\mathbb{N}$ is the subset of $\mathbb{N}_0$ without zero. In our work, we use vectors and matrices over $\mathbb{R}$ and $\mathbb{R}^+_0$. For $\mathbf{v}$ a vector and $i > 0$ an integer, $(\mathbf{v})_i$ is the ith element of $\mathbf{v}$. Similarly, for $\mathbf{A}$ a matrix and $i, j > 0$ integers, $(\mathbf{A})_{i,j}$ is the element in row i and column j of $\mathbf{A}$. We apply scalar functions to vectors element-wise; for example, for $\mathbf{v}_1, \dots, \mathbf{v}_n$ vectors of equal dimension, $\max\{\mathbf{v}_1, \dots, \mathbf{v}_n\}$ is the vector whose ith element is defined as $\max\{(\mathbf{v}_1)_i, \dots, (\mathbf{v}_n)_i\}$ for each i.

A function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is *monotonically increasing* if $x < y$ implies $\sigma(x) \leq \sigma(y)$; it is *Boolean* if it ranges over $\{0, 1\}$; and it is *unbounded* if, for each $y \in \mathbb{R}$, there exists $x \in \mathbb{R}$ such that $\sigma(x) > y$.

A *multiset* over $\mathbb{R}$ is a function $S : \mathbb{R} \rightarrow \mathbb{N}_0$ that assigns to each $x \in \mathbb{R}$ the number $S(x)$ of its occurrences. Such S is said to *contain* some $x \in \mathbb{R}$ if $S(x) > 0$. Moreover, S is *finite* if it contains finitely many elements, and the *cardinality* of such S is $|S| = \sum_{x \in \mathbb{R}} S(x)$. We often write S as a list of possibly repeated real numbers enclosed in double-braces $\{\{ \dots \}\}$. By a slight abuse of notation, we often treat sets as multisets where each element occurs only once. Finally, $\mathcal{F}(\mathbb{R})$ and $\mathcal{F}(\mathbb{R}^+_0)$ are the sets of all finite multisets over $\mathbb{R}$ and $\mathbb{R}^+_0$, respectively.

For Col a finite nonempty 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*, $\mathcal{E}^c \subseteq \mathcal{V} \times \mathcal{V}$ is a set of directed *edges* for each colour $c \in \mathsf{Col}$, and *labelling* λ assigns to each $v \in \mathcal{V}$ a vector $\lambda(v)$ 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 $v, u \in \mathcal{V}$ and each $c \in \mathsf{Col}$, and it is *Boolean* if $(\lambda(v))_i \in \{0, 1\}$ for each $v \in \mathcal{V}$ and each $i \in \{1, \dots, \delta\}$. To improve readability, we often write $\mathbf{v}_\lambda$ instead of $\lambda(v)$, or even just $\mathbf{v}$ when λ is clear from the context. Analogously, for indexed labellings, we often write $\mathbf{v}_{\lambda_\ell}$ or just $\mathbf{v}_\ell$ instead of $\lambda_\ell(v)$. A (Col, δ)-*graph neural network* (*GNN*) $\mathcal{N}$ with $L \geq 1$ *layers* is a tuple

$$\langle\ \{\mathbf{A}_\ell\}_{1 \leq \ell \leq L},\ \{\mathbf{B}^c_\ell\}_{c \in \mathsf{Col} \text{ and } 1 \leq \ell \leq L},\ \{\mathbf{b}_\ell\}_{1 \leq \ell \leq L},\ \{\mathsf{agg}_\ell\}_{1 \leq \ell \leq L},\ \sigma,\ \mathsf{cls}\ \rangle, \tag{3}$$

where, for each $\ell \in \{1, \dots, L\}$ and each $c \in \mathsf{Col}$, $\mathbf{A}_\ell$ and all the $\mathbf{B}^c_\ell$ 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 $\lambda_0, \dots, \lambda_L$ of vertex labellings such that $\lambda_0 = \lambda$ and, for each $\ell \in \{1, \dots, L\}$ and $v \in \mathcal{V}$, the value of $\lambda_\ell(v)$ is given by

$$\mathbf{v}_\ell = \sigma\Big(\mathbf{A}_\ell \mathbf{v}_{\ell-1} + \sum_{c \in \mathsf{Col}} \mathbf{B}^c_\ell \, \mathsf{agg}_\ell\big(\{\!\{ \mathbf{w}_{\ell-1} \mid \langle v, w\rangle \in \mathcal{E}^c \}\!\}\big) + \mathbf{b}_\ell\Big). \tag{4}$$

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 $\mathsf{cls}(\lambda_L(v))$.

In the GNN literature, input and output layers may not be required to have the same dimension δ, and moreover the final step may not use the classification function cls. In our work, however, these observations ensure that GNN outputs are Boolean graph that we can straightforwardly interpret as logical facts. We discuss these issues in detail in Section 3. Also, any GNN where the dimensions of the input and output feature vectors differ can be padded to a GNN where the dimensions are the same, so the requirement on dimensions being the same does not affect the expressivity of GNNs.

3. Encoding/decoding schemes

To realise a dataset transformation using a GNN, we must first encode the input dataset into a graph that can be directly processed by the 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. To overcome these issues, in Section 3.1 we introduce the canonical encoding scheme that converts a dataset into a graph in a straightforward way so that the resulting graph closely reflects the structure of the input dataset. In Section 3.2 we present formal definitions that capture possible relationships between Datalog programs and GNN-based transformations that use the canonical encoding. Finally, in Section 3.3 we discuss how one can extend our results to other encoding schemes described in the literature.

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, describe vertex connectivity using binary facts, and encode the 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 [15,16,26]. 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.

Definition 1. Let Col be a finite, nonempty set of colours and let $\delta \in \mathbb{N}$ be a dimension. A (Col, δ)-*signature* is a set that contains a unary predicate U_i for each $i \in \{1, \dots, \delta\}$, and a binary predicate E^c for each colour $c \in \mathsf{Col}$. A (Col, δ)-*fact* is a fact with a predicate U_i or E^c from the (Col, δ)-signature. A (Col, δ)-*dataset* contains only (Col, δ)-facts.

A consistent indexing scheme for predicates simplifies the presentation of our results, so Definition 1 stipulates that a (Col, δ)-signature consists of predicates of the form U_i and E^c. In examples, however, we use predicates labelled by English words to make them easier to understand. Furthermore, in the rest of this paper, we assume that terms occurring in datasets correspond one-to-one to vertices of coloured graphs—that is, each ground term t is paired with a unique vertex v^t. This is without loss of generality since the result of applying a GNN to a coloured graph depends only on the graph structure and the feature vectors, and not on the identity of vertices. We next define the canonical GNN-based transformation of (Col, δ)-datasets.

Definition 2. The *canonical encoding* $\mathsf{enc}(D)$ of a (Col, δ)-dataset D is the Boolean (Col, δ)-graph $\langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda\rangle$ that contains a vertex $v^t \in \mathcal{V}$ for each ground term t occurring in D and an edge $\langle v^t, v^s\rangle \in \mathcal{E}^c$ for each fact $E^c(t, s) \in D$, and where $(\mathbf{v}^t)_i = 1$ if and only if $U_i(t) \in D$.

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.

Fig. 1 shows the canonical encoding of the book recommendation dataset from Section 1. We introduce colours c_A and c_L to represent the binary predicates *Author* and *Likes*, respectively, and we use *User* and *Novel* as unary predicates U_1, and U_2, respectively. We can thus see the facts from Section 1 as a (Col, δ)-dataset where $\mathsf{Col} = \{c_A, c_L\}$ and $\delta = 2$. We introduce vertices v^D, v^{CP}, v^{TI}, and v^B in $\mathsf{enc}(D)$ for the constants *Dostoevsky*, *Crime_and_Punishment*, *The_Idiot*, and *Bob*, respectively. Next, we introduce a c_L-coloured edge $\langle v^B, v^{CP}\rangle$ for the fact *Likes*(*Bob*, *Crime_and_Punishment*), and c_A-coloured edges $\langle v^D, v^{TI}\rangle$ and $\langle v^D, v^{CP}\rangle$ for facts *Author*(*Dostoevsky*, *The_Idiot*) and *Author*(*Dostoevsky*, *Crime_and_Punishment*), respectively. Finally, we set to one the first element of the feature vector of v^B to represent the fact *User*(*Bob*), and the second element of the feature vectors of v^{TI} and v^{CP} to represent facts *Novel*(*The_Idiot*) and *Novel*(*Crime_and_Punishment*), respectively.

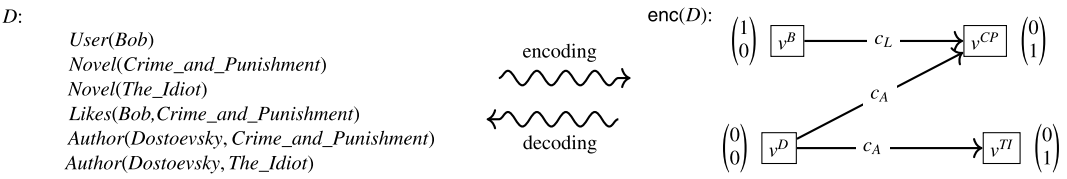


Fig. 1. Dataset D and its canonical encoding $\mathsf{enc}(D)$.

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, their encodings as (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, for each vertex $v \in \mathcal{V}$, v 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 clearly produces only regular graphs, and there is a one-to-one correspondence between (Col, δ)-datasets and regular (Col, δ)-graphs. Our results from Section 7 can thus be equivalently framed as characterising expressivity of GNN transformations of regular graphs in terms of Datalog programs.

A (Col, δ)-graph $\mathcal{G}$ that is Boolean but not regular contains zero-labelled, 'isolated' vertices disconnected from all other vertices. When such $\mathcal{G}$ is decoded into a (Col, δ)-dataset, such vertices do not produce any facts in $\mathsf{dec}(\mathcal{G})$ and thus several nonregular Boolean graphs can produce the same (Col, δ)-dataset. However, as evident from equation (4), information in our GNN model is exchanged only between neighbouring vertices, so each 'isolated' zero-labelled vertex is transformed by a GNN in the same way—that is, the vector labelling such a vertex in the GNN's output does not depend on any other vertices but only on the GNN's matrices. Consequently, such vertices do not seem interesting for our study of GNN expressivity.

Definition 2 explains why our definition of a GNN $\mathcal{N}$ in Section 2.2 uses a classification function and requires the dimensions of the input and output layers of $\mathcal{N}$ to be the same. In particular, we do not see a natural correspondence between non-Boolean graphs and datasets; hence, the use of the classification function in the last layer ensures that graph $\mathcal{N}(\mathsf{enc}(D))$ is Boolean so we can decode it into a (Col, δ)-dataset. The second restriction allows us to decode $\mathsf{dec}(\mathcal{N}(\mathsf{enc}(D)))$ using the same (Col, δ)-signature that is used by D, which simplifies the presentation. Our results could be easily extended to GNNs where the dimensions of the input and output layers differ, but then the input and output datasets would need to use distinct signatures.

An obvious consequence of Definition 2 is that the transformation $T_{\mathcal{N}}$ cannot derive binary facts: the binary facts in D and $T_{\mathcal{N}}(D)$ coincide for each for each (Col, δ)-dataset D. In Section 3.3, we discuss how one can overcome this issue in a way that still allows us to understand clearly the core properties of GNN models.

3.2. Relating GNNs to rules

The canonical encoding/decoding scheme bridges the gap between GNNs and rules in the sense that both can be seen as transformations of datasets. This, in turn, allows us to formally compare a transformation $T_{\mathcal{P}}$ induced by a program with a transformation $T_{\mathcal{N}}$ induced by a GNN as specified in Definition 4.

Definition 4. A Datalog program $\mathcal{P}$ is *sound* for a (Col, δ)-GNN $\mathcal{N}$ if $T_{\mathcal{P}}(D) \subseteq T_{\mathcal{N}}(D)$ for each (Col, δ)-dataset D. Conversely, $\mathcal{P}$ is *complete* for $\mathcal{N}$ if $T_{\mathcal{N}}(D) \subseteq T_{\mathcal{P}}(D)$ for each (Col, δ)-dataset D. Finally, $\mathcal{P}$ is *equivalent* to $\mathcal{N}$ if it is both sound and complete for $\mathcal{N}$.

In other words, if a program is sound or complete for a GNN, then the results obtained by applying the GNN to any dataset always contain or are contained in, respectively, the results obtained by applying the rules to the same dataset. A program equivalent to a GNN can thus be reliably used to explain all the predictions of the GNN on arbitrary input datasets. The central question that we address in this paper is to identify conditions under which a (Col, δ)-GNN $\mathcal{N}$ is equivalent to a Datalog program, and to determine whether such program can be computed from $\mathcal{N}$. Before we start answering these questions in Section 4, we briefly discuss how one can take into account other encoding schemes considered in the literature.

3.3. The effects of noncanonical encoding/decoding schemes

Several encodings that differ substantially from the canonical one have been proposed in the literature, and some were used to characterise the expressivity of GNNs. For example, to be able to derive binary facts, [35] presented an alternative encoding where the input graph is transformed into another graph with additional vertices whose feature vectors encodes both unary and binary facts. As another example, [30] presented a GNN variant that can express the k-WL isomorphism test, but an input graph in this approach is transformed substantially before passing it to the GNN. Preprocessing in these cases involves nontrivial transformations that can substantially increase the graph size, and it is not immediately clear whether and how this affects the expressivity of the end-to-end transformation.

In this section we present a framework that allows us to tease apart the effects of the encoding from the core properties of the GNNs. In particular, we argue that noncanonical encoding/decoding schemes can often be described by a pair of programs $\mathcal{P}_{\mathsf{enc}}$ and

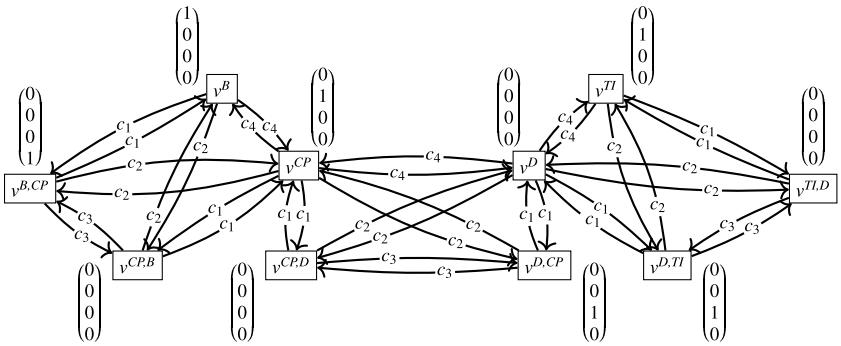


Fig. 2. Encoding of dataset D from Fig. 1 by [35].

$\mathcal{P}_{\mathsf{dec}}$, possibly expressed in an extension of Datalog, which convert a dataset into a (Col, δ)-dataset and back, respectively. 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 $T_{\mathcal{P}_{\mathsf{dec}}}(T_{\mathcal{N}}(T_{\mathcal{P}_{\mathsf{enc}}}(D)))$. 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. The shape of the rules and the Datalog constructs used in the three programs then allow us to attribute precisely the contribution of each step to the overall expressivity.

3.3.1. The encoding by [35]

[35] presented an encoding scheme that introduces vertices for pairs of constants in the input dataset. The encoding is applicable to a function-free dataset D over unary predicates $A_1, \dots, A_\epsilon$ and binary predicates $R_{\epsilon+1}, \dots, R_\delta$ (again, practical examples might use more meaningful predicate names), and it transforms D into a symmetric (Col, δ)-graph over 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 and b occurring together in a binary fact in D. All (i.e., unary and binary) predicates are assigned fixed positions in the 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. Edges of the coloured graph indicate different types of 'connections' between constants; for example, vertices v^a and $v^{a,b}$ are connected by edges of colour c_1 to indicate that constant a occurs first in the constant pair $\langle a, b\rangle$.

Fig. 2 shows the encoding of dataset D from Fig. 1. As in the canonical encoding, we introduce vertices v^B, v^{CP}, v^D, and v^{TI} to represent the constants of D. Furthermore, we additionally introduce vertices $v^{D,CP}$, $v^{CP,D}$, $v^{D,TI}$, $v^{TI,D}$, $v^{B,CP}$, and $v^{CP,B}$ to represent the relationships between the constants of D. We do not introduce vertices such as $v^{CP,TI}$ and $v^{TI,CP}$ because constants $Crime\_And\_Punishment$ and The_Idiot do not occur together in a fact of D. We then encode unary and binary facts using feature vectors labelling the graph's vertices. To this end, we assign the ith position in the feature vectors to the predicate with subindex i; thus, each feature vector contains an encoding of unary predicates followed by an encoding of binary predicates. In our example, *User*, *Novel*, *Author*, and *Likes* correspond to A_1, A_2, R_3, and R_4, respectively, so we assign positions 1, 2, 3, and 4, respectively, to these predicates. Then, we represent the facts in D by setting to one the relevant elements of feature vectors labelling the corresponding vertices. For example, to represent $Novel(The\_Idiot)$, we set the second element of the feature vector of v^{TI} to one; similarly, to represent $Likes(Bob, Crime\_and\_Punishment)$, we set the fourth element of the feature vector of $v^{B,CP}$ to one. Finally, we connect with coloured edges all pairs of vertices of G_D that refer to related constants. For example, we connect vertices v^D and $v^{D,CP}$ by edges of colour c_1 to indicate that *Dostoevsky* occurs in the first position of the constant pair of $v^{D,CP}$. Similarly, we connect $v^{D,CP}$ and v^{CP} by edges of colour c_2. We connect $v^{CP,D}$ and $v^{D,CP}$ by edges of colour c_3 to indicate that the constant pairs of the two vertices are inverses of each other. Finally, we connect v^{CP} and v^D by edges of colour c_4 to indicate that the two constants occur jointly in a fact of D.

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 the pair (a, b) using the term $g(a, b)$; thus, vertex $v^{a,b}$ in the original encoding can be represented by the vertex $v^{g(a,b)}$ in our encoding. For uniformity, we represent each constant a using the term $f(a)$. 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 $v^{g(a,b)}$. Based on this idea, the encoding program $\mathcal{P}_{\mathsf{enc}}$ contains the following rules instantiated for each $i \in \{1, \dots, \epsilon\}$ and each $j \in \{\epsilon + 1, \dots, \delta\}$.

$$A_i(x) \to U_i(f(x)) \qquad R_j(x, y) \to U_j(g(x, y))$$

$$R_j(x, y) \to E^{c_1}(f(x), g(x, y)) \qquad R_j(x, y) \to E^{c_1}(g(x, y), f(x)) \qquad R_j(x, y) \to E^{c_1}(f(y), g(y, x)) \qquad R_j(x, y) \to E^{c_1}(g(y, x), f(y))$$

$$R_j(x, y) \to E^{c_2}(f(y), g(x, y)) \qquad R_j(x, y) \to E^{c_2}(g(x, y), f(y)) \qquad R_j(x, y) \to E^{c_2}(f(x), g(y, x)) \qquad R_j(x, y) \to E^{c_2}(g(y, x), f(x))$$

$$R_j(x, y) \to E^{c_3}(g(x, y), g(y, x)) \qquad R_j(x, y) \to E^{c_3}(g(y, x), g(x, y))$$

$$R_j(x, y) \to E^{c_4}(f(x), f(y)) \qquad R_j(x, y) \to E^{c_4}(f(y), f(x))$$

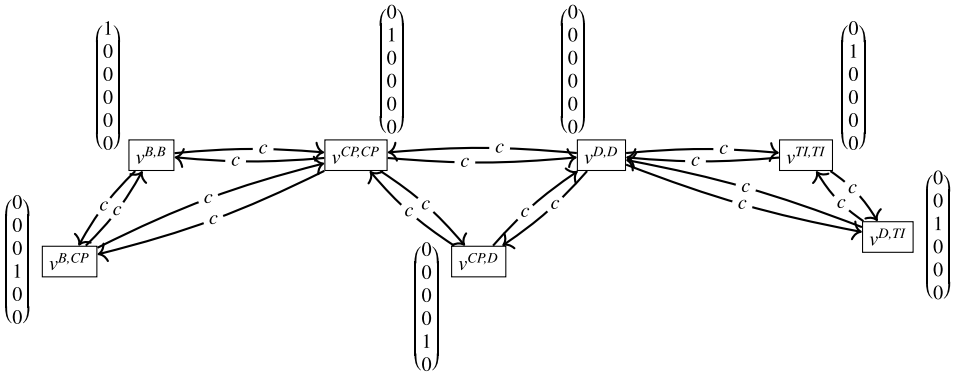


Fig. 3. Encoding of dataset D from Fig. 1 by [17].

The first two rules 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 $j \in \{\epsilon+1,\dots,\delta\}$. The remaining rules encode the adjacency relationships between terms: colour c_1 connects terms of the form $g(a,b)$ and $f(a)$, colour c_2 connects terms of the form $g(a,b)$ and $f(b)$, colour c_3 connects terms of the form $g(a,b)$ and $g(b,a)$, and colour c_4 connects terms of the form $f(a)$ and $f(b)$ where a and b occur jointly in a binary fact.

Program $\mathcal{P}_{\mathsf{dec}}$ capturing the decoder contains the following rules instantiated for each $i \in \{1,\dots,\epsilon\}$ and each $j \in \{\epsilon+1,\dots,\delta\}$. Intuitively, these rules just 'read off' the facts from the labels of the vertices of the form $v^{f(a)}$ and $v^{g(a,b)}$.

$$U_i(f(x)) \rightarrow A_i(x) \qquad\qquad U_j(g(x,y)) \rightarrow R_j(x,y)$$

It is straightforward to show that for each dataset D, the graph obtained by applying the encoding by [35] outlined earlier 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; moreover, the same can be said about the decoder. Thus, for each GNN $\mathcal{N}$ with an equivalent program $\mathcal{P}_{\mathcal{N}}$, the composition $T_{\mathcal{P}_{\mathsf{dec}}}(T_{\mathcal{P}_{\mathcal{N}}}(T_{\mathcal{P}_{\mathsf{enc}}}(D)))$ of the three programs is equivalent to the application of a (function-free) Datalog program.

A limitation of this encoding is that the transformation's output can contain a fact of the form $S(a,b)$ only if the input dataset contains a fact of the form $R(a,b)$ or $R(b,a)$. Intuitively, the presence of $R(a,b)$ or $R(b,a)$ in the input ensures that the resulting (Col, δ)-graph contains a vertex $v^{g(a,b)}$ for representing binary facts of the form $S(a,b)$ for S an arbitrary binary predicate. This limitation can be overcome by an alternative encoding that introduces terms $g(a,b)$ for *all* constants a and b occurring in the input, without requiring a and b to occur jointly in a fact of the input dataset. 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 the following ones for all combinations of unary and binary predicates, and colours. The chaining of $\mathcal{P}_{\mathsf{enc}}$, $\mathcal{P}_{\mathcal{N}}$, and $\mathcal{P}_{\mathsf{dec}}$ now captures a different transformation even if $\mathcal{P}_{\mathcal{N}}$ remains the same.

$$A_i(x) \wedge A_j(y) \rightarrow E^{c_1}(f(x), g(x,y)) \qquad A_i(x) \wedge A_j(y) \rightarrow E^{c_1}(g(x,y), f(x))$$
$$R_i(x,z) \wedge A_j(y) \rightarrow E^{c_1}(f(x), g(x,y)) \qquad R_i(z,x) \wedge A_j(y) \rightarrow E^{c_1}(f(x), g(x,y))$$
$$R_i(x,z) \wedge A_j(y) \rightarrow E^{c_1}(f(y), g(y,x)) \qquad R_i(z,x) \wedge A_j(y) \rightarrow E^{c_1}(f(y), g(y,x))$$
$$R_i(x,z) \wedge A_j(y) \rightarrow E^{c_1}(g(x,y), f(x)) \qquad R_i(z,x) \wedge A_j(y) \rightarrow E^{c_1}(g(x,y), f(x))$$
$$R_i(x,z) \wedge A_j(y) \rightarrow E^{c_1}(g(y,x), f(y)) \qquad R_i(z,x) \wedge A_j(y) \rightarrow E^{c_1}(g(y,x), f(y))$$

3.3.2. *The encoding by [17]*

The encoding presented in Section 3.3.1 is a variant of an encoding initially introduced by Liu et al. [17]. Their encoding is applicable to a function-free dataset D over unary predicates $A_1,\dots,A_\epsilon$ and binary predicates $R_{\epsilon+1},\dots,R_{\epsilon+\epsilon'}$, where $\epsilon, \epsilon' \in \mathbb{N}$. The encoding transforms D into a symmetric (Col, δ)-graph over a single colour $\mathsf{Col} = \{c\}$ where $\delta = \epsilon + 2\epsilon'$. Moreover, all constants in D must be ordered using an arbitrary, but fixed total order $\prec$, and the encoded graph contains a vertex $v^{a,b}$ for each pair of (not necessarily distinct) constants a and b that occur jointly in a fact of D and that satisfy $a \preceq b$. As in Section 3.3.1, all predicates are assigned fixed positions in vectors; however, additional positions are introduced to represent inverse connections. For example, assuming $a \preceq b$, facts of the form $R_j(a,b)$ and $R_j(b,a)$ are mapped to distinct positions in the feature vector labelling $v^{a,b}$: $(\mathbf{v}^{a,b})_j$ is set to 1 if $R_j(a,b) \in D$, and $(\mathbf{v}^{a,b})_{j+\epsilon'}$ is set to 1 if $R_j(b,a) \in D$. Unary predicates for a constant a are represented in the vertex $v^{a,a}$: $(\mathbf{v}^{a,a})_i$ is set to 1 if $A_i(a) \in D$. Edges connect pairs of vertices that have a constant in common. For example, vertices $v^{a,a}$ and $v^{a,b}$ are connected, as are vertices $v^{a,b}$ and $v^{b,c}$.

Fig. 3 shows the encoding of the dataset D from Fig. 1, where we assume that constants are ordered lexicographically. We introduce vertices $v^{B,B}$, $v^{CP,CP}$, $v^{D,D}$, and $v^{TI,TI}$ in G_D to represent the constants of D, and we additionally introduce vertices $v^{CP,D}$, $v^{D,TI}$, and $v^{B,CP}$ to represent the relationships between constants. As in Section 3.3.1, we do not introduce vertices for pairs of constants that do

not occur together in a fact of D; however, we also avoid introducing vertices such as $v^{D,CP}$ and $v^{TI,D}$ because the constants labelling these vertices are not ordered lexicographically. Now let *User*, *Novel*, *Author*, and *Likes* correspond to predicates A_1, A_2, R_3, and R_4, respectively, so these predicates are mapped to positions 1 to 4 of the feature vectors, and let positions 5 and 6 correspond to the inverse connections via binary predicates *Author* and *Likes*, respectively. We represent the facts in D by setting to one the relevant elements of feature vectors labelling the corresponding vertices. Note that the binary fact $Author(Dostoevsky, Crime\_and\_Punishment)$ is represented in position 5 of the vertex $v^{CP,D}$ because it is an inverse connection for the pair of constants $\langle Crime\_and\_Punishment, Dostoevsky\rangle$. Finally, we introduce c-coloured edges between all pairs of vertices of G_D that mention the same constant.

We next show how to capture this encoding using rules. As in Section 3.3.1, we require value invention, so we represent the vertex $v^{a,b}$ in the original encoding using a vertex $v^{f(a,b)}$. In doing so, however, we must honour the ordering on constants. We can achieve this in Datalog with ordering, where the body of a rule is allowed to refer to a built-in predicate $\preceq$ representing an arbitrary total ordering on constants. Literal $x \preceq y$ is satisfied by each substitution μ such that $x\mu \preceq y\mu$. With such an extension in place, the encoding program $\mathcal{P}_{\mathsf{enc}}$ contains the rules

$$A_i(x) \to U_i(f(x,x))$$
$$R_j(x,y) \wedge x \preceq y \to U_j(f(x,y)) \qquad R_j(y,x) \wedge x \preceq y \to U_{j+\epsilon'}(f(x,y))$$

instantiated for each $i \in \{1,\dots,\epsilon\}$ and each $j \in \{\epsilon+1,\dots,\epsilon+\epsilon'\}$, and the rules

$$\alpha \wedge \beta \wedge x \preceq y \to E^c(f(x,x), f(x,y)) \qquad \alpha \wedge \beta \wedge x \preceq y \to E^c(f(x,y), f(x,x))$$
$$\beta \wedge x \preceq y \wedge \gamma \wedge x \preceq z \to E^c(f(x,y), f(x,z)) \qquad \beta \wedge x \preceq y \wedge \gamma \wedge x \preceq z \to E^c(f(x,z), f(x,y))$$
$$\beta \wedge y \preceq x \wedge \gamma \wedge x \preceq z \to E^c(f(y,x), f(x,z)) \qquad \beta \wedge y \preceq x \wedge \gamma \wedge x \preceq z \to E^c(f(x,z), f(y,x))$$
$$\beta \wedge y \preceq x \wedge \gamma \wedge z \preceq x \to E^c(f(y,x), f(z,x)) \qquad \beta \wedge y \preceq x \wedge \gamma \wedge z \preceq x \to E^c(f(z,x), f(y,x))$$

instantiated for all $\alpha \in \{A_i(x), A_i(y) \mid 1 \le i \le \epsilon\}$, $\beta \in \{R_j(x,y), R_j(y,x) \mid \epsilon < j \le \epsilon+\epsilon'\}$, and $\gamma \in \{R_k(x,z), R_k(z,x) \mid \epsilon < k \le \epsilon+\epsilon'\}$. The first three rules ensure that all unary and binary facts in the input dataset are encoded as facts of the form $U_i(f(a,a))$ and $U_j(f(a,b))$, and the remaining rules connect each pair of terms that share a constant.

The decoding program $\mathcal{P}_{\mathsf{dec}}$ contains the following rules instantiated for each $i \in \{1,\dots,\epsilon\}$ and each $j \in \{\epsilon+1,\dots,\epsilon+\epsilon'\}$, which simply read off the relevant predicates from the constants in the transformed dataset.

$$U_i(f(x,x)) \to A_i(x) \qquad U_j(f(x,y)) \to R_j(x,y) \qquad U_{j+\epsilon'}(f(x,y)) \to R_j(y,x)$$

3.3.3. The encoding by [30]

[30] 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 $i \in \{1,\dots,\delta_1\}$. 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, under certain assumptions, the transformation of $\mathcal{G}_1$ into $\mathcal{G}_2$ can be captured by a program $\mathcal{P}_{\mathsf{enc}}$ that transforms a function-free (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 is a consequence of the encoding since the k-GNN model itself operates similarly to a standard GNN. To achieve a concise, more readable presentation, we first make some simplifying and nonessential assumptions, and we discuss their impact at the end of this section. First, while [30] 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 the simpler, so-called *local neighbourhood* approach for connecting vertices, where two vertices are connected if they differ in exactly one constant, and the distinct constants are connected in the original graph. Our encoding also requires extending Datalog not only with function symbols, but also with stratified negation-as-failure not [5]. Under these assumptions, the encoding program $\mathcal{P}_{\mathsf{enc}}$ consists of the following rules instantiated for all $i,j,k \in \{1,\dots,\delta_1\}$.

$$A_i(x) \wedge A_j(y) \wedge A_k(z) \wedge x \not\approx y \wedge x \not\approx z \wedge y \not\approx z \wedge R(y,z) \to E^c(g(x,y), g(x,z))$$
$$A_i(x) \wedge A_j(y) \wedge A_k(z) \wedge x \not\approx y \wedge x \not\approx z \wedge y \not\approx z \wedge R(y,z) \to E^c(g(y,x), g(z,x))$$
$$A_i(x) \wedge A_j(y) \wedge x \not\approx y \wedge \mathsf{not}\ R(x,y) \to U_{i,j,0}(g(x,y))$$
$$A_i(x) \wedge A_j(y) \wedge x \not\approx y \wedge R(x,y) \to U_{i,j,1}(g(x,y))$$

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 $\mathcal{G}_2$). The first two rules 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, the last two rules identify the type of the subgraph of $\mathcal{G}_1$ that a and b participate in. Specifically, a fact of the form $U_{i,j,0}(g(a,b))$ states 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 $\mathcal{G}_1$.

Fig. 4 shows the encoding of our running example dataset. For example, the encoding includes an edge $\langle v^{B,D}, v^{B,TI}\rangle$ because constant *Bob* appears in the first position of the pairs annotating these two vertices, and the fact $Author(Dostoevsky, The\_Idiot)$ appears in D. Binary predicates are treated analogously. Note that the resulting graph has two connected components, but this is not necessarily the case in general. For example, if we extend D with a binary fact mentioning both $Crime\_And\_Punishment$ and The_Idiot in this

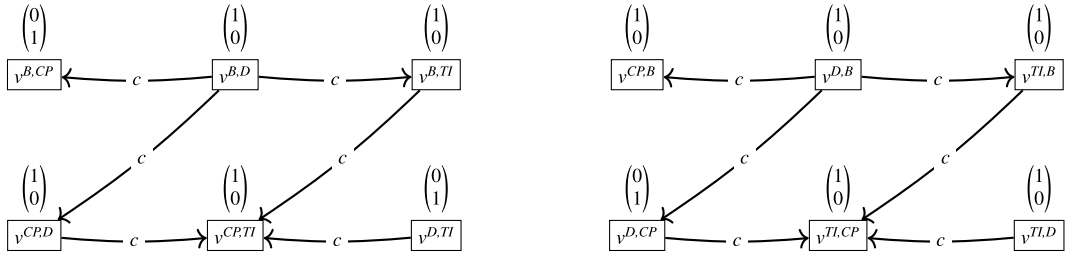


Fig. 4. Encoding of dataset D from Fig. 1 by [30].

order, then the encoding of D consists of a single connected component since Fig. 4 would additionally contain a c-labelled edge from $v^{D,CP}$ to $v^{D,TI}$.

Our assumptions about the encoding can be relaxed. First, to cover larger values of k, we can use analogous rules with more body atoms and function symbols of arity k in the head. Second, to extend our treatment to sets of vertices, we can use a single tuple to represent each set (e.g., by considering a total order on the set of all possible vertices). Third, covering the more general *global neighbourhood* approach to connecting vertices in $\mathcal{G}_2$ requires using unsafe rules and additional negated body atoms.

4. Monotonic GNNs with max-sum aggregation

In this section we introduce monotonic max-sum GNNs. A key feature of our definition is the use of the max-k-sum aggregation function, which generalises both max and sum aggregation.

Definition 5. For each $k \in \mathbb{N}_0 \cup \{\infty\}$, each finite multiset $S \in \mathcal{F}(\mathbb{R}_0^+)$ of nonnegative reals, and $\ell = \min(k, |S|)$, let

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

Each occurrence of a number is counted separately in this definition; for example, max-3-sum$\{\{ 0, 1, 1, 2, 2, 2, 5 \}\} = 9$ because the three largest numbers in S are the one occurrence of 5 and two occurrences of 2. Note that max-1-sum is equivalent to max, and that max-∞-sum is equivalent to sum.

Monotonic max-sum GNNs, which we define next, are restricted to satisfy two key properties. First, we will show that transformations induced by max-sum GNNs are monotonic under injective homomorphisms, thus capturing the essence of Datalog rule application as described in Section 2.1. Second, we will show that we can replace each occurrence of max-∞-sum with max-k-sum for some nonnegative integer k without changing the output of the transformation on any (Col, δ)-dataset. Note that the choice of such k is independent of the dataset, which in turn means that the ability to aggregate over an unbounded number of vertex neighbours does not contribute to the GNN's expressivity. These two properties are ensured by requiring matrix weights to be nonnegative and by placing suitable restrictions on the activation and classification functions. We show later that dataset transformations induced by such GNNs can be captured using Datalog rules with inequalities in the body, where the latter provide counting capabilities. In addition, we also introduce the subclass of monotonic max GNNs, where the aggregation function in each layer is restricted to max. We show later that such GNNs are monotonic under (not necessarily injective) homomorphisms and thus, similarly to Datalog without inequalities, lack counting capabilities.

Definition 6. A *monotonic max-sum* (Col, δ)-GNN is a GNN of form (3) satisfying the following conditions:

1. 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;
2. for each $\ell \in \{1, \dots, L\}$, the aggregation function agg_ℓ is max-k_ℓ-sum for some $k_\ell \in \mathbb{N}_0 \cup \{\infty\}$;
3. the activation function σ is monotonically increasing and unbounded, and the codomain of σ is $\mathbb{R}_0^+$; and
4. 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 $t' \geq t$.

Such a GNN is a *monotonic max* GNN if the aggregation function in each of its layers is max-1-sum.

The aggregation functions in Definition 6 are applicable only to real multisets that contain just nonnegative elements, but this is not a problem: we shall apply monotonic max-sum GNNs to regular (and thus Boolean) graphs, and the activation function ranges over nonnegative numbers; hence, each vertex in each layer is labelled by nonnegative numbers only. Furthermore, note that ReLU satisfies Condition 3, but the ELU and the sigmoid functions are incompatible with this condition.

The conditions of Definition 6 allow us to prove Proposition 1. To simplify our presentation, we assume here and in the rest of this paper that all (Col, δ)-datasets are function-free. This assumption is without loss of generality because we will focus on canonical transformations induced by a GNN and transformations induced by a Datalog program, both of which treat ground terms with function symbols in the same way as constants.

Proposition 1. *For each monotonic max-sum* (Col, δ)*-GNN* $\mathcal{N}$*, operator* $T_{\mathcal{N}}$ *is monotonic under injective homomorphisms; moreover, if* $\mathcal{N}$ *is a monotonic max GNN, then* $T_{\mathcal{N}}$ *is monotonic under homomorphisms.*

Proof. To prove that $T_{\mathcal{N}}$ is monotonic under injective homomorphisms, we consider arbitrary (Col, δ)-datasets D and D', as well as an arbitrary injective homomorphism h from D to D', and we show that $h(T_{\mathcal{N}}(D)) \subseteq T_{\mathcal{N}}(D')$. 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. Moreover, let $\lambda_0, \dots, \lambda_L$ and $\lambda'_0, \dots, \lambda'_L$ be the sequences of labellings of vertices of $\mathcal{G}$ and $\mathcal{G}'$, respectively, after applying $\mathcal{N}$ to the two graphs.

Graphs $\mathcal{G}$ and $\mathcal{G}'$ satisfy the following property $(*)$: for each edge $\langle v^a, v^b \rangle \in \mathcal{E}^c$, we have $\langle v^{h(a)}, v^{h(b)} \rangle \in \mathcal{E}'^c$. In particular, $\langle v^a, v^b \rangle \in \mathcal{E}^c$ implies $E^c(a, b) \in D$ by the definition of the canonical encoding; since h is an injective homomorphism from D to D', we have $E^c(h(a), h(b)) \in D'$, which in turn ensures $\langle v^{h(a)}, v^{h(b)} \rangle \in \mathcal{E}'^c$.

We next prove by induction on ℓ the following property $(\diamondsuit)$: for each $\ell \in \{0, \dots, L\}$, each $i \in \{1, \dots, \delta_\ell\}$, and each $v^a \in \mathcal{V}$, we have $(\mathbf{v}^a_{\lambda_\ell})_i \leq (\mathbf{v}^{h(a)}_{\lambda'_\ell})_i$.

For the base case $\ell = 0$, consider arbitrary $v^a \in \mathcal{V}$ and $i \in \{1, \dots, \delta\}$. By the definition of the canonical encoding, $(\mathbf{v}^a_{\lambda_0})_i \in \{0, 1\}$, and $(\diamondsuit)$ holds trivially if $(\mathbf{v}^a_{\lambda_0})_i = 0$. Moreover, if $(\mathbf{v}^a_{\lambda_0})_i = 1$, the definition of the canonical encoding ensures $U_i(a) \in D$; since h is an injective homomorphism from D to D', we have $U_i(h(a)) \in D'$; but then, the definition of the canonical encoding ensures $(\mathbf{v}^{h(a)}_{\lambda'_0})_i = 1$. Thus, $(\diamondsuit)$ holds, as required.

For the induction step, assume that $(\diamondsuit)$ holds for some $\ell - 1$, and consider an arbitrary vertex $v^a \in \mathcal{V}$ and arbitrary $i \in \{1, \dots, \delta_\ell\}$. By definition, the values of $(\mathbf{v}^a_{\lambda_\ell})_i$ and $(\mathbf{v}^{h(a)}_{\lambda'_\ell})_i$ are computed by Eqs. (5) and (6), respectively.

$$(\mathbf{v}^a_{\lambda_\ell})_i = \sigma\left(\sum_{j=1}^{\delta_{\ell-1}} (\mathbf{A}_\ell)_{i,j} \, (\mathbf{v}^a_{\lambda_{\ell-1}})_j + \sum_{c \in \mathsf{Col}} \sum_{j=1}^{\delta_{\ell-1}} (\mathbf{B}^c_\ell)_{i,j} \, \text{max-}k_\ell\text{-sum}\{\!\{ \, (\mathbf{w}_{\lambda_{\ell-1}})_j \mid \langle v^a, w \rangle \in \mathcal{E}^c \, \}\!\} + (\mathbf{b}_\ell)_i \right) \tag{5}$$

$$(\mathbf{v}^{h(a)}_{\lambda'_\ell})_i = \sigma\left(\sum_{j=1}^{\delta_{\ell-1}} (\mathbf{A}_\ell)_{i,j} \, (\mathbf{v}^{h(a)}_{\lambda'_{\ell-1}})_j + \sum_{c \in \mathsf{Col}} \sum_{j=1}^{\delta_{\ell-1}} (\mathbf{B}^c_\ell)_{i,j} \, \text{max-}k_\ell\text{-sum}\{\!\{ \, (\mathbf{w}'_{\lambda'_{\ell-1}})_j \mid \langle v^{h(a)}, w' \rangle \in \mathcal{E}'^c \, \}\!\} + (\mathbf{b}_\ell)_i \right) \tag{6}$$

By the induction hypothesis, $(\mathbf{v}^a_{\lambda_{\ell-1}})_j \leq (\mathbf{v}^{h(a)}_{\lambda'_{\ell-1}})_j$ for each $j \in \{1, \dots, \delta_{\ell-1}\}$; moreover, the elements of $\mathbf{A}_\ell$ are nonnegative since $\mathcal{N}$ is a monotonic max-sum GNN. Thus, all components of these vectors are nonnegative, so the first sum in the argument of σ in Eq. (5) is smaller or equal than that in Eq. (6). Furthermore, consider an arbitrary colour $c \in \mathsf{Col}$ and an arbitrary $j \in \{1, \dots, \delta_{\ell-1}\}$, and let S be an arbitrary set of up to k_ℓ vertices such that $\langle v^a, w^b \rangle \in \mathcal{E}^c$ for each $w^b \in S$ and

$$\text{max-}k_\ell\text{-sum}\{\!\{ \, (\mathbf{w}_{\lambda_{\ell-1}})_j \mid \langle v^a, w \rangle \in \mathcal{E}^c \, \}\!\} = \sum_{w^b \in S} (\mathbf{w}^b_{\lambda_{\ell-1}})_j.$$

Such S exists even if $k_\ell = \infty$ since vertex v^a has a finite number of neighbours in $\mathcal{G}$. Property $(*)$ ensures $\langle v^{h(a)}, w^{h(b)} \rangle \in \mathcal{E}'^c$ for each $w^b \in S$; moreover, h is injective so the edge is distinct for each w^b. This fact and the induction hypothesis ensure that

$$\text{max-}k_\ell\text{-sum}(\{\!\{ \, (\mathbf{w}_{\lambda_{\ell-1}})_j \mid \langle v^a, w \rangle \in \mathcal{E}^c \, \}\!\}) = \sum_{w^b \in S} (\mathbf{w}^b_{\lambda_{\ell-1}})_j \leq \sum_{w^b \in S} (\mathbf{w}^{h(b)}_{\lambda'_{\ell-1}})_j \leq \text{max-}k_\ell\text{-sum}(\{\!\{ \, (\mathbf{w}'_{\lambda'_{\ell-1}})_j \mid \langle v^{h(a)}, w' \rangle \in \mathcal{E}'^c \, \}\!\}).$$

All elements of $\mathbf{B}^c_\ell$ are nonnegative, so the second sum in the argument of σ in Eq. (5) is smaller or equal than the corresponding sum in Eq. (6). Thus, the argument of σ in Eq. (5) is smaller or equal than the argument of σ in Eq. (6). Finally, function σ is monotonically increasing, which ensures $(\diamondsuit)$, as required.

We now complete the proof that h is an injective homomorphism from $T_{\mathcal{N}}(D)$ to $T_{\mathcal{N}}(D')$. All constants of $T_{\mathcal{N}}(D)$ appear in D, so h is defined on all constants of $T_{\mathcal{N}}(D)$. To prove $h(T_{\mathcal{N}}(D)) \subseteq T_{\mathcal{N}}(D')$, we consider the possible forms of facts in $T_{\mathcal{N}}(D)$.

- Consider an arbitrary unary fact $U_i(a) \in T_{\mathcal{N}}(D)$. The definition of the canonical decoding ensures $\mathsf{cls}((\mathbf{v}^a_{\lambda_L})_i) = 1$. Property $(\diamondsuit)$ implies $(\mathbf{v}^a_{\lambda_L})_i \leq (\mathbf{v}^{h(a)}_{\lambda'_L})_i$. But then, $\mathsf{cls}((\mathbf{v}^{h(a)}_{\lambda'_L})_i) = 1$ since cls is a step function and is thus monotonically increasing. Finally, the definition of the canonical decoding ensures $U_i(h(a)) \in T_{\mathcal{N}}(D')$, as required.
- Consider an arbitrary binary fact $E^c(a, b) \in T_{\mathcal{N}}(D)$. The definition of canonical decoding ensures that $\mathcal{N}(\mathcal{G})$ contains the edge $\langle v^a, v^b \rangle$; moreover, applying $\mathcal{N}$ to $\mathcal{G}$ does not change the graph edges, so $\langle v^a, v^b \rangle \in \mathcal{E}^c$. Property $(*)$ then ensures $\langle v^{h(a)}, v^{h(b)} \rangle \in \mathcal{E}'^c$, and so $\mathcal{N}(\mathcal{G}')$ contains the edge $\langle v^{h(a)}, v^{h(b)} \rangle$. But then, $E^c(h(a), h(b)) \in T_{\mathcal{N}}(D')$, as required.

The proof for the case when $\mathcal{N}$ is a monotonic max GNNs is analogous, with the difference that $k_\ell = 1$ in each layer ensures that the result of neighbour aggregation is now determined by a single neighbouring vertex. We thus do not need to argue existence of a distinct $w^{h(b)}$ for each $w^b \in S$, which is the only part of the proof that depends on h being injective. □

In Section 7.1, Theorem 3, we show that the conditions of Definition 6 also ensure that unbounded aggregation can be replaced with aggregation over a bounded number of neighbours without affecting the output on any input dataset.

5. Checking rule soundness

In this section, we present a method for checking whether a constant-free Datalog rule is sound for a monotonic max-sum GNN. For example, we can verify whether the recommender system from Section 1 recommends books written by authors whose other

books have been liked by the user—a property expressed by the rule (1). Our method is suitable for practical use. Moreover, in the following sections we present several algorithms for extracting an equivalent program from a monotonic max-sum GNN that work by enumerating candidate rules of a certain form and applying our soundness check to each candidate.

Before proceeding, we rule out two edge cases. First, applying a GNN to a graph does not affect the graph edges, which means that applying $T_{\mathcal{N}}$ to a dataset cannot derive any binary facts. Thus, a rule with a binary head atom can be sound for $\mathcal{N}$ only if it is a tautology. Second, the result of applying $T_{\mathcal{N}}$ to any dataset does not depend on the identity of constants in the dataset. Thus, if a rule with constants is sound for $\mathcal{N}$, this rule is necessarily implied by another constant-free rule that is also sound for $\mathcal{N}$. In other words, GNNs cannot represent rules with binary facts in the head, or rules with constants. Consequently, we restrict our attention to checking soundness of an arbitrary constant-free rule r with a unary head atom of the form $U(x)$.

Checking if such a rule r is sound for a monotonic max-sum GNN $\mathcal{N}$ directly using Definition 4 would involve applying T_r and $T_{\mathcal{N}}$ to infinitely many (Col, δ)-datasets, which is clearly impossible in practice. However, $T_{\mathcal{N}}$ is monotonic under injective homomorphisms by Proposition 1, which allows for a straightforward and practical test described in Proposition 2.

Proposition 2. *Let $\mathcal{N}$ be a monotonic max-sum (Col, δ)-GNN, and let r be a constant-free Datalog rule over the (Col, δ)-signature with a unary head atom $U(x)$, a possibly empty set A of body atoms, and a possibly empty set I of body inequalities. Moreover, for each variable y occurring in r, let a_y be a distinct constant uniquely associated with y, and let b be another constant distinct from all a_y. Then, r is sound for $\mathcal{N}$ if and only if, for each substitution μ mapping the variables of r to the constants a_y such that $\mu(y_1) \neq \mu(y_2)$ for each inequality $y_1 \not\approx y_2 \in I$, it is the case that $U(x)\mu \in T_{\mathcal{N}}(D_\mu)$, where $D_\mu = A\mu$ if the head variable x occurs in A, and otherwise $D_\mu = A\mu \cup \{E^c(b, \mu(x))\}$ for $c \in \mathsf{Col}$ an arbitrary but fixed colour.*

Proof. Fix an arbitrary GNN $\mathcal{N}$, rule r with $U(x)$, A, and I, as in the proposition, and distinct constants b and a_y uniquely associated with each variable y occurring in r.

For the ($\Rightarrow$) direction of the claim, assume that rule r is sound for $\mathcal{N}$ and consider an arbitrary substitution μ and dataset D_μ as specified in the claim. Each constant in the range of μ appears in D_μ: the image of each body variable clearly appears in $A\mu$, and the image of the head variable x also appears in $A\mu$ if x occurs in A, or in the additional fact $E^c(b, \mu(x))$ if x does not occur in A. Hence, μ is a substitution mapping the variables of r to the constants of D_μ, so $A\mu \subseteq D_\mu$ implies $U(x)\mu \in T_r(D_\mu)$ by the definition of T_r. Finally, r is sound for $\mathcal{N}$, so $U(x)\mu \in T_{\mathcal{N}}(D_\mu)$, as required.

For the ($\Leftarrow$) direction, assume that $U(x)\mu \in T_{\mathcal{N}}(D_\mu)$ for each substitution μ as specified in the claim, and consider an arbitrary (Col, δ)-dataset D; we next prove $T_r(D) \subseteq T_{\mathcal{N}}(D)$. To this end, consider an arbitrary fact in $T_r(D)$. Due to the shape of r, this fact is of the form $U(x)\nu$ for some substitution ν from the variables of r to the constants in D such that $D \models B_i\nu$ for each literal B_i (i.e., an atom of A or an inequality of I) in the body of r. Let h be the function that maps each constant d in the range of ν to a_y where y is an arbitrarily chosen variable of r such that $\nu(y) = d$—that is, if r contains several y such that $\nu(y) = d$, we choose one such y and define $h(d) = a_y$. Note that h is an injective mapping from the range of ν to the set $\{a_y \mid y \text{ is a variable of } r\}$. Now, let μ be the substitution defined by $\mu(y) = h(\nu(y))$ for each variable y of r, and let D_μ be as specified in the claim. Note that $D \models B_i\nu$ and the fact that h is injective ensure $D_\mu \models B_i\mu$ for each $B_i \in I$, and so μ is a substitution as specified in the claim. From this, it follows that $U(x)\mu \in T_r(D_\mu)$ as in the proof of the ($\Rightarrow$) direction. Now, let D' be D extended with the fact $E^c(b, \nu(x))$ if head variable x does not occur not in A, and $D' = D$ otherwise. Furthermore, let g be the mapping of constants to constants that is defined as h^{-1} on all constants in the range of μ and as the identity on all other constants. Since r is constant-free, mapping g is clearly an injective homomorphism from D_μ to D', so g is also an injective homomorphism from $T_{\mathcal{N}}(D_\mu)$ to $T_{\mathcal{N}}(D')$ by Proposition 1, which in turn implies that $g(T_{\mathcal{N}}(D_\mu)) \subseteq T_{\mathcal{N}}(D')$. Recall that $U(x)\mu \in T_{\mathcal{N}}(D_\mu)$, and so $U(x)\nu \in T_{\mathcal{N}}(D')$, where we used the observation that $g(U(x)\mu) = h^{-1}(U(x)\mu) = U(x)\nu$. Finally, note that the possible extra fact in D' that is not in D connects a constant not in D to a constant in D, but not the other way round; thus, Eq. (3) ensures that $T_{\mathcal{N}}(D')$ and $T_{\mathcal{N}}(D)$ coincide on facts derived over the constants of D. Since all constants in the range of ν are in D, we have $U(x)\nu \in T_{\mathcal{N}}(D)$, as required. □

In the definition of D_μ, the possible additional fact over E^c ensures that the relevant head atom $U(x)\mu$ is derived if the rule r is sound, but unsafe: as explained in Section 2.1, $U(x)\mu$ is then derived for each substitution μ that satisfies the body of r and maps the unsafe variable of $U(x)$ to an arbitrary constant of D_μ.

When $\mathcal{N}$ is a monotonic max GNN, a more efficient soundness check is possible: instead of exploring all possible groundings of r, we can check just one grounding where each variable is mapped to a distinct constant. This is captured by Proposition 3.

Proposition 3. *Let $\mathcal{N}$ be a monotonic max (Col, δ)-GNN, and let r be a constant-free Datalog rule over the (Col, δ)-signature with a unary head atom $U(x)$ and a possibly empty set A of body atoms. Moreover, let a_y and b be constants as in Proposition 2; let μ be the substitution defined by $\mu(y) = a_y$ for each variable y occurring in r; and let D_μ be as in Proposition 2. Then, r is sound for $\mathcal{N}$ if and only if $U(x)\mu \in T_{\mathcal{N}}(D_\mu)$.*

Proof. Fix $\mathcal{N}$, r, $U(x)$, A, I, a_y, b, μ, and D_μ as in the proposition. The proof of the ($\Rightarrow$) direction is as in Proposition 2, so we focus on the ($\Leftarrow$) direction. Assume that $U(x)\mu \in T_{\mathcal{N}}(D_\mu)$, and consider an arbitrary (Col, δ)-dataset D; we next prove that $T_r(D) \subseteq T_{\mathcal{N}}(D)$. To this end, consider an arbitrary fact $U(x)\nu \in T_r(D)$ for ν a substitution satisfying $D \models B_i\nu$ for each atom B_i in the body of r. Let h be the mapping of constants to constants defined as $h(\mu(y)) = \nu(y)$ on the constants in the range of μ and as the identity on other constants. This mapping is well defined since μ is injective and ν is defined for each variable y in r. Let D' be D extended with the fact $E^c(b, \nu(x))$ if the head variable x does not occur in A, and $D' = D$ otherwise. Since $A\nu \subseteq D$, mapping h is a homomorphism from D_μ to D'. By Proposition 1, mapping h is then also a homomorphism from $T_{\mathcal{N}}(D_\mu)$ to $T_{\mathcal{N}}(D')$, so $h(T_{\mathcal{N}}(D_\mu)) \subseteq T_{\mathcal{N}}(D')$. Moreover, $U(x)\mu \in T_{\mathcal{N}}(D_\mu)$ by our assumption, which implies $h(U(x)\mu) = U(x)\nu \in T_{\mathcal{N}}(D')$. Again, as in the proof of Proposition 2, $T_{\mathcal{N}}(D')$ and $T_{\mathcal{N}}(D)$ coincide on the facts over the constants of D, so $U(x)\nu \in T_{\mathcal{N}}(D)$, as required. □

The following example shows how to apply Propositions 2 and 3.

Example 1. Let $\mathcal{N}$ be a monotonic max-sum $(\{c\}, 2)$-GNN with one layer where $\mathbf{A}_1$, $\mathbf{B}_1^c$, and $\mathbf{b}_1$ are defined as shown below, the activation function is ReLU, the aggregation function is max, and the classification function is a step function with threshold one.

$$\mathbf{A}_1 = \mathbf{B}_1^c = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \qquad \mathbf{b}_1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

To test whether rules $r_1 = \top \to U_1(x)$ and $r_2 = \top \to U_2(x)$ are sound for $\mathcal{N}$ using Proposition 2, we introduce distinct constants a_y and b. Both rules contain just one variable, so we need to consider just one substitution $\mu = \{x \mapsto a_x\}$. Thus, we have $D_\mu = \{E^c(b, a_x)\}$, and applying $\mathcal{N}$ to the canonical encoding of D_μ shows that $T_\mathcal{N}(D_\mu) = \{U_1(a_x), U_1(b)\}$. By Proposition 2, $U_1(a_x) \in T_\mathcal{N}(D_\mu)$ implies that r_1 is sound for $\mathcal{N}$, and $U_2(a_x) \notin T_\mathcal{N}(D_\mu)$ implies that r_2 is not sound for $\mathcal{N}$. Furthermore, $\mathcal{N}$ is a max GNN, so Proposition 3 is applicable, and it allows us to reach the same conclusion using the same substitution μ.

To further illustrate Proposition 2, we next consider rule $r_3 = U_1(x) \wedge E^c(x, y) \wedge E^c(x, z) \wedge U_1(y) \wedge U_1(z) \wedge y \not\approx z \to U_2(x)$. There are 27 different substitutions mapping the variables of r to the constants a_x, a_y, and a_z, but only 18 of them respect the inequality $y \not\approx z$. However, discarding symmetric cases leaves only the following substitutions μ_1 and μ_2. These result in datasets D_{μ_1} and D_{μ_2}, that in turn lead to datasets $T_\mathcal{N}(D_{\mu_1})$ and $T_\mathcal{N}(D_{\mu_2})$, respectively, as follows.

$$\mu_1 = \{x \mapsto a_x, y \mapsto a_y, z \mapsto a_z\} \qquad \mu_2 = \{x \mapsto a_y, y \mapsto a_y, z \mapsto a_z\}$$
$$D_{\mu_1} = \{U_1(a_x), E^c(a_x, a_y), E^c(a_x, a_z), U_1(a_y), U_1(a_z)\} \qquad D_{\mu_2} = \{U_1(a_y), E^c(a_y, a_y), E^c(a_y, a_z), U_1(a_y), U_1(a_z)\}$$
$$T_\mathcal{N}(D_{\mu_1}) = \{U_1(a_x), U_1(a_y), U_1(a_z), U_2(a_x)\} \qquad T_\mathcal{N}(D_{\mu_2}) = \{U_1(a_y), U_1(a_z), U_2(a_y)\}$$

Now $U_2(x)\mu_1 \in T_\mathcal{N}(D_{\mu_1})$ and $U_2(x)\mu_2 \in T_\mathcal{N}(D_{\mu_2})$, so Proposition 2 allows us to conclude that r_3 is sound for $\mathcal{N}$.

6. Explaining specific derivations

Towards computing a Datalog program that is equivalent to a monotonic max-sum GNN, in this section we focus on explaining a GNN's derivations on a *specific dataset*—that is, given a monotonic max-sum (Col, δ)-GNN $\mathcal{N}$, a (Col, δ)-dataset D, and a fact $U_i(a) \in T_\mathcal{N}(D)$, we show how to construct a Datalog rule r that is sound for $\mathcal{N}$ and that ensures $U_i(a) \in T_r(D)$.

Unless we restrict the syntactic structure of r, this problem can be solved trivially: we let D' be the conjunction obtained from D by replacing each constant b with a distinct variable x_b uniquely associated with b, and we let r be $D' \to U_i(x_a)$; we clearly have $U_i(a) \in T_r(D)$, and Proposition 2 ensures that rule r is sound for $\mathcal{N}$. This method, however, has at least two drawbacks. First, the rule r can have an arbitrary form, which does not help us understand the expressive power of GNNs. Second, the body of r is likely going to be large, which makes r overly specific: the rule can be applied only to datasets that contain a homomorphic image of D. Consequently, r is unlikely to provide a human-readable explanation for the derivation of $U_i(a)$ by $\mathcal{N}$.

To address the first drawback, we construct a rule r that is of a specific *tree-like* shape. We will show in Section 7 that each $\mathcal{N}$ is equivalent to a Datalog program $\mathcal{P}_\mathcal{N}$ consisting only of tree-like rules; moreover, the number of such rules is finite up to variable renaming, which will provide the basis for our rule extraction algorithm. To address the second drawback, our algorithm includes several optimisations that aim to minimise the size of the extracted tree-like rule. The extracted rule is thus more general and more readable, which makes the rule more likely to provide a good explanation for the inference. We show in Section 10 that our optimisations are very effective in practice.

The rest of this section is organised as follows. In Section 6.1, we characterise the syntactic structure of tree-like rules. Then, in Section 6.2 we present our rule extraction algorithm, and in Section 6.3 we discuss how to optimise the result of the rule extraction and obtain a smaller rule satisfying the same properties.

6.1. Tree-like rules

We will show that each derivation of monotonic max-sum GNNs can be explained using a sound rule of the form $\Gamma \to U(x)$, where Γ is a *tree-like* conjunction for x and U is the unary predicate. Intuitively, we can check whether a conjunction is tree-like as follows: we construct a graph whose vertices are the variables of Γ and that contains a directed edge from y to z for each atom $E^c(y, z)$ in Γ; next, we check whether this graph is a directed tree rooted at x; finally, we check that each inequality in Γ is of the form $z_1 \not\approx z_2$ where variables z_1 and z_2 are children of the same parent variable in the graph. Such inequalities provide a limited capability for counting; for example, conjunction $E^c(y, z_1) \wedge E^c(y, z_2) \wedge z_1 \not\approx z_2$ is true precisely for those values of y that are connected via E^c to at least two distinct constants. Definition 7 formalises these intuitions, and it also introduces intuitive notions of a variable *fan-out* (i.e., the number of children) and *depth*.

Definition 7. A *tree-like conjunction for a variable* x is defined inductively as follows.

- The empty conjunction $\top$ is tree-like for x.
- For each unary predicate U, atom $U(x)$ is tree-like for x.
- For all tree-like conjunctions Γ_1 and Γ_2 for x sharing no variables other than x, conjunction $\Gamma_1 \wedge \Gamma_2$ is tree-like for x.
- For each binary predicate E^c, and all tree-like conjunctions $\Gamma_1, \dots, \Gamma_n$ for distinct variables $y_1, \dots, y_n$ where no Γ_i contains x and no Γ_i and Γ_j with $i \neq j$ share a variable, conjunction $\bigwedge_{i=1}^n \left(E^c(x, y_i) \wedge \Gamma_i \right) \wedge \bigwedge_{1 \leq i < j \leq n} y_i \not\approx y_j$ is tree-like for x.

Let Γ be a tree-like conjunction for a variable x, and let y be a variable in Γ. The *fan-out* of y in Γ is the number of distinct variables z_i for which $E^c(y, z_i)$ is a conjunct of Γ for some colour c. The *depth* of y is the maximal number n for which there exist variables $y_0, \dots, y_n$ and predicates $E^{c_1}, \dots, E^{c_n}$ such that $y_0 = x$, atom $E^{c_i}(y_{i-1}, y_i)$ is a conjunct of Γ for each $i \in \{1, \dots, n\}$, and $y_n = y$. The *depth* of the conjunction Γ is the maximum depth of a variable in Γ.

For d and f natural numbers, a tree-like conjunction Γ is (d, f)-*tree-like* if, for each variable y in Γ, the depth i of y is at most d and the fan-out of y is at most $f(d - i)$. Moreover, a Datalog rule is (d, f)-*tree-like* if it is of form $\Gamma \rightarrow U(x)$, where Γ is a (d, f)-tree-like conjunction for x. Finally, a Datalog program is (d, f)-*tree-like* if it consists only of (d, f)-tree-like rules.

Note that the body of a tree-like rule can be $\top$; 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 $U(a)$ for each constant a occurring in D.

We note that tree-like conjunctions include, modulo syntactic variation, all concepts of the $\mathcal{ALCQ}$ description logic [27] constructed from $\top$, atomic concepts, and concepts of the form $\geq n\, R.C$ and $C_1 \sqcap C_2$; however, our definition is more expressive, permitting conjunctions 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 cannot be directly represented as $\mathcal{ALCQ}$ concepts.

[26] showed that, if the expressiveness of any aggregate-combine GNN can be characterised using a logical formula, then it can also be characterised using an $\mathcal{ALCQ}$ concept. Monotonic max-sum GNNs are a subclass of aggregate-combine GNNs modulo the fact that the latter can contain just one colour; however, we believe that the results by [26] can be straightforwardly extended to multiple colours. Since our results show that monotonic max-sum GNNs can be characterised using a logical formula, the results by [26] imply that each monotonic max-sum GNN can be characterised using an $\mathcal{ALCQ}$ concept; however, the concept in question can contain negation and/or disjunction, which are not needed in our work. Thus, our characterisation and the characterisation by [26] seem to be incomparable. Furthermore, it is unclear whether tree-like Datalog rules are the smallest Datalog class that can characterise monotonic max-sum GNNs, and we leave investigating this question for future work.

6.2. Rule extraction algorithm

Algorithm 1 Rule extraction algorithm.

Inputs:
$\mathcal{N}$: a monotonic max-sum (Col, δ)-GNN with L layers, dimensions $\delta_0, \dots, \delta_L$, and aggregation numbers $k_1, \dots, k_L$
D : a (Col, δ)-dataset
$U_i(a)$: a fact in $T_{\mathcal{N}}(D)$

Let $\langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda \rangle$ be the graph $\mathsf{enc}(D)$
Let $\lambda_0, \dots, \lambda_L$ be the labellings of vertices $\mathcal{V}$ when $\mathcal{N}$ is applied to $\mathsf{enc}(D)$
$\Gamma := \top$
$\theta := \{x \mapsto a\}$
$\mu_L(x) := \{i\}$
for ℓ **from** L **down to** 1 **do**
  $\theta' := \theta$
  for each mapping $y \mapsto b$ in θ' **do**
    $\mu_{\ell-1}(y) := \emptyset$
    for each $h \in \{1, \dots, \delta_{\ell-1}\}$ **do**
      if $(\mathbf{A}_\ell)_{j,h} \neq 0$ for some $j \in \mu_\ell(y)$ **then**
        $\mu_{\ell-1}(y) := \mu_{\ell-1}(y) \cup \{h\}$
      for each $c \in \mathsf{Col}$ such that $(\mathbf{B}^c_\ell)_{j,h} \neq 0$ for some $j \in \mu_\ell(y)$ **do**
        $V := \{v \in \mathcal{V} \mid \langle v^b, v \rangle \in \mathcal{E}^c \text{ and } (\mathbf{v}_{\ell-1})_h \neq 0\}$
        $MS := \text{max-}k_\ell\text{-sum}\{\!\{ (\mathbf{v}_{\ell-1})_h \mid v \in V \}\!\}$
        $Z := \emptyset$
        while $V \neq \emptyset$ and $\text{sum}\{\!\{ (\mathbf{v}^{\theta(z)}_{\ell-1})_h \mid z \in Z \}\!\} < MS$ **do**
          Let $v \in V$ be a vertex with the highest value of $(\mathbf{v}_{\ell-1})_h$ and remove v from V
          Let d be the constant such that v is v^d, and let z be a fresh variable
          $\Gamma := \Gamma \wedge E^c(y, z) \wedge \bigwedge \{z \not\approx z' \mid z' \in Z\}$
          $\theta := \theta \cup \{z \mapsto d\}$
          $\mu_{\ell-1}(z) := \{h\}$
          $Z := Z \cup \{z\}$
$\Gamma := \Gamma \wedge \bigwedge \{U_j(y) \mid y \mapsto b \text{ in } \theta, j \in \mu_0(y), \text{ and } U_j(b) \in D\}$
return $\Gamma \rightarrow U_i(x)$

Algorithm 1 takes as input a monotonic max-sum (Col, δ)-GNN $\mathcal{N}$, a (Col, δ)-dataset D, and a fact $U_i(a) \in T_{\mathcal{N}}(D)$, and it produces a rule r that is sound for $\mathcal{N}$ and that satisfies $U_i(a) \in T_r(D)$. In other words, rule r never derives a fact that is not also derived by $T_{\mathcal{N}}$ (i.e., r never 'fabricates' facts) on any dataset, and it explains why $T_{\mathcal{N}}$ derives $U_i(a)$ on D. Rule r is of the form $\Gamma \rightarrow U_i(x)$, where Γ is a tree-like conjunction for x.

Conjunction Γ is constructed by considering the layers of the GNN in reverse. During this process, the algorithm keeps several auxiliary pieces of information. Specifically, the algorithm produces an auxiliary substitution θ that maps the variables in Γ to the constants in D such that $\Gamma\theta \subseteq D$. Moreover, for each variable y in Γ and each layer ℓ, the algorithm maintains a set $\mu_\ell(y) \subseteq \{1, \dots, \delta_\ell\}$, which is used to optimise the construction of Γ due to the sparsity of the matrices of $\mathcal{N}$. Intuitively, our construction ensures that the value of the jth element of the feature vector labelling a vertex $v^{\theta(y)}$ in layer ℓ when $\mathcal{N}$ is applied to D can be computed as follows:

$$(\mathbf{v}_\ell^{\theta(y)})_j = \sigma\left(\sum_{h=1}^{\delta_{\ell-1}} (\mathbf{A}_\ell)_{j,h}(\mathbf{v}_{\ell-1}^{\theta(y)})_h + \sum_{c \in \mathsf{Col}} \sum_{h=1}^{\delta_{\ell-1}} (\mathbf{B}_\ell^c)_{j,h} \text{max-}k_\ell\text{-sum}\{\!\{ (\mathbf{v}_{\ell-1}^{\theta(z)})_h \mid E^c(y,z) \text{ in } \Gamma \}\!\} + (\mathbf{b}_\ell)_j\right). \tag{7}$$

Now, if $(\mathbf{A}_\ell)_{j,h} = 0$ (which happens often in practice since monotonic max-sum GNNs can be trained effectively even if all negative weights are clamped to zero), then the hth element of the feature vector labelling vertex $v^{\theta(y)}$ in layer $\ell - 1$ is irrelevant to the computation of $(\mathbf{v}_\ell^{\theta(y)})_j$, and thus also irrelevant to the computation of $(\mathbf{v}_L^a)_i$ and the derivation of $U_i(a)$. Analogous reasoning applies if $(\mathbf{B}_\ell^c)_{j,h} = 0$. To detect such situations, the set $\mu_{\ell-1}(y)$ collects positions $h \in \{1, \dots, \delta_{\ell-1}\}$ such that the value of $(\mathbf{v}_{\ell-1}^{\theta(y)})_h$ is relevant in the above computation.

The construction of the rule $\Gamma \to U_i(x)$ is shown in Algorithm 1. The algorithm encodes input dataset D into a coloured graph (line 1), and then applies the input GNN $\mathcal{N}$ to the encoding and extracts the vertex labellings (line 2). Next, it initialises Γ to the empty conjunction (line 3), ensures that θ maps the 'root' variable x to the target constant a (line 4), and initialises $\mu_L(x)$ to contain i (line 5) since the value of $(\mathbf{v}_L^{\theta(x)})_i$ is clearly relevant. The algorithm then iterates through the layers of $\mathcal{N}$ in descending order (lines 6–23), and, for each ℓ between L and one, the algorithm uses the GNN parameters for layer ℓ to extend Γ and θ, as well as to construct sets $\mu_{\ell-1}$. Specifically, the algorithm considers each variable y introduced thus far by iterating over the domain of the current substitution θ (lines 7–8), and for each y it considers each position $h \in \{1, \dots, \delta_{\ell-1}\}$ (line 10). As explained above, if position j of $\mathbf{v}_\ell^{\theta(y)}$ is relevant and $(\mathbf{A}_\ell)_{j,h} \neq 0$, then position h of $\mathbf{v}_{\ell-1}^{\theta(y)}$ is relevant too. This condition is checked in line 11, and h is added to $\mu_{\ell-1}(y)$ in line 12 if necessary. The algorithm next considers each colour $c \in \mathsf{Col}$ (lines 13–23). If position j of $\mathbf{v}_\ell^{\theta(y)}$ is relevant and $(\mathbf{B}_\ell^c)_{j,h} \neq 0$, then colour c needs to be taken into account when computing $\mathbf{v}_\ell^{\theta(y)}$, and position h of $\mathbf{v}_{\ell-1}$ can be relevant too for each neighbour v of $v^{\theta(y)}$. Thus, the algorithm identifies in line 14 the set V containing each neighbour v of $v^{\theta(y)}$ of colour c such that $(\mathbf{v}_{\ell-1})_h$ is nonzero and can thus contribute to the value of $(\mathbf{v}_\ell^{\theta(y)})_j$, and it computes the aggregation value MS for these vertices (line 15). The algorithm next identifies the subset of the vertices of V that actually contribute to MS (lines 17–23). To this end, the algorithm explores in line 18 the candidate vertices by decreasing value of $(\mathbf{v}_{\ell-1})_h$: when the sum over all such values reaches MS in line 17, all relevant vertices have been identified. For each such v, the algorithm identifies the constant d of D corresponding to v and introduces a fresh variable z (line 18). The algorithm then extends Γ by the atom $E^c(y,z)$ and an inequality $z \not\approx z'$ between z and any other variable z' that has already been introduced thus far in this loop (line 20), and it extends θ by mapping z to d (line 21). Thus, $D \models \Gamma\theta$ reflects the fact that D contains the constants corresponding to the relevant vertices that produce the aggregation value of MS. In addition, the algorithm identifies the position h in the feature vector of $\mathbf{v}_{\ell-1}^{\theta(z)}$ as relevant (line 22). Finally, the algorithm records z as having been processed by adding it to Z (line 23); thus, the set Z contains all children variables of y, which is needed to generate the relevant inequalities in line 20. When the loop in lines 6–23 finishes, for each variable y in Γ, the set $\mu_0(y)$ enumerates all positions that are relevant in $\mathbf{v}_0^{\theta(y)}$. All of these correspond to unary facts in the input dataset, so the algorithm simply extends Γ with the relevant atoms that check the presence of such facts (line 24).

Algorithm 1 clearly terminates since the number of iterations of in lines 17–23 is bounded by the maximum degree of a vertex in the canonical encoding of D. The following theorem shows that the resulting rule satisfies the desired properties.

Theorem 1. *When applied to a max-sum* (Col, δ)*-GNN* $\mathcal{N}$*, a* (Col, δ)*-dataset* D*, and a fact* $U_i(a) \in T_{\mathcal{N}}(D)$*, Algorithm 1 produces a* $(L, |\mathsf{Col}| \cdot \delta_{\mathcal{N}} \cdot \mathsf{deg}(D))$*-tree-like rule* r *that is sound for* $\mathcal{N}$ *and that satisfies* $U_i(a) \in T_r(D)$*, where* L *is the number of layers of* $\mathcal{N}$*,* $\delta_{\mathcal{N}} = \max(\delta_0, \dots, \delta_L)$ *for the layer dimensions* $\delta_0, \dots, \delta_L$*, and* $\mathsf{deg}(D)$ *is the maximum degree of a vertex in* $\mathsf{enc}(D)$*.*

Proof. Assume that Algorithm 1 is applied to some $\mathcal{N}$, D, and $U_i(a)$ as stated in the claim. Let $\mathcal{G}$, $\mathcal{V}$, $\{\mathcal{E}^c\}_{c \in \mathsf{Col}}$, and $\lambda_0, \dots, \lambda_L$ be as stated in the algorithm, and let $r = \Gamma \to U_i(x)$ be the resulting rule. Moreover, let θ be the final substitution, and let μ_ℓ with $\ell \in \{0, \dots, L\}$ be the final mappings computed by the algorithm. Note that $\theta(x) = a$. The construction of Γ ensures that an atom A is added to Γ only if $A\theta \in D$. Moreover, an inequality $z \not\approx z'$ is added to Γ only when variables z and z' are introduced for vertices corresponding to distinct constants b and b', and θ is defined on these variables as $\theta(z) = b$ and $\theta(z') = b'$; thus, $\theta(z) \neq \theta(z')$ for each $z \not\approx z' \in \Gamma$. Hence, $D \models \Gamma\theta$, which implies that $U_i(a) \in T_r(D)$, as required. Finally, Γ is $(L, \delta_{\mathcal{N}} \cdot |\mathsf{Col}| \cdot \mathsf{deg}(D))$-tree-like for x since at most $\mathsf{deg}(D)$ fresh variables can be introduced for each y, ℓ, c, and $i \in \{1, \dots, \delta_\ell\}$.

To show that r is sound for $\mathcal{N}$, we consider an arbitrary dataset D' and an atom in $T_r(D')$; this atom is clearly of the form $U_i(a')$ for a' a constant occurring in D'. We prove that $U_i(a') \in T_{\mathcal{N}}(D')$. Since $U_i(a') \in T_r(D')$, there exists a substitution θ' such that $D' \models \Gamma\theta'$ and $x\theta' = a'$. Let $\mathcal{G}' = \langle \mathcal{V}', \{\mathcal{E}'^c\}_{c \in \mathsf{Col}}, \lambda' \rangle$ be the canonical encoding of D', and let $\lambda'_0, \dots, \lambda'_L$ be the labellings of the vertices of $\mathcal{G}'$ when $\mathcal{N}$ is applied to $\mathcal{G}'$. Since the range of the activation function σ is $\mathbb{R}_0^+$, all components of all feature vectors of $\mathcal{G}$ and $\mathcal{G}'$ are nonnegative. We assign to each variable in r a *level* as follows: the level of variable x is L, and the level of any other variable in r is the value of the index ℓ from the loop in lines 6–23 when the variable is first introduced. We next prove by induction on ℓ the following claim.

Claim. *Inequality* $(\mathbf{v}_{\lambda_\ell}^{\theta(y)})_j \leq (\mathbf{v}_{\lambda'_\ell}^{\theta'(y)})_j$ *holds for each* $\ell \in \{0, \dots, L\}$*, each variable* y *in* r *whose level is at least* ℓ*, and each* $j \in \mu_\ell(y)$*.*

Proof. For the base case $\ell = 0$, consider an arbitrary variable y in r (which is of level at least zero by construction) and $j \in \mu_0(y)$. The values of $(\mathbf{v}_{\lambda_0}^{\theta(y)})_j$ and $(\mathbf{v}_{\lambda'_0}^{\theta'(y)})_j$ are in $\{0, 1\}$, so we need to prove that $(\mathbf{v}_{\lambda_0}^{\theta(y)})_j = 1$ implies $(\mathbf{v}_{\lambda'_0}^{\theta'(y)})_j = 1$. Assume that $(\mathbf{v}_{\lambda_0}^{\theta(y)})_j = 1$. The

definition of the canonical encoding then ensures $U_j(y)\theta \in D$. But then, line 24 of Algorithm 1 ensures that Γ contains an atom $U_j(y)$. Moreover, we assumed that $D' \models \Gamma\theta'$, so we have $U_j(y)\theta' \in D'$, and thus $(\mathbf{v}^{\theta'(y)}_{\lambda_0})_j = 1$, as required.

For the induction step, consider an arbitrary $\ell \in \{1, \dots, L\}$, and assume that $(\mathbf{v}^{\theta(y)}_{\lambda_{\ell-1}})_j \leq (\mathbf{v}^{\theta'(y)}_{\lambda_{\ell-1}})_j$ for each variable y in r of level $\ell' \geq \ell - 1$ and each $j \in \mu_{\ell-1}(y)$. Consider an arbitrary variable y in r of level $\ell' \geq \ell$ and arbitrary $j \in \mu_\ell(y)$. By Eq. (4), the values of $(\mathbf{v}^{\theta(y)}_{\lambda_\ell})_j$ and $(\mathbf{v}^{\theta'(y)}_{\lambda_\ell})_j$ are given by equations (8) and (9), respectively.

$$(\mathbf{v}^{\theta(y)}_{\lambda_\ell})_j = \sigma\left(\sum_{h=1}^{\delta_{\ell-1}} (\mathbf{A}_\ell)_{j,h} (\mathbf{v}^{\theta(y)}_{\lambda_{\ell-1}})_h + \sum_{c \in \mathsf{Col}} \sum_{h=1}^{\delta_{\ell-1}} (\mathbf{B}^c_\ell)_{j,h} \text{ max-}k_\ell\text{-sum}\{\!\{ (\mathbf{w}_{\lambda_{\ell-1}})_h \mid \langle v^{\theta(y)}, w\rangle \in \mathcal{E}^c \}\!\} + (\mathbf{b}_\ell)_j\right) \tag{8}$$

$$(\mathbf{v}^{\theta'(y)}_{\lambda'_\ell})_j = \sigma\left(\sum_{h=1}^{\delta_{\ell-1}} (\mathbf{A}_\ell)_{j,h} (\mathbf{v}^{\theta'(y)}_{\lambda'_{\ell-1}})_h + \sum_{c \in \mathsf{Col}} \sum_{h=1}^{\delta_{\ell-1}} (\mathbf{B}^c_\ell)_{j,h} \text{ max-}k_\ell\text{-sum}\{\!\{ (\mathbf{w}_{\lambda'_{\ell-1}})_h \mid \langle v^{\theta'(y)}, w\rangle \in \mathcal{E}'^c \}\!\} + (\mathbf{b}_\ell)_j\right) \tag{9}$$

We show that each summand of Eq. (8) is smaller than or equal to the corresponding summand of equation (9). Towards this goal, consider an arbitrary $h \in \{1, \dots, \delta_{\ell-1}\}$.

If $h \notin \mu_{\ell-1}(y)$, since h is not added to $\mu_{\ell-1}(y)$ in line (12) during the algorithm's execution, the condition of line (10) ensures that $(\mathbf{A}_\ell)_{j,h} = 0$. If $h \in \mu_{\ell-1}(y)$, since the level ℓ' of y satisfies $\ell' \geq \ell > \ell - 1$, the induction hypothesis ensures $(\mathbf{v}^{\theta(y)}_{\lambda_{\ell-1}})_h \leq (\mathbf{v}^{\theta'(y)}_{\lambda'_{\ell-1}})_h$. Thus, the first sum in the argument of σ in Eq. (8) is smaller than or equal to the corresponding sum in Eq. (9).

Due to $\ell' > \ell - 1$, variable y is in the domain of θ in the iteration of the loop in lines 6–23 for $\ell - 1$. Now consider an arbitrary colour $c \in \mathsf{Col}$. If the condition in line 13 is not satisfied for c, then $(\mathbf{B}^c_\ell)_{j,h} = 0$, so the summand in equation (8) for c and position h is zero and is smaller or equal to the corresponding summand of Eq. (9). If the condition in line 13 is satisfied, then the loop in lines 10–23 is evaluated for $\ell - 1$, y, h, and c. Let Z be the value of this set at the end of the iteration. If $Z = \emptyset$, then

$$0 = \text{max-}k_\ell\text{-sum}\{\!\{ (\mathbf{v}_{\lambda_{\ell-1}})_h \mid \langle v^{\theta(y)}, v\rangle \in \mathcal{E}^c \}\!\} \leq \text{max-}k_\ell\text{-sum}\{\!\{ (\mathbf{v}_{\lambda'_{\ell-1}})_h \mid \langle v^{\theta'(y)}, v\rangle \in \mathcal{E}'^c \}\!\},$$

where the equality holds due to the condition in line 17, and the inequality holds since the components of all feature vectors are nonnegative. Now assume that $Z \neq \emptyset$. For each pair of distinct variables $z, z' \in Z$, line 20 of Algorithm 1 ensures that Γ contains $z \not\approx z'$. Thus, $D \models \Gamma\theta$ implies $\theta(z) \neq \theta(z')$—that is, each $z \in Z$ corresponds to a distinct constant $\theta(z)$ occurring in D, which in turn corresponds to a distinct vertex $v^{\theta(z)}$. Analogously, $D' \models \Gamma\theta'$ implies that each $z \in Z$ corresponds to a distinct constant $z\theta'$ occurring in D', which in turn corresponds to a distinct vertex $v^{\theta'(z)}$. Moreover, for each $z \in Z$, line 20 ensures that Γ contains $E^c(y, z)$. Thus, $D \models \Gamma\theta$ implies $E^c(y, z)\theta \in D$, which implies $\langle v^{\theta(y)}, v^{\theta(z)}\rangle \in \mathcal{E}^c$. Analogously, $D' \models \Gamma\theta'$ implies $E^c(y, z)\theta' \in D'$, which implies $\langle v^{\theta'(y)}, v^{\theta'(z)}\rangle \in \mathcal{E}'^c$. We next use these observations to show that

$$\text{max-}k_\ell\text{-sum}\{\!\{ (\mathbf{v}_{\lambda_{\ell-1}})_h \mid \langle v^{\theta(y)}, v\rangle \in \mathcal{E}^c \}\!\} \leq \sum_{z \in Z} (\mathbf{v}^{\theta(z)}_{\lambda_{\ell-1}})_h \leq \sum_{z \in Z} (\mathbf{v}^{\theta'(z)}_{\lambda'_{\ell-1}})_h \leq \text{max-}k_\ell\text{-sum}\{\!\{ (\mathbf{v}_{\lambda'_{\ell-1}})_h \mid \langle v^{\theta'(y)}, v\rangle \in \mathcal{E}'^c \}\!\}.$$

The first inequality holds because the condition in line 17 is falsified when the loop finishes. For the second inequality, each variable $z \in Z$ is introduced in an iteration of the loop in lines 6–23 for $\ell - 1$, so each $z \in Z$ is of level $\ell - 1$; furthermore, line 22 ensures that $h \in \mu_{\ell-1}(z)$; hence, we have $(\mathbf{v}^{\theta(z)}_{\lambda_{\ell-1}})_h \leq (\mathbf{v}^{\theta'(z)}_{\lambda'_{\ell-1}})_h$ by the induction hypothesis. The third inequality holds because $|Z| \leq k_\ell$; this is trivial if $k_\ell = \infty$, and otherwise note that the body of the loop in lines 17–23 can clearly be executed at most k_ℓ times since we always select a neighbour with the highest value of the hth element in the corresponding feature vector. This reasoning holds for each h and c, so each summand of equation (8) is indeed smaller or equal that the corresponding summand of Eq. (9). Finally, all elements of $\mathbf{A}_\ell$ and the $\mathbf{B}^c_\ell$ are nonnegative and function σ is monotonically increasing, which completes the proof of the induction step. □

We now complete the argument that r is sound for $\mathcal{N}$. Note that $U_i(x)\theta = U_i(a)$ and $U_i(x)\theta' = U_i(a')$, and line (5) ensures $i \in \mu_L(x)$; thus, the above claim guarantees that

$$1 = \mathsf{cls}((\mathbf{v}^{\theta(x)}_{\lambda_L})_i) \leq \mathsf{cls}((\mathbf{v}^{\theta'(x)}_{\lambda'_L})_i) = 1.$$

The first equality holds due to $U_i(a) \in T_r(D)$ and the definition of canonical decoding. The inequality holds by the claim and the fact that cls is monotonically increasing. The last equality holds because the range of cls is $\{0, 1\}$. But then, the definition of the canonical decoding ensures $U_i(a') \in T_r(D')$, as required. □

The sparsity of the matrices significantly affects the size of the extracted rule. If the matrices are not sparse, the algorithm can produce a tree-like rule whose body has depth L and a fan-out of $L \cdot |\mathsf{Col}| \cdot \delta_{\mathcal{N}} \cdot \deg(D)$, and it can introduce up to δ binary predicates for each variable. Thus, the rule can contain at most $(|\mathsf{Col}| \cdot \delta_{\mathcal{N}} \cdot \deg(D))^L \cdot \delta$ atoms in a worst case, which is a potentially very large number. With sparse matrices, the number of atoms can be reduced dramatically, in the extreme case to zero. In Section 10 we show empirically that our algorithm produces rules containing at most a few dozen atoms. Moreover, the optimisation techniques described in Section 6.3 further reduces the number of body atoms to about four or less.

We next provide an example illustrating an application of Algorithm 1.

Example 2. Let $\mathsf{Col} = \{c\}$, let $\delta = 2$, and let $\mathcal{N}$ be a monotonic max-sum (Col, δ)-GNN with two layers both using the sum aggregation function and such that the only nonzero parameters (including the bias vectors) are

$$\mathbf{A}_1 = \begin{pmatrix} 0.1 & 0 \\ 0 & 0 \end{pmatrix}, \qquad \mathbf{B}^c_1 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \qquad \text{and} \qquad \mathbf{A}_2 = \begin{pmatrix} 0 & 0 \\ 0.5 & 0.5 \end{pmatrix}.$$

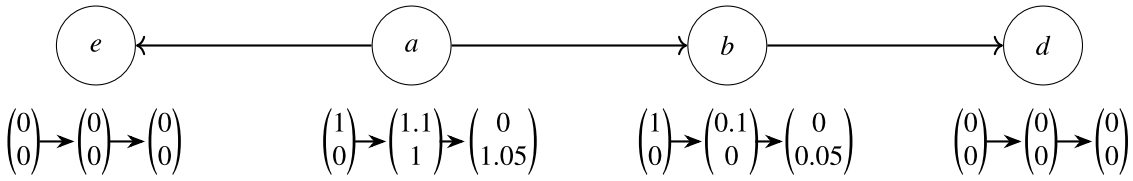


Fig. 5. The result of applying $\mathcal{N}$ to $\mathsf{enc}(D)$ in Example 2.

Moreover, let the activation function be ReLU, and let the classification threshold be 0.5. Finally, let

$$D = \{U_1(a), E^c(a,b), U_1(b), E^c(b,d), E^c(a,e)\}.$$

We next apply Algorithm 1 to $\mathcal{N}$, D, and $U_2(a)$. The algorithm first applies $\mathcal{N}$ to $\mathsf{enc}(D)$. The result is shown in Fig. 5. The labellings λ_0, λ_1, and λ_2 of each vertex are presented left-to-right below each vertex.

The algorithm then initialises $\Gamma := \top$, $\theta := \{x \mapsto a\}$, and $\mu_2(x) := \{2\}$. In the iteration of line 6 for $\ell = 2$, the iteration of line 10 considers only variable x, and positions $h = 1$ and $h = 2$. For $h = 1$, $(\mathbf{A}_2)_{2,1} \neq 0$ ensures $\mu_1(x) = \{1\}$; moreover, $\mathbf{B}^c_2 = 0$ ensures that no further steps are made in this iteration. Similarly, for $h = 2$, $(\mathbf{A}_2)_{2,2} \neq 0$ leads to $\mu_1(x) = \{1, 2\}$ without any further steps. Next, in the iteration of line 6 for $\ell = 1$, the iteration of line 10 considers only variable x, and positions $h = 1$ and $h = 2$. For $h = 1$, $(\mathbf{A}_1)_{1,1} \neq 0$ ensures $\mu_0(x) = \{1\}$. Then, since $(\mathbf{B}^c_1)_{1,1} \neq 0$, the algorithm computes V as $V := \{v^b\}$; note that v^e is not included in V because $(\mathbf{v}^e_{\lambda_0})_1 = 0$. Thus, the algorithm introduces a fresh variable y, extends Γ with $E^c(x, y)$, extends θ by setting $y \mapsto b$, and defines $\mu_0(y) = \{1\}$. For $h = 2$, $(\mathbf{A}_1)_{1,2} = 0$, so $\mu_0(x)$ is not extended any further. Then, since $(\mathbf{B}^c_2)_{2,2} \neq 0$, the algorithm attempts to identify the set V of neighbours v of v^a such that $(\mathbf{v}_{\lambda_0})_2 > 0$; however, no such neighbours exist, so no further steps are made. The algorithm then updates Γ using μ_0: for x, we have $\mu_0(x) = \{1\}$; thus, due to $U_1(a) \in D$, atom $U_1(x)$ is added to Γ. Atom $U_1(y)$ is added to Γ using analogous reasoning. Thus, we have $\Gamma = U_1(x) \wedge E^c(x, y) \wedge U_1(y)$, so the algorithm returns the rule $U_1(x) \wedge E^c(x, y) \wedge U_1(y) \rightarrow U_2(x)$.

6.3. Optimising the extracted rule

Although Algorithm 1 uses the sparsity of the GNN's matrices to reduce the size of the resulting rule $\Gamma \rightarrow U_i(x)$, conjunction Γ can nevertheless often contain redundant atoms—that is, one can often find a proper subset $\Gamma' \subsetneq \Gamma$ such that rule $\Gamma' \rightarrow U_i(x)$ is still sound for $\mathcal{N}$ (and it is obvious that this rule still derives $U_i(a)$ on D). One can remove all redundant atoms from Γ by considering each subset $\Gamma' \subsetneq \Gamma$ and applying the soundness check from Proposition 2 or 3 to the corresponding rule; however, the number of candidate subsets Γ' can be very large, which renders such a naïve approach infeasible in practice.

Algorithm 2 Optimising the extracted rule.

Inputs:
$\mathcal{N}$: a monotonic max-sum (Col, δ)-GNN
$\Gamma \rightarrow U_i(x)$: a tree-like rule that is sound for $\mathcal{N}$

1: Let S be a list of the unary and binary atoms of Γ sorted in decreasing order of relevance
2: $\Gamma' := \top$
3: **while** $\Gamma' \rightarrow U_i(x)$ is not sound for $\mathcal{N}$ **do**
4: Let A be the first element of S, and remove A from S
5: $\Gamma' := \Gamma' \wedge A$
6: **repeat**
7: $E := \{E^c(y_1, y_2) \in \Gamma \setminus \Gamma' \mid \text{variable } y_2 \text{ occurs in } \Gamma'\}$
8: Remove all atoms in E from S
9: $\Gamma' := \Gamma' \wedge \bigwedge E$
10: **until** $E = \emptyset$
11: $\Gamma' := \Gamma' \wedge \bigwedge\{y_1 \not\approx y_2 \in \Gamma \setminus \Gamma' \mid \text{variables } y_1 \text{ and } y_2 \text{ both occur in } \Gamma'\}$
12: **return** $\Gamma' \rightarrow U_i(x)$

Algorithm 2 presents a heuristic that reduces the number of considered subsets of Γ. In particular, the algorithm orders the unary and binary atoms of Γ according to some relevance criterion (line 1), and then constructs a subset Γ' in lines 3–11 by considering each atom A of Γ in decreasing order of relevance. Intuitively, the relevance criterion should be chosen such that extending Γ' with atoms of higher relevance is more likely to make the rule $\Gamma' \rightarrow U_i(x)$ sound for $\mathcal{N}$. We discuss two such criteria at the end of this section, and in Section 10 we show empirically that these indeed produce small rules in practice. In each iteration, the next highest relevant atom A is selected (line 4) and added to Γ' (line 5). This, however, can create a disconnected conjunction, so in lines 6–10 the algorithm ensures that A is connected via binary atoms to Γ'. Moreover, in line 11 the algorithm ensures that all relevant inequalities of Γ are added to Γ' as well. This process is repeated until $\Gamma' \rightarrow U_i(x)$ becomes sound for $\mathcal{N}$ (line 3). Since $\Gamma \rightarrow U_i(x)$ is sound for $\mathcal{N}$,

the algorithm returns $\Gamma' = \Gamma$ in the worst case. Thus, the algorithm clearly terminates and returns a rule that is sound for $\mathcal{N}$ and that derives the fact $U_i(a)$ on the dataset D.

To understand the intuition behind our relevance criteria, assume that Algorithm 1 produces a rule $\Gamma \to U_i(x)$ that explains the derivation of $U_i(a)$ on a dataset D using a substitution θ—that is, we have $D \models \Gamma\theta$ and $U_i(x)\theta = U_i(a)$. Algorithm 2 would ideally produce a rule $\Gamma' \to U_i(x)$ that also derives $U_i(a)$ on D via θ, but where Γ' is a least subset of Γ such that, if $\mathcal{N}$ is applied to the unary and binary atoms of $\Gamma'\theta$ (i.e., a relevant subset of D), then $(\mathbf{v}_L^{\theta(x)})_i$ is the least possible value exceeding the classification threshold and thus ensuring that $U_i(a)$ is derived. Constructing a rule that satisfies this condition exactly would be very hard, though, not least because of the nonlinearities inherent in the GNN model. To obtain a practical approach, Algorithm 2 uses the notion of atom relevance to estimate the contribution of each atom $B \in \Gamma$ to the value of $(\mathbf{v}_L^{\theta(x)})_i$. We developed the following two criteria for computing such estimates.

- To determine the relevance of an atom $B \in \Gamma$, our first criterion extracts the conjunction $\Gamma_B \subseteq \Gamma$ that contains B and all atoms needed to 'connect' B with the 'root' variable x, applies $\mathcal{N}$ to the canonical encoding of $\Gamma_B\theta$, and returns $(\mathbf{v}_L^{\theta(x)})_i$ as the relevance value of B. Intuitively, Γ_B identifies the path leading from the atom B to the head atom $U_i(x)$, and the relevance of B aims to estimate the contribution of this path to the value of $(\mathbf{v}_L^{\theta(x)})_i$ on dataset D.
- Our second criterion assigns the relevance of zero to each binary atom, and computes the relevance of each unary atom as follows. Let ν be a substitution that maps each variable of Γ to a distinct constant, and let D' be the dataset containing the unary and binary atoms of $\Gamma\nu$. Our approach can be seen as rewriting the equation (4) for vertex $v^{\theta(x)}$ and layer L using other instances of Eq. (4) for all relevant layers and vertices, until the value of $(\mathbf{v}_L^{\theta(x)})_i$ is expressed as a function of values of the form $(\mathbf{v}_0^{\theta(y)})_j$, where y is a variable in r; crucially, in all such equations, all nonlinearities are removed (i.e., σ is taken to be the identity function). The resulting equation for $(\mathbf{v}_L^{\theta(x)})_i$ is a linear combination over vectors of the form $(\mathbf{v}_0^{\theta(y)})_j$, where each such vector is multiplied by a coefficient corresponding to a sum of products of elements of matrices $\mathbf{A}_\ell$ and $\mathbf{B}_\ell^c$. Then, the relevance of each unary atom $U_j(y)$ is the coefficient of $(\mathbf{v}_0^{\theta(y)})_j$ in this equation.

7. Equivalence of monotonic max-sum GNNs and Datalog programs

In this section we show that each monotonic max-sum GNN is equivalent to a Datalog program consisting of tree-like rules. Towards this goal, in Section 7.1 we show that for each monotonic max-sum GNN with unbounded aggregation functions, there exists an equivalent GNN that uses only bounded aggregation functions. Next, in Section 7.2 we show that these aggregation bounds are computable from the GNN's parameters. Finally, in Section 7.3 we show how to extract an equivalent Datalog program from a monotonic max-sum GNN without unbounded aggregation.

Throughout the rest of this section, we fix a monotonic max-sum (Col, δ)-GNN $\mathcal{N}$ of form (3) 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 $\mathcal{N}$.

7.1. Limiting neighbour aggregation

In this section, we show that for each aggregation function max-k_ℓ-sum of $\mathcal{N}$ with $k_\ell = \infty$, there exists an integer $C_\ell \in \mathbb{N}_0$ such that, if we replace max-k_ℓ-sum with max-C_ℓ-sum, the transformation induced by the GNN remains the same. In other words, we show that, to apply the 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 7.2 we show that capacities C_ℓ can be algorithmically computed from $\mathcal{N}$.

The definition of C_ℓ hinges upon a crucial property of monotonic max-sum GNNs: each set of possible values that can occur in a position of any feature vector when the GNN is applied to a (Col, δ)-dataset has a nonnegative smallest element. To establish this property, we first introduce a family of sets that contain all of these possible values.

Definition 8. A *multiset family* of dimension γ is a mapping $\mathbf{Y}$ that assigns to each colour $c \in \mathsf{Col}$ a finite multiset $\mathbf{Y}^c$ of vectors of dimension γ. For each $\ell \in \{1, \dots, L\}$, each $i \in \{1, \dots, \delta_\ell\}$, each vector $\mathbf{x}$ of dimension $\delta_{\ell-1}$, and each multiset family $\mathbf{Y}$ also of dimension $\delta_{\ell-1}$, let

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

Then, sets $\mathcal{X}_{\ell,i}$ with $\ell \in \{0, \dots, L\}$ and $i \in \{1, \dots, \delta_\ell\}$ are defined by induction on ℓ as follows.

- If $\ell = 0$, then $\mathcal{X}_{0,i} = \{0, 1\}$.
- If $\ell \geq 1$, then $\mathcal{X}_{\ell,i}$ is the least set such that $\sigma(\mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y})) \in \mathcal{X}_{\ell,i}$ for all $\mathbf{x}$ and $\mathbf{Y}$ where
  - $\mathbf{x}$ is a vector of dimension $\delta_{\ell-1}$ such that $(\mathbf{x})_j \in \mathcal{X}_{\ell-1,j}$ for each position $j \in \{1, \dots, \delta_{\ell-1}\}$, and
  - $\mathbf{Y}$ is a multiset family of dimension $\delta_{\ell-1}$ such that $(\mathbf{y})_j \in \mathcal{X}_{\ell-1,j}$ for each colour $c \in \mathsf{Col}$, each vector $\mathbf{y} \in \mathbf{Y}^c$, and each position $j \in \{1, \dots, \delta_{\ell-1}\}$.

Intuitively, $\mathcal{X}_{\ell,i}$ contains all real numbers that can occur in the ith position of a vector labelling a vertex at layer ℓ when $\mathcal{N}$ is applied to the canonical encoding of some (Col, δ)-dataset. Note that some of these sets may be infinite whenever there exists at least one k_ℓ that is equal to ∞. The following lemma formalises our intuition behind sets $\mathcal{X}_{\ell,i}$.

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

Proof. Let $\mathsf{enc}(D) = \langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda \rangle$. The proof is by induction on ℓ. For the base case $\ell = 0$, Definitions 2 and 8 ensure $(\mathbf{v}_0)_i \in \{0, 1\} = \mathcal{X}_{0,i}$ for each i. For the induction step, we assume that the claim holds for $\ell - 1$ with $\ell \in \{1, \dots, L\}$, and we show that it holds for ℓ as well. To this end, consider an arbitrary $i \in \{1, \dots, \delta_\ell\}$ and note that the value of $(\mathbf{v}_\ell)_i$ is given by

$$(\mathbf{v}_\ell)_i = \sigma\left(\sum_{j=1}^{\delta_{\ell-1}} (\mathbf{A}_\ell)_{i,j} (\mathbf{v}_{\ell-1})_j + \sum_{c \in \mathsf{Col}} \sum_{j=1}^{\delta_{\ell-1}} (\mathbf{B}^c_\ell)_{i,j} \max\text{-}k_\ell\text{-}\mathrm{sum}(\{\!\{ (\mathbf{u}_{\ell-1})_j \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}) + (\mathbf{b}_\ell)_i \right). \tag{10}$$

Consider also $\mathbf{x} = \mathbf{v}_{\ell-1}$, the multiset family $\mathbf{Y}$ of dimension $\delta_{\ell-1}$ defined by $\mathbf{Y}^c = \{\!\{ \mathbf{u}_{\ell-1} \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}$ for every $c \in \mathsf{Col}$, 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 (10) with the definition of $\mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y})$ in Definition 8, we have $\sigma(z) = (\mathbf{v}_\ell)_i$. The definition of $\mathcal{X}_{\ell,i}$ ensures $\sigma(z) \in \mathcal{X}_{\ell,i}$; so $(\mathbf{v}_\ell)_i \in \mathcal{X}_{\ell,i}$, as desired. □

Theorem 2. *Each set* $\mathcal{X}_{\ell,i}$ *satisfies* $\mathcal{X}_{\ell,i} \subseteq \mathbb{R}^+_0$*, and, for each* $\alpha \in \mathbb{R}$*, the set* $\{\alpha' \in \mathcal{X}_{\ell,i} \mid \alpha' \leq \alpha\}$ *is finite.*

The condition on $\mathcal{X}_{\ell,i}$ in the second claim of Theorem 2 is often called the *descending chain condition*. The proof of the theorem is technically involved, but it is not essential for understanding the remaining results; thus, we present the full proof in A, and we simply mention here that the proof crucially depends on the fact that the activation function σ is unbounded. The theorem ensures that each (possibly infinite) set $\mathcal{X}_{\ell,i}$ has a least element. This is required for Algorithm 3, which we use in the following definition of the *capacity* of $\mathcal{N}$.

Definition 9. The *capacity* of each layer ℓ of $\mathcal{N}$ is defined in Algorithm 3. 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 3 should for now be understood as inductively defining a sequence of numbers C_ℓ for ℓ from L to 1. However, in Section 7.2 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'.

Algorithm 3 Computing $\mathsf{Capacity}(\mathcal{N})$.

1: Let α_L be the threshold of cls
2: **for** ℓ **from** L **down to** 1 **do**
3: **if** all elements of $\mathbf{A}_\ell$ and $\mathbf{B}^c_\ell$ are 0 **or** $\bigcup_i \mathcal{X}_{\ell-1,i} = \{0\}$ **then**
4: $C_\ell := C_{\ell-1} := \cdots := C_1 := 0$
5: **return**
6: $w_\ell :=$ the least nonzero element of $\mathbf{A}_\ell$ and all $\mathbf{B}^c_\ell$
7: $\epsilon_\ell :=$ the least nonzero element of $\bigcup_i \mathcal{X}_{\ell-1,i}$
8: $\beta_\ell :=$ the least natural number such that $\sigma(\beta_\ell) \geq \alpha_\ell$
9: $b_\ell :=$ the least element of $\mathbf{b}_\ell$
10: $C_\ell := \min(k_\ell, \max(0, \lceil \frac{\beta_\ell - b_\ell}{w_\ell \cdot \epsilon_\ell} \rceil))$
11: $\alpha_{\ell-1} := \frac{\beta_\ell - b_\ell}{w_\ell}$
return $C_1, \dots, C_L$

Theorem 3 states the main result of this subsection: if we let $\mathcal{N}'$ be the GNN obtained from $\mathcal{N}$ by replacing each k_ℓ by C_ℓ, then $T_{\mathcal{N}}(D) = T_{\mathcal{N}'}(D)$ for each (Col, δ)-dataset D; in other words, we can give up the ability to aggregate an unbounded number of neighbours without affecting the GNN's predictions. To understand the intuition behind this result, 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 3 performs an early return (via line 5) or layer 0 if this does not happen. Indeed, assume that $\mathsf{cls}((\mathbf{v}_{\lambda_L})_i) = 1$ for some v and i. 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 step function, which implies that $(\mathbf{v}_{\lambda_L})_i \geq \alpha_L$ for α_L the threshold of cls. Moreover, $(\mathbf{v}_{\lambda_L})_i$ is produced from the $(\mathbf{v}_{\lambda_{L-1}})_j$ and the $(\mathbf{u}_{\lambda_{L-1}})_j$, where u ranges over the neighbours of v. If we assume that ϵ_ℓ is the least nonzero value that each u can contribute to $(\mathbf{v}_{\lambda_L})_i$, it suffices to have $\lceil \frac{\beta_\ell - b_\ell}{w_\ell \cdot \epsilon_\ell} \rceil$ nonzero neighbours to reach the minimum integer value β_L needed to surpass the threshold α_L; moreover, if $\beta_\ell < b_\ell$, then we need no neighbours at all. Thus, we can replace k_ℓ with $\max(0, \lceil \frac{\beta_\ell - b_\ell}{w_\ell \cdot \epsilon_\ell} \rceil)$ 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 a lower bound on the value of each $(\mathbf{v}_{\lambda_{L-1}})_j$ and $(\mathbf{u}_{\lambda_{L-1}})_j$ that would suffice, on its own, to reach β_L in layer L; thus, we can apply analogous reasoning to it.

Theorem 3. *Let* $\mathcal{N}'$ *be the* (Col, δ)*-GNN obtained from* $\mathcal{N}$ *by replacing* k_ℓ *with* C_ℓ *for each* $\ell \in \{1, \dots, L\}$*. Then, for each* (Col, δ)*-dataset* D*, we have* $T_{\mathcal{N}}(D) = T_{\mathcal{N}'}(D)$*.*

Proof. Consider an arbitrary (Col, δ)-dataset D, and let $\langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda \rangle$ be $\mathsf{enc}(D)$. Let $\lambda_0, \dots, \lambda_L$ and $\lambda'_0, \dots, \lambda'_L$ be the labellings of the vertices $\mathcal{V}$ induced by applying $\mathcal{N}$ and $\mathcal{N}'$ to $\mathsf{enc}(D)$, respectively. For each $v \in \mathcal{V}$, each $\ell \in \{0, \dots, L\}$, and each $i \in \{1, \dots, \delta_\ell\}$, the values of $(\mathbf{v}_{\lambda_\ell})_i$ and $(\mathbf{v}_{\lambda'_\ell})_i$ are computed using equations (11) and (12).

$$(\mathbf{v}_{\lambda_\ell})_i = \sigma\left(\sum_{j=1}^{\delta_{\ell-1}} (\mathbf{A}_\ell)_{i,j} (\mathbf{v}_{\lambda_{\ell-1}})_j + \sum_{c \in \mathsf{Col}} \sum_{j=1}^{\delta_{\ell-1}} (\mathbf{B}^c_\ell)_{i,j} \max\text{-}k_\ell\text{-sum}(\{\!\{ (\mathbf{u}_{\lambda_{\ell-1}})_j \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}) + (\mathbf{b}_\ell)_i\right) \tag{11}$$

$$(\mathbf{v}_{\lambda'_\ell})_i = \sigma\left(\sum_{j=1}^{\delta_{\ell-1}} (\mathbf{A}_\ell)_{i,j} (\mathbf{v}_{\lambda'_{\ell-1}})_j + \sum_{c \in \mathsf{Col}} \sum_{j=1}^{\delta_{\ell-1}} (\mathbf{B}^c_\ell)_{i,j} \max\text{-}C_\ell\text{-sum}(\{\!\{ (\mathbf{u}_{\lambda'_{\ell-1}})_j \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}) + (\mathbf{b}_\ell)_i\right) \tag{12}$$

By Definition 6, the elements of $\mathbf{A}_\ell$ and all $\mathbf{B}^\ell_c$ are nonnegative (while the elements of $(\mathbf{b}_\ell)_i$ may be negative) and σ is monotonically increasing. We first prove that for each $v \in \mathcal{V}$, each $\ell \in \{0, \dots, L\}$, and each $i \in \{1, \dots, \delta_\ell\}$, we have $(\mathbf{v}_{\lambda_\ell})_i \geq (\mathbf{v}_{\lambda'_\ell})_i$. We prove this by induction on ℓ. The base case holds trivially since $\lambda_0 = \lambda'_0$. For the induction step, we assume that the claim holds for $\ell - 1$, and we prove that it holds for ℓ as well. To this end, consider arbitrary $v \in \mathcal{V}$ and $i \in \{1, \dots, \delta_\ell\}$. The inductive hypothesis ensures that $(\mathbf{u}_{\lambda_{\ell-1}})_j \geq (\mathbf{u}_{\lambda'_{\ell-1}})_j$ for all $u \in \mathcal{V}$ and $j \in \{1, \dots, \delta_{\ell-1}\}$. Furthermore, by definition, $C_\ell \leq k_\ell$. Thus, subtracting the right-hand side of equality (12) from the one of equality (11) yields a positive value, so $(\mathbf{v}_{\lambda_\ell})_i \geq (\mathbf{v}_{\lambda'_\ell})_i$.

Now let ℓ_{st} be the largest $\ell \in \{1, \dots, L\}$ such that either all elements of $\mathbf{A}_\ell$ and $\mathbf{B}^c_\ell$ for each $c \in \mathsf{Col}$ are 0, or $\bigcup_{j=1}^{\delta_{\ell-1}} \mathcal{X}_{\ell-1,j} = \{0\}$; if such ℓ does not exist, then let $\ell_{\mathsf{st}} = 0$. If $\ell_{\mathsf{st}} > 0$, then Algorithm 3 returns early when the loop of line 2 reaches ℓ_{st}; otherwise, the algorithm completes all iterations. Hence, α_ℓ is defined for each $\ell_{\mathsf{st}} \leq \ell \leq L$, and β_ℓ is defined for each $\ell_{\mathsf{st}} < \ell \leq L$. We next prove the following claim.

Claim. *For each vertex $v \in \mathcal{V}$, layer $\ell \in \{\ell_{\mathsf{st}}, \dots, L\}$, and position $i \in \{1, \dots, \delta_\ell\}$, either $(\mathbf{v}_{\lambda_\ell})_i = (\mathbf{v}_{\lambda'_\ell})_i$ or $(\mathbf{v}_{\lambda_\ell})_i > (\mathbf{v}_{\lambda'_\ell})_i \geq \alpha_\ell$.*

Proof. The two possibilities in the claim are clearly mutually exclusive. We now prove the claim by induction on $\ell \in \{\ell_{\mathsf{st}}, \dots, L\}$.

For the base case, if $\ell_{\mathsf{st}} = 0$, the first alternative holds trivially since $\lambda_0 = \lambda'_0$. If $\ell_{\mathsf{st}} > 0$, consider an arbitrary $i \in \{1, \dots, \delta_\ell\}$. By the definition of ℓ_{st}, we have two possibilities. The first one is that all elements of $\mathbf{A}_{\ell_{\mathsf{st}}}$ and $\mathbf{B}^c_{\ell_{\mathsf{st}}}$ for each $c \in \mathsf{Col}$ are all zero, in which case Eqs. (11) and (12) ensure that $(\mathbf{v}_{\lambda_{\ell_{\mathsf{st}}}})_i = (\mathbf{v}_{\lambda'_{\ell_{\mathsf{st}}}})_i = \sigma((\mathbf{b}_{\ell_{\mathsf{st}}})_i)$. The second possibility is that $\mathcal{X}_{\ell-1,j} = \{0\}$ for each $j \in \{1, \dots, \delta_{\ell-1}\}$; then, Eq. (11) ensures $(\mathbf{v}_{\lambda_{\ell_{\mathsf{st}}}})_i = \sigma((\mathbf{b}_{\ell_{\mathsf{st}}})_i)$. Moreover, we have shown that $(\mathbf{v}_{\lambda_{\ell_{\mathsf{st}}}})_i \geq (\mathbf{v}_{\lambda'_{\ell_{\mathsf{st}}}})_i$; however, since all the elements in the sum in Eq. (12) other than $(\mathbf{b}_\ell)_i$ are nonnegative by Definition 6, we have $(\mathbf{v}_{\lambda'_{\ell_{\mathsf{st}}}})_i \geq \sigma((\mathbf{b}_{\ell_{\mathsf{st}}})_i)$. Hence, $(\mathbf{v}_{\lambda_\ell})_i = (\mathbf{v}_{\lambda'_\ell})_i$ and the first possibility holds.

For the induction step, consider an arbitrary layer $\ell \in \{\ell_{\mathsf{st}} + 1, \dots, L\}$ (which implies $\ell_{\mathsf{st}} < L$), vertex $v \in \mathcal{V}$, and position $i \in \{1, \dots, \delta_\ell\}$. We assume that $(\mathbf{v}_{\lambda_\ell})_i \neq (\mathbf{v}_{\lambda'_\ell})_i$ (since otherwise the first possibility holds); 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 it remains to show $(\mathbf{v}_{\lambda'_\ell})_i \geq \alpha_\ell$. Since $\ell > \ell_{\mathsf{st}}$, Algorithm 3 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 $C_\ell = \min(k_\ell, \max(k'_\ell, 0))$. 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 $j \in \{1, \dots, \delta_{\ell-1}\}$. We next consider the following four cases.

Case 1. There exists $j \in \{1, \dots, \delta_{\ell-1}\}$ such that $(\mathbf{A}_\ell)_{i,j} > 0$ and $(\mathbf{v}_{\lambda'_{\ell-1}})_j \geq \alpha_{\ell-1}$. All summands in Eq. (12) except $(\mathbf{b}_\ell)_i$ are nonnegative, so the argument of σ 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. $C_\ell > 0$ and there exist $c \in \mathsf{Col}$, $j \in \{1, \dots, \delta_{\ell-1}\}$, 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 equation (12) except $(\mathbf{b}_\ell)_i$ are nonnegative and $\max\text{-}C_\ell\text{-sum}(\{\!\{ (\mathbf{w}_{\lambda'_{\ell-1}})_j \mid \langle v, w \rangle \in \mathcal{E}^c \}\!\}) \geq (\mathbf{u}_{\lambda'_{\ell-1}})_j$ due to $C_\ell > 0$, and so the argument of σ in Eq. (12) is greater or equal to $(\mathbf{B}^\ell_c)_{i,j}(\mathbf{u}_{\lambda'_{\ell-1}})_j + (\mathbf{b}_\ell)_i \geq w_\ell \alpha_{\ell-1} + b_\ell = \beta_\ell$. 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 3. $C_\ell = 0$ and case 1 does not hold. If $C_\ell = k_\ell$, then the sum over $c \in \mathsf{Col}$ in both Eqs. (11) and (12) is always equal to 0. Furthermore, since case 1 does not hold, the induction hypothesis ensures that for each $j \in \{1, \dots, \delta_{\ell-1}\}$ such that $(\mathbf{A}_\ell)_{i,j} > 0$, we have $(\mathbf{v}_{\lambda'_{\ell-1}})_j = (\mathbf{v}_{\lambda_{\ell-1}})_j$. Thus, it follows that $(\mathbf{v}_{\lambda_\ell})_i = (\mathbf{v}_{\lambda'_\ell})_i$, as desired. If $C_\ell \neq k_\ell$, then $C_\ell = \max(0, k'_\ell)$, which together with $C_\ell = 0$, implies that $k'_\ell \leq 0$, and so $\beta_\ell \leq b_\ell$. Then, since all summands in the argument of σ other than $(\mathbf{b}_\ell)_i$ are nonnegative, the argument of σ in equation (12) is greater or equal than $(\mathbf{b}_\ell)_i \geq b_\ell \geq \beta_\ell$; moreover, since σ is monotonically increasing and $\sigma(\beta_\ell) \geq \alpha_\ell$, we have $(\mathbf{v}_{\lambda'_\ell})_i \geq \alpha_\ell$, as desired.

Case 4. None of cases 1–3 hold. Since case 1 does not hold, the induction hypothesis ensures that for each $j \in \{1, \dots, \delta_{\ell-1}\}$ such that $(\mathbf{A}_\ell)_{i,j} > 0$, we have $(\mathbf{v}_{\lambda'_{\ell-1}})_j = (\mathbf{v}_{\lambda_{\ell-1}})_j$. Furthermore, since case 2 does not hold, the induction hypothesis ensures that for each $c \in \mathsf{Col}$ and $j \in \{1, \dots, \delta_{\ell-1}\}$ 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 $\{\!\{ (\mathbf{u}_{\lambda_\ell})_j \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\} = \{\!\{ (\mathbf{u}_{\lambda'_\ell})_j \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}$. Finally, since cases 1 and 3 do not hold, we have $C_\ell > 0$. Combining these observations with our assumption that $(\mathbf{v}_{\lambda_\ell})_i \neq (\mathbf{v}_{\lambda'_\ell})_i$, it can only be the case that $C_\ell \neq k_\ell$ (so $C_\ell < k_\ell$) and there must exist at least one $j \in \{1, \dots, \delta_{\ell-1}\}$ 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. Now, recall that all summands in the argument of σ in Eq. (12) except $(\mathbf{b}_\ell)_i$ are nonnegative, so its jth value is greater than or equal to $(\mathbf{B}^c_\ell)_{i,j} \max\text{-}C_\ell\text{-sum}(\{\!\{ (\mathbf{u}_{\lambda'_\ell})_j \mid \langle v, u \rangle \in \mathcal{E}^c \}\!\}) + (\mathbf{b}_\ell)_i$. However, we have observed that there exist more than C_ℓ elements different from zero in this multiset, and

Lemma 1 and the definition of ϵ_ℓ ensure that each of them is greater than or equal to ϵ_ℓ. Thus, this value is greater than or equal to $w_\ell C_\ell \epsilon_\ell + b_\ell \geq \beta_\ell$, where the last inequality follows from $C_\ell = k'_\ell$ as $C_\ell > 0$. However, $\sigma(\beta_\ell) \geq \alpha_\ell$, and since σ is monotonic, we have $(\mathbf{v}_{\lambda'_\ell})_i \geq \alpha_\ell$, which concludes our proof by induction. □

Having proved the claim, we are now ready to complete the proof of the theorem. We consider an arbitrary constant a in D and an arbitrary unary predicate U_i in the (Col, δ)-signature with $i \in \{1, \dots, \delta\}$, and we show that $U_i(a) \in T_{\mathcal{N}}(D)$ if and only if $U_i(a) \in T_{\mathcal{N}'}(D)$; this implies the theorem since $T_{\mathcal{N}}(D)$ and $T_{\mathcal{N}'}(D)$ clearly contain the same binary atoms. By the definition of the canonical encoding and decoding, as well as the definitions of $\mathcal{N}$ and $\mathcal{N}'$, it suffices to show that $\mathsf{cls}((\mathbf{v}^a_{\lambda_L})_i) = 1$ if and only if $\mathsf{cls}((\mathbf{v}^a_{\lambda'_L})_i) = 1$. We have first shown that $(\mathbf{v}^a_{\lambda_L})_i \geq (\mathbf{v}^a_{\lambda'_L})_i$; then, by the above claim, either $(\mathbf{v}^a_{\lambda_L})_i = (\mathbf{v}^a_{\lambda'_L})_i$ or $(\mathbf{v}^a_{\lambda'_L})_i \geq \alpha_L$, since $L \geq \ell_{\mathsf{st}}$. If $(\mathbf{v}^a_{\lambda_L})_i = (\mathbf{v}^a_{\lambda'_L})_i$, our equivalence holds trivially. If $(\mathbf{v}^a_{\lambda'_L})_i \geq \alpha_L$, then the definition of α_L in Algorithm 3 ensures that $\mathsf{cls}((\mathbf{v}^a_{\lambda'_L})_i) = 1$. Since $(\mathbf{v}^a_{\lambda_L})_i \geq (\mathbf{v}^a_{\lambda'_L})_i$, we have $\mathsf{cls}((\mathbf{v}^a_{\lambda_L})_i) = 1$ and our equivalence holds as well. □

7.2. Enumerating sets $\mathcal{X}_{\ell,i}$

The definition of ϵ_ℓ in Algorithm 3 uses the smallest nonzero element of the sets $\mathcal{X}_{\ell,i}$. These sets can be infinite, so it is not obvious that one can effectively compute ϵ_ℓ. To answer this question, we next show that the sets $\mathcal{X}_{\ell,i}$ can be enumerated from the smallest element to the largest, and so we can compute the smallest nonzero element of each $\mathcal{X}_{\ell,i}$ or verify that no such element exists. These enumerations are computed inductively: to compute any element in the sequence of the values of $\mathcal{X}_{\ell,i}$, we must enumerate all elements of the sets $\mathcal{X}_{\ell-1,j}$ smaller than a particular value.

In what follows, let $\mathcal{X}^{>\alpha}_{\ell,i} = \{\alpha' \in \mathcal{X}_{\ell,i} \mid \alpha' > \alpha\}$. Each set $\mathcal{X}_{\ell,i}$ can be enumerated algorithmically using the function $\mathsf{Next}(\ell, i, \alpha)$ of Algorithm 4 as follows: for α a special symbol $\triangleright$, $\mathsf{Next}(\ell, i, \triangleright)$ returns the smallest element of $\mathcal{X}_{\ell,i}$ (note that $\mathcal{X}_{\ell,i}$ cannot be empty by construction); moreover, for $\alpha \in \mathbb{R}$ a real number, $\mathsf{Next}(\ell, i, \alpha)$ 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, or $\triangleleft$ otherwise. In Algorithm 4 we use the following notation: for $\mathbf{x}$ a vector, j an index, and r a real number, $\mathbf{x}[j \leftarrow r]$ is the vector obtained by replacing the jth element of $\mathbf{x}$ with r.

The algorithm is based on the observation that, since matrices $\mathbf{A}_\ell$ and $\mathbf{B}^c_\ell$ contain only nonnegative elements and the activation function σ is monotonically increasing, we can enumerate the values computed by Eq. (4) for 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 $(\mathbf{v}_\ell)_i$ for a vertex v at layer ℓ: vector $\mathbf{x}$ reflects possible $\mathbf{v}_{\ell-1}$ for v at the previous layer, multisets $\mathbf{Y}^c$ of the multiset family $\mathbf{Y}$ reflect possible $\mathbf{u}_{\ell-1}$ for the neighbours u of v via c, 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 it 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 each $\alpha \in \mathbb{R}$, Theorem 2 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 4 captures the formal properties of this algorithm. The proof relies on the techniques used in the proof of Theorem 2 so we defer it to Appendix A.

Theorem 4. *Algorithm 4 terminates on all inputs. Moreover, for each $\ell \in \{0, \dots, L\}$ and each $i \in \{1, \dots, \delta_\ell\}$,*

- $\mathsf{Next}(\ell, i, \triangleright)$ *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 worst-case complexity of Algorithm 4 remains unclear, and resolving it requires an understanding of how the elements of the matrices of $\mathcal{N}$ influence the number of the recursive calls to Next. We leave such an investigation for future work.

Theorem 4 ensures that the capacity C_ℓ of a GNN can be computed using Algorithm 3. Specifically, the smallest positive number in each $\mathcal{X}_{\ell,i}$ can be obtained by calling $\mathsf{Next}(\ell, i, \triangleright)$ and, if 0 is returned, calling $\mathsf{Next}(\ell, i, 0)$. Theorem 4 ensures that such call terminates and returns the expected value. All other steps can be easily computed from the parameters of $\mathcal{N}$.

7.3. Extracting the equivalent Datalog program

We now show that $\mathcal{N}$ is equivalent to a tree-like program $\mathcal{P}_\mathcal{N}$ that can be constructed from the parameters of $\mathcal{N}$. Essentially, the construction requires computing the capacity $C_\mathcal{N}$ using Algorithms 3 and 4 from Sections 7.1 and 7.2, respectively, enumerating all tree-like rules of size that is determined by L, $\delta_0, \dots, \delta_L$, and $C_\mathcal{N}$, and collecting all rules that are sound for $\mathcal{N}$; the latter can be checked using the techniques from Section 5.

Theorem 5. *GNN $\mathcal{N}$ is equivalent to the program $\mathcal{P}_\mathcal{N}$ that contains $E^c(x, y) \rightarrow E^c(x, y)$ for each $c \in \mathsf{Col}$ and, up to variable renaming, each $(L, |\mathsf{Col}| \cdot \delta_\mathcal{N} \cdot C_\mathcal{N})$-tree-like rule that is sound for $\mathcal{N}$, where $\delta_\mathcal{N} = \max(\delta_0, \dots, \delta_L)$.*

Algorithm 4 Computing $\mathsf{Next}(\ell, i, \alpha)$.

if $\ell = 0$ **then**
  if $\alpha = \triangleright$ or $\alpha < 0$ **then return** 0
  else if $\alpha < 1$ **then return** 1
  else return $\triangleleft$
Let $\mathbf{Y}_\emptyset$ be such that $\mathbf{Y}^c_\emptyset = \emptyset$ for each $c \in \mathsf{Col}$
$z := \mathsf{Val}(\ell, i, \mathsf{Start}(\ell), \mathbf{Y}_\emptyset)$
if $\alpha = \triangleright$ **then return** $\sigma(z)$
$F := \{\langle \mathsf{Start}(\ell), \mathbf{Y}_\emptyset, z\rangle\}$
while $F \neq \emptyset$ **do**
  Choose and remove $\langle \mathbf{x}, \mathbf{Y}, z\rangle$ in F with least z
  if $\sigma(z) > \alpha$ **then return** $\sigma(z)$
  for each $\mathbf{x}' \in \mathsf{Expand}(\ell, \mathbf{x})$ **do**
    $z' := \mathsf{Val}(\ell, i, \mathbf{x}', \mathbf{Y})$
    if $z' > z$ **then** add $\langle \mathbf{x}', \mathbf{Y}, z'\rangle$ to F
  for each $c \in \mathsf{Col}$ **do**
    for each $\mathbf{y} \in \mathbf{Y}^c$ **and** $\mathbf{y}' \in \mathsf{Expand}(\ell, \mathbf{y})$ **do**
      $\mathbf{Y}' := \mathbf{Y}$ and $\mathbf{Y}'^c := (\mathbf{Y}'^c \setminus \{\mathbf{y}\}) \cup \{\mathbf{y}'\}$
      $z' := \mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y}')$
      if $z' > z$ **then** add $\langle \mathbf{x}, \mathbf{Y}', z'\rangle$ to F
  if $\mathsf{Start}(\ell)$ contains a nonzero **then**
    $V := \{\mathsf{Start}(\ell)\}$
  else
    $V := \mathsf{Expand}(\ell, \mathsf{Start}(\ell))$
  for each $\mathbf{y}' \in V$ **do**
    $\mathbf{Y}' := \mathbf{Y}$ and $\mathbf{Y}'^c := \mathbf{Y}'^c \cup \{\mathbf{y}'\}$
    $z' := \mathsf{Val}(\ell, i, \mathbf{x}, \mathbf{Y}')$
    if $z' > z$ **then** add $\langle \mathbf{x}, \mathbf{Y}', z'\rangle$ to F
return $\triangleleft$

function $\mathsf{Start}(\ell)$
  return the vector $\mathbf{x}$ of dimension $\delta_{\ell-1}$ where $(\mathbf{x})_j = \mathsf{Next}(\ell - 1, j, \triangleright)$ for $j \in \{1, \dots, \delta_{\ell-1}\}$

function $\mathsf{Expand}(\ell, \mathbf{v})$
  $V := \emptyset$
  for each j **from** 1 **to** $\delta_{\ell-1}$ **do**
    $v' := \mathsf{Next}(\ell - 1, j, (\mathbf{v})_j)$
    if $v' \neq \triangleleft$ **then** $V := V \cup \{\mathbf{v}[j \leftarrow v']\}$
  return V

Proof. We show that $T_{\mathcal{N}}(D) = T_{\mathcal{P}_{\mathcal{N}}}(D)$ for each (Col, δ)-dataset D. Each rule in $\mathcal{P}_{\mathcal{N}}$ with a unary predicate in the head is sound by definition, and each rule in it with a binary predicate in the head is trivially sound since the application of $\mathcal{N}$ does not remove edges; thus, $T_{\mathcal{P}_{\mathcal{N}}}(D) \subseteq T_{\mathcal{N}}(D)$. 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 prove that $\alpha \in T_{\mathcal{P}_{\mathcal{N}}}(D)$. Theorem 3 ensures $T_{\mathcal{N}}(D) = T_{\mathcal{N}'}(D)$, where $\mathcal{N}'$ is the (Col, δ)-GNN obtained from $\mathcal{N}$ by replacing k_ℓ with C_ℓ for each $\ell \in \{1, \dots, L\}$, so $\alpha \in T_{\mathcal{N}'}(D)$. If α is a binary fact, then $\alpha \in D$ since the application of $T_{\mathcal{N}}$ does not introduce any binary facts; furthermore, $\alpha \in T_{\mathcal{P}_{\mathcal{N}}}(D)$, because $\alpha \in T_r(D)$ for $r \in T_{\mathcal{P}_{\mathcal{N}}}(D)$ the rule $E^c(x, y) \rightarrow E^c(x, y)$ where E^c is the predicate of α. Now assume that α is a unary fact, and let r be a rule obtained by running Algorithm 1 on inputs $\mathcal{N}'$, D, and α. The loop for line 17 of the algorithm can make only C_ℓ iterations: the aggregation function in layer ℓ is max-C_ℓ-sum so the sum in the condition of line 17 reaches MS after at most C_ℓ distinct elements are chosen from the set V in line 18. Hence, r must be $(L, |\mathsf{Col}| \cdot \delta_{\mathcal{N}} \cdot C_\ell)$-tree-like since at most C_ℓ fresh variables can be introduced for each y, ℓ', c, and $i \in \{1, \dots, \delta_\ell\}$. By the same argument as in the proof of Theorem 1, $\alpha \in T_r(D)$ and r is sound for $\mathcal{N}'$. But then, r is sound for $\mathcal{N}$ because $\mathcal{N}$ and $\mathcal{N}'$ are equivalent by Theorem 3. Hence, $r \in \mathcal{P}_{\mathcal{N}}$, which together with $\alpha \in T_r(D)$ implies $\alpha \in T_{\mathcal{P}_{\mathcal{N}}}(D)$, as required. □

Program $\mathcal{P}_{\mathcal{N}}$ is thus computable in principle. However, the capacity $C_{\mathcal{N}}$ can be large; furthermore, the number of tree-like rules that need to be considered is exponential in $|\mathsf{Col}| \cdot \delta_{\mathcal{N}} \cdot C_{\mathcal{N}}$ (and so in the size of the (Col, δ)-signature), and doubly exponential in the number of layers L; finally, the soundness check is worst-case exponential in the size of the rule. Therefore, such a naïve approach is mainly of theoretical interest, and we leave the development of practical rule extraction procedures for monotonic max-sum GNNs for future work.

Next, we show that, if $\mathcal{N}$ is a monotonic max GNN, then it suffices to enumerate only all $(L, |\mathsf{Col}| \cdot \delta_{\mathcal{N}})$-tree-like rules without inequalities. This simplifies the procedure by making the rules smaller, and by allowing for a soundness check that is linear in the size of the rule. In Section 8 we use these observations to develop a practical algorithm for extracting equivalent programs from monotonic max GNNs, and in Section 10 we show empirically that equivalent programs can indeed be extracted in practice from monotonic GNNs of nontrivial sizes.

Theorem 6. *If $\mathcal{N}$ is a monotonic max (Col, δ)-GNN, then $\mathcal{N}$ is equivalent to the program $\mathcal{P}_{\mathcal{N}}$ that contains $E^c(x, y) \rightarrow E^c(x, y)$ for each $c \in \mathsf{Col}$ and, up to variable renaming, each $(L, |\mathsf{Col}| \cdot \delta_{\mathcal{N}})$-tree-like rule without inequalities that is sound for $\mathcal{N}$, where $\delta_{\mathcal{N}} = \max\{\delta_0, \dots, \delta_L\}$.*

Proof. We show that $T_{\mathcal{P}_{\mathcal{N}}}(D) = T_{\mathcal{N}}(D)$ for each (Col, δ)-dataset D. Each rule in $\mathcal{P}_{\mathcal{N}}$ is sound and thus $T_{\mathcal{P}_{\mathcal{N}}}(D) \subseteq T_{\mathcal{N}}(D)$. To prove $T_{\mathcal{N}}(D) \subseteq T_{\mathcal{P}_{\mathcal{N}}}(D)$, consider an arbitrary unary fact $\alpha \in T_{\mathcal{N}}(D)$; the case of binary facts is the same as in the proof of Theorem 5. Let r be a rule obtained by running Algorithm 1 on inputs $\mathcal{N}$, D, and α. The loop for line 17 in the algorithm can make just one iteration: the aggregation function in layer ℓ is max so the sum in the condition of line 17 reaches MS after at most one element is chosen from the set V in line 18. Thus, r must be $(L, \delta_{\mathcal{N}} \cdot |\mathsf{Col}|)$-tree-like since at most one fresh variable can be introduced for each y, ℓ, c, and $i \in \{1, \dots, \delta_\ell\}$, and moreover r does not contain inequalities because set Z in the algorithm contains at most one variable. Then, by the same argument as the proof of Theorem 1, $\alpha \in T_r(D)$ and r is sound for $\mathcal{N}$. 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. □

8. Practical extraction of equivalent Datalog programs from monotonic max GNNs

In this section we present an optimised, practical algorithm for extracting an equivalent Datalog program from a monotonic max GNN. The basic principles can be extended to monotonic max-sum GNNs as well, but the resulting approach is likely to still be impractical because the soundness check in Proposition 2 must consider a large number of substitutions.

Let $\mathcal{N}$ be a monotonic max (Col, δ)-GNN with L layers of dimensions $\delta_0, \dots, \delta_L$, and let $\delta_{\mathcal{N}} = \max(\delta_0, \dots, \delta_L)$. Our optimised algorithm follows the basic strategy suggested by Theorem 6, which involves enumerating all $(L, |\mathsf{Col}| \cdot \delta_{\mathcal{N}})$-tree-like rules without inequalities and including into the output the rules that are sound for $\mathcal{N}$; the latter can be checked using Proposition 3. However, the algorithm greatly reduces the number of candidate rules by using the following two observations.

- Analogously to Section 6.2, the sparsity in the GNN's matrices makes some body atoms irrelevant. For example, to check whether $r = B_1 \wedge \cdots \wedge B_n \rightarrow H$ is sound for $\mathcal{N}$, we test whether $H\nu \in T_{\mathcal{N}}(\{B_1\nu, \dots, B_n\nu\})$ for ν a substitution mapping each variable in r to a distinct constant. However, if the feature vector elements corresponding to some atom $B_i\nu$ are all multiplied by zero weights during the application of $\mathcal{N}$, then this atom cannot affect the derivation of $H\nu$ and is thus irrelevant. Consequently, rule r is sound for $\mathcal{N}$ if and only if rule $r' = B_1 \wedge \cdots \wedge B_{i-1} \wedge B_{i+1} \wedge \cdots \wedge B_n \rightarrow H$ is sound for $\mathcal{N}$. Thus, our optimised algorithm considers only r' as a candidate, but not r.
- Assume that a rule r subsumes a rule r', so $T_{r'}(D) \subseteq T_r(D)$ for each (Col, δ)-dataset D. If r is sound for $\mathcal{N}$, then only r needs to be included into the output Datalog program, but not r' (even though r' is sound for $\mathcal{N}$ too). Therefore, if our optimised algorithm proves that r is sound, it will not further consider any other rule that is subsumed by r.

In Section 8.1, we use our first observation and show how to exploit matrix sparsity to construct a tree-like conjunction Γ^i_{all} for each $i \in \{1, \dots, \delta\}$ that provides an 'upper bound' on the bodies of candidate rules that need to be considered—that is, our candidates will be tree-like rules of the form $\Gamma \rightarrow U_i(x)$ where Γ consists only of atoms from Γ^i_{all}. In particular, if the matrices of $\mathcal{N}$ contain no zeros, Γ^i_{all} is simply the body of the largest $(L, |\mathsf{Col}| \cdot \delta_{\mathcal{N}})$-tree-like rule for x; however, if these matrices are sparse, we can exclude irrelevant atoms and obtain a much smaller Γ^i_{all}. Then, in Section 8.2, we present our optimised algorithm, which enumerates and checks for soundness all candidate rules in a systematic and efficient way, all while exploiting subsumption relations between rules as per our second observation.

8.1. Reducing the space of candidate rules

Algorithm 5 takes as input a monotonic max GNN $\mathcal{N}$ and an index $i \in \{1, \dots, \delta\}$, and it constructs a conjunction Γ^i_{all} satisfying the property outlined above. The algorithm initialises Γ^i_{all} to the empty conjunction in line 1. Furthermore, for each layer ℓ, the algorithm maintains a relevance mapping μ_ℓ, which maps each variable of Γ^i_{all} to a subset of $\{1, \dots, \delta_\ell\}$. These mappings capture a notion of relevance similarly to Algorithm 1; however, whereas in Algorithm 1 this is for a specific input dataset D and substitution θ, mappings μ_ℓ are defined in Algorithm 5 for an arbitrary dataset and an arbitrary grounding of Γ^i_{all}. Thus, $h \notin \mu_\ell(y)$ means that for each tree-like rule $\Gamma \rightarrow U_i(x)$ mentioning y with $\Gamma \subseteq \Gamma^i_{all}$, each dataset D, and each substitution ν such that $\Gamma\nu \subseteq D$, the value of $(\mathbf{v}^{\nu(y)}_\ell)_h$ is irrelevant in the computation of $(\mathbf{v}^{\nu(x)}_L)_i$ when applying $\mathcal{N}$ to the dataset's encoding. The value $\mu_L(x)$ is initialised to $\{i\}$ in line 3 just like in Algorithm 1. Relevance is then propagated backwards through the layers of $\mathcal{N}$ in lines 4–15. In particular, assume that $j \in \mu_\ell(y)$—that is, the value of $(\mathbf{v}^{\nu(y)}_\ell)_j$ is relevant. Then, the value of $(\mathbf{v}^{\nu(y)}_{\ell-1})_h$ is relevant to the computation of $(\mathbf{v}^{\nu(y)}_\ell)_j$ only if $(\mathbf{A}_\ell)_{j,h} \neq 0$. The latter is checked in the condition in line 9, and, if the condition holds, h is added to $\mu_{\ell-1}(y)$ in line 10. Analogously, the value of $(\mathbf{v}^{\nu(z)}_{\ell-1})_h$ is relevant in layer ℓ only if $\nu(z)$ is a neighbour of $\nu(y)$ connected by an edge of colour c and $(\mathbf{B}^c_\ell)_{j,h} \neq 0$. The latter is checked in the condition in line 11; if the condition holds, Γ^i_{all} is extended in line 13 with an atom $E^c(y, z)$ that can match this neighbour, and h is added to $\mu_{\ell-1}(z)$ in line 15. After relevance is propagated through all layers, the domain of μ_0 contains all variables that produce matches in layer zero, and, for each such variable y, the set $\mu_0(y)$ contains the indexes of all relevant positions. Thus, Γ^i_{all} is extended in line 16 with the corresponding unary atoms.

Algorithm 5 Computing the conjunction Γ^i_{all}.

Inputs:
$\mathcal{N}$: a monotonic max (Col, δ)-GNN with L layers and dimensions $\delta_0, \dots, \delta_L$
i : a natural number from $\{1, \dots, \delta\}$

1: $\Gamma^i_{all} := \top$
2: $T := \{x\}$
3: $\mu_L(x) := \{i\}$
4: **for** ℓ **from** L **down to** 1 **do**
5: $\quad T' := T$
6: $\quad$ **for each** variable $y \in T'$ **do**
7: $\quad\quad \mu_{\ell-1}(y) := \emptyset$
8: $\quad\quad$ **for each** $h \in \{1, \dots, \delta_{\ell-1}\}$ **do**
9: $\quad\quad\quad$ **if** $(\mathbf{A}_\ell)_{j,h} \neq 0$ for some $j \in \mu_\ell(y)$ **then**
10: $\quad\quad\quad\quad \mu_{\ell-1}(y) := \mu_{\ell-1}(y) \cup \{h\}$
11: $\quad\quad\quad$ **for each** $c \in \mathsf{Col}$ such that $(\mathbf{B}^c_\ell)_{j,h} \neq 0$ for some $j \in \mu_\ell(y)$ **do**
12: $\quad\quad\quad\quad$ Let z be a fresh variable
13: $\quad\quad\quad\quad \Gamma^i_{all} := \Gamma^i_{all} \wedge E^c(y, z)$
14: $\quad\quad\quad\quad T := T \cup \{z\}$
15: $\quad\quad\quad\quad \mu_{\ell-1}(z) := \{h\}$
16: $\Gamma^i_{all} := \Gamma^i_{all} \wedge \bigwedge\{U_j(y) \mid y \in \mathsf{dom}(\mu_0)$ and $j \in \mu_0(y)\}$
17: **return** Γ^i_{all}

Example 3. Let $\mathcal{N}$ be a monotonic max (Col, δ)-GNN with one layer, where $\mathsf{Col} = \{c\}$, $\delta = 3$, and the model's matrices are

$$\mathbf{A}_1 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \mathbf{B}^c_1 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}.$$

Consider the application of Algorithm 5 to $\mathcal{N}$ and $i = 1$. The algorithm initialises Γ^1_{all} to the empty conjunction, sets $\mu_1(x) = \{1\}$, and then considers the loop in lines 4–15 for $\ell = 0$. Condition in line 9 holds for $j = 1$ and $h = 2$, so $\mu_0(x) = \{2\}$ is set in line 10. Note that $(\mathbf{A}_1)_{2,3} \neq 0$, but $2 \notin \mu_1(x)$, so 3 is not added to $\mu_0(x)$. Moreover, $(\mathbf{B}^c_1)_{1,h} \neq 0$ only for $h = 1$ and $h = 2$; thus, the algorithm introduces variables y^c_1 and y^c_2, with $\mu_0(y^c_1) = \{1\}$ and $\mu_0(y^c_2) = \{2\}$, respectively, and it extends Γ^1_{all} to $E^c(x, y^c_1) \wedge E^c(x, y^c_2)$. Finally, conjunction Γ^1_{all} is extended with unary atoms producing

$$\Gamma^1_{all} = U_2(x) \wedge E^c(x, y^c_1) \wedge U_1(y^c_1) \wedge E^c(x, y^c_2) \wedge U_2(y^c_2).$$

Note that there are only $2 \cdot 3 \cdot 3 = 12$ conjunctions contained in Γ^1_{all} that are tree-like for x: we can include $U_2(x)$ or not, and, for each $i \in \{1, 2\}$, we can include nothing, $E^c(x, y^c_i)$, or $E^c(x, y^c_i) \wedge U_i(y^c_i)$. Hence, the rule extraction algorithm only needs to consider 12 candidate tree-like rules with $U_1(x)$ in the head. In contrast, an analogous calculation reveals that the total number of $(1, 3)$-tree-like rules with $U_1(x)$ in the head is $2^3 \cdot (2^3 + 1) \cdot (2^3 + 1) \cdot (2^3 + 1) = 5,832$.

Lemma 2. *For each monotonic max* (Col, δ)*-GNN* $\mathcal{N}$*, each rule over the* (Col, δ)*-signature of the form* $\Gamma \to U_i(x)$ *that is sound for* $\mathcal{N}$*, and* Γ^i_{all} *the result of applying Algorithm 5 to* $\mathcal{N}$ *and* i*, there exists a subset* $\Gamma' \subseteq \Gamma^i_{all}$ *such that* Γ' *is tree-like for* x*, rule* $\Gamma' \to U_i(x)$ *is sound for* $\mathcal{N}$*, and rule* $\Gamma' \to U_i(x)$ *subsumes* $\Gamma \to U_i(x)$*.*

Proof. Fix a monotonic max GNN $\mathcal{N}$ and a rule $r = \Gamma \to U_i(x)$ that is sound for $\mathcal{N}$, as well as the set Γ^i_{all} as in the formulation of the lemma. Without loss of generality, we assume that Γ and Γ^i_{all} share only the variable x. Let ν be a substitution mapping each variable occurring in r to a distinct constant not occurring in r, and let A be the set of body atoms in Γ; clearly, $U_i(x)\nu \in T_r(A\nu)$. Let $r' = \Gamma' \to U_i(x)$ be the rule obtained by applying Algorithm 1 to $\mathcal{N}$, $A\nu$, and $U_i(x)\nu$, and let θ be the substitution constructed during this application. Furthermore, let ρ be a mapping of each variable y occurring in Γ' to the (unique) variable z in r such that $\theta(y) = \nu(z)$. As shown in the proof of Theorem 1, substitution θ witnesses $H\nu \in T_{r'}(A\nu)$; thus, on the one hand, we have $U_i(x)\theta = U_i(x)\nu$, which implies $\rho(x) = x$, and, on the other hand, we have $\Gamma'\theta \subseteq A\nu$, which implies $\Gamma'\rho \subseteq \Gamma$. Therefore, r' subsumes r. Theorem 1 ensures that Γ' is tree-like for x and that r' is sound for $\mathcal{N}$. Finally, note that the construction of Γ' closely follows the construction of Γ^i_{all} in Algorithm 5. In particular, the set T in Algorithm 5 plays the role of the domain of the substitution θ of Algorithm 1; moreover, since $\mathcal{N}$ is a monotonic max GNN, the loop in lines 17–23 is always evaluated at most once. Thus, whenever Algorithm 1 introduces an atom $E^c(y, z)$ to Γ', Algorithm 5 introduces the same atom (up to variable renaming) to Γ^i_{all}. Consequently, Γ' is contained in Γ^i_{all} up to variable renaming, as required. □

The following proposition is a simple consequence of Lemma 2 and it ensures that Γ^i_{all} satisfies the required properties.

Proposition 4. *Let* $\mathcal{N}$ *be a monotonic max* (Col, δ)*-GNN, and let* $\mathcal{O}_\mathcal{N}$ *be the program that contains* $E^c(x, y) \to E^c(x, y)$ *for each* $c \in \mathsf{Col}$ *and each rule* $\Gamma \to U_i(x)$ *such that* $\Gamma \subseteq \Gamma^i_{all}$*, conjunction* Γ *is tree-like for* x*, rule* $\Gamma \to U_i(x)$ *is sound for* $\mathcal{N}$*,* $i \in \{1, \dots, \delta\}$*, and* Γ^i_{all} *is the result of applying Algorithm 5 to* $\mathcal{N}$ *and* i*. Then,* $\mathcal{O}_\mathcal{N}$ *is equivalent to* $\mathcal{N}$*.*

Proof. Fix $\mathcal{N}$ and $\mathcal{O}_\mathcal{N}$ as in the claim, let $\mathcal{P}_\mathcal{N}$ be the program as specified in Theorem 6, and consider an arbitrary (Col, δ)-dataset D. All rules in $\mathcal{O}_\mathcal{N}$ are sound for $\mathcal{N}$, so the inclusion $T_{\mathcal{O}_\mathcal{N}}(D) \subseteq T_\mathcal{N}(D)$ holds trivially. For the other direction, Theorem 6 ensures $T_\mathcal{N}(D) \subseteq T_{\mathcal{P}_\mathcal{N}}(D)$. Furthermore, the definition of $\mathcal{O}_\mathcal{N}$ ensures that $\mathcal{O}_\mathcal{N}$ contains the same rules with binary predicates in the head as $\mathcal{P}_\mathcal{N}$. Moreover, each rule $r \in \mathcal{P}_\mathcal{N}$ with a unary predicate in the head is sound for $\mathcal{N}$, so Lemma 2 ensures that $\mathcal{O}_\mathcal{N}$ contains a rule r' that subsumes r—that is, $T_r(D) \subseteq T_{r'}(D)$. Thus, we have $T_{\mathcal{P}_\mathcal{N}}(D) \subseteq T_{\mathcal{O}_\mathcal{N}}(D)$, so $T_\mathcal{N}(D) \subseteq T_{\mathcal{O}_\mathcal{N}}(D)$, as required. □

8.2. Exploring the space of candidate rules

By Proposition 4, program $\mathcal{O}_\mathcal{N}$ equivalent to $\mathcal{N}$ can be obtained by considering each $i \in \{1, \dots, \delta\}$ and each subset of $\Gamma \subseteq \Gamma^i_{all}$ that is tree-like for x, and checking whether $\Gamma \to U_i(x)$ is sound for $\mathcal{N}$. However, the search space of such a procedure can still be large. Algorithm 6 uses rule subsumption to considerably prune the set of considered rules. The loop in lines 2–19 considers all possible head predicates U_i. In each iteration, the algorithm maintains two sets: *frontier* contains conjunctions that cover the remaining search space, and *min_pos* accumulates conjunctions that constitute the bodies of the rules with unary head predicates that should be included into the resulting program. The two sets jointly satisfy the following invariant: each rule $\Gamma' \to U_i(x)$ in $\mathcal{O}_\mathcal{N}$ is subsumed by a rule $\Gamma'' \to U_i(x)$ for some $\Gamma'' \in \mathit{frontier} \cup \mathit{min\_pos}$. This invariant is established in line 5. In each iteration of the loop in lines 6–16, a candidate conjunction $\Gamma \in \mathit{frontier}$ is selected and a check is made to determine whether the rule $\Gamma \to U_i(x)$ is sound for $\mathcal{N}$. If the check succeeds, conjunction Γ is moved from *frontier* to *min_pos* in line 10, so the invariant remains preserved. Moreover, all rules produced by conjunctions in *min_pos* that are subsumed by $\Gamma \to U_i(x)$ are removed in line 9; this does not affect the algorithm's correctness, but it ensures that the resulting program does not contain redundant rules. If $\Gamma \to U_i(x)$ is not sound for $\mathcal{N}$, then no rule of the form $\Gamma_n \to U_i(x)$ with $\Gamma_n \subseteq \Gamma$ is sound for $\mathcal{N}$ either; hence, such rules do not need to be checked and are removed from *frontier* in line 13. Finally, to ensure that the removal of all Γ_n (including Γ itself) from *frontier* does not affect the invariant, *frontier* is extended in line 16 with each conjunction $\Gamma_s \subseteq \Gamma^i_{all}$ that extends one such Γ_n with one atom and does not contain a conjunction in *min_pos*. Set *frontier* eventually becomes empty, whereas *min_pos* contains the bodies of all relevant rules with predicate U_i in the head. These rules are added to the output program in line 19.

Algorithm 6 Optimised extraction of an equivalent program.

Inputs:
$\mathcal{N}$: a monotonic max (Col, δ)-GNN with L layers and dimensions $\delta_0, \dots, \delta_L$

1: $\mathcal{P} := \{E^c(x, y) \to E^c(x, y) \mid c \in \mathsf{Col}\}$
2: **for each** $i \in \{1, \dots, \delta\}$ **do**
3: Construct Γ^i_{all} using Algorithm 5
4: $\mathit{min\_pos} := \emptyset$
5: $\mathit{frontier} := \{\top\}$
6: **while** $\mathit{frontier} \neq \emptyset$ **do**
7: Choose an arbitrary $\Gamma \in \mathit{frontier}$
8: **if** $\Gamma \to U_i(x)$ is sound for $\mathcal{N}$ **then**
9: Remove from *min_pos* each Γ_p such that $\Gamma \subseteq \Gamma_p$
10: Remove Γ from *frontier* and add Γ to *min_pos*
11: **else**
12: **while** there exists $\Gamma_n \in \mathit{frontier}$ such that $\Gamma_n \subseteq \Gamma$ **do**
13: Remove a smallest such Γ_n from *frontier*
14: **for each** $\Gamma_s \subseteq \Gamma^i_{all}$ that is tree-like for x and extends Γ_n by a single atom **do**
15: **if** $\Gamma_p \not\subseteq \Gamma_s$ for each $\Gamma_p \in \mathit{min\_pos}$ **then**
16: Add Γ_s to *frontier*
17: **for each** $\Gamma \in \mathit{min\_pos}$ **do**
18: **if** $\mathcal{P}$ does not contain $\Gamma \to U_i(x)$ up to variable renaming **then**
19: Add $\Gamma \to U_i(x)$ to $\mathcal{P}$
20: **return** $\mathcal{P}$

Theorem 7. *For $\mathcal{N}$ a monotonic max (Col, δ)-GNN, Algorithm 6 outputs a program equivalent to $\mathcal{N}$.*

Proof. Fix a monotonic max (Col, δ)-GNN $\mathcal{N}$. Algorithm 6 always terminates on input $\mathcal{N}$ since each update to *frontier* removes a conjunction from it and possibly adds one or more conjunctions with a strictly larger number of atoms; moreover, Γ^i_{all} provides an upper bound to the number of atoms in conjunctions in *frontier*. Hence, *frontier* eventually becomes empty so the loop in line 6 terminates.

Let $\mathcal{P}$ be the output of Algorithm 6 on input $\mathcal{N}$; we next show that $\mathcal{P}$ is equivalent to $\mathcal{N}$. Each rule in $\mathcal{P}$ is clearly sound for $\mathcal{N}$, so we only need to show that $\mathcal{P}$ is complete for $\mathcal{N}$. By Proposition 4, it suffices to show that each rule of $\mathcal{O}_\mathcal{N}$, the program defined in the proposition, is subsumed by a rule in $\mathcal{P}$. This holds trivially for rules with a binary predicate in the head, so we focus on rules with unary head predicates. We fix an arbitrary $i \in \{1, \dots, \delta\}$ and consider the execution of the for-loop in line 2 for i. We then prove that the following property $(*)$ holds after line 5 and after each iteration of the loop in lines 6–16:

for each rule $\Gamma' \to U_i(x) \in \mathcal{O}_{\mathcal{N}}$, there exists a conjunction $\Gamma'' \in \mathit{min\_pos} \cup \mathit{frontier}$ such that $\Gamma'' \subseteq \Gamma'$.

The empty conjunction $\top$ is a subset of each conjunction Γ' considered in property $(*)$, so the property clearly holds after line 5. Now assume that $(*)$ holds just after line 6, and let Γ be the conjunction chosen in line 7. Sets $\mathit{min\_pos}$ and $\mathit{frontier}$ can be updated in one of the following two ways.

- Assume that the rule $\Gamma \to U_i(x)$ is sound for $\mathcal{N}$ in line 8. Each Γ_p that is removed from $\mathit{min\_pos}$ in line 9 satisfies $\Gamma \subseteq \Gamma_p$, so property $(*)$ clearly holds after line 9. Moreover, Γ is simply moved from $\mathit{frontier}$ to $\mathit{min\_pos}$ in line 10, so property $(*)$ holds after line 10 as well.
- Assume that the rule $\Gamma \to U_i(x)$ is not sound for $\mathcal{N}$ in line 8. Property $(*)$ can be affected only in line 13 when some Γ_n is removed from $\mathit{frontier}$; thus, consider an arbitrary $\Gamma' \to U_i(x) \in \mathcal{O}_{\mathcal{N}}$ such that $\Gamma_n \subseteq \Gamma'$ and property $(*)$ does not hold for $\Gamma' \to U_i(x)$ after removing Γ_n from $\mathit{frontier}$. Since $\Gamma_n \subseteq \Gamma$ by line 12, rule $\Gamma_n \to U_i(x)$ is not sound for $\mathcal{N}$. Therefore, $\Gamma_n \subsetneq \Gamma'$ and there exists a conjunction $\Gamma_s \subseteq \Gamma^i_{all}$ that is tree-like for x, extends Γ_n by a single atom, and $\Gamma_s \subseteq \Gamma'$. By our assumption, property $(*)$ does not hold for $\Gamma' \to U_i(x)$, so for each $\Gamma_p \in \mathit{min\_pos}$, we have $\Gamma_p \not\subseteq \Gamma'$ and thus $\Gamma_p \not\subseteq \Gamma_s$. But then, conjunction Γ_s is added to $\mathit{frontier}$ in line 16, which ensures that property $(*)$ holds again for Γ'.

Property $(*)$ thus holds after each iteration of the while-loop from line 6. When the loop finishes, $\mathit{frontier}$ is empty so property $(*)$ ensures that for each rule $\Gamma' \to U_i(x) \in \mathcal{O}_{\mathcal{N}}$, there exists $\Gamma'' \in \mathit{min\_pos}$ such that $\Gamma'' \subseteq \Gamma'$. Hence, after line 17, each rule $\Gamma' \to U_i(x) \in \mathcal{O}_{\mathcal{N}}$ is subsumed by a rule of $\mathcal{P}$.

The process is repeated for each $i \in \{1, \dots, \delta\}$, so at the end of the execution, each rule in $\mathcal{O}_{\mathcal{N}}$ is subsumed by a rule in the program $\mathcal{P}$, as required. □

9. Complete characterisation of monotonic max GNNs

We now complement our results from Section 7 and show that for monotonic max GNNs, the converse holds as well: for each tree-like Datalog program without inequalities over the signature that consists of unary predicates $U_1, \dots, U_\delta$ and binary predicates E^c with $c \in \mathsf{Col}$, we can construct an equivalent monotonic max GNN. Towards this goal, in the rest of this section we fix a program $\mathcal{P}$ consisting of (d, f)-tree-like rules without inequalities. To ensure that the program can be equivalent to the canonical transformation induced by a GNN, which neither introduces nor removes any binary atom, we assume that $\mathcal{P}$ additionally contains $E^c(x, y) \to E^c(x, y)$ for each $c \in \mathsf{Col}$. Now, let $\tau_1, \dots, \tau_n$ be a sequence containing, up to variable renaming, each (d, f)-tree-like conjunction for variable x without inequalities ordered by the increasing depth—that is, for each i and j with $i < j$, the depth of τ_i is less than or equal to the depth of τ_j. By definition, each τ_i can be written as

$$\tau_i = \varphi_{i,0} \wedge \bigwedge_{k=1}^{m_i} \Big(E^{c_k}(x, y_k) \wedge \varphi_{i,k} \Big), \tag{13}$$

where $\varphi_{i,0}$ is a conjunction of unary atoms over the variable x, each $\varphi_{i,k}$ with $k \in \{1, \dots, m_i\}$ is a $(d-1, f)$-tree-like conjunction for y_k, and for each pair of distinct $k, k' \in \{1, \dots, m_i\}$, conjunctions $\varphi_{i,k}$ and $\varphi_{i,k'}$ do not have variables in common. Note that m_i may be zero, conjunctions $\varphi_{i,k}$ can be $\top$, and colours c_k need not be distinct.

We next define the monotonic max (Col, δ)-GNN $\mathcal{N}_{\mathcal{P}}$ of form (3). 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. Dimensions δ_ℓ are defined as follows:

- $\delta_0 = \delta_L = \delta$, and
- δ_ℓ for each $\ell \in \{1, \dots, L-1\}$ is the number of conjunctions among $\tau_1, \dots, \tau_n$ 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 each $c \in \mathsf{Col}$, each $\ell \in \{1, \dots, L\}$, each $i \in \{1, \dots, \delta_\ell\}$, and each $j \in \{1, \dots, \delta_{\ell-1}\}$:

$$(\mathbf{A}_\ell)_{i,j} = \begin{cases} 1 & \text{if one of the following holds:} \\ & \bullet\ \ell = 1 \text{ and } \tau_i \text{ contains } U_j(x); \\ & \bullet\ \ell \in \{2, \dots, L-1\} \text{ and} \\ & \quad \text{either } i \in \{1, \dots, \delta_{\ell-1}\} \text{ and } i = j, \\ & \quad \text{or } i \in \{\delta_{\ell-1}+1, \dots, \delta_\ell\} \text{ and } \varphi_{i,0} = \tau_j; \\ & \bullet\ \ell = L \text{ and } \mathcal{P} \text{ contains rule } \tau_j \to U_i(x) \text{ up to variable renaming;} \\ 0 & \text{otherwise;} \end{cases}$$

$$(\mathbf{B}^c_\ell)_{i,j} = \begin{cases} 1 & \text{if both of the following hold:} \\ & \bullet\ \ell \in \{2, \dots, L-1\} \text{ and} \\ & \bullet\ \text{there exists } k \in \{1, \dots, m_i\} \text{ such that} \\ & \quad c = c_k, \text{and} \\ & \quad \text{formulas } \varphi_{i,k} \text{ and } \tau_j \text{ are equal up to variable renaming;} \\ 0 & \text{otherwise;} \end{cases}$$

$$(\mathbf{b}_\ell)_i = \begin{cases} 1 - \sum_{j=1}^{\delta_{\ell-1}} \left((\mathbf{A}_\ell)_{i,j} + \sum_{c \in \mathsf{Col}} (\mathbf{B}^c_\ell)_{i,j} \right) & \text{if one of the following holds:} \\ & \bullet\ \ell = 1; \\ & \bullet\ \ell \in \{2, \dots, L-1\} \text{ and } i \in \{\delta_{\ell-1}+1, \dots, \delta_\ell\}; \\ 0 & \text{otherwise.} \end{cases}$$

To understand the intuition behind the construction of $\mathcal{N}_\mathcal{P}$, assume that $\mathcal{N}_\mathcal{P}$ is applied to the canonical encoding of D, and consider a vector $\mathbf{v}_\ell$ labelling a vertex corresponding to some constant a of D in layer $\ell \in \{1, \dots, L-1\}$. Then, the ith 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 the variable x to a. This is formally captured by Lemma 3. The final layer L of $\mathcal{N}_\mathcal{P}$ simply realises a disjunction over all relevant bodies of rules in $\mathcal{P}$ for a given rule head.

Lemma 3. *For each* (Col, δ)*-dataset* D*, each layer* $\ell \in \{1, \dots, L-1\}$ *of* $\mathcal{N}_\mathcal{P}$*, each position* $i \in \{1, \dots, \delta_\ell\}$*, each constant* a *in* D*, and for* $\mathbf{v}_\ell$ *the labelling of the vertex* $v = v^a$ *corresponding to* a *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* a *such that* $D \models \tau_i \nu$*, and*
- $(\mathbf{v}_\ell)_i = 0$ *otherwise.*

Proof. For an arbitrary (Col, δ)-dataset D, let $\langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda \rangle$ be the canonical encoding $\mathsf{enc}(D)$ of D, and consider the application of $\mathcal{N}_\mathcal{P}$ to this graph. We prove the claim by induction on ℓ.

For the base case $\ell = 1$, consider an arbitrary constant a and an arbitrary position $i \in \{1, \dots, \delta_1\}$, and let v be the vertex corresponding to a. Let J_1 and J'_1 be the sets of indices defined as follows.

$$J_1 = \{j \in \{1, \dots, \delta_0\} \mid (\mathbf{v}_0)_j = 1\} \tag{14}$$

$$J'_1 = \{j \in \{1, \dots, \delta_0\} \mid (\mathbf{A}_1)_{i,j} = 1\} \tag{15}$$

Recall that $(\mathbf{v}_0)_j \in \{0, 1\}$ and $(\mathbf{A}_1)_{i,j} \in \{0, 1\}$ for each $j \in \{1, \dots, \delta_0\}$; furthermore, $\mathbf{B}^c_1$ has all elements equal to 0 for each $c \in \mathsf{Col}$, and so $(\mathbf{b}_1)_i = 1 - |J'_1|$. Thus, the argument of σ in the computation of $(\mathbf{v}_1)_i$ is equal to

$$|J_1 \cap J'_1| + 1 - |J'_1|, \tag{16}$$

which is equal to 1 if $J'_1 \subseteq J_1$, and otherwise it less than or equal to 0. Hence, $(\mathbf{v}_1)_i = 1$ if $J'_1 \subseteq J_1$, and otherwise $(\mathbf{v}_1)_i = 0$. Thus, to prove the claim, we show that $J'_1 \subseteq J_1$ if and only if $D \models \tau_i \nu$ for $\nu = \{x \mapsto a\}$; since $i \in \{1, \dots, \delta_1\}$, formula τ_i may contain only atoms over x by definition, and there may be between zero and δ_0 such atoms. For the ($\Leftarrow$) direction, assume that $D \models \tau_i \nu$ for $\nu = \{x \mapsto a\}$, 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 $U_j(a) \in D$, so our encoding ensures $(\mathbf{v}_0)_j = 1$; hence, $j \in J_1$, as required. For the ($\Rightarrow$) direction, assume that $J'_1 \subseteq J_1$. By the definitions of $\mathbf{A}_1$ and J'_1, for each $U_j(x) \in \tau_i$, we have $j \in J'_1$; however, since $J'_1 \subseteq J_1$, we have $j \in J_1$ and so $U_j(a) \in D$. Hence, $D \models \tau_i \nu$ for $\nu = \{x \mapsto a\}$.

For the induction step, consider $\ell \in \{1, \dots, L-1\}$ such that the claim holds for $\ell - 1$; consider also an arbitrary constant b and an arbitrary position $i \in \{1, \dots, \delta_\ell\}$, and let v be the vertex corresponding to b. We have two possibilities. The first one is $i \in \{1, \dots, \delta_{\ell-1}\}$; then, $(\mathbf{A}_\ell)_{i,j} = 1$ if and only if $j = i$, $(\mathbf{B}^c_\ell)_{i,j} = 0$ for each j, 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 possibility is $i \in \{\delta_{\ell-1}+1, \dots, \delta_\ell\}$. For each $c \in \mathsf{Col}$, let $J_{\ell,c}$ and $J'_{\ell,c}$ be the sets of indexes defined as follows.

$$J_{\ell,c} = \{j \in \{1, \dots, \delta_{\ell-1}\} \mid \text{there exists a vertex } u \text{ such that } \langle v, u \rangle \in \mathcal{E}^c \text{ and } (\mathbf{u}_{\ell-1})_j = 1\} \tag{17}$$

$$J'_{\ell,c} = \{j \in \{1, \dots, \delta_{\ell-1}\} \mid (\mathbf{B}^c_\ell)_{i,j} = 1\} \tag{18}$$

Since $\varphi_{i,0}$ is a conjunction of unary atoms that mention x, there exists $j_0 \in \{1, \dots, \delta_{\ell-1}\}$ such that $\varphi_{i,0} = \tau_{j_0}$. Furthermore, recall that $(\mathbf{A}_\ell)_{i,j} \in \{0, 1\}$ and $(\mathbf{v}_{\ell-1})_j \in \{0, 1\}$, and $(\mathbf{B}^c_\ell)_{i,j} \in \{0, 1\}$ for each $j \in \{1, \dots, \delta_{\ell-1}\}$ and $c \in \mathsf{Col}$; moreover, $(\mathbf{b}_\ell)_i = -\sum_{c \in \mathsf{Col}} |J'_{\ell,c}|$, because, by construction, $(\mathbf{A}_\ell)_{i,i} = 1$ and $(\mathbf{A}_\ell)_{i,j} = 0$ for each $j \in \{1, \dots, \delta_{\ell-1}\}$ such that $j \neq i$. Thus, the argument of σ in the computation of $(\mathbf{v}_{\lambda_\ell})_i$ is equal to

$$(\mathbf{v}_{\ell-1})_{j_0} + \sum_{c \in \mathsf{Col}} \left(|J_{\ell,c} \cap J'_{\ell,c}| - |J'_{\ell,c}| \right), \tag{19}$$

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 b such that $D \models \tau_i \nu$.

For the ($\Leftarrow$) direction, assume that such a substitution ν exists. Then $D \models \varphi_{i,0} \nu$; 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 $j \in J'_{\ell,c}$; by the definition of $\mathbf{B}^c_\ell$ and $J'_{\ell,c}$, there exists $k \in \{1, \dots, m_i\}$ such that $c = c_k$, and τ_j and $\varphi_{i,k}$ are equal up to variable renaming. Let $d = \nu(y_k)$, and let $u = v^d$ be the vertex corresponding to d. Our assumption implies $D \models E^c(x, y_k)\nu$ so $\langle v, u \rangle \in \mathcal{E}^c$. Our assumption also implies $D \models \varphi_{i,k} \nu$, which ensures that there exists a substitution ν_k mapping x to d such that $D \models \tau_j \nu_k$. Finally, since $i > \delta_{\ell-1}$, formula τ_i is a tree-like conjunction for x of depth at most $\ell - 1$, and so τ_j (which is equal to $\varphi_{i,k}$ up to variable renaming) is a tree-like formula for x of depth at most $\ell - 2$, which implies $j \leq \delta_{\ell-1}$. Thus, we can apply the induction hypothesis for layer $\ell - 1$, position j, constant d (i.e., vertex u), and substitution ν_k, and we obtain that $(\mathbf{u}_{\ell-1})_j = 1$. Consequently, $j \in J_{\ell,c}$, 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 $D \models \varphi_{i,0}\{x \mapsto b\}$. Furthermore, since $i > \delta_{\ell-1}$, formula τ_i is a tree-like conjunction for x of depth at most $\ell - 1$, and so for each $k \in \{1, \dots, m_i\}$, formula $\varphi_{i,k}$ is a tree-like conjunction of depth at most $\ell - 2$, which implies that there exists $j_k \in \{1, \dots, \delta_{\ell-1}\}$ such that $\varphi_{i,k}$ is equal to τ_{j_k} up to variable renaming. Furthermore, $(\mathbf{B}^c_\ell)_{i,j_k} = 1$ and so $j_k \in J'_{\ell,c}$, which implies $j_k \in J_{\ell,c}$ by our assumption. Thus, there exists a vertex u for a constant d in 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 d such that $D \models \tau_{j_k}\nu_k$. Moreover, since τ_{j_k} is equal to $\varphi_{i,k}$ up to variable renaming, there exists a substitution ν'_k mapping y_k to d such that $D \models \varphi_{i,k}\nu'_k$. Note that $\varphi_{i,k}$ has no variables in common with any other $\varphi_{i,k'}$ with $k' \neq 0$, and none of these formulas mention x, so the substitution $\nu = \{x \mapsto b\} \cup \bigcup_{k=1}^{m_i} \nu'_k$ is correctly defined. Moreover, $D \models \varphi_{i,0}\nu$, and $D \models \varphi_{i,k}\nu$ for each $k \in \{1, \dots, m_i\}$, and $D \models E^c(x, y_k)\nu$ since $\langle v, u\rangle \in \mathcal{E}^c$. Thus, $D \models \tau_i\nu$, as required. □

Using Lemma 3, we next show that GNN $\mathcal{N}_\mathcal{P}$ is indeed equivalent to program $\mathcal{P}$, and we also present a bound on the magnitude of the largest dimension δ_{L-1} of $\mathcal{N}$.

Theorem 8. *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 $\langle \mathcal{V}, \{\mathcal{E}^c\}_{c \in \mathsf{Col}}, \lambda\rangle$ be $\mathsf{enc}(D)$, and consider the application of $\mathcal{N}_\mathcal{P}$ to this graph. Program $\mathcal{P}$ contains $E^c(x, y) \rightarrow E^c(x, y)$ for each $c \in \mathsf{Col}$, and $T_{\mathcal{N}_\mathcal{P}}$ neither introduces nor removes binary atoms; thus, $T_\mathcal{P}(D)$ and $T_{\mathcal{N}_\mathcal{P}}(D)$ contain exactly the same binary atoms. To conclude the proof, we consider an arbitrary constant a in D and an arbitrary position $i \in \{1, \dots, \delta_L\}$, and we show that $U_i(a) \in T_{\mathcal{N}_\mathcal{P}}(D)$ if and only if $U_i(a) \in T_\mathcal{P}(D)$. Towards this goal, let v be the vertex corresponding to a in $\mathcal{V}$, and let J_L and J'_L be the following sets of indices.

$$J_L = \{j \in \{1, \dots, \delta_{L-1}\} \mid (\mathbf{v}_{L-1})_j = 1\} \tag{20}$$

$$J'_L = \{j \in \{1, \dots, \delta_{L-1}\} \mid (\mathbf{A}_L)_{i,j} = 1\} \tag{21}$$

By construction, for each $j \in \{1, \dots, \delta_{L-1}\}$, 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 3 ensures that $(\mathbf{v}_{L-1})_j \in \{0, 1\}$. 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, $(\mathsf{cls}(\mathbf{v}_L))_i = 1$ if $J'_L \cap J_L \neq \emptyset$, and $(\mathsf{cls}(\mathbf{v}_L))_i = 0$ otherwise.

Now assume that $U_i(a) \in T_{\mathcal{N}_\mathcal{P}}(D)$. This means that $\mathsf{cls}((\mathbf{v}_L)_i) = 1$, which implies that $J'_L \cap J_L \neq \emptyset$. Consider an arbitrary $j \in J'_L \cap J_L$. Since $j \in J'_L$, there exists a rule of the form $\varphi \rightarrow U_i(x) \in \mathcal{P}$ where φ is equal to τ_j. Furthermore, $j \in J_L$ implies $(\mathbf{v}_{L-1})_j = 1$, and so, by Lemma 3, there exists a substitution ν mapping x to a such that $D \models \varphi\nu$. Hence, $U_i(x)\nu \in T_\mathcal{P}(D)$, and so $U_i(a) \in T_\mathcal{P}(D)$, as required.

Conversely, assume that $U_i(a) \in T_\mathcal{P}(D)$. Fact $U_i(a)$ is produced by a rule $\varphi \rightarrow U_i(x) \in \mathcal{P}$ and a substitution ν mapping x to a such that $D \models \varphi\nu$. Since φ is a tree-like conjunction for x of depth at most $d = L - 2$, there exists $j \in \{1, \dots, \delta_{L-1}\}$ such that φ is equal to τ_j up to variable renaming. Lemma 3 then 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 $\mathsf{cls}((\mathbf{v}_L)_i) = 1$, so $U_i(a) \in T_{\mathcal{N}_\mathcal{P}}(D)$, as required.

Note that each δ_ℓ with $\ell \in \{1, \dots, L\}$ 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 7, 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 $f^i \cdot d \cdot \ \dots \ \cdot (d - i + 1)$ and is thus bounded by $f^i \cdot d!$. By adding up the contribution for each depth, we can have at most $f^d \cdot (d + 1)!$ variables. Each variable is labelled by one of the 2^δ conjunctions of depth zero, and each variable different from the root is connected by one of the $|\mathsf{Col}|$ predicates to its parent. Hence, we can have at most $(|\mathsf{Col}| \cdot 2^\delta)^{f^d \cdot (d+1)!}$ tree-like formulas. □

10. Evaluation

We have implemented our monotonic max-sum GNN model using Python 3.8.5 and Pytorch Geometric v1.7.1. Our implementation supports the canonical encoding/decoding scheme from Section 3.1, as well as the encoding/decoding scheme by [35] from Section 3.3.1. We have also implemented Algorithms 1 and 6 for rule extraction. We next describe the results of our empirical evaluation where we compared the classification performance of our model with that of several state-of-the-art approaches on knowledge graph completion benchmarks. Furthermore, we describe the result of applying our rule extraction algorithms to models trained for knowledge graph completion. All experiments discussed in this section have been performed on a MacBook Pro-equipped with the M2 processor and 8 GB of RAM, running macOS Ventura 13.3. Our implementation of monotonic max-sum GNNs and the code for running our experiments are available online.[1]

Benchmarks. We base our evaluation on the knowledge graph (KG) completion task—the problem of extending a dataset D to another dataset D'. When seen as a classification problem, the aim of KG completion is to learn a Boolean function $f(\cdot, \cdot)$ that takes a function-free dataset D and a fact α over a fixed signature, and that returns *true* on D and α if and only if $\alpha \in D'$. We focus on the *inductive* variant of this problem, where the test dataset may contain facts with constants not occurring in the training or validation datasets (but all predicates in the test dataset are known at training time). Inductive KG completion is more general and challenging than its *transductive* variant [36], where all constants in the test dataset are known during training.

We used the 12 inductive KG completion benchmarks by [23], which were derived from the FB15K-237 [36], NELL-995 [37], and WN18RR [38] KGs. Each of these three KGs produces four distinct benchmarks, so we refer to each benchmark by the name of the

[1] https://github.com/davala-9/mgnns

Table 1
Number of facts for each benchmark and phase (Training, Validation, and Testing).

	FB15K-237				NELL-995				WN18RR			
	v1	v2	v3	v4	v1	v2	v3	v4	v1	v2	v3	v4
Training	4245	9739	17,986	27,203	4687	8219	16,393	7546	5410	15,262	25,901	7940
Validation	489	1166	2194	3352	414	922	1851	876	630	1838	3097	934
Testing	2198	4623	8271	13,138	933	5062	8857	7804	1806	4452	6932	13,763

KG followed by a suffix between v1 and v4. Each benchmark provides disjoint datasets D_{tr}, D_{val}, and D_{tst} for training, validation, and testing, respectively. All benchmarks consist of binary facts only, and Table 1 summarises the numbers of such facts in D_{tr}, D_{val}, and D_{tst}. Each benchmark provides a method for randomly splitting its testing dataset D_{tst} into the incomplete dataset D^I_{tst} and the set D^C_{tst} of additional facts that should hold in the completion of D^I_{tst}. For each fact $\alpha \in D^C_{tst}$, we used (D^I_{tst}, α) as a positive example for testing. Additionally, we randomly constructed $|D^C_{tst}|$ distinct facts $\beta_1, \dots, \beta_n$ that use the predicates of D^I_{tst} and satisfy $\beta_i \notin D^C_{tst}$, and we used each (D^I_{tst}, β_i) as a negative example for testing. Random construction of negative examples was necessary due to the large number of possible facts that are not present in D^C_{tst}, and it also ensured that our evaluation did not favour any specific method of constructing negative examples during training. Furthermore, we maintained an equal number of positive and negative examples to ensure a balance between precision and recall.

Evaluated Systems. We next describe the five systems that we used in our evaluation. We considered monotonic max-sum GNNs with two aggregation functions: sum and max. We compared these with a standard GNN model that uses sum aggregation but without the monotonicity requirement, as well as two state-of-the-art rule-based approaches to KG completion.

- S-MGNN realises a dataset transformation using a monotonic max-sum GNN with sum aggregation. Each GNN uses two layers; the dimension of the middle layer is twice the size of the input layer; and the activation function is ReLU. All benchmarks consist exclusively of binary facts, so the ability to process and derive such facts is essential. Thus, in all GNN-based systems, we used the variant of the encoding/decoding scheme by [35] described in Section 3.3.1 that does not introduce a vertex for each pair of constants in the dataset.
- M-MGNN is analogous to S-MGNN, but it uses max aggregation (instead of sum). Including this system in our evaluation allows us to check whether the increased expressive power obtained by allowing sum aggregation in monotonic GNNs is relevant in practice.
- P-GNN is the same as S-MGNN, but without any restrictions on the sign of the weights of the matrices $\mathbf{A}_\ell$ and $\mathbf{B}^c_\ell$; the letter 'P' in the name stands for 'plain'. This makes P-GNN a variant of a Relational Graph Convolutional Network [15], a standard GNN model that is widely used in practice. We use this model as a baseline to investigate whether and how the monotonicity requirement impacts the performance of monotonic max-sum GNNs on KG completion tasks.
- DRUM [13] learns an end-to-end differentiable model that, given a dataset D and a fact α, returns a score representing the likelihood of α belonging to the completion of D; predictions are then computed by applying a threshold to the score. Additionally, DRUM can produce *chain rules* of the form $B_1(x_0, x_1) \wedge B_2(x_1, x_2) \wedge \cdots \wedge B_n(x_{n-1}, x_n) \rightarrow H(x_0, x_n)$, each associated with a confidence score. These rules aim to explain the model's predictions symbolically; however, [39] showed that the produced rules are neither sound nor complete with respect to the model. Hence, the rules produced by DRUM provide only approximate explanations of the model's predictions.
- AnyBURL [40] uses path sampling to produce chain rules of a slightly more general form, where variables can be replaced with constants. As in DRUM, each produced rule is associated with a confidence score. To determine whether fact α belongs to the completion of D, AnyBURL applies all rules to D once, combines the confidence score of all rules that derive α, and applies a threshold to the result.

Training. For each KG completion benchmark, we trained an instance of each of the five systems described above. The training dataset D_{tr} was first split randomly with a $9:1$ ratio into an incomplete dataset D^I_{tr} and a set D^C_{tr} of facts that should hold in the completion of D^I_{tr}. To train our model, we used (D^I_{tr}, α) for each fact $\alpha \in D^C_{tr}$ as a positive example, and we obtained negative examples (D^I_{tr}, β) by replacing the predicates of positive examples α to obtain new facts β not in D^C_{tr}. We used cross-entropy loss, and we trained our models using the Adam optimisation algorithm parameterised by the standard learning rate (0.01) and weight decay (5×10^{-4}), and a maximum of $50,000$ epochs for training. We trained DRUM and AnyBURL using their (publicly available) code with their default configurations. To choose the classification thresholds for the testing phase, we classified the negative and positive examples in the validation dataset using a range of thresholds ranging from 0.01 to 0.99, and we selected the threshold that maximised accuracy.

Evaluation Results. For each system, we classified all positive and negative testing examples and calculated the precision, recall, accuracy, and F1 score using the standard methods. Additionally, we computed the area under the precision–recall curve (AUC) by considering various classification thresholds. Table 2 shows the results of our evaluation, as well as the training times for all systems apart from AnyBURL (which does not train a model). As one can see, the performance of monotonic max-sum GNNs is competitive with that of the baselines for most benchmarks. For example, S-MGNN and M-MGNN provide the best accuracy and F1 score on nine of our 12 benchmarks. Interestingly, on 11 out of 12 benchmarks, S-MGGN and M-MGNN both achieved a better F1 score than

Table 2
Results for S-MGNN (abbreviated as S), M-MGNN (M), P-GNN (P), DRUM (D), and AnyBURL (A).

		Precision (%)					Recall (%)					Accuracy (%)					F1 Score					AUC					Training (seconds)			
		S	M	P	D	A	S	M	P	D	A	S	M	P	D	A	S	M	P	D	A	S	M	P	D	A	S	M	P	D
FB15K-237	v1	91.5	95.1	98.8	82.2	**100.0**	54.0	**58.0**	41.0	40.5	25.6	74.5	**77.5**	70.3	65.9	62.9	67.9	**72.0**	58.0	54.2	41.1	72.8	**75.4**	65.5	68.6	62.0	5693	9192	1006	15,540
	v2	98.4	93.6	**100.0**	85.3	**100.0**	57.2	**66.5**	48.7	48.5	47.9	78.1	**81.0**	74.3	70.0	74.0	72.4	**77.8**	65.5	61.9	64.5	76.7	**79.8**	75.5	76.0	73.8	19,174	23,523	1658	19,080
	v3	90.1	98.2	99.4	85.6	**100.0**	**73.8**	58.2	71.4	45.2	43.6	83.2	78.6	**85.6**	68.8	71.2	81.4	73.1	**83.1**	59.2	60.7	82.8	78.0	**87.5**	73.3	71.7	32,348	45,104	316	37,776
	v4	96.7	94.7	98.6	88.8	**100.0**	**71.2**	70.3	48.3	41.6	46.0	**84.4**	83.2	73.8	68.2	73.0	**82.0**	80.7	64.8	56.7	63.0	**84.1**	81.0	75.7	73.7	73.0	72,487	42,540	10,287	87,000
NELL-995	v1	85.0	87.6	**100.0**	94.7	97.5	**100.0**	91.2	17.6	18.0	77.0	**91.2**	89.4	58.9	58.5	87.5	**91.9**	89.7	30.0	30.3	86.0	**93.0**	91.6	59.9	52.6	85.7	489	662	79	402
	v2	91.7	**100.0**	99.0	80.7	**100.0**	52.5	42.1	41.4	41.2	**53.1**	73.9	71.0	70.5	65.7	**76.6**	66.8	59.2	58.4	54.5	**69.4**	72.0	65.2	69.4	74.6	**76.3**	864	3575	183	6354
	v3	46.8	47.8	80.0	85.8	**100.0**	**87.5**	87.4	42.1	45.6	47.3	43.9	46.0	65.8	69.0	**73.7**	60.9	61.8	55.1	59.6	**64.2**	84.7	**85.6**	64.3	75.3	69.3	2070	13,456	1814	36,090
	v4	93.4	89.7	97.3	80.3	**100.0**	68.0	**68.6**	46.3	26.8	44.3	**81.6**	80.4	72.5	60.1	72.2	**78.7**	77.7	62.8	40.2	61.4	82.5	81.3	**72.8**	67.1	71.6	1224	16,839	358	3990
WN18RR	v1	98.1	93.3	97.8	97.9	**99.1**	**92.7**	92.1	82.4	73.4	58.5	**95.5**	92.7	90.3	85.9	79.0	**95.3**	92.7	89.5	83.4	73.6	90.8	**93.9**	90.7	92.5	78.8	246	292	132	312
	v2	99.1	99.4	**100.0**	96.3	**100.0**	**84.4**	**84.4**	81.7	69.8	75.7	**91.9**	92.0	90.9	83.6	87.9	91.2	**91.3**	89.9	80.9	86.2	87.2	88.2	**90.0**	87.4	59.8	270	856	575	2856
	v3	99.0	**99.7**	98.4	91.3	**99.7**	59.7	**59.9**	58.3	59.7	48.4	79.6	**79.9**	78.7	77.0	74.1	74.5	**74.9**	73.2	72.2	65.2	67.8	69.3	73.2	**85.2**	60.6	603	2423	594	9984
	v4	98.3	97.9	98.6	98.3	**99.9**	**84.8**	84.7	81.9	65.6	71.6	**91.7**	91.5	90.3	82.2	85.8	**91.1**	90.8	89.4	78.7	83.4	87.3	88.3	88.4	**93.9**	59.0	195	409	171	1638

Table 3
Median number of body atoms for the 10 rules extracted to justify 10 facts chosen arbitrarily from those derived by the monotonic max GNN model on each benchmark, and the total time required to extract all 10 rules.

	FB15K-237				NELL-995				WN18RR			
	v1	v2	v3	v4	v1	v2	v3	v4	v1	v2	v3	v4
# Body atoms per rule (median)	7	4	6	12	1	4	4	4	1	1	1	1
Total extraction time (seconds)	21	20	20	21	11	12	18	16	10	11	11	10

Table 4
Extraction of time (in minutes) and total number of extracted rules produced by applying our implementation of Algorithm 6 to the monotonic max GNN models learned for each benchmark, where *TO* represents a timeout—that is, the fact that the full program had not been extracted after 12 hours.

	FB15K-237				NELL-995				WN18RR			
	v1	v2	v3	v4	v1	v2	v3	v4	v1	v2	v3	v4
Extraction time	*TO*	*TO*	*TO*	*TO*	*TO*	*TO*	1	*TO*	54	13	*TO*	55
Number of rules	−−	−−	−−	−−	−−	−−	142	−−	2340	11,843	−−	5693

P-GNN, although the difference is typically small. This suggests that the monotonicity requirement does not negatively impact the performance of KG completion, and in fact it can be a useful inductive bias. Finally, monotonic GNNs typically perform better with sum than max aggregation: on seven out of 12 benchmarks, S-MGGN exhibited higher accuracy than M-MGNN. This suggests that the additional expressive power provided by sum aggregation can be useful in practice.

Rule Extraction Experiments. For each benchmark, we sampled ten facts from the set of facts derived by M-MGNN on the benchmark, and for each fact we produced a rule $\Gamma \to U_i(x)$ that explains the derivation using Algorithm 1. We then used Algorithm 2 to optimise this rule using the two relevance criteria described in Section 6.3, and we retained the shorter rule.

Table 3 shows the median of the lengths of the ten rules extracted to justify the ten facts from each benchmark, and the total time required to extract all the rules from each benchmark. Rules with a single body atom were common and corresponded to symmetry rules, such as $collaboratesWith(x, y) \to collaboratesWith(y, x)$ and $seeAlso(x, y) \to seeAlso(y, x)$ produced on NELL-995.v1 and WN18RR.v2, respectively. The low median values for rule length offer evidence that our algorithm is capable of extracting simple and understandable rules in practice. The short extraction times also suggest that these rules can be provided in a fast and convenient way.

Our algorithms learned rules that express relationship subsumption such as $worksFor(x, y) \to hiredBy(x, y)$ on multiple NELL-995 datasets, and it learned (half of) inverse relations such as $wonAward(x, y) \to awardWonBy(y, x)$ on multiple FB15K-237 datasets. The longer rules from Table 3 represent conditional subsumption, where a relation R between x and y implies a relation S between x and y if these variables participate in additional relationships. For example, on the FB15K-237 v3 dataset, our algorithm learned the rule

$$wonAward(x, y) \wedge awardNominee(y, z_1) \wedge awardNominee(z_2, x) \wedge producedFilm(x, z_3) \to awardNominee(y, x),$$

which states that, if person x won an award y, then x was a nominee for y, provided that the award y had some nominee z_1, x was an award nominee for some award z_2, and x produced a film z_3. Only some of these additional conditions seem appropriate. For example, atom $awardNominee(y, z_1)$ is relevant to deduce that x was nominated for y because it requires that y is an award with nominations. In contrast, the remaining two atoms make the rule restrictive, but their presence is required for soundness: removing them produces unsound rules, which suggests that the model overfits the training data. As another example, on the NELL-995 v1 dataset, our algorithm learned the rule

$$organisationHired(x, y) \wedge topMemberOf(y, x) \to organisationFired(x, y),$$

which states that top members of organisations are eventually fired. Note that in all the rules discussed in this section, there is a body atom that mentions both x and y because our encoding allows us to predict only facts that mention pairs of constants that occur together in a fact of the input dataset.

We have also applied our implementation of Algorithm 6 for extracting a Datalog program equivalent to a monotonic max GNN and applied it to the trained M-MGNN models for each benchmark. Table 4 shows the program computation times (we chose a timeout of 12 hours) and the number of rules in each of the extracted programs. As shown in the table, the extraction of a faithful program was feasible for the WN18RR benchmarks: these contain a small number of predicates, so the space of the candidate rules is sufficiently small. For the remaining KGs, rule extraction succeeded within the time limit only for the NELL-995.v3 benchmark. This is because the classification threshold selected for this benchmark was very small so the soundness checks succeeded early. This suggests that developing ways to further reduce the space of the candidate rules is an interesting and important avenue for further research.

11. Related work

Rule Extraction From Neural Networks. Methods for extracting rules from neural models that operate on relational data generally proceed by first training a neural model on the data and subsequently extracting rules from the trained model. Early systems of this kind focused on extracting propositional (i.e., variable-free) rules from feed-forward neural networks [41]. Neural-LP [12] is a prominent example of a modern system of this kind, and it is a precursor of DRUM [13]. In Neural-LP, DRUM, and several recent approaches [42,43], the weights of the model are learned using Recurrent Neural Networks, and the extracted rules are chain rules as outlined in Section 10. Neural-LP has also been extended to learn rules with counting [44]. Other approaches in this category extract (typically chain) rules from models based on reinforcement learning [37,45], dynamic neural module networks [10,11], or graph embeddings [46,47]. Beyond chain rules, the approach by [48] extracts rules expressed as axioms in *description logics*—a popular KR formalism—from feed-forward neural networks for image classification. A limitation of these approaches is that the formal relationship between a model and the rules extracted from it is usually left unspecified. Thus, as shown by [49] and [39], the result of applying the rules can, both in principle and in practice, differ from the model's predictions in both directions—that is, the rules can be both unsound and incomplete. Moreover, [39] characterised precisely the expressive power of the DRUM model.

Similarly to our work, techniques for neural learning of monotonic functions on the ordering of real numbers [50–52] also rely on nonnegative matrices and monotonic activation functions. We, however, focus on learning monotonic functions on the ordering induced by the existence of homomorphisms between datasets.

Rule Mining. A second category of systems, often called *rule-mining*, identify frequent path patterns in the training data and represent these patterns as rules. They use heuristics to estimate the quality of the extracted rules, but they do not train a neural model and they provide no formal guarantees about their output. The AMIE+ [53] and AnyBURL systems [40] are prominent examples of this approach, and many similar approaches have been proposed recently [54,55].

Inductive Logic Programming. The techniques listed in the previous paragraphs can be used to automatically learn logical rules from training data. *Inductive Logic Programming (ILP)* is an alternative approach to rule learning, where rules are generated from a dataset and a set of examples of positive and negative inferences so that the application of the generated rules produces all positive and no negative examples [56,57]. Standard ILP techniques satisfy all examples *exactly* and, as a result, are intolerant to noise: small inconsistencies in the input examples can significantly affect the quality of the produced rules [58]. This limitation also applies to some newer rule learning approaches [59–61]. Other recent ILP systems, such as ∂ILP [58], achieve robustness to noise by interpreting the ILP task as a binary classification problem and providing a differentiable implementation of deduction. Traditionally, these techniques focus on learning very expressive rules from small datasets, but their application to large datasets such as those involved in KG completion has not been thoroughly studied.

Explanation and Formal Verification of GNNs. Existing methods for explaining the predictions of a GNN focus on identifying parts of the input graph that are most relevant to a given prediction [62–64]. These techniques are largely independent from the specifics of the model used. In contrast, we focus on a specific type of GNN and provide a logical justification for each prediction. Furthermore, our explanations do not only identify the part of the input dataset that is relevant for a particular prediction, but also produce a sound rule responsible for the prediction. Sälzer and Lange [65] show that it is impossible to verify, in general, whether a rule with an empty body is sound for a plain (unrestricted) GNN with sum aggregation and ReLU activation; however, their conclusions do not apply to our setting since their class of GNNs allows for negative weights and is thus more general than monotonic max-sum GNNs.

The Expressive Power of GNNs. Characterising the expressivity of ML models for data management has steadily gained importance, and computational logic has been widely used to this end. In a pioneering study, [26] showed that a transformation is induced by a GNN and is expressible in first-order logic if and only if it is expressible by a concept query of the $\mathcal{ALCQ}$ description logic [27]. [28] proved an analogous result for a class of GNNs with a dedicated vertex and colour, and [66] did so for a class of recurrent GNNs. Grohe [29] has shown that a graph query can be realised by a GNN in a very large class if and only if it can be expressed in the guarded fragment of first-order logic with two variables and counting terms for the number of assignments satisfying a formula. Furthermore, Benedikt et al. [33] have shown that transformations realised by GNNs with bounded piece-wise linear activation functions correspond to formulas expressed in *Presburguer logic*, a language that is decidable but not comparable with first-order logic; [67] independently reached closely related conclusions. Our Theorem 5 provides a tighter upper bound on the expressive power of monotonic max-sum GNNs. [30] showed that GNNs can express certain types of graph isomorphism tests, and [68] extended this analysis to a more general class of GNNs with very expressive aggregation functions that can aggregate vertices not in the immediate neighbourhood. Rosenbluth et al. [69] showed that there exist functions expressed by GNNs with max aggregation that cannot be expressed by GNNs with sum aggregation. Finally, Sourek et al. [32] characterised the expressivity of GNNs using a hybrid language where each Datalog rule is annotated with a tensor.

Encoding Logic Programs Into Neural Networks. Early approaches to neuro-symbolic reasoning focused on identifying classes of rules expressed in first-order logic for which the immediate consequence operator of certain classes of logic programs can be simulated by recurrent [6], fibring [7], or feed-forward networks [8]. However, their embedding methods are very different from ours: they transform a dataset into a vector of real numbers by encoding groups of facts as vector's components, and they can extract only propositional rules from trained networks. Certain neural-symbolic architectures can simulate the intuition of forward [9,10] and

backwards chaining [11] of Horn clauses. Sadeghian et al. [13] show how programs of chain-like Datalog rules can be encoded into DRUM models. Recently, Dong et al. [9] presented a network architecture that simulates application of arbitrary function-free first-order rules to a dataset; to this end, they use multi-layer perceptrons to learn Boolean operations between ground atoms and combinatorial tensor calculus to simulate quantification. All of these approaches seem fundamentally different to ours in that they transform an arbitrary set of rules into an equivalent model, but do not provide a transformation in the other direction—that is, they cannot transform an arbitrary, trained model into an equivalent set of rules.

12. Conclusion and future work

We introduced a class of GNN-based transformations of datasets that mimic a round of application of Datalog rules with inequalities. The predictions made by our transformations can be explained symbolically, and we have shown that they can be successfully trained in practice. We see many avenues for future work. On the theoretical side, we will develop a full characterisation of monotonic max-sum GNNs. We shall also develop extensions that can capture more complex rules (e.g., transitivity), deal with nonmonotonic extensions (e.g., negation-as-failure and aggregation), and include advanced deep learning features (e.g., attention mechanisms). On the practical side, extracting an equivalent program leaves considerable room for improvement: as we discussed in Section 10, doing so is not always feasible, and, even when it is, the resulting program is often difficult to read because it contains many rules with a large number of body atoms. We believe these drawbacks can be addressed as follows. First, the algorithm for extracting the equivalent program can be optimised by developing heuristics that reduce the space of candidate rules, as well as by parallelising the implementation. Second, instead of extracting rules that are strictly equivalent to the model, one can consider producing rules that *approximate* the model under some suitable notion of approximation. Third, derivations can be explained and model's properties verified without necessarily extracting a complete equivalent program; our results in Sections 5 and 6 can be seen as preliminary steps in this direction.

CRediT authorship contribution statement

David Tena Cucala: Writing – review & editing, Writing – original draft, Software, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Bernardo Cuenca Grau:** Writing – review & editing, Supervision, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization; **Boris Motik:** Writing – review & editing, Supervision, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization; **Egor V. Kostylev:** Writing – review & editing, Validation, Methodology, Formal analysis, Conceptualization.

Data availability

An open link to all relevant data and code has been provided.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was funded by the EPSRC grants AnaLOG (EP/P025943/1) and ConCur (EP/V050869/1). 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.

Appendix A. Proofs of Theorems 2 and 4

Our proofs of Theorems 2 and 4 from Section 7 both depend on Lemma 4, whose formulation and proof are technically involved. In this appendix, we present and prove the lemma, and then we use it to prove both theorems. In the rest of this appendix, let $\mathcal{N}$, $\delta_0, \dots, \delta_L$, and $k_1, \dots, k_L$ be as fixed in Section 7.

To formulate the lemma, we introduce a family of nonempty sequences $\mathcal{S}_{\ell,i}$ of real numbers, where $\ell \in \{0, \dots, L\}$ and $i \in \{i, \dots, \delta_\ell\}$. Intuitively, sequence $\mathcal{S}_{\ell,i}$ enumerates the set $\mathcal{X}_{\ell,i}$ in ascending order. Our definition of these sequences is inductive over the layers of the network $\ell \in \{0, \dots, L\}$. The inductive step for $\ell \in \{1, \dots, L\}$ uses several auxiliary notions that we introduce in Definitions 10–12; there, we assume that $\mathcal{S}_{\ell-1,j}$ has been defined for each $j \in \{1, \dots, \delta_{\ell-1}\}$, and moreover we let $\mathbf{s}_{\ell-1}$ be the vector of dimension $\delta_{\ell-1}$ such that $(\mathbf{s}_{\ell-1})_j$ is the first element of $\mathcal{S}_{\ell-1,j}$ for each $j \in \{1, \dots, \delta_{\ell-1}\}$. Definition 13 later uses all of these auxiliary notions to define the sequences $\mathcal{S}_{\ell,i}$.

Definition 10. For each $\ell \in \{1, \dots, L\}$ and each $i \in \{1, \dots \delta_\ell\}$, an (ℓ, i)-*triple* is a triple of the form $\langle \mathbf{x}, \mathbf{Y}, z \rangle$ whose elements satisfy the following conditions:

- $\mathbf{x}$ is a vector of dimension $\delta_{\ell-1}$ such that for each $j \in \{1, \dots, \delta_{\ell-1}\}$, it is the case that $(\mathbf{x})_j \in \mathcal{S}_{\ell-1,j}$;

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

Definition 11. 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 holds:

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

In the last item of Definition 11, if the first element of $\mathcal{S}_{\ell-1,j}$ is positive, then $\mathbf{s}_{\ell-1} = \mathbf{s}_{\ell-1}[j \leftarrow y']$. This substitution of the first element of $\mathcal{S}_{\ell-1,j}$ by its first positive element simply ensures that $\mathbf{Y}_2$ is never a copy of $\mathbf{Y}_1$ where one of the multisets has been extended with a vector where all components are zeros.

Definition 12. For each $\ell \in \{1, \dots, L\}$ and each $i \in \{1, \dots, \delta_\ell\}$, the sequence $\mathcal{F}_{\ell,i} = f_0, \dots, f_n, \dots$ of finite sets of (ℓ, i)-triples is defined inductively over n as follows.

- For the base case $n = 0$, 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 $n > 0$, if f_{n-1} has been defined and is not empty, then

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

  where $\min(f_n) = \min\{z \mid \text{there exist } \mathbf{x} \text{ and } \mathbf{Y} \text{ such that } \langle \mathbf{x}, \mathbf{Y}, z \rangle \in f_n\}$.

Note that $\min(f_{n-1})$ above is correctly defined since f_{n-1} is nonempty by assumption and always finite. Moreover, f_n may be defined but empty; in that case, f_m is undefined for each $m > n$, so the sequence $\mathcal{F}_{\ell,i}$ is finite. We are now ready to define sequences $\mathcal{S}_{\ell,i}$.

Definition 13. For each $\ell \in \{0, \dots, L\}$ and each $i \in \{1, \dots, \delta_\ell\}$, the sequence $\mathcal{S}_{\ell,i}$ of real numbers is defined inductively as follows.

- For the base case $\ell = 0$, let $\mathcal{S}_{0,i} = (0, 1)$ for each $i \in \{1, \dots, \delta_0\}$.
- For the inductive step, assume that $\mathcal{S}_{\ell-1,j}$ has been defined for each $j \in \{1, \dots, \delta_{\ell-1}\}$, and consider any $i \in \{1, \dots, \delta_\ell\}$. Then, $\mathcal{S}_{\ell,i}$ is the sequence of real numbers whose nth element is $\sigma(\min(f_n))$ if f_n is defined and nonempty.

Since f_0 is always defined and not empty, each $\mathcal{S}_{\ell,i}$ is not empty as well. We are finally ready to state and prove Lemma 4, which shows that sequences $\mathcal{S}_{\ell,i}$ enumerate the elements of $\mathcal{X}_{\ell,i}$ in ascending order.

Lemma 4. *For each $\ell \in \{0, \dots, L\}$ and each $i \in \{1, \dots, \delta_\ell\}$, sequence $\mathcal{S}_{\ell,i}$ satisfies the following conditions:*

1. *each element of $\mathcal{S}_{\ell,i}$ is nonnegative;*
2. *$\mathcal{S}_{\ell,i}$ is strictly monotonically increasing;*
3. *$\mathcal{S}_{\ell,i}$ is either finite or it converges to ∞; and*
4. *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 $(0, 1)$ for each $i \in \{1, \dots, \delta_\ell\}$ by definition, so conditions 1–4 hold trivially. Now consider arbitrary $\ell \in \{1, \dots, L\}$ such that each $\mathcal{S}_{\ell-1,j}$ with $j \in \{1, \dots, \delta_{\ell-1}\}$ satisfies conditions 1–4, and consider arbitrary $i \in \{1, \dots, \delta_\ell\}$.

Condition 1 follows straightforwardly from the fact that each element of $\mathcal{S}_{\ell,i}$ for $\ell \geq 1$ is equal to $\sigma(z)$ for some $z \in \mathbb{R}$, and the range of σ is a subset of $\mathbb{R}_0^+$. Condition 2 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 $\min(f_n) > \min(f_{n-1})$.

We prove condition 3 by contradiction—that is, we assume that $\mathcal{S}_{\ell,i}$ is infinite, but 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 $j \in \{1, \dots, \delta_{\ell-1}\}$ and each $c \in \mathsf{Col}$, let w_j, α_j, and $n_{j,c}$ be as follows:

$$w_j = \min\{(\mathbf{A}_\ell)_{i,j}\} \cup \{(\mathbf{B}_\ell^c)_{i,j} \mid c \in \mathsf{Col}\}; \tag{A.1}$$

$$\alpha_j = \begin{cases} \frac{\bar{\beta} - (\mathbf{b}_\ell)_i}{w_j} & \text{if } w_j \neq 0, \\ \text{undefined} & \text{otherwise;} \end{cases} \tag{A.2}$$

$$\epsilon_j = \begin{cases} \text{the first positive value of } \mathcal{S}_{\ell-1,j} & \text{if such a value exists,} \\ \text{undefined} & \text{otherwise;} \end{cases} \tag{A.3}$$

$$n_{j,c} = \begin{cases} \lceil \frac{\bar{\beta} - (\mathbf{b}_\ell)_i}{(\mathbf{B}^c_\ell)_{i,j} \cdot \epsilon_j} \rceil & \text{if } (\mathbf{B}^c_\ell)_{i,j} \neq 0 \text{ and } \epsilon_j \text{ is defined;} \\ 0 & \text{otherwise.} \end{cases} \tag{A.4}$$

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 jth element is not zero.

To prove the first property, note that, if $(\mathbf{x})_j \geq \alpha_j$, then, since condition 1 of the inductive hypothesis ensures that all elements of $\mathbf{x}$ and vectors in $\mathbf{Y}$ are nonnegative, and 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 prove 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 2 and 3 of the inductive hypothesis, each $\mathcal{S}_{\ell-1,j}$ is countable, monotonically increasing, and either finite or converges to infinity; hence, the 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 take only 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 take only finitely many values. Note also that each $\mathbf{Y}^c$ cannot have infinitely many elements because it does not contain any vector where all elements are 0, and, for each $j \in \{1, \dots, \delta_{\ell-1}\}$, there exist fewer than $n_{j,c}$ elements in $\mathbf{Y}^c$ whose jth component's value is positive, and no elements for which the jth component 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 4. 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 each $j \in \{1, \dots, \delta_{\ell-1}\}$, each $c \in \mathsf{Col}$, and each $\mathbf{y} \in \mathbf{Y}^c$, 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 an arbitrary $\alpha \in \mathcal{X}_{\ell,i}$. By Definition 8, 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 $j \in \{1, \dots, \delta_{\ell-1}\}$, and there also exists a multiset family $\mathbf{Y}_\alpha$ of dimension $\delta_{\ell-1}$ such that $(\mathbf{y})_j \in \mathcal{X}_{\ell-1,j}$ for each $c \in \mathsf{Col}$, each $\mathbf{y} \in \mathbf{Y}^c_\alpha$, and each $j \in \{1, \dots, \delta_{\ell-1}\}$; 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

$$\langle \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset, \mathsf{Val}(\ell, i, \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset) \rangle = t_0, \dots, t_K = \langle \mathbf{x}_\alpha, \mathbf{Y}_\alpha, z_\alpha \rangle$$

such that t_n is a successor of t_{n-1} for each $n \in \{1, \dots, K\}$. Indeed, each multiset $\mathbf{Y}^c_\alpha$ is finite and, starting from t_0, we can reach t_K if, in each step, we change some vector component to the next element in $\mathcal{S}_{\ell-1,j}$ or we add a new vector to some multiset of the multiset family. We now show by induction on n the following statement ($*$): for each $t_n = \langle \mathbf{x}_n, \mathbf{Y}_n, z_n \rangle$ in this sequence, some element of the sequence $\mathcal{F}_{\ell,i}$ 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 $j \in \{1, \dots, \delta_{\ell-1}\}$, 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.

These properties clearly imply $z = z_n$. For the base case, $t_0 \in f_0$ 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 $n \in \{1, \dots, K\}$ has a witness in $\mathcal{F}_{\ell,i}$, and we show that $t_n = \langle \mathbf{x}_n, \mathbf{Y}_n, z_n \rangle$ 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, then, by definition, the last element of $\mathcal{F}_{\ell,i}$ is empty, but we know that some element in the sequence $\mathcal{F}_{\ell,i}$ contains t, 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 $f_{m'}$ form an infinite sequence that is strictly monotonic and whose values are always smaller than $\sigma(z)$; however, this contradicts condition 3. 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 jth component of $\mathbf{x}_{n-1}$ for j such that $(\mathbf{A}_\ell)_{i,j} = 0$, or in the jth component of some $\mathbf{y} \in \mathbf{Y}^c_{n-1}$ for some $c \in \mathsf{Col}$ with $(\mathbf{B}^c_\ell)_{i,j} = 0$; hence, 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 jth component of $\mathbf{x}_{n-1}$ for j such that $(\mathbf{A}_\ell)_{i,j} > 0$, or in the jth 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 jth 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. Thus, t' is 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 witness appears in f_{m-1} but not in f_m. But then, the definition of $\mathcal{S}_{\ell,i}$ ensures that $\sigma(z_\alpha)$ is the $(m-1)$th element of $\mathcal{S}_{\ell,i}$; since $\sigma(z_\alpha) = \alpha$, number α appears in $\mathcal{S}_{\ell,i}$, as desired. □

We now prove Theorem 2, which we restate for convenience.

Theorem 2. *Each set $\mathcal{X}_{\ell,i}$ satisfies $\mathcal{X}_{\ell,i} \subseteq \mathbb{R}^+_0$, and, for each $\alpha \in \mathbb{R}$, the set $\{\alpha' \in \mathcal{X}_{\ell,i} \mid \alpha' \leq \alpha\}$ is finite.*

Proof. By condition 4 of Lemma 4, for each $\ell \in \{0, \dots, L\}$ and each $i \in \{1, \dots, \delta_\ell\}$, the elements of $\mathcal{X}_{\ell,i}$ are precisely the elements of the sequence $\mathcal{S}_{\ell,i}$. By condition 1, all elements in $\mathcal{S}_{\ell,i}$ are nonnegative, so $\mathcal{X}_{\ell,i} \subseteq \mathbb{R}^+_0$. Moreover, assume that the set $\mathcal{X}_{\ell,i} \setminus \mathcal{X}^{>\alpha}_{\ell,i}$ is infinite; then, by condition 4 of Lemma 4, the set $\mathcal{S}_{\ell,i}$ contains infinitely many numbers that are smaller or equal than α. However, condition 2 ensures that $\mathcal{S}_{\ell,i}$ is strictly monotonically increasing, so α is an upper bound of the sequence. This, in turn, contradicts condition 3. Consequently, the set $\mathcal{X}_{\ell,i} \setminus \mathcal{X}^{>\alpha}_{\ell,i}$ is finite. □

Finally, we prove Theorem 4, which we also restate for convenience.

Theorem 4. *Algorithm 4 terminates on all inputs. Moreover, for each $\ell \in \{0, \dots, L\}$ and each $i \in \{1, \dots, \delta_\ell\}$,*

- $\mathsf{Next}(\ell, i, \triangleright)$ *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. By condition 4 of Lemma 4, for each $\ell \in \{0, \dots, L\}$ and each $i \in \{1, \dots, \delta_\ell\}$, the set $\mathcal{X}_{\ell,i}$ contains exactly all the elements of $\mathcal{S}_{\ell,i}$. Furthermore, since $\mathcal{S}_{\ell,i}$ is strictly monotonically increasing by condition 2 of Lemma 4, 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 an arbitrary $\alpha \in \mathbb{R}$, let $\mathcal{S}^{>\alpha}_{\ell,i}$ be the subsequence of $\mathcal{S}_{\ell,i}$ which contains all elements in $\mathcal{S}^{>\alpha}_{\ell,i}$ greater than α. Clearly, $\mathcal{X}^{>\alpha}_{\ell,i}$ is identical to the set of elements in $\mathcal{S}^{>\alpha}_{\ell,i}$. Moreover, since $\mathcal{S}_{\ell,i}$ is strictly monotonically increasing, then either $\mathcal{S}^{>\alpha}_{\ell,i}$ is empty or it contains an element $s^{>\alpha}_{\ell,i}$ that appears in $\mathcal{S}_{\ell,i}$ exactly once and satisfies the following conditions:

1. all elements that precede $s^{>\alpha}_{\ell,i}$ in $\mathcal{S}_{\ell,i}$ are smaller or equal to α; and
2. all elements that follow $s^{>\alpha}_{\ell,i}$ in $\mathcal{S}_{\ell,i}$ are strictly greater than $s^{>\alpha}_{\ell,i}$.

In particular, condition 2 ensures that, if $\mathcal{S}^{>\alpha}_{\ell,i}$ is not empty, then $s^{>\alpha}_{\ell,i}$ is the smallest element of $\mathcal{S}^{>\alpha}_{\ell,i}$. Hence, to show the claims 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 $\mathcal{S}^{>\alpha}_{\ell,i}$ is empty, and otherwise it returns $s^{>\alpha}_{\ell,i}$.

We show both claims simultaneously by induction on ℓ.

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

For the induction step, consider an arbitrary $\ell \in \{1, \dots, L\}$, and assume that both claims above hold for $\ell - 1$. Consider an arbitrary $i \in \{1, \dots, \delta_\ell\}$. To show the first claim, note that the definition of $\mathcal{S}_{\ell,i}$ ensures that the first element of $\mathcal{S}_{\ell,i}$ is $\sigma(z)$ for $z = \mathsf{Val}(\ell, i, \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset)$. Moreover, $\mathbf{s}_{\ell-1}$ is defined as the vector of dimension $\delta_{\ell-1}$ such that for each $j \in \{1, \dots, \delta_{\ell-1}\}$, the value of $(\mathbf{s}_{\ell-1})_j$ is the first element of $\mathcal{S}_{\ell-1,j}$. However, by the 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 4 ensure that $\mathsf{Next}(\ell, i, \triangleright)$ is precisely $\sigma(z)$. To show the second claim, we study the execution of $\mathsf{Next}(\ell, i, \alpha)$ for an arbitrary $\alpha \in \mathbb{R}$. Since $\alpha \neq \triangleright$, the set 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 $\mathcal{S}_{\ell,i} = q_0, q_1, \dots$. 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 it is strictly monotonically increasing. We next show the following claim $(*)$: for each $n \in \{0, \dots, N\}$, 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 $\langle \mathbf{s}_{\ell-1}, \mathbf{Y}_\emptyset, z'\rangle$ 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 $n \in \{0, \dots, N-1\}$ and assume that $F = f_n$ holds 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 $\sigma(z) = q_n$ is selected; since $n < N$ and so $q_n \leq \alpha$, the condition in line 11 is not satisfied, so the algorithm does not exit the loop and eventually starts 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 $\sigma(z) = q_n$. Then, after reaching the state where $F = f_n$, the algorithm's loop runs (at least) K more 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 is exactly f_{n+1}. This concludes the proof of $(*)$.

Assume now that $S^{>\alpha}_{\ell,i}$ is empty, which means that all elements of $S_{\ell,i}$ are smaller than α. Then, N is precisely the number of elements of $S_{\ell,i}$—that is, $N > 0$ and q_{N-1} is the last defined element of $S_{\ell,i}$. By $(*)$, F becomes equal to f_N after a finite number of steps during the execution of $\mathsf{Next}(\ell, i, \alpha)$. 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 $\lhd$, as expected. If $S^{>\alpha}_{\ell,i}$ is not empty, then there exists an element $s^{>\alpha}_{\ell,i}$ satisfying conditions 1, and 2. In particular, the definition of N, condition 1, and the fact that $s^{>\alpha}_{\ell,i} > \alpha$ together ensure that $s^{>\alpha}_{\ell,i}$ is precisely the $(N+1)$th element q_N of $S_{\ell,i}$. Claim $(*)$ ensures that, in the execution of $\mathsf{Next}(\ell, i, \alpha)$, the set F becomes equal to f_N after a finite number of steps. Since $s^{>\alpha}_{\ell,i}$ is the $(N+1)$th element of $S_{\ell,i}$, there exists a triple $\langle x, \mathbf{Y}, z \rangle \in f_N$ with $\sigma(z) = s^{>\alpha}_{\ell,i}$ and every other triple $\langle x', \mathbf{Y}', z' \rangle \in F$ satisfies $z' \geq z$. 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 $\langle x, \mathbf{Y}, z \rangle$). Then, 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 claim. □

References

[1] Y. Lin, Z. Liu, M. Sun, Y. Liu, X. Zhu, Learning entity and relation embeddings for knowledge graph completion, in: Proc. of the 29tnh AAAI Conf. on Artificial Intelligence (AAAI 2015), Austin, TX, USA, 2015, p. 2181–2187.

[2] J. Portisch, H. Paulheim, The DLCC node classification benchmark for analyzing knowledge graph embeddings, in: Proc. of the 21st Int. Semantic Web Conf. (ISWC 2022), Virtual event, 2022, pp. 592–609.

[3] D. Lukovnikov, A. Fischer, J.L. J.S. Auer, Neural network-based question answering over knowledge graphs on word and character level, in: Proc. of the 26th Int. Conf. on World Wide Web (WWW 2017), Perth, Australia, 2017, pp. 1211–1220.

[4] X. Wang, X. He, Y. Cao, M. Liu, T. Chua, KGAT: knowledge graph attention network for recommendation, in: Proc. of the 25th ACM SIGKDD Int. Conf. on Knowledge Discovery & Data Mining (KDD 2019), Anchorage, AK, USA, 2019, p. 950–958.

[5] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, Reading, MA, USA, Reading, MA, USA, 1995.

[6] S. Hölldobler, Y. Kalinke, H. Störr, Approximating the semantics of logic programs by recurrent neural networks, Appl. Intell. 11 (1) (1999) 45–58.

[7] S. Bader, A.S.d. Garcez, P. Hitzler, Computing first-order logic programs by fibring artificial neural networks, in: Proc. of the 18th Int. Florida Artificial Intelligence Research Society Conference (FLAIRS 2005), 2005, pp. 314–319.

[8] S. Bader, P. Hitzler, S. Hölldobler, A. Witzel, A fully connectionist model generator for covered first-order logic programs, in: Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007), Hyderabad, India, 2007, pp. 666–671.

[9] H. Dong, J. Mao, T. Lin, C. Wang, L. Li, D. Zhou, Neural logic machines, in: Proc. of the 7th Int. Conf. on Learning Representations (ICLR 2019), New Orleands, LA, USA, 2019.

[10] A. Campero, A. Pareja, T. Klinger, J. Tenenbaum, S. Riedel, Logical Rule Induction and Theory Learning Using Neural Theorem Proving, (2018). arXiv:abs/1809.02193

[11] T. Rocktäschel, S. Riedel, End-to-end differentiable proving, in: Advances in Neural Information Processing Systems 30 (NIPS 2017), Long Beach, CA, USA, 2017, pp. 3788–3800.

[12] F. Yang, Z. Yang, W.W. Cohen, Differentiable learning of logical rules for knowledge base reasoning, in: Advances in Neural Information Processing Systems 30 (NIPS 2017), Long Beach, CA, USA, 2017, pp. 2319–2328.

[13] A. Sadeghian, M. Armandpour, P. Ding, D.Z. Wang, DRUM: end-to-end differentiable rule mining on knowledge graphs, in: Advances in Neural Information Processing Systems 32 (NeurIPS 2019), Vancouver, Canada, 2019, pp. 15321–15331.

[14] L. Wu, P. Cui, J. Pei, L. Zhao, Graph Neural Networks: Foundations, Frontiers, and Applications, Springer, Singapore, 2023.

[15] M.S. Schlichtkrull, T.N. Kipf, P. Bloem, R. van den Berg, I. Titov, M. Welling, Modeling relational data with graph convolutional networks, in: Proc. of the 15th Int. Semantic Web Conf. (ISWC 2018), 10843, Heraklion, Greece, 2018, pp. 593–607.

[16] M. Pflueger, D.J. Tena Cucala, E.V. Kostylev, GNNQ: a neuro-symbolic approach to query answering over incomplete knowledge graphs, in: Proc. of the 21st Int. Semantic Web Conference (ISWC 2022), 13489, Virtual event, 2022, pp. 481–497.

[17] S. Liu, B. Cuenca Grau, I. Horrocks, E.V. Kostylev, INDIGO: GNN-based inductive knowledge graph completion using pair-wise encoding, in: Advances in Neural Information Processing Systems 34 (NeurIPS 2021), Virtual event, 2021, pp. 2034–2045.

[18] V.N. Ioannidis, A.G. Marques, G.B. Giannakis, A recurrent graph neural network for multi-relational data, in: Proc. of the 2019 IEEE Int. Conference on Acoustics, Speech and Signal Processing (ICASSP 2019), Brighton, UK, 2019, pp. 8157–8161.

[19] M. Qu, Y. Bengio, J. Tang, GMNN: graph Markov neural networks, in: Proc. of the 36th Int. Conf. on Machine Learning (ICML 2019), 97, Long Beach, CA, USA, 2019, pp. 5241–5250.

[20] Z. Yang, W.W. Cohen, R. Salakhutdinov, Revisiting semi-supervised learning with graph embeddings, in: Proc. of the 33rd Int. Conf. on Machine Learning (ICML 2016), 48, New York City, NY, USA, 2016, pp. 40–48.

[21] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: Proc. of the 5th Int. Conf. on Learning Representations (ICLR 2017), Toulon, France, 2017.

[22] M. Zhang, Y. Chen, Link prediction based on graph neural networks, in: Advances in Neural Information Processing Systems 31 (NeurIPS 2018), Montréal, Canada, 2018, pp. 5171–5181.

[23] K.K. Teru, E. Denis, W. Hamilton, Inductive relation prediction by subgraph reasoning, in: Proc. of the 37th Int. Conf. on Machine Learning (ICML 2020), 119, Virtual event, 2020, pp. 9448–9457.

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

[25] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, C. Lutz, OWL 2 Web Ontology Language: Profiles (2nd Edition), 2012, W3C Recommendation, https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/.

[26] P. Barceló, E.V. Kostylev, M. Monet, J. Pérez, J.L. Reutter, J.P. Silva, The logical expressiveness of graph neural networks, in: Proc. of the 8th Int. Conf. on Learning Representations (ICLR 2020), Addis Ababa, Ethiopia, 2020.

[27] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, Cambridge, UK, Cambridge, UK, 2007.

[28] X. Huang, M.A.R. Orth, İ.İ. Ceylan, P. Barceló, A theory of link prediction via relational Weisfeiler-Leman, in: Advances in Neural Information Processing Systems 36 (NeurIPS 2023), New Orleans, LA, USA, 2023.

[29] M. Grohe, The descriptive complexity of graph neural networks, in: Proc. of the 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2023), Boston, MA, USA, 2023, pp. 1–14.

[30] C. Morris, M. Ritzert, M. Fey, W.L. Hamilton, J.E. Lenssen, G. Rattan, M. Grohe, Weisfeiler and Leman go neural: higher-order graph neural networks, in: Proc. of the the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019), Honolulu, HI, USA, 2019, pp. 4602–4609.

[31] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, in: 7th International Conference on Learning Representations, OpenReview.net, 2019.
[32] G. Sourek, F. Zelezný, O. Kuzelka, Beyond graph neural networks with lifted relational neural networks, Mach. Learn. 110 (7) (2021) 1695–1738.
[33] M. Benedikt, C.-H. Lu, B. Motik, Tan, Decidability of graph neural networks via logical characterizations, in: K. Bringmann, M. Grohe, G. Puppis, O. Svensson (Eds.), Proc. of the 51st Int. Colloquium on Automata, Languages, and Programming (ICALP 2024), 297 of LIPIcs, Schloss Dagstuhl—Leibniz-Zentrum für Informatik, Tallinn, Estonia, 2024, pp. 127:1–127:20.
[34] D. Tena Cucala, B. Cuenca Grau, B. Motik, E.V. Kostylev, On the correspondence between monotonic max-sum GNNs and datalog, in: P. Marquis, T.C. Son, G. Kern-Isberner (Eds.), Proc. of the 20th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2023), Rhodes, Greece, 2023, pp. 658–667.
[35] D.J. Tena Cucala, B. Cuenca Grau, E.V. Kostylev, B. Motik, Explainable GNN-based models over knowledge graphs, in: Proc. of the 10th Int. Conf. on Learning Representations (ICLR 2022), Virtual event, 2022.
[36] A. Bordes, N. Usunier, A. García-Durán, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, in: Advances in Neural Information Processing Systems 26 (NIPS 2013), Lake Tahoe, NV, USA, 2013, pp. 2787–2795.
[37] W. Xiong, T. Hoang, W.Y. Wang, DeepPath: a reinforcement learning method for knowledge graph reasoning, in: Proc. of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP 2017), Copenhagen, Denmark, 2017, pp. 564–573.
[38] T. Dettmers, P. Minervini, P. Stenetorp, S. Riedel, Convolutional 2D knowledge graph embeddings, in: Proc. of the 32nd AAAI Conf. on Artificial Intelligence (AAAI 2018), New Orleans, LA, USA, 2018, pp. 1811–1818.
[39] X. Wang, D.J. Tena Cucala, B. Cuenca Grau, I. Horrocks, Faithful rule extraction for differentiable rule learning models, in: Proc. of the 12th Int. Conf. on Learning Representations (ICLR 2024), Viena, Austria, 2024.
[40] C. Meilicke, M.W. Chekol, D. Ruffinelli, H. Stuckenschmidt, Anytime bottom-up rule learning for knowledge graph completion, in: Proc. of the 28th Int. Joint Conf. on Artificial Intelligence (IJCAI 2019), Macao, China, 2019, pp. 3137–3143.
[41] R. Andrews, J. Diederich, A.B. Tickle, Survey and critique of techniques for extracting rules from trained artificial neural networks, Knowl. Based Syst. 8 (6) (1995) 373–389.
[42] T. Liu, Q. Lv, J. Wang, S. Yang, H. Chen, Learning rule-induced subgraph representations for inductive relation prediction, in: Advances in Neural Information Processing Systems 36 (NeurIPS 2023), New Orleans, LA, USA, 2023, pp. 3517–3535.
[43] M. Qu, J. Chen, L.A.C. Xhonneux, Y. Bengio, J. Tang, RNNLogic: learning logic rules for reasoning on knowledge graphs, in: Proc. of the 9th Int. Conf. on Learning Representations (ICLR 2021), Virtual event, 2021.
[44] P. Wang, D. Stepanova, C. Domokos, J.Z. Kolter, Differentiable learning of numerical rules in knowledge graphs, in: Proc. of the 8th Int. Conf. on Learning Representations (ICLR 2020), Addis Ababa, Ethiopia, 2020.
[45] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, A. McCallum, Go for a walk and arrive at the answer: reasoning over paths in knowledge bases using reinforcement learning, in: Proc. of the 6th Int. Conf. on Learning Representations (ICLR 2018), Vancouver, Canada, 2018.
[46] P.G. Omran, K. Wang, Z. Wang, Scalable rule learning via learning representation, in: Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI 2018), Stockholm, Sweden, 2018, pp. 2149–2155.
[47] Z. W, B. Paudel, L. Wang, J. Chen, H. Zhu, W. Zhang, A. Bernstein, H. Chen, Iteratively learning embeddings and rules for knowledge graph reasoning, in: Proc. of the 28th Int. World Wide Web Conf. (WWW 2019), San Francisco, CA, USA, 2019, pp. 2366–2377.
[48] J. Ferreira, M. de Sousa Ribeiro, R. Gonçalves, J. Leite, Looking inside the black-box: logic-based explanations for neural networks, in: Proc. of the 19th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2022), Haifa, Israel, 2022.
[49] D.J. Tena Cucala, B. Cuenca Grau, B. Motik, Faithful approaches to rule learning, in: Proc. of the 19th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2022), Haifa, Israel, 2022, pp. 484–493.
[50] S. You, D. Ding, K.R. Canini, J. Pfeifer, M.R. Gupta, Deep lattice networks and partial monotonic functions, in: Advances in Neural Information Processing Systems 30 (NIPS 2017), Long Beach, CA, USA, 2017, pp. 2981–2989.
[51] M.R. Gupta, A. Cotter, J. Pfeifer, K. Voevodski, K.R. Canini, A. Mangylov, W. Moczydlowski, A.V. Esbroeck, Monotonic calibrated interpolated look-up tables, J. Mach. Learn. Res. 17 (109) (2016) 1–7.
[52] A. Wehenkel, G. Louppe, Unconstrained monotonic neural networks, in: Advances in Neural Information Processing Systems 32 (NeurIPS 2019), Vancouver, Canada, 2019, pp. 1543–1553.
[53] L. Galárraga, C. Teflioudi, K. Hose, F.M. Suchanek, Fast rule mining in ontological knowledge bases with AMIE+, VLDB J. 24 (6) (2015) 707–730.
[54] N. Ahmadi, V. Huynh, V.V. Meduri, S. Ortona, P. Papotti, Mining expressive rules in knowledge graphs, ACM J. Data Inf. Qual. 12 (2) (2020) 1–27.
[55] Y. Gu, Y. Guan, P. Missier, Efficient Rule Learning with Template Saturation for Knowledge Graph Completion, (2020). arXiv:abs/2003.06071
[56] S. Muggleton, Inductive logic programming, New Gener. Comput. 8 (4) (1991) 295–318.
[57] A. Cropper, S. Dumancic, Inductive logic programming at 30: a new introduction, J. Artif. Intell. Res. 74 (2022) 765–850.
[58] R. Evans, E. Grefenstette, Learning explanatory rules from noisy data, J. Artif. Intell. Res. 61 (2018) 1–64.
[59] A. Cropper, S.H. Muggleton, Logical minimisation of meta-rules within meta-interpretive learning, in: Proc. of the 24th Int. Conf. on Inductive Logic Programming (ILP 2014), 9046, Nancy, France, 2014, pp. 62–75.
[60] X. Si, M. Raghothaman, K. Heo, M. Naik, Synthesizing datalog programs using numerical relaxation, in: Proc. of the 28th Int. Joint Conf. on Artificial Intelligence (IJCAI 2019), Macao, China, 2019, pp. 6117–6124.
[61] M. Raghothaman, J. Mendelson, D. Zhao, M. Naik, B. Scholz, Provenance-guided synthesis of datalog programs, ACM Program. Lang. 4 (62) (2020) 1–27.
[62] Z. Ying, D. Bourgeois, J. You, M. Zitnik, J. Leskovec, GNNExplainer: generating explanations for graph neural networks, in: Advances in Neural Information Processing Systems 32 (NeurIPS 2019), Vancouver, Canada, 2019, pp. 9240–9251.
[63] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, X. Zhang, Parameterized explainer for graph neural network, in: Advances in Neural Information Processing Systems 33 (NeurIPS 2020), Virtual event, 2020.
[64] W. Lin, H. Lan, B. Li, Generative causal explanations for graph neural networks, in: Proc. of the 38th Int. Conf. on Machine Learning (ICML 2021), 139, Virtual event, 2021, pp. 6666–6679.
[65] M. Sälzer, M. Lange, Fundamental limits in formal verification of message-passing neural networks, in: Proc. of the 11th Int. Conf. on Learning Representations (ICLR 2023), Kigali, Rwanda, 2023.
[66] M. Pflueger, D.J. Tena Cucala, E.V. Kostylev, Recurrent graph neural networks and their connections to bisimulation and logic, in: Proc. of the 38th AAAI Conf. on Artificial Intelligence (AAAI 2024), Vancouver, Canada, 2024.
[67] P. Nunn, M. Sälzer, F. Schwarzentruber, N. Troquard, A Logic for Reasoning About Aggregate-Combine Graph Neural Networks, (2024). arXiv:abs/2405.00205
[68] F. Geerts, J.L. Reutter, Expressiveness and approximation properties of graph neural networks, in: The 10th Int. Conf. on Learning Representations (ICLR 2022), Virtual event, 2022.
[69] E. Rosenbluth, J. Tönshoff, M. Grohe, Some might say all you need is sum, in: Proc. of the 32nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013), Macao, China, 2023, pp. 4172–4179.