Graph neural networks with polynomial activations have limited expressivity

Sammy Khalife
Johns Hopkins University, Department of Applied Mathematics and Statistics

Abstract

The expressivity of Graph Neural Networks (GNNs) can be entirely characterized by appropriate fragments of the first order logic. Namely, any query of the two variable fragment of graded modal logic (GC2) interpreted over labeled graphs can be expressed using a GNN whose size depends only on the depth of the query (uniformity). As pointed out by [2, 9], this description holds for a family of activation functions of the underlying neural network, leaving the possibility for a hierarchy of logics uniformly expressible by GNNs depending on the chosen activation function. In this article, we show that such hierarchy indeed exists by proving that GC2 queries cannot be uniformly expressed by GNNs with polynomial activations and aggregations. This implies a separation between the expressivity of GNNs with polynomial and those with non polynomial activations (such as Rectified Linear Units) and partially answers an open question formulated by [2, 9].

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Machine learning theory; Mathematics of computing; Computing methodologies → Neural networks

Keywords and phrases Graph Neural Networks; Uniform Expressivity; Polynomial Activations; Logic

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Graph Neural Networks (GNNs) are deep learning architectures for input data that incorporates some relational structure represented as a graph, and have proven to be very performant and efficient for various types of learning problems [20, 4, 7, 6, 16, 11, 8, 13, 17, 19, 3, 5, 15]. Understanding the *expressive power* of GNNs, and its dependence on the activation function of the underlying neural networks is one of the basic tasks towards a rigorous study of their computational capabilities.

In this context, several approaches have been conducted in order to describe and characterize the expressivity of GNNs. The first approach consists in comparing GNNs to other standard computation models on graphs such as the *color refinement* or the *Wesfeiler-Leman* algorithms. This *reduction* type of approach stands to reason as the computational models of GNNs, Wesfeiler-Leman/color refinement algorithms are intimately connected: they all fall under the paradigm of trying to discern something about the global structure of a graph from local neighborhood computations. In that regard, it has been proven [14, 18] that the color refinement algorithm precisely captures the expressivity of GNNs. More precisely, there is a GNN distinguishing two nodes of a graph if and only if colour refinement assigns different colours to these nodes. This results holds if one supposes that the size of the underlying neural networks are allowed to grow with the size of the input graph. Hence, in his survey, [9] emphasizes the fact that this equivalence has been established only for unbounded GNN, and asks: *Can GNNs with bounded size simulate color refinement?* In [1], the authors answer by the negative if the underlying neural network are supposed to have Rectified Linear Unit (ReLU) activation functions. In [12] the authors provide a generalization of this result, for GNNs with piecewise polynomial activation functions. Furthermore, explicit lower bounds on the neural network size to simulate the color refinement can be derived for piecewise-polynomial activation functions given upper bounds on the number of regions of a neural network with piecewise-polynomial activation.

The second line of research to study the expressive power of GNNs is to characterize the types of boolean queries interpreted over labeled graphs that a GNN can *simulate*. Given a boolean query Q (taking as input a graph, or a graph and one of its vertices), *does there exist a GNN whose output characterizes the output of Q?* For example, *can a GNN express if a vertex of a graph is part of a clique of given size?* Furthermore, *can we characterize the set of queries that can be simulated by GNNs?* In that context if the size of the GNN does not depend on the size of the input, graph, then we say that the GNN expresses the query *uniformly* (by size we mean the total number of weights of the underlying neural networks of the GNN, and supposing that the number of iterations of the GNN does not depend on the size of the input either). Uniform expressivity is interessing from a practical standpoint as it captures the expressivity of GNNs of fixed size with respect to the input graphs.

Several mathematical answers to such questions have been already obtained. Notably, a complete description of the logic of the queries that the GNNs can express uniformly has been derived. The suitable formal logic interpreted over labelled graphs is a two variable fragment of *graded model logic* (GC2). Any GNN expresses a query of this logic, and conversely, any query of this logic can be expressed by a GNN whose size and iterations only depends on the depth of the query [2, 9]. For specific activation functions such as ReLUs, the size of a GNN required to express a given query of GC2 does not depend on the size of the input graph, but only on the number of *subformulas* (or *depth*) of the query. The known proofs of this result [2, 9] provide an explicit construction of a GNN with ReLU activations that expresses a given query. In recent results [10], the author provides a more general description of the logics expressible by GNNs, and also treat the non-uniform case. The uniform case is obtained for rational piecewise-linear activations (or equivalently, rational ReLUs), and a non-uniform result is presented for GNNs with general arbitrary real weights and activation functions. The author also presents new results about the expressivity of GNNs if random initialisation is allowed on the features of the vertices. In this article, we focus on uniform expressivity and consider the following question: *What is the impact of the activation and aggregation functions on the logic uniformly expressed by GNNs?*

Main contributions. In this article we show a separation between polynomial and non-polynomial activations (and in particular, piecewise linear activations) with respect to the logic expressible by GNNs with those activation functions. More precisely, we prove that GNNs with polynomial activation and aggregation functions cannot express all GC2 queries (uniformly), although GNNs with piecewise linear activations and a linear aggregation function can. This result holds even if: i) the weights of the polynomial GNNs are arbitrary real numbers with infinite precision, and ii) the weights of the GNNs with piecewise polynomial are restricted to integers (also, the underlying neural networks are supposed to have finitely many linear pieces). This shows how the power of graph neural networks can change immensely if one changes the activation function of the neural networks. Our result constitutes an additional step towards a complete understanding of the impact of the activation function on the formal expressivity of GNNs.

The rest of this article is organized as follows. Section 2 presents the definitions of GNNs and the background logic. In Section 3, we state our main result and compare it to the existing ones. Section 4 presents an overview of the proof of our main result, as well as proofs of the technical lemmata are presented. We conclude with some remarks and open questions in Section 5.

2 Preliminaries

2.1 Graph Neural Networks (GNNs)

We assume the input graphs of GNNs to be finite, undirected, simple, and vertex-labeled: a graph is a tuple $G = (V(G), E(G), P_1(G), \cdots, P_\ell(G))$ consisting of a finite vertex set $V(G)$, a binary edge relation $E(G) \subset V(G)^2$ that is symmetric and irreflexive, and unary relations $P_1(G), \cdots, P_\ell(G) \subset V(G)$ representing $\ell > 0$ vertex labels. In the following, we suppose that the $P_i(G)$'s form a partition of the set of vertices of G, i.e. each vertex has a unique label. Also, the number ℓ of labels, which we will also call *colors*, is supposed to be fixed and does not grow with the size of the input graphs. This setup introduced by [2, 9] allow to model the presence of features of the vertices of input graphs in practical applications. In order to describe the logic of GNNs, we also take into account access to the color of the vertices into the definition of the logic considered, as we shall see in Section 2.2. When there is no ambiguity about which graph G is being considered, $N(v)$ refers to the set of neighbors of v in G not including v. $|G|$ will denote the number of vertices of G. We use simple curly brackets for a set $X = \{x \in X\}$ and double curly brackets for a multiset $Y = \{\{y \in Y\}\}$.

▶ **Definition 1** (Neural network). *Fix an* activation function $\sigma : \mathbb{R} \to \mathbb{R}$. *For any number of hidden layers $k \in \mathbb{N}$, input and output dimensions w_0, $w_{k+1} \in \mathbb{N}$, a $\mathbb{R}^{w_0} \to \mathbb{R}^{w_{k+1}}$* neural network with σ activation *is given by specifying a sequence of k natural numbers $w_1, w_2, \cdots, w_k$ representing widths of the hidden layers and a set of $k+1$ affine transformations $T_i : \mathbb{R}^{w_{i-1}} \to \mathbb{R}^{w_i}$, $i = 1, \ldots, k+1$. Such a NN is called a $(k+1)$-layer NN, and is said to have k hidden layers. The function $f : \mathbb{R}^{w_0} \to \mathbb{R}^{w_{k+1}}$ computed or represented by this NN is:*

$$f = T_{k+1} \circ \sigma \circ T_k \circ \cdots T_2 \circ \sigma \circ T_1.$$

In the following, the *Rectified Linear Unit* activation function $\mathrm{ReLU} : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is defined as $\mathrm{ReLU}(x) = \max(0, x)$. The *Sigmoid* activation function $\mathrm{Sigmoid} : \mathbb{R} \to (0,1)$ is defined as $\mathrm{Sigmoid}(x) = \frac{1}{1+e^{-x}}$.

▶ **Definition 2** (Graph Neural Network (GNN)). *A GNN is characterized by:*

- *A positive integer T called the number of iterations, positive integers $(d_t)_{t\in\{1,\cdots,T\}}$ and $(d'_t)_{t\in\{0,\cdots,T\}}$ for inner dimensions. $d_0 = d'_0 = \ell$ is the input dimension of the GNN (number of colors) and d_T is the output dimension.*
- *a sequence of combination and aggregation functions $(\mathsf{comb}_t, \mathsf{agg}_t)_{t\in\{1,\cdots,T\}}$. Each aggregation function agg_t maps each finite multiset of vectors of $\mathbb{R}^{d_{t-1}}$ to a vector in $\mathbb{R}^{d'_t}$. For any $t \in \{1, \cdots, T\}$, each combination function $\mathsf{comb}_t : \mathbb{R}^{d_{t-1}+d'_t} \longrightarrow \mathbb{R}^{d_t}$ is a neural network with given activation function $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$.*

The update rule of the GNN at iteration $t \in \{0, \cdots, T-1\}$ for any labeled graph G and vertex $v \in V(G)$, is given by:

$$\xi^{t+1}(v) = \mathsf{comb}(\xi^t(v), \mathsf{agg}\{\{\xi^t(w) : w \in \mathcal{N}_G(v)\}\})$$

Each vertex v is initially attributed an indicator vector $\xi^0(v)$ of size ℓ, encoding the color of the node v: the colors being indexed by the palette $\{1, \cdots, \ell\}$, $\xi^0(v) = e_i$ (the i-th canonical vector) if the color of the vertex v is i. We say that a GNN has polynomial activations provided the underlying neural network comb *has polynomial activation functions.*

▶ Remark 3. The type of GNN in our Definition 2 is sometimes referred to as *aggregation-combine* GNNs *without global readout*. Here are a few variants that can be found in the litterature:

- *Recurrent* GNNs, allowing comb_t and agg_t functions, do not depend on the iteration t.
- GNNs *with global readout*, which allow the aggregation functions to also take as input the embeddings of all the vertices of the graph. See Remark 16 for more details on the logic side of things.
- General *Message-passing* GNNs that allow operations before the aggregation on the neigbhors *as well as the current vertex.* As mentioned in [10][Remark 4.1], it is not clear whether this type of GNN increases expressivity.

▶ Remark 4. Since aggregation functions are defined on multisets of varying size, we say that an aggregation function is polynomial provided for every size of the input multiset, the aggregation function is a (symmetric) multivariate polynomial of the entries of the multiset.

2.2 Logical background: GC2 and queries

The first-order language of graph theory we consider is built up in the usual way from a *vocabulary* containing variables $x_1, x_2, \cdots, x_m$, the relations symbols E and $=$, the logical connectives $\wedge, \vee, \neg, \rightarrow$, and the quantifiers $\forall$ and $\exists$.

Given a number of colors ℓ and a colored graph $G = (V(G), E(G), P1(G), ..., P^\ell(G))$, where $P1(G), \cdots, P^\ell(G) \subset V(G)$ represent the vertex colors (one color per vertex),

- The universe A of the logic is given by $A = V$.
- the set of symbols S of the first order logic we consider is composed of:
  - the 2-ary edge relation symbol E: $(x, y) \in A^2$ are related if and only if $(x, y) \in E$.
  - function symbols $\mathsf{col}_1, \cdots, \mathsf{col}_\ell$. $\mathsf{col}_i(v \in G)$ returns 1 if the color of v is the i-th one.
- The pair (A, I) allows us to interpret the first order logic, where the map I is a function from A^2 to $\{0, 1\}$ (or equivalently, a subset of the set of pair of vertices).

The quantifiers, variables, and set of symbols form the *alphabet* of the logic considered. The set of *formulas* in the logic is a set of strings over the alphabet defined inductively (see the appendix for a formal definition). The pair (A, I), called an S-structure for the logic, allows us to interpret the formulas of the logic (here, in the space of graphs). Any graph is naturally associated to an S structure in order to interpret the sentences of formula of the logic. For example, the following formula interpreted over a graph G expresses that no vertex of G is isolated: $\forall x \exists y E(x, y)$. Similarly, the formula $\forall x \neg E(x, x)$ expresses the fact that we do not want any self loops. A more interesting example is given by

$$\psi := \forall x \forall y [E(x, y) \rightarrow E(y, x) \wedge x \neq y] \tag{1}$$

expresses that G is undirected and loop-free. Similarly,

$$\begin{aligned} \phi := \forall x \exists y \exists z (\neg(y = z) \wedge E(x, y) \wedge E(x, z)) \\ \wedge \forall w (E(x, w) \rightarrow ((w = y) \vee (w = z))) \end{aligned} \tag{2}$$

expresses that every node x of the considered graph has exactly two out-neigbhors.

Since GNNs can output values for every vertex of a graph, the formulas we will be using to have to take as "input" some vertex variable; we call such variable a *free* variable. Concretely, a free variable is a variable which is not bound to a quantifier. Namely, the Formulas 1 and 2, contain no free variable. However, the formula $\phi(x) := \exists y \exists z (\neg(y = z) \wedge E(x, y) \wedge E(x, z))$ has a single free variable x. We interpret ϕ with (G, v) as a S-structure, where G is a graph and v one of its vertices (and the assignment that maps x to v). In this case, ϕ expresses that vertex v has two out-neighbors in G.

Any formula in $FO(S)$ (the first order logic on S) can be thought of as a 0/1 function on the class of all interpretations of $FO(S)$:

1. If ϕ is a sentence, then any graph G is an S-structure and is mapped to 0 or 1, depending on whether G satisfies ϕ or not.
2. If $\phi(x)$ is a formula with a single free variable x, then any pair (G, v), where $G = (V, E)$ is a graph and $v \in V$, is an interpretation with G as the S-structure and the assignment β maps x to v. Thus, every pair (G, v) is mapped to 0 or 1, depending on whether (G, β) satisfies ϕ or not. This example can be extended to handle formulas with multiple free variables, where we may want to model 0/1 functions on subsets of vertices.

The *depth* of a formula ϕ is defined recursively as follows (for a formal and general definition of depth, cf. Definition 30). If ϕ is of the form in then its depth is 0. If $\phi = \neg\phi'$ or $\phi = \forall x\phi'$ or $\phi = \exists x\phi'$, then the depth of ϕ is the depth of ϕ' plus 1. If $\phi = \phi_1 \star \phi_2$ with $\star \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$, then the depth of ϕ is the 1 more than the maximum of the depths of ϕ_1 and ϕ_2. In order to characterize the logic of GNNs, we are interested in a fragment of the first order logic, defined as follows.

▶ **Definition 5** (Guarded model logic (GC) and GC2 [2, 9])**.** *The fragment of guarded logic GC is formed using quantifiers that restrict to range over the neighbours of the current nodes. GC-formulas are formed from the atomic formulas by the Boolean connectives and quantification restricted to formulas of the form $\exists^{\geq p} y(E(x,y) \wedge \psi))$, where x and y are distinct variables and x appears in ψ as a free variable. Note that every formula of GC has at least one free variable. We refer to the 2-variable fragment of GC as GC2, also known as graded modal logic (with two variables).*

A depth-recursive definition can be given as follows: a graded modal logic formula F is either $\mathsf{Col}(x)$ (returning 1 or 0 for one of the palette colors) or one of the following:

$$\neg\phi(x), \quad \phi(x) \wedge \psi(x), or \quad \exists^{\geq N} y(E(x,y) \wedge \phi(y))$$

where N is a positive integer and ϕ and ψ are GC2 formulas of smaller depth than F.

▶ **Example 6** ([2])**.** All graded modal logic formulas are unary by definition, so all of them define unary queries. Suppose $\ell = 2$ (number of colors), and for illustration purposes $\mathsf{Col}_1 = \text{Red}$, $\mathsf{Col}_2 = \text{Blue}$. Let:

$$\gamma(x) := \text{Blue}(x) \wedge \exists y(E(x,y) \wedge \exists^{\geq 2} x(Edge(y,x) \wedge \text{Red}(x))$$

γ queries if x has blue color, and has at least one neigbhor which has at least two red neighbors. Then γ is in GC2. Now,

$$\delta(x) := \text{Blue}(x) \wedge \exists y(\neg E(x,y) \wedge \exists^{\geq 2} x E(y,x) \wedge \text{Red}(x))$$

is not in GC2 because the use of the guard $\neg E(x,y)$ is not allowed. However,

$$\eta(x) := \neg(\exists y(E(x,y) \wedge \exists^{\geq 2} x E(y,x) \wedge \text{Blue}(x))$$

is in GC2 because the negation $\neg$ is applied to a formula in GC2.

2.3 Color refinement

For the proof of our main result, we will need two additional definitions that we include for completeness:

▶ **Definition 7** (Embeddings and refinement). *Given a set X, an embedding ξ is a function that takes as input a graph G and a vertex $v \in V(G)$, and returns an element $\xi(G,v) \in X$. We say that an embedding ξ refines an embedding ξ' if and only if for any graph G and any $v \in V(G), \xi(G,v) = \xi(G,v') \implies \xi'(G,v) = \xi'(G,v')$. When the graph G is clear from context, we use $\xi(v)$ as shorthand for $\xi(G,v)$.*

▶ **Definition 8** (Color refinement). *Given a graph G, and $v \in V(G)$, let $(G,v) \mapsto \mathsf{col(G,v)}$ be the function which returns the color of the node v. The color refinement refers to a procedure that returns a sequence of embeddings cr^t, computed recursively as follows:*

- $cr^0(G,v) = \mathsf{col}(G,v)$
- *For* $t \geq 0$, $\mathsf{cr}^{t+1}(G,v) := (\mathsf{cr}^t(G,v), \{\{\mathsf{cr}^t(G,w) : w \in N(v)\}\})$

In each round, the algorithm computes a coloring that is finer than the one computed in the previous round, that is, cr^{t+1} refines cr^t. For some $t \leq n := |G|$, this procedure stabilises: the coloring does not become strictly finer anymore.

The following connection between color refinement and GNNs will be useful to prove our main result. Notably, the theorem holds regardless of the choice of the aggregation function agg and the combination function comb.

▶ **Theorem 9** ([14, 18]). *Let d be a positive integer, and let ξ be the output of a GNN after d iterations. Then cr^d refines ξ, that is, for all graphs G, G' and vertices $v \in V(G)$, $v' \in V(G')$, $\mathsf{cr}^{(d)}(G,v) = \mathsf{cr}^d(G',v') \implies \xi(G,v) = \xi(G',v')$.*

3 Uniform expressivity of GNNs

▶ **Definition 10** ([9]). *Suppose that ξ is the vertex embedding computed by a GNN. We say that a GNN expresses uniformly a unary query Q there is a real $\epsilon < \frac{1}{2}$ such that for all graphs G and vertices $v \in V(G)$.*

$$\begin{cases} \xi(G,v) \geq 1-\epsilon & \text{if } v \in Q(G) \\ \xi(G,v) \leq \epsilon & \text{if } v \notin Q(G) \end{cases}$$

We are now equipped to state the known previous results regarding the expressivity of GNNs:

▶ **Theorem 11.** *[2, 9] Let Q be a unary query expressible in graded modal logic GC2. Then there is a GNN whose size depends only on the depth of the query, that expresses Q uniformly.*

▶ Remark 12. Let ℓ be the number of colors of the vertices in the input graphs, the family of GNNs with $\mathsf{agg} = \mathsf{sum}$, and $\mathsf{comb}(x,y) = \mathrm{ReLU}(Ax + By + C)$ (where $A \in \mathbb{N}^{\ell\times\ell}$, $B \in \mathbb{N}^{\ell\times\ell}$ and $C \in \mathbb{N}^{\ell}$) is sufficient to uniformly express all queries of GC2 uniformly. This result follows from the constructive proof in [2]. Furthermore, for each query Q of depth q, there is a GNN of this type with at most q iterations that expresses uniformly Q.

▶ **Example 13.** Let Q be the following GC2 query:

$$Q(x) := \mathrm{Red}(x) \wedge (\exists y E(x,y) \wedge \mathrm{Blue}(y))$$

asking if the vertex x has red color, and if it has a neighbor with blue color. Writing the subformulas of Q: $\mathrm{sub}(Q) = (Q1, Q2, Q3, Q4)$ with $Q_1 = \mathrm{Red}$, $Q_2 = \mathrm{Blue}$, $Q_3 = \exists(E(x,y) \wedge Q_2(y)$, and $Q_4 = Q = Q_1 \wedge Q_3$, let

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, c = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix}$$

and let σ be the clipped ReLU function, i.e. $\sigma(\cdot) := \min(1, \max(0, \cdot))$ (the clipped ReLU can be computed by a neural network with ReLU activations). Then, it can be verified that Q can be computed in 4 iterations with the update rule:

$$\xi^0(v) = 1, \quad \xi^{t+1}(v) := \sigma(A\xi^t(v) + B(\sum_{w \in N(v)} \xi^t(w)))$$

i.e. $Q_i(v) = \xi^4(v)_i$. In particular, $Q_i(v) = 1 \iff \xi^4(v)_i = 1$. Note that in that case, we are able to *compute exactly* the query.

▶ Remark 14. The ability of GNNs to compute exactly GC2 queries is used in the proof of Theorem 11 included in the first section of the appendix. (We also report that in the constructive proof of [2], the matrices A and B should be replaced by their transpose). We emphasize here that one cannot mimic the proof for sigmoid activations, even by replacing exact *computation* by uniform expressivity.

Theorem 11 has a partial converse, with a slight weakening:

▶ **Theorem 15.** *[2] Let Q be a unary query expressible by a GNN and also expressible in first-order logic. Then Q is expressible in GC2.*

▶ Remark 16. The right logic for GNNs *with global readout* (cf. Remark 3) is not GC2. If $C2$ is the fragment of the first order logic with counting quantifiers ($\exists^{\geq p}$) and with at most 2 variables; then we have the following result [2]: *Let Q be a Boolean or unary query expressible in C2. Then there is a GNN with global readout that expresses Q.*

In contrast with Theorem 11, we prove:

▶ **Theorem 17.** *There are GC2 queries that no GNN with polynomial activations and aggregations can uniformly express.*

Equivalently, if $\mathcal{L}_P$ (resp. $\mathcal{L}_{ReLU}$) is the set of logical queries uniformly expressible by GNNs with polynomial aggregation and activations (resp. piecewise linear activations and polynomial aggregation), then we have the following strict inclusion: $\mathcal{L}_P \subsetneq (\mathcal{L}_{ReLU} = \text{GC2})$. A natural question suggested by this result is: *what is the fragment of GC2 that polynomial GNNs can uniformly express?* As detailed in the next section, the query used in our proof uses logical negation. Although we do not settle the above question entirely, we can obtain the following corollary by immediate contradiction, as GNNs with polynomial activations and aggregations can simulate logical negation:

▶ **Corollary 18.** *There are queries of GC2 using only the guarded existential quantifiers with counting $\exists^{\geq K} E$, the logical and $\wedge$ and the atomic formulas Col(.), that GNNs with polynomial activations and aggregations cannot uniformly express.*

4 Proof of our main result

Overview. To prove our result we construct a query that no GNN with polynomial activation can uniformly express. We prove this statement by contradiction: on the one hand we interpret the embedding returned by a GNN with polynomial activations on a set of given input graphs, as a polynomial of some parameters of the graph structure. On the other hand, we interpret our query on the same set of input graphs. We show that if GNN were to uniformly express the query, then the polynomial obtained by the first evaluation cannot verify the constraints imposed by the query. Our approach uses fundamental properties of multivariate polynomials and can easily extend to a large family of queries.

Our set of inputs are formed using rooted unicolored trees of the form shown in Figure 1 which is a tree of depth two whose depth one vertices have prescribed degrees $k_1, \cdots, k_m$, with $k_1, \cdots, k_m \geq 1$. We first collect three elementary Lemmata, one that will be useful to *extract* monomials of largest degree in a multivariate polynomial (Lemma 19). Since the trees are parameterized by m-tuples of integers $k_1, \ldots, k_m$, the embedding of the root node computed by the GNN at any iteration is a function of these m integers. Since the activations are polynomials, these embeddings of the root node are multivariate symmetric polynomial functions of $k_1, \ldots, k_m$ (Lemma 20). Furthermore, the degree of these polynomials is bounded by a constant independent of m. Our proof of Theorem 17 builds on these results combined with fundamental properties of symmetric multivariate polynomials of bounded degree.

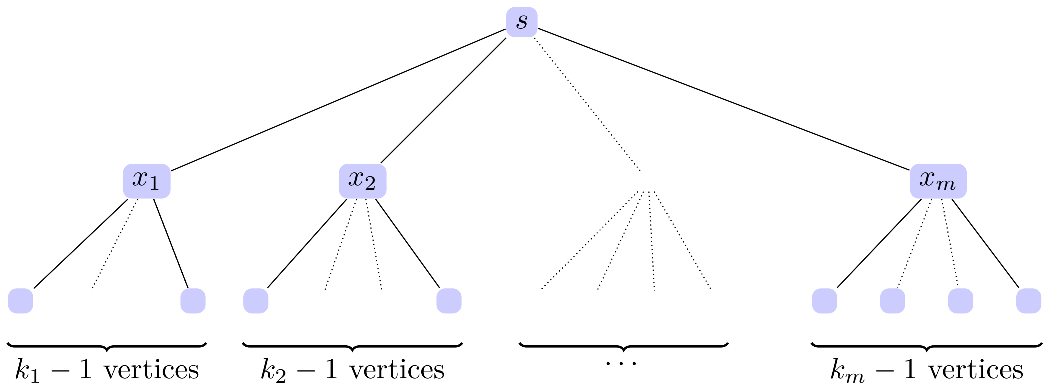

Figure 1 $T[k_1, \cdots, k_m]$

▶ **Lemma 19.** *Let p be a positive integer and let $S \subset \mathbb{N}^p$ be a finite subset of integral vectors of the nonnegative orthant, such that $|S| \geq 2$. Then there exist $x^* \in S$ and $u \in \mathbb{N}^p$ such that for any $x \in S - \{x^*\}$, $\langle x^*, u\rangle > \langle x, u\rangle$.*

Proof. Consider one vector x^* maximizing $\|x\|_2^2$ over S, i.e. $\|x^*\|_2^2 = \max_{x\in S}\|x\|_2^2$ then let $x \in S - \{x^*\}$. Such x^* and x exist because S is finite and $|S| \geq 2$.

- Case 1: x is not colinear to x^*. It follows from the Cauchy-Schwarz inequality, that $\langle x^*, x\rangle < \|x^*\|_2\|x\|_2$. Hence

$$\langle x^*, x^* - x\rangle = \|x^*\|_2^2 - \langle x^*, x\rangle > \|x^*\|_2(\|x^*\|_2 - \|x\|_2) > 0$$

- Case 2: $x \in S - \{x^*\}$ is colinear to x^*, i.e. $x = \lambda x^*$ with $\lambda \in \mathbb{R}$. Since x^* is maximizing the 2-norm on S, then $0 \leq \lambda < 1$. Then

$$\langle x^*, x^* - x\rangle = \|x^*\|(1 - \lambda) > 0$$

In both cases, $\langle x^*, x^* - x\rangle > 0$. Hence, we can set $u := x^* \in S \subset \mathbb{N}^p$, and the Lemma is proved. ◀

▶ **Lemma 20.** *Let $\xi^t(T[k_1, \ldots, k_m], s)$ be the embedding of the tree displayed in Figure 1 obtained via a GNN with polynomial activation and aggregation functions after t iterations, where $\xi^0(v) = 1$ for all vertices $v \in V(T[k_1, \ldots, k_m])$. Then, for any iteration t, there exists a symmetric multivariate polynomial F such that $\xi^t(T[k_1, \ldots, k_m], s) = F(k_1, \cdots, k_m)$. Furthermore, the degree of F does not depend on m, but only on the underlying neural network and t.*

Proof. For clarity, we will perform two separate inductions, one for the existence of the symmetric polynomial, and the second for the degree boundedness.

We first prove by induction on t that, for any vertex $v \in V(T[k_1, \cdots, k_m])$, $\xi^t(T[k_1, \ldots, k_m], v)$ is a polynomial function of the k_i's.

Base case: for $t = 0$ this is trivial since all vertices are initialised with the constant polynomial 1, whose degree does not depend on m.

Induction step: Suppose the property is true at iteration t, i.e for each node w, $\xi^t(T[k_1, \ldots, k_m], w)$ is a multivariate polynomial of the k_i's. Since

$$\begin{aligned}\xi^{t+1}(T[k_1, \ldots, k_m], v) = \mathsf{comb}(&\xi^t(T[k_1, \ldots, k_m], v),\\ &\mathsf{agg}(\{\{\xi^t(T[k_1, \cdots, k_m], w) : w \in N(v)\}\}))\end{aligned}$$

where comb is a neural network with polynomial activations, hence a multivariate polynomial. Also, agg is supposed polynomial in the entries of its multiset argument. Then by composition, $\xi^{t+1}(T[k_1, \ldots, k_m], v)$ is a multivariate polynomial of $k_1, \cdots, k_m$.

To conclude on the symmetry of the polynomial, we need the intermediary claim:

Claim: Let $T[k_1, \cdots, k_m]$ be two rooted trees given by Figure 1, and let s be its source vertex. Then for any iteration t, the color refinement embedding $\mathsf{cr}^t(s)$ is invariant by permutation of the k_i's.

Proof of claim. By induction of $t \geq 0$, we prove the following claim: for any integer $t \geq 0$, $\mathsf{cr}^t(s)$ and the multiset $\{\{\mathsf{cr}^t(x_1), \cdots, \mathsf{cr}^t(x_m)\}\}$ are invariant by permutation of the k_i's.

Base case: $(t = 0)$ is obvious since $\mathsf{cr}^0(v) = 1$ for any vertex $v \in T[k_1, \cdots, k_m]$.

Induction step: Suppose that for some iteration t, $\mathsf{cr}^t(s)$ and $\{\{\mathsf{cr}^t x_1, \cdots, \mathsf{cr}^t(x_m)\}\}$ are invariant by permutation of the k_i's. $\mathsf{cr}^{t+1}(s)$ being invariant by permutation of the k_i's is a consequence of: $\mathsf{cr}^{t+1}(s) = (\mathsf{cr}^t(s), \{\{\mathsf{cr}^t(x_1), \cdots, \mathsf{cr}^t(x_m)\}\})$ ◀

We know from Theorem 9 that the color refinement algorithm refines any GNN at any iteration. Since the tuple obtained by color refinement for the vertex s is invariant with respect to permutations of the k_i's, $\xi^t(T[k_1, \ldots, k_m], s)$ is also invariant with respect to permutations of the k_i's.

Finally, the degree boundedness also follows by induction:

Base case: At the first iteration $(t = 0)$, P_m is constant equal to 1 $(q_1 = 1)$ for any m.

Induction step: Suppose that for any iteration $t \leq T$, there exists $q_t \in \mathbb{N}$ (that does not depend on m nor the vertex $v \in V(T[k_1, \cdots, k_m])$), such that for any integer m, and for any iteration t, $\mathsf{deg}(P_m) \leq q_t$. Then, using again the update rule:

$$\begin{aligned}\underbrace{\xi^{t+1}(T[k_1, \cdots, k_m], v)}_{Q_m} = \mathsf{comb}(&\underbrace{\xi^t(T[k_1, \cdots, k_m], v)}_{R_m},\\ &\underbrace{\mathsf{agg}(\{\{\xi^t(T[k_1, \cdots, k_m], w) : w \in N(v)\}\})}_{S_m})\end{aligned}$$

R_m and S_m are multivariate polynomials of m variables. By the induction hypothesis, there exist r_t and s_t, such that for any m, $\mathsf{deg}(R_m) \leq q_t$ and $\mathsf{deg}(S_m) \leq q_t$.

The function comb is a bivariate polynomial of degree independent of m (neural network with a polynomial activation function). Let v be its degree. Hence, the degree of Q_m is at most $v \times q_t^2$. Hence the property remains true at $t + 1$, setting $q_{t+1} := v \times q_t^2$. ◀

Proof of Theorem 17. Consider the following query of GC2:

$$Q(s) = \neg\left(\exists^{\geq 1} x(E(s,x) \wedge \exists^{\leq 1} sE(x,s))\right) = \forall x E(s,x) \exists^{\geq 2} sE(x,s)$$

Q is true if and only if all the neigbhors of the node s have degree at least 2. We will prove by contradiction that any bounded GNN with polynomial activations and aggregations cannot uniformly express the query Q. Let $P_m := \xi^t(T[k_1, \cdots, k_m], s)$ be the embedding of the source node of $T[k_1, \cdots, k_m]$ returned by a GNN with polynomial activations, after a fixed number of iterations t. Suppose that it can uniformly express the query Q, then;

$$\begin{cases} P_m(k_1, \cdots, k_m) \geq 1 - \epsilon & \text{if } s \in Q(T[k_1, \cdots, k_m]) \\ P_m(k_1, \cdots, k_m) \leq \epsilon & \text{if } s \notin Q(T[k_1, \cdots, k_m]) \end{cases}$$

Let $\tilde{P}_m := P_m - \frac{1}{2}$ and $\epsilon' := \frac{1}{2} - \epsilon$. Interpreting the query Q over $T[k_1, \cdots, k_m]$ implies the following constraints on the sequence of polynomial $\tilde{P}_m$:

$$\exists \epsilon' > 0 \text{ such that } \forall k \in \mathbb{N}^m \begin{cases} \exists i \in \{1, \cdots, m\}, k_i = 0 \implies \tilde{P}_m(k) \leq -\epsilon' \\ \forall i \in \{1, \cdots, m\}, k_i > 0 \implies \tilde{P}_m(k) \geq \epsilon' \end{cases} \tag{3}$$

Let q be the degree of P_m, $q := \mathsf{deg}(P_m) = \mathsf{deg}(\tilde{P}_m)$ and let S be the set of exponents of the monomials of P_m, i.e.

$$S := \{(\alpha_1, \cdots, \alpha_m) \in \mathbb{N}^m : \alpha_1 + \cdots + \alpha_m \leq q \text{ and } k_1^{\alpha_1} \cdots k_m^{\alpha_m} \text{ is a monomial of } \tilde{P}_m\}$$

First, note that $q \geq 2$ ($\tilde{P}_m$ cannot be linear, otherwise its zero locus would contain a union of several hyperplanes as soon as $m \geq 2$). This insures that $|S| \geq 2$.

Then using Lemma 19 (with $p = m$) tells us there exists $\alpha^* \in S$ and $u = (u_1, \cdots, u_m) \in \mathbb{N}^m$ such that for any $\alpha' \in S - \{\alpha^*\}$,

$$\langle \alpha^*, u \rangle > \langle \alpha', u \rangle$$

Claim 1: *The (univariate) monomial $t^{\langle \alpha^*, u \rangle}$ is the monomial of $\tilde{P}_m(t^{u_1}, \cdots, t^{u_m})$ of* largest degree.

Proof.

$$\tilde{P}_m = \sum_{\alpha \in S} \gamma_\alpha k_1^{\alpha_1} \cdots k_m^{\alpha_m} \implies \tilde{P}_m(t^{u_1}, \cdots, t^{u_{m-1}}, t^{u_m}) = \sum_{\alpha \in S} \gamma_\alpha t^{\langle \alpha, u \rangle}$$

Hence, the monomial of largest degree of $\tilde{P}_m(t^{u_1}, \cdots, t^{u_m})$ is the one such that $\langle \alpha, u \rangle$ is (strictly) maximized when $\alpha \in S$. By construction, it is α^*. ◀

Now, suppose that $\tilde{P}_m$ has bounded degree (i.e. there is a uniform bound on the degree of the polynomials $\tilde{P}_m$ that is independent of m). Then, any monomial of largest degree of $\tilde{P}_m$ does not contain all the variables. Without loss of generality (by symmetry of $\tilde{P}_m$), we can suppose that the monomial $k_1^{\alpha_1^*} \cdots k_m^{\alpha_m^*}$ does not contain k_m, i.e. $\alpha_m^* = 0$.

Claim 2: *In these conditions, the (univariate) monomial $t^{\langle \alpha^*, u \rangle}$ is also* the *monomial of* of largest degree of $P_m(t^{u_1}, \cdots, t^{u_{m-1}}, 0)$.

Proof. Evaluating $\tilde{P}_m$ in $(t^{u_1}, \cdots, t^{u_{m-1}}, 0)$ removes the contribution of each monomial of P_m containing the last variable, and keeps only the contribution of the monomials containing it:

$$\tilde{P}_m = \sum_{\alpha \in S} \gamma_\alpha k_1^{\alpha_1} \cdots k_m^{\alpha_m} \implies \tilde{P}_m(t^{u_1}, \cdots, t^{u_{m-1}}, 0) =$$

$$\sum_{\alpha \in S \alpha = (\alpha_1, \cdots, \alpha_{m-1}, 0)} \gamma_\alpha t^{\langle \alpha, u \rangle}$$

Therefore, the monomial of largest degree of $\tilde{P}_m(t^{u_1}, \cdots, t^{u_{m-1}}, 0)$ is the one such that $\langle \alpha, u \rangle$ is (strictly) maximized when $\alpha \in S$ and $\alpha_m = 0$. By construction, such α is α^*. ◀

However, Conditions 3 imply that the leading monomial of $\tilde{P}_m(t^{u_1}, \cdots, t^{u_m})$ must have a positive coefficient but the leading monomial of $\tilde{P}_m(t^{u_1}, \cdots, t^{u_{m-1}}, 0)$ must have a negative coefficient, by taking both limits when t tends to $+\infty$ (this limit can be constant). This is a contradiction because Claims 1 and 2 imply that these coefficients are both γ_{α^*}.

Hence if P_m expresses Q, the leading monomial of $\tilde{P}_m$ contains all the variables, and $\mathsf{deg}(P_m) = \mathsf{deg}(\tilde{P}_m) \geq m$. This gives a contradiction with Lemma 19 since $P_m = \tilde{P}_m + \frac{1}{2}$ is supposed to be computed by a GNN. Therefore, no GNN with polynomial activations and aggregations can uniformly express the query Q.

◀

5 Discussion and open problems

The expressive capabilities of GNNs are accurately characterized by the color refinement (or Wesfeiler-Leman) algorithm and fragments of the 2-variable counting logic. It is valuable to comprehend the expressivity of machine learning architectures, and in particular those of GNNs, as it can help in selecting an appropriate architecture for a given problem and facilitates the comparison of various architectures and approaches. It is also essential to note that expressivity is just one facet of practical use of GNNs and the related machine learning algorithms. This paper does not delve into other crucial aspects such as the GNN's ability to generalize from provided data, and the computational efficiency of learning and inference. In particular, we have not investigated the ability of a GNN to learn a logical query from examples (without knowing the query in advance), for instance from a sample complexity standpoint. We believe that theoretical investigations on the expressivity of GNNs and logical expressivity can also suggest potential avenues to integrate logic-based and statistical reasoning in machine learning methods, and in particular in GNN architectures. These investigations also include questions of independent mathematical interest, some of which remain open. We list some below that are closely related to the results presented in this article:

Question 1: Can GNNs with sigmoidal activations and $\mathsf{agg} = \mathsf{sum}$ can uniformly express GC2 queries?

Question 2: Can GNNs with sigmoidal activations and polynomial activations uniformly express GC2 queries?

We conjecture that the answer to both 1 and 2 is negative. The answer to Question 3 seems less clear.

Question 3: Can GNNs with polynomial activation functions and a non polynomial aggregation function express uniformly GC2 queries?

More generally, we believe to be of interest to characterize the logics expressible by GNNs with certain activations and aggregation functions:

Question 4: What is the fragment of GC2 uniformly expressible by GNNs with polynomial activations and aggregation functions?

▶ Remark 21. Note that the proof of Theorem 11 can easily be extended to a larger family of queries. Namely, for any integer $p \geq 2$, let

$$Q_p(s) := \neg\left(\exists^{\geq 1} x(E(s,x) \wedge \exists^{\leq (p-1)} sE(x,s))\right) = \forall x E(s,x) \exists^{\geq p} sE(x,s)$$

Q_p queries if vertex s has neighbors whose degree are all at least p. Then any $(Q_p)_{p\in\mathbb{N}}$ cannot be expressed by any GNN with polynomial activations.

▶ Remark 22. The proof of Theorem 17 was initially attempted using queries of the form:

$$\tilde{Q}_p(s) = \neg\left(\exists^{\geq 1} x(E(s,x) \wedge \exists^{\geq (p+1)} sE(x,s))\right) = \forall x E(s,x) \exists^{\leq p} sE(x,s)$$

Which expresses that all the neighbors of s have degree at most p. Note the similarity between Q_p from Remark 21 and $\tilde{Q}_p$. Although $\tilde{Q}_p$ also seems a good candidate that cannot be expressed uniformly by a GNN with polynomial activations and aggregations, we could not conclude with the same approach as in the proof of Theorem 17, due to the following interesting fact:

There exists $\epsilon > 0$ and a sequence of symmetric polynomial $(p_m)_{m\in\mathbb{N}} \in \mathbb{R}[x_1, \cdots, x_m]$ of bounded degree (i.e. there exists an integer q such that for any m, $\mathsf{deg}(p_m) \leq q$) and for any m, p_m is greater than ϵ on the vertices of the unit hypercube $\{0,1\}^m$, and less than $-\epsilon$ on all the other points of $\mathbb{N}^m$. $p_m = 1 - \sum_{i=1}^m x_i^2 + \sum_{i=1}^m x_i^4$ is an example of such sequence of symmetric polynomials.

References

1. Anders Aamand, Justin Chen, Piotr Indyk, Shyam Narayanan, Ronitt Rubinfeld, Nicholas Schiefer, Sandeep Silwal, and Tal Wagner. Exponentially improving the complexity of simulating the weisfeiler-lehman test with graph neural networks. *Advances in Neural Information Processing Systems*, 35:27333–27346, 2022.
2. Pablo Barceló, Egor V Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations (ICLR 2020)*, 2020.
3. Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.
4. Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.
5. Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*, 2021.
6. Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.
7. Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. Cognitive graph for multi-hop reading comprehension at scale. *arXiv preprint arXiv:1905.05460*, 2019.
8. David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

9 Martin Grohe. The logic of graph neural networks. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–17. IEEE, 2021.

10 Martin Grohe. The descriptive complexity of graph neural networks. *arXiv preprint arXiv:2303.04613*, 2023.

11 William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

12 Sammy Khalife and Amitabh Basu. On the power of graph neural networks and the role of the activation function. *arXiv preprint arXiv:2307.04661*, 2023.

13 Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

14 Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

15 Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020.

16 Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

17 Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

18 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

19 Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

20 Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 2018.

A Proof that piecewise linear GNNs (and AGG=SUM) are as expressive as GC2

Proof. One can reformulate the claim of the Theorem as follows.

Claim. Let Q be a query in GC2, and let sub$(Q) = (Q_1, Q_2, \cdots, Q_d)$ be an enumeration of the sub-formulas of Q. Then, there exists a GNN returning an embedding ξ^t such that for graph G and any vertex $v \in V(G)$, $\xi^t \in \{0,1\}^d$, and for any $i \in \{1, \cdots, d\}$, $\xi_i^{t+1}(v) = 1 \iff Q_i(v) = 1$.

The overall GNN will take as input the graph G as well as for each node of v, $\xi^0(v) \in \{0,1\}^\ell$ encoding the colors of each node of G; and after L iterations, outputs for each node a vector $\xi^d(v) \in \{0,1\}^d$. Furthemore, at each intermediate iteration $t \in \{1, \cdots, d-1\}$, the constructed GNN will verify $\xi^t(v) \in \{0,1\}^d$. This property will be crucial for the inductive argument to go through.

In order to prove the claim, we simply need to find appropriate $(\mathsf{comb}_t)_{1 \le t \le d}$ and $(\mathsf{agg}_t)_{1 \le t \le d}$ functions such that if ξ^t verifying the update rule:

$$\xi^{t+1}(G, v) = \mathsf{comb}_t(\xi^t(G, v), \mathsf{agg}_t(\{\{\xi^t(G, v)) : v \in \mathcal{N}_G(v)\}\}))$$

then ξ^t have the property we are seeking, i.e. it computes the given query Q. We will prove that we can find such comb_t and agg_t functions by induction on the depth of Q.

Base case. If Q has depth 1, Q is one of ℓ color queries, and this can be computed via a GNN in one iteration, whose underlying neural network is the projection onto the $i-$th coordinate, i.e.

- $\mathsf{comb}_0(x, y) = \mathrm{proj}_i(x)$
- agg_0 can be chosen as any aggregation function.

Induction step. Let suppose Q be a query of depth $d > 1$.

First construction. By the induction hypothesis, we here suppose that we have access to some $(\mathsf{comb}_t)_{1 \le t \le d-1}$ and $(\mathsf{comb}_t)_{1 \le t \le d-1}$ such that for any $j \in \{1, \cdots, d-1\}$ and for any $1 \le i \le j$, $Q_j(v) = 1 \iff \xi_j^i(v) = 1$. Recall that sub$(Q) = (Q_1, Q_2, \cdots, Q_d)$, i.e. the Q_is form an enumeration of the sub-formulas of Q. In particular, $Q_d = Q$. In the following construction, ξ^i keeps "in memory" the output of the subformulas Q_j, for $j \le i$. For each case described above , we show that we are able to construct comb_d and agg_d in the following form:

- $\mathsf{comb}_d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d, (x, y) \mapsto \sigma(A_d X + B_d Y + c_d)$, where $\sigma(\cdot) = \min(1, \max(0, \cdot))$ is the clipped ReLU. Note that a clipped ReLU can be computed by a Neural Network with ReLU activations.
- agg_d is the sum function.

Due to the inductive nature of GC2, either one of the following holds:

- Case 1: there exist subformulas Q_j and Q_k of Q, such that $\ell(Q_j) + \ell(Q_k) = d$ and $Q = Q_j \wedge Q_k$
- Case 2: $Q(x) = \neg Q_j(x)$ where Q_j is a query of depth $d - 1$
- Case 3: there exists a subformula Q_j of Q such that $\ell(Q_j) = d - 1$ and $Q(x) = \exists^{\ge N} y(E(x, y) \wedge Q_j(y))$

We will first give general conditions on the update of the combinations and aggregate functions, and then conclude that these conditions are actually be met by constant comb and agg functions:

- A_d gets the same first $d - 1$ rows as A_{d-1}.

- B_d gets the same first $d-1$ rows as B_{d-1}
- c_d get the same first $d-1$ coordinates as c_{d-1},

- Case 1: The d-th row of A_d gets all zeros except: $(A_d)_{jd} = 1$, $(A_d)_{kd} = 1$. and the d-th row of B_t. Set $(c_d)_d = 0$.
- Case 2: The d-th row of A_d gets all zeros except: $(A_d)_{jd} = 1$. The d-th row of B_d is set to 0 except $(B_d)_{jd} = -1$. Set $(c_d)_d = -1$.
- Case 3: The d-th row of A_d gets all zeros except: $(A_d)_{jd} = 1$. The d-th row of B_d is set to 0 except $(B_d)_{jd} = -1$. Set $(c_d)_d = -N+1$.

What remains to prove is the following:

$$\text{For any } i \in \{1, \cdots, d\}, \quad \xi_i^d(G, v) = 1 \iff Q_i(v) = 1$$

Due to our update rule described for each case, the first $d-1$ coordinates of $\xi^d(G, v)$ are the same as $\xi^{d-1}(G, v)$. Hence, the property is true for $i \leq d-1$ by immediate induction. We are left to show that $\xi_d^d(G, v) = 1 \iff Q(v) = Q_d(v) = 1$. Here, we use the fact that:

- for every node v, every coordinate of $\xi^{d-1}(v)$ is in $\{0, 1\}$.
- σ is the clipped ReLU activation function: $\sigma(\cdot) = \min(1, \max(0, \cdot))$.

Since $\xi^d(v) = \sigma(A_t \xi^{d-1}(v) + B_t \sum_{w \in N(v)} \xi^{d-1}(w) + c_t)$ This follows from an immediate discussion on the three cases described previously, and ends the induction on d.

An important feature of the update described above is that (A_d, B_d, c_d) verifies all the conditions imposed for every $(A_t, B_t, c_t)_{1 \leq t \leq d}$ to compute all subformulas Q_t. The updates of A_t, B_t and c_t are only made for the t-th row and t-th entry, and do not depend on the previous columns but only on the query Q. Hence, we may as well start from the beginning by setting (A_t, B_t, c_t) to (A_d, B_d, c_d), instead of changing these matrices at every iteration t. In these conditions, the combination function comb_t, parametrized by A_t, B_t and c_t can be defined independently of t. The same holds for agg_t as it can be chosen as the sum for any iteration.

Second construction. We refer to the proof of [9] for an approach where the GNN is non-recurrent (each comb_t in that case depends on t). However, we insist on the fact that ξ^t is a $\{0, 1\}$ is a vector so that the proof with a clipped ReLU activation goes through. ◀

The reason for which the constructive proof in [2] does not extend to other activations (namely, sigmoid) is that for a fixed positive integer p, we need $f_p : \mathbb{R} \to \mathbb{R}$ such that for some $0 < \epsilon < \frac{1}{2}$, and for any $x_1, \cdots, x_N \in [0, \frac{1}{2} - \epsilon] \cup [\frac{1}{2} + \epsilon, 1]$,

$$f_p(\sum_{i=1}^{N} x_i) \geq \frac{1}{2} + \epsilon \iff \text{there are at least } p \ x_i\text{'s such that } x_i \geq \frac{1}{2} + \epsilon$$

Such f_p does not exist: on the one hand, it is necessary that $f(x) \geq \frac{1}{2} + \epsilon$ for any $x \geq p$. Furthermore, it is possible to pick $x_1, \cdots, x_N$ such that for any i, $x_i \in [0, \frac{1}{2} - \epsilon]$ but $x_1 + \cdots x_N \geq p$. This in turn would imply $f_p(\sum_{i=1}^{N} x_i) \geq \frac{1}{2} + \epsilon$ but all x_i's are smaller than $\frac{1}{2}$. The reason the constructive proof above goes through follows from the fact that the previous property becomes verified if one restricts to $\{0\} \cup \{1\}$ and $f_p(.) = \text{cReLU}(\cdot - p + 1)$ where cReLU is the clipped ReLU.

B Logic background: general definitions

▶ **Definition 23.** *A* first order logic *is given by a countable set of symbols, called the* alphabet *of the logic:*

1. *Boolean connectives:* $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$
2. *Quantifiers:* $\forall, \exists$
3. *Equivalence/equality symbol:* $\equiv$
4. *Variables:* $x_0, x_1, \ldots$ *(finite or countably infinite set)*
5. *Punctuation:* (,) *and* ,.
6. **a.** *A (possibly empty) set of* constant *symbols.*
   b. *For every natural number* $n \geq 1$*, a (possibly empty) set of* n-ary function *symbols.*
   c. *For every natural number* $n \geq 1$*, a (possibly empty) set of* n-ary relation *symbols.*

▶ Remark 24. Items 1-5 are common to *any* first order logic. Item 6 changes from one system of logic to another. Example: In Graph theory, the first order logic has:

- no constant symbols
- no function symbol
- a single 2-ary relation symbol E (which is interpreted as the edge relation between vertices). When graphs are supposed labeled with ℓ colors: ℓ function symbols $\mathsf{col}_1, \cdots, \mathsf{col}_\ell$. $\mathsf{col}_i(v \in G)$ returns 1 if the color of v is the i-th one.

The set of symbols from Item 6 is called the *vocabulary* of the logic. It will be denoted by S and the first order logic based on S will be denoted by $FO(S)$.

▶ **Definition 25.** *The set of* terms *in a given first order logic* $FO(S)$ *is a set of strings over the alphabet defined inductively as follows:*

1. *Every variable and constant symbol is a term.*
2. *If* f *is an* n*-ary function symbol, and* $t_1, \ldots, t_n$ *are terms, then* $f(t_1, \ldots, t_n)$ *is a term.*

▶ **Definition 26.** *The set of* formulas *in a given first order logic is a set of strings over the alphabet defined inductively as follows:*

1. *If* t_1, t_2 *are terms, then* $t_1 \equiv t_2$ *is a formula.*
2. *If* R *is an* n*-ary relation symbol, and* $t_1, \ldots, t_n$ *are terms, then* $R(t_1, \ldots, t_n)$ *is a formula.*
3. *If* ϕ *is a formula, then* $\neg\phi$ *is a formula.*
4. *If* ϕ_1, ϕ_2 *are formulas, then* $(\phi_1 \vee \phi_2)$*,* $\phi_1 \wedge \phi_2$*,* $\phi_1 \rightarrow \phi_2$ *and* $\phi_1 \leftrightarrow \phi_2$ *are formulas.*
5. *If* ϕ *is a formula and* x *is a variable, then* $\forall x\phi$ *and* $\exists x\phi$ *are formulas.*

The set of all variable symbols that appear in a term t will be denoted by $\mathrm{var}(t)$. The set of *free variables in a formula* is defined using the inductive nature of formulas:

1. $\mathrm{free}(t_1 \equiv t_2) = \mathrm{var}(t_1) \cup \mathrm{var}(t_2)$
2. $\mathrm{free}(R(t_1, \ldots, t_n)) = \mathrm{var}(t_1) \cup \ldots \cup \mathrm{var}(t_n)$
3. $\mathrm{free}(\neg\phi) = \mathrm{free}(\phi)$
4. $\mathrm{free}(\phi_1 \star \phi_2) = \mathrm{var}(\phi_1) \cup \mathrm{var}(\phi_2)$, where $\star \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$
5. $\mathrm{free}(\forall x\phi) = \mathrm{free}(\phi) \setminus \{x\}$
6. $\mathrm{free}(\exists x\phi) = \mathrm{free}(\phi) \setminus \{x\}$

▶ Remark 27. The same variable symbol may be a free symbol in ϕ, but appear bound to a quantifier in a subformula of ϕ.

▶ **Definition 28.** *The set of* sentences *in a first order logic are all the formulas with no free variables, i.e.,* $\{\phi : \mathrm{free}(\phi) = \emptyset\}$.

▶ **Definition 29.** *Given a first order logic* $FO(S)$*, an* S-structure *is a pair* $\mathcal{U} = (A, I)$ *where* A *is a nonempty set, called the* domain/universe *of the structure, and* I *is a map defined on* S *such that*

1. $I(c)$ *is an element of* A *for every constant symbol* c.
2. $I(f)$ *is a function from* A^n *to* A *for every* n*-ary function symbol* f.
3. $I(R)$ *is a function from* A^n *to* $\{0, 1\}$ *(or equivalently, a subset of* A^n*) for every* n*-ary relation symbol* R.

Given an S*-structure* $\mathcal{U} = (A, I)$ *for* $FO(S)$*, an* assignment *is a map from the set of variables in the logic to the domain* A*. An* interpretation *of* $FO(S)$ *is a pair* $(\mathcal{U}, \beta)$*, where* $\mathcal{U}$ *is an* S*-structure and* β *is an assignment.*

We say that an interpretation $(\mathcal{U}, \beta)$ *satisfies* a formula ϕ, if this assignment restricted to the free variables in ϕ evaluates to 1, using the standard Boolean interpretations of the symbols of the first order logic in Items 1-5 of Definition 23.

▶ **Definition 30.** *The* depth *of a formula* ϕ *is defined recursively as follows. If* ϕ *is of the form in points 1. or 2. in Definition 26 , then its depth is 0. If* $\phi = \neg\phi'$ *or* $\phi = \forall x\phi'$ *or* $\phi = \exists x\phi'$*, then the depth of* ϕ *is the depth of* ϕ' *plus 1. If* $\phi = \phi_1 \star \phi_2$ *with* $\star \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$*, then the depth of* ϕ *is the 1 more than the maximum of the depths of* ϕ_1 *and* ϕ_2.

This is equivalent to the depth of the tree representing the formula, based on the inductive definition. The length/size *of the formula is the total number nodes in this tree. Up to constants, this is the number of leaves in the tree, which are called the* atoms *of the formula.*