arXiv:2508.01067v2 [cs.LO] 18 Feb 2026

Expressive Power of Graph Transformers via Logic

Veeti Ahvonen[1], Maurice Funk[2], Damian Heiman[1], Antti Kuusisto[1], and Carsten Lutz[2]

[1]Tampere University, Finland
[2]Leipzig University, Germany

February 19, 2026

Abstract

Transformers are the basis of modern large language models, but relatively little is known about their precise expressive power on graphs. We study the expressive power of graph transformers (GTs) by Dwivedi and Bresson (2020) and GPS-networks by Rampásek et al. (2022), both under soft-attention and average hard-attention. Our study covers two scenarios: the theoretical setting with real numbers and the more practical case with floats. With reals, we show that in restriction to vertex properties definable in first-order logic (FO), GPS-networks have the same expressive power as graded modal logic (GML) with the global modality. With floats, GPS-networks turn out to be equally expressive as GML with the counting global modality. The latter result is absolute, not restricting to properties definable in a background logic. We also obtain similar characterizations for GTs in terms of propositional logic with the global modality (for reals) and the counting global modality (for floats).

1 Introduction

Transformers have emerged as a powerful machine learning architecture serving as the basis of modern large language models such as GPTs (Vaswani et al., 2017) and finding success also, e.g., in computer vision (Dosovitskiy et al., 2021). Recently, transformers have received significant attention in the field of graph learning, traditionally dominated by graph neural networks (GNNs) (Scarselli et al., 2009) and related formalisms like graph convolutional networks (Kipf and Welling, 2017). This shift is driven by well-known challenges GNNs face in handling long-range interactions, including issues such as over-squashing (Alon and Yahav, 2021) and over-smoothing (Li et al., 2018). Whereas GNNs rely primarily on local message passing, transformers can attend globally to any vertex in the graph. The literature now includes many graph learning models incorporating transformers. An important distinction is between 'pure' transformer models, which ignore the graph structure and result in 'bags-of-vertices' models (Yun et al., 2019; Kreuzer et al., 2021), and hybrids that combine transformers and GNN-style message passing (Rampásek et al., 2022).

To understand the limitations of learning models and their relationships, an expanding literature characterizes the expressive power of such models using logical formalisms. While transformers on words as used in GPTs connect to versions of linear temporal logic and first-order logic with counting (Li and Cotterell, 2025; Chiang et al., 2023), GNNs relate to variants of graded modal logic (GML) (Barceló et al., 2020; Benedikt et al., 2024).

In this article, we provide logical characterizations of graph learning models that incorporate transformers. Our characterizations are uniform in that we do not impose a constant bound on graph size. We are primarily interested in models that combine GNN message passing layers with transformer layers, and focus in particular on the rather general GPS-networks of Rampásek et al. (2022). In addition, we also consider pure bags-of-vertices graph transformers (GTs) (Yun et al., 2019; Dwivedi and Bresson, 2020). For both models, we study the case where features are vectors of real numbers, as in most theoretical studies, and also the case where they are floats, as in real-life implementations. We study both soft-attention and average hard-attention in the transformer layers. We focus on these models in their 'naked' form, without positional (or structural) encodings. Such encodings—often based on the graph Laplacian, homomorphism counts, and notions of graph centrality—enrich each vertex with information regarding its position in the graph. While they play an important role in graph learning with transformers, there is an uncomfortably large zoo of them. Therefore, we believe that to characterize expressive power, it is natural to begin with the naked case, providing a foundation for analyzing models with encodings. We focus on vertex classification as a basic learning task, but many of our results also generalize to graph classification tasks, see the end of Section 4.3 and Appendix D.4.

To survey our results, we start with the case of real numbers. Our first main result is that in restriction to vertex properties expressible in first-order logic (FO), GPS-networks based on reals have the same expressive power as GML with the (non-counting) global modality (GML + G), stated in Theorem 4. As all our results, this applies to both soft-attention and average hard-attention, assuming sum aggregation in message-passing layers as in (Barceló et al., 2020). While it is unsurprising that adding transformer layers to a GNN corresponds to adding a global feature to the logic GML, it was far from clear that this feature is the non-counting global modality, rather than, say, its counting version. Our result implies that GPS-networks cannot globally count in an absolute way, as in 'the graph contains at least 10 vertices labeled p'. In contrast, they can globally count in a relative way, as in 'the graph contains more vertices labeled p than vertices labeled q'. This, however, is not expressible in FO. The proof of our result is non-trivial and requires the introduction of a new type of bisimilarity (global-ratio graded bisimilarity $\sim_{G\%}$) and the proof of a new van Benthem/Rosen-style result that essentially states: if an FO-formula $\varphi(x)$ is invariant under $\sim_{G\%}$, then it is equivalent to a GML + G-formula. We also prove that, relative to FO, real-based GTs have the same expressive power as propositional logic PL with only the non-counting global modality (PL + G), stated as Theorem 10.

We next discuss our results regarding floating-point numbers. Our main result for this case is that float-based GPS-networks have the same expressive power as GML with the counting global modality (GML + GC), stated as Theorem 16. In contrast to the case of the reals, this characterization is absolute rather than relative to FO. It applies to any reasonable aggregation function including sum, max and mean. We consider it remarkable that transitioning from reals to floats results in incomparable expressive power: while relative global counting is no longer possible, absolute global counting becomes expressible. Our proof techniques leverage the underflow phenomenon of floats. Via techniques from (Ahvonen et al., 2024b), it follows that with *floats*, GPS-networks have the same expressive power as GNNs with counting global readout (GNN + GC). We also show that float-based GTs have the same expressive power as PL with the counting global modality (PL + GC), stated in Theorem 14. We note that the logic PL + GC was recently used to study links between entropy and description complexity (Jaakkola et al., 2025). We also briefly discuss our characterizations with floats in restriction to word-shaped graphs, how the float results generalize for graph classification and non-Boolean classification tasks, and how they could

be modified to cover positional encodings. Finally, we emphasize that the paper focuses exclusively on theoretical analysis and does not involve any experimental evaluation.

Related Work. To our knowledge, the expressive power of graph transformers and GPS-networks has not yet been studied from the perspective of logic. Regarding transformers *over words*, the closest to our work are (Jerad et al., 2025; Li and Cotterell, 2025), which characterized fixed-precision transformers that use 'causal masking' with soft-attention, average hard-attention and unique hard-attention, via the past fragment of linear temporal logic (LTL). In contrast, our characterizations exclude masking and focus on transformers on graphs rather than words. Similar characterizations via variations of LTL are given, for instance, in (Yang et al., 2024, 2025).

Regarding logic-based lower and upper bounds for the expressive power of transformers over words, we mention (Chiang et al., 2023), which established the logic $\mathsf{FOC}[+;\mathsf{MOD}]$ as an upper bound for the expressivity of fixed-precision transformer encoders and as a lower bound for transformer encoders working with reals. Merrill and Sabharwal (2023) showed that first-order logic with majority quantifiers is an upper bound for log-precision transformers. Yang and Chiang (2024) gave a lower bound for future-masked soft-attention transformers with unbounded input size via the past fragment of Minimal Tense Logic with counting terms ($\mathsf{K}_t[\#]$). Barceló et al. (2024) established two lower bounds for hard-attention transformers working with reals: first-order logic extended with unary numerical predicates (FO(Mon)) in the case of unique hard-attention and linear temporal logic extended with unary numerical predicates and counting formulas (LTL(C,+)) for average hard-attention. For a more comprehensive study of precise characterizations and upper/lower bounds on the expressive power of different transformer architectures on words, see the survey (Strobl et al., 2024).

Regarding non-logic-based studies of the expressive power of graph transformers, Kreuzer et al. (2021) showed that graph transformers with positional encodings are universal function approximators in the non-uniform setting. In the uniform setting, Rosenbluth et al. (2024) proved that GNN + GCs and GPS-networks using reals have incomparable expressive power w.r.t. function approximation. By contrast, this paper proves that with floats, GNN + GCs and GPS-networks are equally expressive w.r.t. both graph and vertex properties, and also w.r.t. non-Boolean graph classification which amounts to expressing functions over floating-point numbers on graphs. Recently, Sälzer et al. (2025) studied the complexity of verifying float-based GNNs, and showed that it is PSPACE-complete.

Barceló et al. (2020) pioneered work on the expressive power of graph neural networks by characterizing aggregate-combine GNNs with reals in restriction to FO via graded modal logic, and the same GNNs extended with a global readout mechanism (essentially the same as our GNN + GCs) with the two-variable fragment of FO with counting quantifiers. Grohe (2023) connected GNNs to the circuit complexity class TC^0, utilizing dyadic rationals. Ahvonen et al. (2024b) gave logical characterizations of recurrent and constant-iteration GNNs with both reals and floats, making similar assumptions to ours on float operations such as summation. Pfluger et al. (2024) characterized recurrent GNNs that use reals in terms of the graded two-way μ-calculus relative to a logic LocMMFP. We also mention (Benedikt et al., 2024), which characterized GNNs with bounded activation functions via logics involving Presburger quantifiers.

2 Preliminaries

We let $\mathbb{Z}_+$ denote the set of positive integers and $\mathbb{N}$ the set of non-negative integers. For $n \in \mathbb{Z}_+$, define $[n] := \{1, \ldots, n\}$. Also, set $\mathbb{B} := \{0, 1\}$. For a set S, we let $\mathcal{M}(S)$ denote the set of multisets over S, i.e., the set of functions $S \to \mathbb{N}$. We use double curly brackets $\{\{\cdots\}\}$ to denote multisets. For a multiset M, $M_{|k}$ denotes the **k-restriction** of M, i.e., the multiset given by $M_{|k}(x) = \min\{M(x), k\}$.

For $\mathbf{x} \in X^n$ and $i \in [n]$, let $\mathbf{x}_i$ denote the ith component of $\mathbf{x}$. For a matrix $M \in X^{n \times m}$, we use $M_{i,*}$, $M_{*,j}$ and $M_{i,j}$ to denote, respectively, the ith row (from the top), the jth column (from the left), and the jth entry in the ith row of M. For a sequence $(M^{(1)}, \ldots, M^{(k)})$ of matrices in $X^{n \times m}$, their **concatenation** is the matrix $M \in X^{n \times km}$ such that $M^{(\ell)}_{i,j} = M_{i,(\ell-1)m+j}$ for all $\ell \in [k]$. For non-empty sets X and Y, let X^+ denote the set of non-empty sequences over X, while $f \colon X^+ \to^{|\cdot|} Y^+$ is a notation for functions that map each sequence in X^+ to a sequence of the same length in Y^+.

2.1 Graphs and Feature Maps

For a finite domain $D \neq \emptyset$, a dimension $d \in \mathbb{N}$ and a non-empty set W, a **feature map** is a function $f \colon D \to W^d$ that maps each $x \in D$ to a **feature vector** $f_x \in W^d$. Typically, D consists of graph vertices and W is $\mathbb{R}$ or a set of floating-point numbers. If D is ordered by $<^D$, then we can identify f with a **feature matrix** $M \in W^{|D| \times d}$, where the row $M_{i,*}$ contains the feature vector of the ith element of D w.r.t. $<^D$, the column $M_{*,j}$ containing the jth vector components.

We work with vertex-labeled directed graphs, allowing self-loops, and simply refer to them as *graphs*. Let LAB denote a countably infinite set of **(vertex) label symbols**. We assume an ordering $<^{\mathrm{LAB}}$ of LAB, also inducing an ordering $<^L$ of every $L \subseteq \mathrm{LAB}$. Finite subsets of LAB are denoted by Π. Given $\Pi \subseteq \mathrm{LAB}$, a **$\Pi$-labeled graph** is a tuple $\mathcal{G} = (V, E, \lambda)$, where V is a finite non-empty set of **vertices**, $E \subseteq V \times V$ is a set of **edges** and $\lambda \colon V \to 2^\Pi$ a **vertex labeling function**. Here, 2^Π denotes the power set of Π. For convenience, we set $V(\mathcal{G}) := V$, $E(\mathcal{G}) := E$, and $\lambda(\mathcal{G}) := \lambda$. A **pointed graph** is a pair $(\mathcal{G}, v)$ with $v \in V$. The set of **out-neighbors** of $v \in V(\mathcal{G})$ is $\mathrm{Neigh}_\mathcal{G}(v) := \{\, u \mid (v, u) \in E \,\}$. We may identify λ with a feature map $\lambda' \colon V \to \mathbb{B}^{|\Pi|}$ where $\lambda'(v)_i = 1$ if the ith vertex label symbol (w.r.t. $<^\Pi$) is in $\lambda(v)$ and else $\lambda'(v)_i = 0$. Thus, Π-labeled graphs can be seen as graphs where each vertex is labeled with a single vector from $\mathbb{B}^{|\Pi|}$. We assume w.l.o.g. that for any graph $\mathcal{G}$, $V(\mathcal{G}) = [n]$ for some $n \in \mathbb{Z}_+$. Hence, we can identify feature maps of graphs with feature matrices and use labeling functions, feature maps and feature matrices interchangeably.

2.2 Graph Transformers and GNNs

We next discuss the computing architectures relevant to this article: graph transformers, GPS-networks, and GNNs. We view them as vertex classifiers that produce Boolean classifications. For this section, fix an arbitrary Π-labeled graph $\mathcal{G} = (V, E, \lambda)$ with $|\Pi| = \ell$ and $|V| = n$. In what follows, we will often speak of the input/hidden/output dimension of learning models and their components. For brevity, we abbreviate these to I/H/O dimension.

Basic Components.

A multilayer perceptron (or feedforward neural network) can be intuitively understood as a structure that propagates an input through a series of layers, where in each layer every "neuron" computes a weighted sum of its incoming signals, applies an activation function, and passes the result forward to the next layer. The formal details, along with some auxiliary notions, are given below.

A **perceptron layer** P of I/O dimension (d_I, d_O) consists of a **weight matrix** $W \in \mathbb{R}^{d_O \times d_I}$, a **bias term** $b \in \mathbb{R}^{d_O \times 1}$ and an **activation function** $\alpha\colon \mathbb{R} \to \mathbb{R}$. Given an input vector $\mathbf{x} \in \mathbb{R}^{d_I}$, P computes the vector $P(\mathbf{x}) := \alpha\big(b + W\mathbf{x}\big)$, where α is applied element-wise. A **multilayer perceptron** (MLP) F of I/O dimension (d_I, d_O) is a sequence $(P^{(1)}, \dots, P^{(m)})$ of perceptron layers, where each $P^{(i)}$ has I/O dimension (d_{i-1}, d_i), where $d_0 = d_I$ and $d_m = d_O$ and $P^{(m)}$ uses the identity activation function. Given a vector $\mathbf{x} \in \mathbb{R}^{d_I}$, F computes the vector $F(\mathbf{x}) := P^{(m)}(\cdots P^{(2)}(P^{(1)}(\mathbf{x}))\cdots)$. For a matrix $X \in \mathbb{R}^{n \times d_I}$, we let $F(X)$ denote the $\mathbb{R}^{n \times d_O}$-matrix, where F is applied row-wise for X. An MLP is **α-activated** if every layer uses α, except the last, which always uses the identity function. Unless otherwise stated, MLPs are ReLU-activated, where $\mathrm{ReLU}(x) = \max(0, x)$. An MLP is **simple** if it is ReLU-activated and has only two perceptron layers.

Next, we introduce aggregation functions and readout gadgets, which are essential components of graph transformers, GPS-networks and graph neural networks. An **aggregation function** of dimension d_I is a function $\mathrm{AGG}\colon \mathcal{M}(\mathbb{R}^{d_I}) \to \mathbb{R}^{d_I}$ which typically is (pointwise) sum, max or mean. It is **set-based** if $\mathrm{AGG}(M) = \mathrm{AGG}(M_{|1})$ for all $M \in \mathcal{M}(\mathbb{R}^{d_I})$. A **readout gadget** of I/O dimension (d_I, d_O) is a tuple $R := (F, \mathrm{AGG})$, where F and AGG are as above. Given a matrix $X \in \mathbb{R}^{n \times d_I}$, it computes the matrix $R(X) \in \mathbb{R}^{n \times d_O}$ where each row is the same, defined by $R(X)_{i,*} := F\big(\mathrm{AGG}(\{\!\{\, X_{j,*} \mid j \in [n] \,\}\!\})\big)$.

Graph Neural Networks.

A (constant-iteration) graph neural network can be intuitively viewed as a distributed system in which vertices of the input graph exchange information synchronously for a fixed number of rounds. In each round, every vertex updates its feature vector based on its previous feature vector and an aggregated representation of the feature vectors of its out-neighbors. The formal details follow below.

A **message-passing layer** of dimension d is a pair $L = (\mathrm{COM}, \mathrm{AGG})$, where COM is an MLP of I/O dimension $(2d, d)$ and AGG is an aggregation function of dimension d. A **message-passing layer with counting global readout** of dimension d is a pair (L, R), where L is defined as above and R is a readout gadget of I/O dimension (d, d). A **message-passing layer with non-counting global readout** (L, R) of dimension d is defined analogously, but the aggregation function of R is set-based.

A **graph neural network** (GNN) over (Π, d) is a tuple $G = (P, L^{(1)}, \dots, L^{(k)}, C)$ where P is an MLP of I/O dimension (ℓ, d), each $L^{(i)} = (\mathrm{COM}^{(i)}, \mathrm{AGG}^{(i)})$ is a message passing layer of dimension d, and C is an MLP of I/O dimension $(d, 1)$ that induces a function $\mathbb{R}^d \to \mathbb{B}$ called a **(Boolean vertex) classification head**. The MLP C does not have to be ReLU-activated, and can use, e.g., the Heaviside function σ, defined such that $\sigma(x) = 1$ if $x > 0$ and $\sigma(x) = 0$ if $x \leq 0$. Over a graph $\mathcal{G}$, G computes a sequence $\lambda^{(0)}, \dots, \lambda^{(k)}$ of feature maps and a final feature map $G(\mathcal{G})$ as follows: $\lambda^{(0)} := P(\lambda)$ and $\lambda^{(i+1)} := \lambda^{(i)} + L^{(i+1)}\big(\lambda^{(i)}\big)$,

where for each $v \in V$, $L^{(i+1)}\big(\lambda^{(i)}\big)$ maps v to the feature vector

$$\mathrm{COM}^{(i+1)}\Big(\lambda_v^{(i)}, \mathrm{AGG}^{(i+1)}\big(\{\{\lambda_u^{(i)} \mid (v,u) \in E\}\}\big)\Big).$$

Finally, $G(\mathcal{G}) := C(\lambda^{(k)})$.[1] The final feature map $G(\mathcal{G})$ labels each vertex with 0 or 1, and we say that G **accepts** a vertex v of $\mathcal{G}$ if $G(\mathcal{G})$ maps v to 1. Note that we follow the convention of (Rampásek et al., 2022; Rosenbluth et al., 2024) by including skip connections around message-passing layers, which refers to the fact that $\lambda^{(i+1)}$ is not simply defined as $L^{(i+1)}\big(\lambda^{(i)}\big)$. It is easy to see that GNNs have the same expressive power with and without skip connections.

We define **graph neural networks with counting global readout** (GNN + GCs) and **graph neural networks with non-counting global readout** (GNN + Gs) analogously, except each $L^{(i)}$ is a message-passing layer $(\hat{L}^{(i)}, R^{(i)})$ with counting and non-counting global readout, respectively. They behave analogously to GNNs, except that $\lambda^{(i+1)} := \hat{\lambda}^{(i+1)} + R^{(i+1)}(\hat{\lambda}^{(i+1)})$ where $\hat{\lambda}^{(i+1)} := \lambda^{(i)} + \hat{L}^{(i+1)}(\lambda^{(i)})$. In other words, each vertex's feature vector additionally reflects the global information aggregated from all vertices in the graph.

Example 1. We can define a GNN over $(\{p\}, 1)$ that accepts precisely the vertices that satisfy the following property $\mathcal{P}$: 'a vertex has at least 5 out-neighbors that have the label p'. The initial MLP assigns the feature vector (1) to each vertex labeled p, and (0) to others. The first (and only) message-passing layer uses summation as the aggregation function; if the sum is at least 5, the vertex has the property $\mathcal{P}$. Similarly, a GNN + G can express that a graph contains a vertex with the property $\mathcal{P}$ by adding a second layer that sums over all vertices in the graph, and a GNN + GC can express, e.g., that a graph contains exactly 3 vertices with the property $\mathcal{P}$.

Self-Attention and Graph Transformers.

A graph transformer is intuitively quite similar to a GNN; each vertex of a graph updates its feature vector in synchronous rounds. The key difference is that instead of aggregation and combination functions, a global attention mechanism is used to obtain information from all vertices. A GPS-network is then a variation of graph transformers that combines the attention mechanism with the update mechanism of GNNs. The formal details are given below.

A **self-attention head** H of I/H dimension (d, d_h) over $\mathbb{R}$ is defined w.r.t. an **attention-function** $\alpha\colon \mathbb{R}^+ \to^{|\cdot|} \mathbb{R}^+$ and three $\mathbb{R}^{d\times d_h}$-matrices: the **query-matrix** W_Q, the **key-matrix** W_K and the **value-matrix** W_V. Given a matrix $X \in \mathbb{R}^{n\times d}$, it computes the $n \times d_h$-matrix

$$H(X) := \alpha\left(\frac{(XW_Q)(XW_K)^\mathsf{T}}{\sqrt{d_h}}\right)(XW_V),$$

where α is applied row-wise. A **self-attention module** of dimension d over $\mathbb{R}$ is a tuple $\mathrm{SA} := (H^{(1)}, \ldots, H^{(k)}, W_O)$, where each $H^{(i)}$ is a self-attention head of I/H dimension (d, d_h) and $W_O \in \mathbb{R}^{kd_h\times d}$ is an **output matrix**. Let $\mathcal{H}(X)$ denote the concatenation of $H^{(1)}(X), \ldots, H^{(k)}(X)$. Now, SA computes the matrix $\mathrm{SA}(X) := \mathcal{H}(X)W_O$. For brevity, we may omit 'self' from 'self-attention'.

[1] Note that messages flow in the direction opposite to the edges. However, this is only a convention and could be reversed by using modal logics that operate over the inverse edge relation.

A **transformer layer** of dimension d is a pair $(\mathrm{SA}, \mathrm{FF})$, where SA is an attention module and FF is an MLP of dimension d. A **GPS-layer** of dimension d is a tuple $(\mathrm{SA}, \mathrm{MP}, \mathrm{FF})$, where MP is a message-passing layer.

A **graph transformer** (GT) over (Π, d) is a tuple $T = (P, L^{(1)}, \ldots, L^{(k)}, C)$, where P and C are as for GNNs and each $L^{(i)}$ is a transformer layer $(\mathrm{SA}^{(i)}, \mathrm{FF}^{(i)})$ of dimension d. A **GPS-network** N over (Π, d) is defined like a GT except that each $L^{(i)}$ is a GPS-layer $(\mathrm{SA}^{(i)}, \mathrm{MP}^{(i)}, \mathrm{FF}^{(i)})$ of dimension d. This definition does not include the optional normalization layers. For more about normalization layers, see e.g. (Rampásek et al., 2022). Analogously to a GNN, a GPS-network N computes over a graph $\mathcal{G}$ a sequence of feature maps and a final feature map $N(\mathcal{G})$ as follows: $\lambda^{(0)} := P(\lambda)$,

$$\begin{aligned} \lambda_B^{(i+1)} &:= \lambda^{(i)} + B^{(i+1)}\big(\lambda^{(i)}\big), \text{ where } B \in \{\mathrm{SA}, \mathrm{MP}\}, \\ \lambda_{\mathrm{SA+MP}}^{(i+1)} &:= \lambda_{\mathrm{SA}}^{(i+1)} + \lambda_{\mathrm{MP}}^{(i+1)}, \\ \lambda^{(i+1)} &:= \lambda_{\mathrm{SA+MP}}^{(i+1)} + \mathrm{FF}^{(i+1)}\big(\lambda_{\mathrm{SA+MP}}^{(i+1)}\big). \end{aligned}$$

Finally, $N(\mathcal{G}) := C(\lambda^{(k)})$. A GT T computes a feature map $T(\mathcal{G})$ analogously to N, but without the modules $\mathrm{MP}^{(i)}$:

$$\lambda^{(i+1)} := \lambda_{\mathrm{SA}}^{(i+1)} + \mathrm{FF}^{(i+1)}\big(\lambda_{\mathrm{SA}}^{(i+1)}\big).$$

Acceptance for GTs and GPS-networks is defined analogously to GNNs.

We focus on the two most commonly used attention functions. For $\mathbf{x} \in \mathbb{R}^\ell$, let $\mathcal{I}_{\mathbf{x}} = \{\, i \in [\ell] \mid \mathbf{x}_i = \max(\mathbf{x}) \,\}$ where max returns the largest entry in vector $\mathbf{x}$. We define the **average hard** (AH) and **softmax** functions:

1. $\mathrm{AH}(\mathbf{x})_i := \frac{1}{|\mathcal{I}_{\mathbf{x}}|}$ if $i \in \mathcal{I}_{\mathbf{x}}$ and $\mathrm{AH}(\mathbf{x})_i := 0$ otherwise,
2. $\mathrm{softmax}(\mathbf{x})_i := \frac{e^{x_i - b}}{\sum_{j \in [\ell]} e^{x_j - b}}$, where $b = \max(\mathbf{x})$.[2]

Example 2. For $\mathbf{x} = (5, 7, 1, 7) \in \mathbb{R}^4$, $\mathrm{softmax}(\mathbf{x}) \approx (0.063, 0.468, 0.001, 0.468)$ and $\mathrm{AH}(\mathbf{x}) = (0, \frac{1}{2}, 0, \frac{1}{2})$.

Attention heads that use AH or softmax are called average hard-attention heads and soft-attention heads, respectively. The same naming applies to attention modules, transformer layers, graph transformers, GPS-layers and GPS-networks. Later in Example 5 in Section 3, we show that there exists a soft-attention graph transformer that expresses the property: 'at least half of the vertices in the studied graph have the label symbol p'.

2.3 Logics

We define the logics used in this paper. Let Π be a finite set of vertex label symbols. With a first-order (FO) formula φ over Π, we mean a formula of first-order logic over the vocabulary that contains a unary relation symbol for each $p \in \Pi$ and a binary edge relation symbol E (equality is included). A **Π-formula of graded modal logic with the counting global modality** (GML + GC) is defined by the grammar $\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond_{\geq k}\varphi \mid \langle \mathsf{G} \rangle_{\geq k}\varphi$, where $p \in \Pi$ and $k \in \mathbb{N}$. We use $\vee$, $\rightarrow$ and $\leftrightarrow$ as abbreviations in the usual way, and for $D \in \{\Diamond, \langle \mathsf{G} \rangle\}$, we define that $D_{<k}\varphi := \neg D_{\geq k}\varphi$ and $D_{=k}\varphi := D_{\geq k}\varphi \wedge D_{<k+1}\varphi$.

[2] This is also known as the "stable" or "safe" softmax due to its numerical stability (Blanchard et al., 2019), in contrast to the version of softmax without the biases $-b$.

The semantics of GML + GC is defined over pointed graphs. In the field of modal logic, Π-labeled graphs are often called Kripke models. For a Π-formula φ of GML + GC and a pointed Π-labeled graph $(\mathcal{G}, v)$, the truth of φ in $(\mathcal{G}, v)$ (denoted by $\mathcal{G}, v \models \varphi$) is defined as follows. $\mathcal{G}, v \models \top$ holds always. For $p \in \Pi$, $\mathcal{G}, v \models p$ iff $p \in \lambda(v)$. The cases $\neg$ and $\wedge$ are defined in the usual way. For diamonds,

$$\begin{aligned} \mathcal{G}, v \models \Diamond_{\geq k}\varphi &\text{ iff } |\{u \in \mathrm{Neigh}_{\mathcal{G}}(v) \mid \mathcal{G}, u \models \varphi\}| \geq k \\ \mathcal{G}, v \models \langle \mathsf{G} \rangle_{\geq k}\varphi &\text{ iff } |\{u \in V(\mathcal{G}) \mid \mathcal{G}, u \models \varphi\}| \geq k. \end{aligned}$$

Graded modal logic with global modality (GML + G) is the fragment of GML + GC where diamonds $\langle \mathsf{G} \rangle_{\geq k}$ are allowed only if $k = 1$. For simplicity, we let $\langle \mathsf{G} \rangle := \langle \mathsf{G} \rangle_{\geq 1}$. **Graded modal logic** (GML) is the fragment of GML + G without diamonds $\langle \mathsf{G} \rangle$. **Modal logic** (ML) is the fragment of GML where we allow diamonds $\Diamond_{\geq k}$ only if $k = 1$, and we let $\Diamond := \Diamond_{\geq 1}$. **Propositional logic** (PL) is the fragment of ML without diamonds $\Diamond$. The logics ML + GC, ML + G, PL + GC and PL + G are defined in the expected way.

Example 3. The property 'no vertex is a dead-end' is expressed by the GML + GC-formula $\langle \mathsf{G} \rangle_{=0} \Diamond_{=0} \top$. No GML + GC-formula can express the property 'at least half of the vertices in the graph have label p'. A formal proof via graded bisimulations is straightforward.

2.4 Equivalence of Vertex Classifiers

A **(vertex) property over** Π is a mapping f that assigns to each Π-labeled graph $\mathcal{G}$ a feature map $\lambda' \colon V(\mathcal{G}) \to \{0, 1\}$ and is invariant under isomorphisms. A **vertex classifier** is any object C that defines a vertex property. Note that each of our computing models is a vertex classifier. Each Π-formula φ of any logic introduced above also corresponds to a vertex classifier (where for FO, φ must have a single free variable) which maps each Π-labeled graph $\mathcal{G}$ to the feature map λ_φ with $\lambda_\varphi(v) = 1$ if $\mathcal{G}, v \models \varphi$ and $\lambda_\varphi(v) = 0$ otherwise.

We say that vertex classifiers C_1 and C_2 are **equivalent** if they define the same vertex property. Two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ of vertex classifiers **have the same expressive power** if for each $C_1 \in \mathcal{C}_1$ there is an equivalent $C_2 \in \mathcal{C}_2$, and vice versa. We say that $\mathcal{C}_1$ and $\mathcal{C}_2$ **have the same expressive power relative to** FO, if for each vertex property f definable by a formula $\varphi(x) \in$ FO, there is some $C_1 \in \mathcal{C}_1$ that defines f if and only if there is some $C_2 \in \mathcal{C}_2$ that defines f.

3 Characterizing Real-Based Transformers

We provide characterizations of the expressive power of GPS-networks and of GTs, over the reals and relative to FO, in terms of the logics GML + G and PL + G, respectively. The characterizations apply to both soft-attention and average hard-attention. We start with GPS-networks.

Theorem 4. *Relative to* FO*, the following have the same expressive power:* GML + G*, soft-attention* GPS*-networks, and average hard-attention* GPS*-networks.*

We discuss Theorem 4 before giving a proof. An interesting comparison is to the results of Barceló et al. (2020), who prove that relative to FO, GNNs without transformer layers and without global readout have the same expressive power as GML. They also show that,

when counting global readout is admitted, GNNs can express all of GML + GC.[3] Relative to FO, GPS-networks thus sit properly in the middle between GNNs and GNN + GCs: they can express global properties such as P_1 = 'the graph contains a vertex labeled p', but cannot express absolute global counting, such as P_2 = 'the graph contains at least 2 vertices labeled p'.

Let us also discuss absolute expressive power, dropping FO as a background logic. GNNs are by definition a special case of GPS-networks. Conversely, Property P_1 witnesses that GPS-networks are strictly more expressive than GNNs, also in an absolute sense. Likewise, Property P_2 shows that some GNN + GCs do not have an equivalent GPS-network. It remains open whether every GPS-network is equivalent to a GNN + GC. It is proved in (Rosenbluth et al., 2024) that this is not the case for graph classification, but the result does not immediately transfer to vertex classification.

While GPS-networks cannot express global properties that involve absolute counting, they can express global properties with relative counting, and so can GTs. This is not visible in Theorem 4 because such properties do not fall within FO. We demonstrate relative counting in the example below.

Example 5. There is a 1-layer soft-attention GT (P, L, C) that accepts precisely the vertices of those graphs that satisfy the property: 'at least half of the vertices in the graph have the label p'. The initial MLP P maps each feature vector to a 2-dimensional feature vector, where the first component encodes the labeling by p and the other is 0. The soft-attention module of L then has $W_Q = W_K = [0, 0]^\mathsf{T}$, $W_V = [1, 0]^\mathsf{T}$ and $W_O = [0, 1]$ and the MLP outputs a zero matrix. After the last skip connection, the second column of the matrix consists of the value x, where x tells the ratio of how many vertices have the label p; the final classification head C outputs 1 if $x \geq 0.5$ and 0 otherwise.

We now survey the proof of Theorem 4, full details are in Appendix B. The easier direction is to show that every GML + G-formula can be translated into an equivalent GPS-network. We extend the corresponding construction of (Barceló et al., 2020) for GML, using self-attention heads to handle subformulae of the form $\langle \mathsf{G} \rangle \varphi$.

Lemma 6. *For every* GML + G*-formula, there is an equivalent* GPS*-network. This applies to both soft-attention and average hard-attention.*

A notable difference to the proof of (Barceló et al., 2020) is that we use a step function as an activation function, rather than truncated ReLU. Intuitively, this is because truth values are represented as 0 and 1 in feature vectors, but both soft-attention and average hard-attention may deliver an arbitrarily small (positive) fractional value and there seems to be no way to 'rectify' this into a 1 without using a step function. This is similar to what happens in GNNs that use arithmetic mean as the aggregation function (Schönherr and Lutz, 2026).

The difficult direction in the proof of Theorem 4 is to show that every GPS-network that expresses an FO-property is equivalent to a GML + G-formula. In (Barceló et al., 2020), this direction is proved by first showing that GNNs are invariant under graded bisimulation and then applying a van Benthem/Rosen-style result from finite model theory (Otto, 2019) which says that every FO-formula invariant under graded bisimulation is equivalent to

[3] They actually show that GNNs with counting global readout capture all of C_2—the two-variable fragment of FO with counting quantifiers—but only on undirected graphs; this fails for directed graphs, as GNN + GCs cannot express, for instance, the C_2-formula $\exists y\, E(x, y) \wedge E(y, x)$.

a GML-formula. GPS-networks, however, are not invariant under graded bisimulations because these do not preserve global properties. We thus introduce a stronger version of graded bisimilarity that also takes into account the multiplicities with which graded bisimulation types are realized, and prove a corresponding van Benthem/Rosen theorem.

Let Π be a finite set of vertex label symbols. A **graded bisimulation** between two Π-labeled graphs $\mathcal{G}_1 = (V_1, E_1, \lambda_1)$ and $\mathcal{G}_2 = (V_2, E_2, \lambda_2)$ is a binary relation $Z \subseteq V_1 \times V_2$ that satisfies the following conditions:

atom for all $(v_1, v_2) \in Z$, $\lambda_1(v_1) = \lambda_2(v_2)$.

graded forth for all pairs $(u_1, u_2) \in Z$, for all $k \geq 1$: for all pairwise distinct vertices $v_1, \ldots, v_k \in \mathrm{Neigh}_{\mathcal{G}_1}(u_1)$ there are pairwise distinct vertices $v'_1, \ldots, v'_k \in \mathrm{Neigh}_{\mathcal{G}_2}(u_2)$ with $(v_1, v'_1), \ldots, (v_k, v'_k) \in Z$.

graded back for all pairs $(u_1, u_2) \in Z$, for all $k \geq 1$: for all pairwise distinct vertices $v'_1, \ldots, v'_k \in \mathrm{Neigh}_{\mathcal{G}_2}(u_2)$ there are pairwise distinct $v_1, \ldots, v_k \in \mathrm{Neigh}_{\mathcal{G}_1}(u_1)$ with $(v_1, v'_1), \ldots, (v_k, v'_k) \in Z$.

We write $(\mathcal{G}_1, v_1) \sim (\mathcal{G}_2, v_2)$ if there is a graded bisimulation Z between $\mathcal{G}_1$ and $\mathcal{G}_2$ with $(v_1, v_2) \in Z$.

A **graded bisimulation type over** Π is a maximal set t of Π-labeled pointed graphs such that $(\mathcal{G}_1, v_1) \sim (\mathcal{G}_2, v_2)$ for all $(\mathcal{G}_1, v_1), (\mathcal{G}_2, v_2) \in t$. For a pointed Π-labeled graph $(\mathcal{G}, v)$, we use $\mathsf{tp}_{\mathcal{G}}(v)$ to denote the unique graded bisimulation type t over Π such that $(\mathcal{G}, v) \in t$.

Pointed graphs $(\mathcal{G}_1, v_1)$, $(\mathcal{G}_2, v_2)$ are **global-ratio graded bisimilar**, written $(\mathcal{G}_1, v_1) \sim_{G\%} (\mathcal{G}_2, v_2)$, if $(\mathcal{G}_1, v_1) \sim (\mathcal{G}_2, v_2)$ and there exists a rational number $q > 0$ such that for every graded bisimulation type t,

$$|\{v \in V_1 \mid \mathsf{tp}_{\mathcal{G}_1}(v) = t\}| = q \cdot |\{v \in V_2 \mid \mathsf{tp}_{\mathcal{G}_2}(v) = t\}|.$$

Note that the ratios between graded bisimulation types above are closely related to relative counting as in Example 5.

Example 7. We illustrate that global-ratio graded bisimilarity provides a middle ground between graded bisimilarity and isomorphism.

First consider the $\{p\}$-labeled graphs $\mathcal{G}_1 = (\{v_1\}, \emptyset, \lambda_1)$ such that $\lambda_1(v_1) = \{p\}$ and $\mathcal{G}_2 = (\{u_1, u_2\}, \emptyset, \lambda_2)$ such that $\lambda_2(u_1) = \{p\}$ and $\lambda_2(u_2) = \emptyset$. Then $(\mathcal{G}_1, v_1) \sim (\mathcal{G}_2, u_1)$, but $(\mathcal{G}_1, v_1) \not\sim_{G\%} (\mathcal{G}_2, u_1)$ as the graded bisimulation type of $(\mathcal{G}_2, u_2)$ does not occur in $\mathcal{G}_1$.

Now consider the graph $\mathcal{G}_3 = (\{w_1, w_2\}, \emptyset, \lambda_3)$ with $\lambda_3(w_1) = \lambda_3(w_2) = \{p\}$. Then $(\mathcal{G}_1, v_1) \sim_{G\%} (\mathcal{G}_3, w_1)$, taking $q = \frac{1}{2}$, but $\mathcal{G}_1$ and $\mathcal{G}_3$ are not isomorphic.

A vertex classifier such as a GPS-network or an FO-formula $\varphi(x)$ is **invariant under** $\sim_{G\%}$ if for all pointed graphs $(\mathcal{G}_1, v_1)$ and $(\mathcal{G}_2, v_2)$, $(\mathcal{G}_1, v_1) \sim_{G\%} (\mathcal{G}_2, v_2)$ implies that $\mathcal{G}_1 \models \varphi(v_1)$ if and only if $\mathcal{G}_2 \models \varphi(v_2)$. A layer-by-layer analysis of GPS-networks shows the following.

Lemma 8. *Let N be a soft-attention or average hard-attention* GPS*-network. Then N is invariant under $\sim_{G\%}$.*

It follows that GPS-networks cannot distinguish (G_1, v_1) and (G_3, w_1) from Example 7. In contrast, (G_1, v_1) and (G_2, u_1) can be distinguished: by Lemma 6, one can employ a GPS-network equivalent to the GML + G-formula $\langle \mathsf{G} \rangle \neg p$.

We now give the announced van Benthem/Rosen theorem.

Theorem 9. *For every* FO*-formula* $\varphi(x)$ *over* Π*, the following are equivalent:*

1. φ *is invariant under* $\sim_{G\%}$*;*
2. φ *is equivalent to a* GML + G*-formula over all (finite!)* Π*-labeled pointed graphs.*

The direction "2 ⇒ 1" of Theorem 9 is easy to prove by a straightforward induction on the structure of GML + G-formulae. The difficult part is to show the "1 ⇒ 2" direction, that is, every FO formula $\varphi(x)$ invariant under $\sim_{G\%}$ is equivalent to a GML + G-formula. To achieve this, we combine and extend techniques from (Otto, 2004) and (Otto, 2019). The former provides a van Benthem/Rosen theorem that links global (ungraded) bisimulation to ML + G, and the latter a theorem of the same kind that links graded bisimulation to GML.

Our proof consists of a sequence of results saying that if an FO-formula $\varphi(x)$ is invariant under $\sim_{G\%}$, then it is also invariant under certain other notions of bisimulation that become increasingly weaker.[4] We finally arrive at a notion of bisimulation that is called global c-graded ℓ-bisimulation, where c is a counting bound and ℓ a depth bound, both derived from $\varphi(x)$. Importantly, this notion of bisimulation has only a finite number of bisimulation types, and each type can be distinguished from the others using a characteristic GML + G-formula. We can thus construct the desired GML + G-formula by taking the disjunction of all characteristic formulae for bisimulation types in which $\varphi(x)$ is true. To make sure that the ratio-property of $\sim_{G\%}$ is respected, we replace several constructions from (Otto, 2004) with more careful ones.

Combining Theorem 9 and Lemma 8 completes the proof sketch of Theorem 4. Moreover, as a special case, the construction in the proof of Lemma 6 shows that PL + G-formulae can be translated into GTs with both soft-attention and average hard-attention. A minor extension of our techniques used to prove Theorem 9, then shows the following.

Theorem 10. *Relative to* FO*, the following have the same expressive power:* PL + G*, soft-attention* GT*s, average hard-attention* GT*s.*

4 Characterizing Float-Based Transformers

We give characterizations of GPS-networks and GTs based on floating-point numbers via the logics GML + GC and PL + GC, respectively. Before the characterizations, we introduce floats and float-based GPS-networks and GTs.

4.1 Floating-Point Numbers and Arithmetic

We define the concepts of floating-point numbers based on the IEEE 754 standard (IEEE, 2019). Let $p, q \in \mathbb{Z}_+$. A **floating-point number** (over p and q) is a string of the form

$$b_0 b_1 \cdots b_{p+q} \in \{0,1\}^{p+q+1}.$$

[4] While this provides a good intuition, it is not strictly true. For technical reasons, the intermediate notions of bisimulation sometimes get stronger in certain respects, e.g. they may use up-and-down features as used for modal logics with the converse modality.

The bit b_0 is called the **sign**, the string $\mathbf{e} = b_1 \cdots b_q$ the **exponent** and $\mathbf{s} = b_{q+1} \cdots b_{p+q}$ the **significand**. Let $a = 2^{p-1}$, $b = 2^{q-1} - 1$, and let e and s be the non-negative integers represented in binary by $\mathbf{e}$ and $\mathbf{s}$. Then the above floating-point number is interpreted as the real number $(-1)^{b_0} \frac{s}{a} 2^{e-b}$. As an exception, the float with $\mathbf{s} = 0^p$, $\mathbf{e} = 1^q$ and $b_0 = 0$ (resp. $b_0 = 1$) corresponds to ∞ (resp. $-\infty$). When the context is clear, we identify a float with the real number (or ∞, $-\infty$) that it represents. A float is **normalized** if $b_{q+1} \neq 0$, and **subnormalized** if $b_{q+1} = 0$ and $\mathbf{e} = 0^q$. A **floating-point format** $\mathcal{F}(p, q)$ over p and q consists of all normalized and subnormalized floating-point numbers over p and q and the symbols ∞, $-\infty$, and NaN ('not-a-number'), and when clear we may write $\mathcal{F}$ instead of $\mathcal{F}(p, q)$.

Next, we discuss basic arithmetic operations over floating-point formats: addition $+$, subtraction $-$, multiplication $\cdot$, division $\div$ and square root $\sqrt{x}$. The definition of each of the operations is to first "compute" to unlimited precision in real arithmetic (extended with ∞ and $-\infty$) and then rounding to the nearest float in the format, with ties rounding to the float with an even least significant bit. Undefined results, such as $\frac{\infty}{\infty}$, are mapped to NaN. If any input is NaN, the output is NaN, i.e., our NaN is *silent* and propagated through the computation. The exponential function $\exp(x)$ over floats is not a basic operation and is implemented in a standard way, using basic operations, range reductions and polynomial approximations. Background and a discussion on these concepts is in Appendix C.1.

4.2 Float-Based Transformers

We introduce float-based GTs, GPS-networks and GNNs. To define them, we replace reals with floats, but we must also carefully specify how float operations are performed. One reason is that many float operations (e.g. sum) are not associative due to rounding errors between operations. Thus, switching the order of operations can affect the outcome.

Example 11. Consider the sum of the real numbers -1, 1 and 4 representable in the format $\mathcal{F}(2, 3)$: here we have $(-1 + 1) + 4 = 0 + 4 = 4$ but $-1 + (1 + 4) = -1 + 4 = 3$. Note that in the latter equation, the precise sum of 1 and 4 would be 5, which is not representable in the format $\mathcal{F}(2, 3)$ and is thereby rounded to the nearest number; both 4 and 6 are equally near, and 4 has the even least significant bit.

The softmax function, the sum aggregation function and some matrix multiplications in attention heads take a sum over the features of vertices in the studied graph, and are thus affected by this non-associativity issue. In the worst case, this can violate the isomorphism invariance of these learning models, which is undesirable. For example, in typical real-life implementations, the set V of vertices in the studied graph is associated with some implementation-related, implicit linear order $<^V$ (that is not part of the actual graph). Then isomorphism invariance can be violated if the sum aggregation sums in the order $<^V$. Hence, it is better to order the floats instead of the vertices. We make the natural assumption that floats are always summed in increasing order, which results in models that are isomorphism invariant. This is further justified by numerical stability (Wilkinson, 1959; Robertazzi and Schwartz, 1988; Higham, 1993).

Given a floating-point format $\mathcal{F}$, we let $\mathrm{SUM}_{\mathcal{F}}$ denote the operation that maps a multiset N of floats to the sum $f_1 + \cdots + f_\ell$ where each f_i appears $N(f_i)$ times and the floats appear and are summed in increasing order. We recall from (Ahvonen et al., 2024b) the following important result on *boundedness* of float sums.

Proposition 12. *For all floating-point formats $\mathcal{F}$, there exists a $k \in \mathbb{N}$ such that for all multisets M over floats in $\mathcal{F}$, we have* $\mathrm{SUM}_{\mathcal{F}}(M) = \mathrm{SUM}_{\mathcal{F}}(M_{|k})$.

To see that the above proposition holds, observe that $\mathrm{SUM}_{\mathcal{F}}$ repeatedly adds the same float the number of times it appears in the sum. In the format $\mathcal{F}(2,2)$, the number $\frac{1}{2}\cdot 2^{-1} = 0.25$ is exactly representable. Summing 0.25 repeatedly in this format gives 0.25 after one addition, 0.50 after two additions, 0.75 after three additions, 1.0 after four additions, and 1.0 after five additions, since 1.25 is not in $\mathcal{F}(2,2)$ and rounds to 1.0. Thus, the sum "saturates beyond a threshold". Using this phenomenon, it is easy to obtain a proof of Proposition 12

We say that an aggregation function AGG is **bounded** if there exists a $k \in \mathbb{N}$ such that $\mathrm{AGG}(M) = \mathrm{AGG}(M_{|k})$ for all M. Apart from sum, also mean aggregation is similarly bounded. Furthermore, we assume that softmax is implemented for a floating-point format $\mathcal{F}$ by using the above sum $\mathrm{SUM}_{\mathcal{F}}$ in the denominator, and the remaining operations are carried out in the natural order, i.e., we first calculate the bias b, then values $x_j - b$, then the exponents, and finally the division. Likewise, we assume AH is implemented for $\mathcal{F}$ by calculating the denominator in $\frac{1}{|\mathcal{I}_{\mathbf{x}}|}$ using the same approach as (Li and Cotterell, 2025), i.e., calculating it as $\mathrm{SUM}_{\mathcal{F}}(M)$, where M is the multiset over $\mathcal{F}$ consisting of precisely $|\mathcal{I}_{\mathbf{x}}|$ instances of the float 1, and then performing the division.

Float-Based Learning Models.

Floating-point GT*s, denoted by* GT[F]*, are defined in the same way as* GT*s based on reals, except that they use floats in feature vectors and float operations where the order of operations is as specified above. Likewise for* GPS*-networks,* GNN*s,* MLP*s, etc. We further assume that these models always use aggregation functions that are bounded. This is a natural assumption as sum, max and mean are all bounded on floats by the above findings.*

We call these learning models **simple** when the MLPs are simple[5] and the aggregation functions are $\mathrm{SUM}_{\mathcal{F}}$. In fact, GTs and GPS-networks were originally defined based on simple MLPs (Dwivedi and Bresson, 2020; Rampásek et al., 2022). We do not fix a single float format for all GT[F]s, GPS[F]-networks, GNN[F]s, etc.; instead, each of them is associated with *some* float format. In our translations, the format is assumed arbitrary when translating them into logics, but can be chosen freely in the other direction.

4.3 Characterizations

Next, we provide logical characterizations for GT[F]s and GPS[F]-networks with both soft and average hard-attention. The characterizations are absolute, i.e., they do not require relativizing to a background logic such as FO. Our float-based GT[F]s and GPS[F]-networks also do not require step function activated MLPs aside from the classification heads.

First, we make an observation about float-based multiplication relevant to our translation techniques. When multiplying two floating-point numbers that are very close to zero, **underflow** occurs: the exact result is so small that all significant bits are lost, and the output is 0. For instance, underflow can occur in attention heads in some matrix multiplications. The following proposition demonstrates this phenomenon. In the proposition and the proof that follows, we identify each float with the real number that it represents.

[5]An exception is the final Boolean vertex classifier, which is otherwise a simple MLP but uses the Heaviside function.

Proposition 13. *Let $\mathcal{F}$ be a floating-point format, let f be the smallest positive float in $\mathcal{F}$ and let k be some even integer such that $\frac{k}{2}$ is accurately representable in $\mathcal{F}$. For all $F \in \mathcal{F}$, $|F| \leq |\frac{1}{k}|$ if and only if $F \cdot (\frac{k}{2} f) = 0$.*

For the proof, note that since $\frac{k}{2}$ is accurately representable in $\mathcal{F}$, then so is also the precise product $\frac{k}{2} f$. Now, we observe that $|F| \leq |\frac{1}{k}|$ if and only if the precise product $F \cdot (\frac{k}{2} f)$ belongs to the closed interval $[-\frac{1}{2} f, \frac{1}{2} f]$. All numbers in this interval round to 0.

Now, we give our logical characterization for GT[F]s. Recall that float-based computing models by definition use bounded aggregation functions, and as explained, this is a natural assumption. By 'constant local aggregation functions', we intuitively mean that in message-passing layers, vertices cannot distinguish if a message was received from an out-neighbor or from any other vertex.

Theorem 14. *The following have the same expressive power:* $\mathrm{PL} + \mathrm{GC}$*, soft-attention* GT[F]*s and average hard-attention* GT[F]*s (and* $\mathrm{GNN} + \mathrm{GC}$[F]*s with constant local aggregation functions). This also holds when the* GT[F]*s and* $\mathrm{GNN} + \mathrm{GC}$[F]*s are simple.*

We provide a more detailed proof for Theorem 14 in Appendix D.1, but we *sketch* the proof here. In the direction from GT[F]s to logic, the general idea is that for each vertex v we simulate its feature vector $\mathbf{x}_v$ after each transformer layer by simulating each bit of $\mathbf{x}_v$ by a single formula, i.e., a sequence of formulae simulates the whole vector $\mathbf{x}_v$. As a last step, we combine these formulae recursively into a single formula that simulates the output of the classification head. There are two key insights for simulating bits. First, each 'local step' of a GT[F] where a vertex does not need to know the features of any other vertices (e.g. MLPs or matrix products XW_Q, XW_K and XW_V) can be expressed as a function $f_{\mathcal{F}} \colon \mathcal{F}^n \to \mathcal{F}^m$. As floats are bit strings, we can identify $f_{\mathcal{F}}$ with a *partial* function $f_{\mathbb{B}} \colon \{0,1\}^{kn} \to \{0,1\}^{km}$, where k is the number of bits in $\mathcal{F}$. PL is expressively complete for expressing Boolean combinations, i.e., each function $g \colon \{0,1\}^n \to \{0,1\}$ has an equivalent PL-formula as $g(\mathbf{x})$ is simply a Boolean combination of the values in $\mathbf{x}$. Thus, we can construct an equivalent PL-formula for each output bit of $f_{\mathbb{B}}$, and the full function $f_{\mathbb{B}}$ can be simulated by a sequence of formulae. Second, for the remaining 'non-local' steps, it suffices to know the features of other vertices in the 'global sense', i.e., the edges of the graph are not used. Due to Proposition 12, the float sums appearing in attention heads are bounded for some k, i.e., after k copies of a float F, further instances of F do not affect the sum. Since the attention heads sum over the features of all vertices, it suffices for a vertex to be able to distinguish a bounded number of each possible feature vector appearing in the graph, and we can count up to this bound with the counting global modality.

For the converse, to translate a $\mathrm{PL} + \mathrm{GC}$-formula φ into a simple GT[F], we use a similar strategy as with reals: we compute the truth values of the subformulae of φ one at a time, using multiple transformer layers per subformula. Again the truth value of each subformula is represented as 0 or 1 in the feature vector for each vertex, i.e., in the feature matrix there is a column for each subformula that is used to encode the truth value of the subformula in each vertex. There are also some auxiliary columns in the feature matrix. The translation involves making use of the properties of floats and floating-point operations. The operators $\neg$ and $\wedge$ are easy to handle by using the MLPs of the transformer layers. The hardest part is to simulate modalities $\langle \mathsf{G} \rangle_{\geq k}$ by using MLPs *and* attention modules. Assume that we are computing the truth value of a subformula of the form $\langle \mathsf{G} \rangle_{\geq k} \psi$ and the truth value of ψ has already been encoded into a column i of the current feature matrix. Then we build an attention head that checks if the number ℓ of 1s in the column i is at least k.

Intuitively, we construct a query matrix W_Q and a key matrix W_K such that the matrix $\mathrm{softmax}\big((XW_Q)(XW_K)^{\mathsf{T}}/\sqrt{d_h}\big)$ has an entry in every row that has the value $\frac{1}{\ell}$ (or a suitable rounded value). A similar construction is also possible by using the average-hard function. Then due to Proposition 13, we can construct a value matrix which uses underflow to check if $\frac{1}{\ell} \leq \frac{1}{k}$. After that, by using MLPs, we can distinguish when $\ell \geq k$ and when $\ell < k$. This completes the proof sketch.

Before characterizing GPS[F]-networks, we prove a helpful characterization of float-based GNNs.

Theorem 15. *The following pairs have the same expressive power (denoted by $\equiv$):*

$$\mathrm{GNN[F]} \equiv \mathrm{GML}, \mathrm{GNN} + \mathrm{G[F]} \equiv \mathrm{GML} + \mathrm{G} \text{ and } \mathrm{GNN} + \mathrm{GC[F]} \equiv \mathrm{GML} + \mathrm{GC}.$$

This also holds when each type of GNN[F] *is simple.*

This theorem follows from the proof techniques of Theorem 3.2 of (Ahvonen et al., 2024b), which showed that (simple) recurrent float GNNs are equally expressive as a recursive rule-based bisimulation invariant logic called the graded modal substitution calculus (GMSC). Unlike the float GNNs in that paper, our GNN[F]s are not recurrent, meaning they only scan the neighborhood of a vertex up to some fixed depth. The corresponding constant-iteration fragment of GMSC is GML. The techniques in (Ahvonen et al., 2024b) generalize for global readouts and modalities, see Appendix D.2 for the technical details of the proof.

We now characterize float-based GPS-networks.

Theorem 16. *The following have the same expressive power:* $\mathrm{GML} + \mathrm{GC}$*, soft-attention* GPS[F]*-networks, average hard-attention* GPS[F]*-networks and* $\mathrm{GNN} + \mathrm{GC[F]}$*s. This also holds in the case where the* GPS[F]*-networks and* $\mathrm{GNN} + \mathrm{GC[F]}$*s are simple.*

The result follows from Theorems 14 and 15. Importantly, any transformer layer and message-passing layer can be simulated by a GPS-layer of a higher dimension by appending the inputs and outputs of the transformer and message-passing layer with zeros on the GPS side. The technical details of the proof are in Appendix D.3.

We make some final observations. As seen in Example 5, 'relative global counting' is expressible by real-based GTs. However, the same construction does not work for GT[F]s as, due to Proposition 12, the softmax-function and average hard function lose accuracy in a drastic way with large graphs. For the same reason, Lemma 8 fails with floats; GT[F]s and GPS[F]-networks are not invariant under the bisimilarity $\sim_{G\%}$. However, Theorems 14, 15 and 16 show that with float-based GTs, GPS-networks and GNNs, 'absolute counting' is possible (locally or globally depending on the model), since the matching logics can count.

We note that our results with floats immediately hold when restricted to word-shaped graphs, i.e., graphs where the domain is a prefix $[n]$ of positive integers, the edge relation is the successor relation over $[n]$, and $\lambda_{\mathbf{w}}(v)$ is a singleton for each vertex. For example, a GT over word-shaped graphs is just an 'encoder-only transformer without causal masking'. A popular example is BERT (Devlin et al., 2019), which is such a model inspired by (Vaswani et al., 2017). In Appendix E.1, we study unique hard-attention graph transformers over word-shaped graphs. In Appendix D.4, we also consider generalizations of our float results for graph classification tasks and non-Boolean classification tasks.

We also briefly discuss how our float results could be modified to cover positional encodings. Often, each GNN[F], GT[F], or GPS[F]-network $A = (P, L^{(1)}, \ldots, L^{(k)}, C)$ (with input

dimension ℓ) is associated with a **positional encoding** (or PE) π over $\mathcal{F}$, i.e., a mapping that assigns to each graph $\mathcal{G}$ a function $\pi(\mathcal{G})\colon V(\mathcal{G}) \to \mathcal{F}^\ell$. For example, a popular PE is LapPE (Rampásek et al., 2022). Now, A with π computes over $\mathcal{G}$ a sequence of feature maps similarly to A without π (see Section 2.2), but for each vertex v in $\mathcal{G}$, we define $\lambda_v^{(0)} := P(\lambda)_v + \pi(\mathcal{G})_v$. Our characterizations with floats can be modified to cover PEs by simply adding proposition symbols to the logic that encode the PE of each vertex, see Appendix E.2 for more details. Positional encodings are an important future topic that warrants further study.

5 Conclusion

We have given logical characterizations for GPS-networks and graph transformers, based on reals and on floats. As future work, it would be interesting to lift all our characterizations from vertex to graph classification, and to more comprehensively study the expressive power of GPS-networks and GTs enriched with common forms of positional encodings such as graph Laplacians. Our results in the float case in fact already lift to graph classification tasks and also to non-Boolean classification, and they also hold when restricted to word-shaped graphs; this is covered in Appendices D.4 and E. Another interesting open question is whether, in the case of the reals, every GPS-network can be expressed as a GNN + GC.

6 Acknowledgments

Veeti Ahvonen was supported by the Vilho, Yrjö and Kalle Väisälä Foundation. Damian Heiman was supported by the Magnus Ehrnrooth Foundation. Antti Kuusisto was supported by the project *Perspectives on computational logic*, funded by the Research Council of Finland, project number 369424. Carsten Lutz was supported by DFG project LU 1417/4-1.

References

Veeti Ahvonen, Damian Heiman, and Antti Kuusisto. Descriptive Complexity for Neural Networks via Boolean Networks. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:22, Dagstuhl, Germany, 2024a. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-310-2. doi: 10.4230/LIPIcs.CSL.2024.9.

Veeti Ahvonen, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Logical characterizations of recurrent graph neural networks with reals and floats. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024b. URL `http://papers.nips.cc/paper_files/paper/2024/hash/bca7a9a0dd85e2a68420e5cae27eccfb-Abstract-Conference.html`.

Veeti Ahvonen, Maurice Funk, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Expressive power of graph transformers via logic, 2025. URL `https://arxiv.org/abs/2508.01067v1`.

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Pablo Barceló, Alexander Kozachinskiy, Anthony Widjaja Lin, and Vladimir V. Podolskii. Logical languages accepted by transformer encoders with hard attention. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=gbrHZq07mq.

Michael Benedikt, Chia-Hsuan Lu, Boris Motik, and Tony Tan. Decidability of graph neural networks via logical characterizations. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 127:1–127:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi: 10.4230/LIPICS.ICALP.2024.127. URL https://doi.org/10.4230/LIPIcs.ICALP.2024.127.

Pierre Blanchard, Desmond J. Higham, and Nicholas J. Higham. Accurate computation of the log-sum-exp and softmax functions, 2019. URL https://arxiv.org/abs/1909.03469.

David Chiang. Transformers in uniform tc^0, 2025. URL https://arxiv.org/abs/2409.13629.

David Chiang, Peter Cholak, and Anand Pillay. Tighter bounds on the expressivity of transformer encoders. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of ICML 2023*, volume 202 of *Proceedings of Machine Learning Research*, pages 5544–5562. PMLR, 2023.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *CoRR*, abs/2012.09699, 2020. URL https://arxiv.org/abs/2012.09699.

Martin Grohe. The descriptive complexity of graph neural networks. In *LICS*, pages 1–14, 2023. doi: 10.1109/LICS56636.2023.10175735.

Yiding Hao, Dana Angluin, and Robert Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Trans. Assoc. Comput. Linguistics*, 10:800–810, 2022. doi: 10.1162/TACL_A_00490.

Stan P. Hauke and Przemyslaw Andrzej Walega. Aggregate-combine-readout gnns are more expressive than logic C2. *CoRR*, abs/2508.06091, 2025. doi: 10.48550/ARXIV.2508.06091. URL https://doi.org/10.48550/arXiv.2508.06091.

William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002. ISSN 0022-0000. doi: https://doi.org/10.1016/S0022-0000(02)00025-9. URL https://www.sciencedirect.com/science/article/pii/S0022000002000259. Special Issue on Complexity 2001.

Nicholas J. Higham. The accuracy of floating point summation. *SIAM J. Sci. Comput.*, 14 (4):783–799, 1993. doi: 10.1137/0914050.

IEEE. Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019. doi: 10.1109/IEEESTD.2019.8766229.

Reijo Jaakkola, Antti Kuusisto, and Miikka Vilander. Relating description complexity to entropy. *Journal of Computer and System Sciences*, 149:103615, 2025. ISSN 0022-0000. doi: https://doi.org/10.1016/j.jcss.2024.103615. URL https://www.sciencedirect.com/science/article/pii/S0022000024001107.

Selim Jerad, Anej Svete, Jiaoda Li, and Ryan Cotterell. Unique hard attention: A tale of two sides, 2025. URL https://arxiv.org/abs/2503.14615.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 21618–21629, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/b4fd1d2cb085390fbbadae65e07876a7-Abstract.html.

Jiaoda Li and Ryan Cotterell. Characterizing the expressivity of transformer language models, 2025. URL https://arxiv.org/abs/2505.23623.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18)*, pages 3538–3545. AAAI Press, 2018. doi: 10.1609/AAAI.V32I1.11604.

William Merrill and Ashish Sabharwal. A logic for expressing log-precision transformers. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/a48e5877c7bf86a513950ab23b360498-Abstract-Conference.html.

Peter Nilsson, Ateeq Ur Rahman Shaik, Rakesh Gangarajaiah, and Erik Hertz. Hardware implementation of the exponential function using taylor series. In *2014 NORCHIP*, pages 1–4, 2014. doi: 10.1109/NORCHIP.2014.7004740.

Martin Otto. Modal and guarded characterisation theorems over finite transition systems. *Annals of Pure and Applied Logic*, 130(1-3):173–205, 2004. doi: 10.1016/j.apal.2004.04.003.

Martin Otto. Graded modal logic and counting bisimulation. *CoRR*, abs/1910.00039, 2019. URL http://arxiv.org/abs/1910.00039.

Maximilian Pfluger, David Tena Cucala, and Egor V. Kostylev. Recurrent graph neural networks and their connections to bisimulation and logic. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(13):14608–14616, 2024. doi: 10.1609/aaai.v38i13.29377.

Ladislav Rampásek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/5d4834a159f1547b267a05a4e2b7cf5e-Abstract-Conference.html.

Thomas G. Robertazzi and Stuart C. Schwartz. Best "ordering" for floating-point addition. *ACM Trans. Math. Softw.*, 14(1):101–110, 1988. doi: 10.1145/42288.42343.

Eran Rosenbluth, Jan Tönshoff, Martin Ritzert, Berke Kisin, and Martin Grohe. Distinguished in uniform: Self-attention vs. virtual nodes. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=AcSChDWL6V.

Marco Sälzer, François Schwarzentruber, and Nicolas Troquard. Verifying quantized graph neural networks is pspace-complete. *CoRR*, abs/2502.16244, 2025. doi: 10.48550/ARXIV.2502.16244. URL https://doi.org/10.48550/arXiv.2502.16244.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Moritz Schönherr and Carsten Lutz. Logical characterizations of gnns with mean aggregation. In *Proc. of AAAI*. AAAI Press, 2026.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? A survey. *Trans. Assoc. Comput. Linguistics*, 12:543–561, 2024. doi: 10.1162/TACL_A_00663. URL https://doi.org/10.1162/tacl_a_00663.

Ping Tak Peter Tang. Table-driven implementation of the exponential function in IEEE floating-point arithmetic. *ACM Trans. Math. Softw.*, 15(2):144–157, 1989. doi: 10.1145/63522.214389. URL https://doi.org/10.1145/63522.214389.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

James Hardy Wilkinson. Rounding errors in algebraic processes. In *Information Processing, Proceedings of the 1st International Conference on Information Processing, UNESCO, Paris 15-20 June 1959*, pages 44–53. UNESCO (Paris), 1959.

Andy Yang and David Chiang. Counting like transformers: Compiling temporal counting logic into softmax transformers. In *First Conference on Language Modeling*, 2024. URL `https://openreview.net/forum?id=FmhPg4UJ9K`.

Andy Yang, David Chiang, and Dana Angluin. Masked hard-attention transformers recognize exactly the star-free languages. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

Andy Yang, Michaël Cadilhac, and David Chiang. Knee-deep in c-rasp: A transformer depth hierarchy, 2025. URL `https://arxiv.org/abs/2506.16055`.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper_files/paper/2019/file/9d63484abb477c97640154d40595a3bb-Paper.pdf`.

A Preliminaries

A.1 Word-shaped graphs

For a word $\mathbf{w} = w_1 \cdots w_n \in \Pi^+$, the **word-shaped** graph of $\mathbf{w}$ is $\mathcal{G}_\mathbf{w} = (V_\mathbf{w}, E_\mathbf{w}, \lambda_\mathbf{w})$ where $V_\mathbf{w} = [n]$, $E_\mathbf{w}$ is the successor relation over $[n]$, and $\lambda_\mathbf{w}(i) = \{w_i\}$ for all $i \in V_\mathbf{w}$.

When graph transformers are restricted to word-shaped graphs, a GT becomes an ordinary "encoder-only transformer without causal masking". For example, the popular BERT (Devlin et al., 2019) inspired by (Vaswani et al., 2017) is such a model.

A.2 Transformers with unique hard-attention

In Appendices D and E, we also study transformers that use unique hard-attention instead of soft-attention or average hard-attention. Given $\mathbf{x} \in \mathbb{R}^p$, we let $\mathcal{I}_\mathbf{x} = \{\, i \in [p] \mid \mathbf{x}_i = \max(\mathbf{x}) \,\}$. The unique hard function $\mathrm{UH}\colon \mathbb{R}^+ \to^{|\cdot|} \mathbb{R}^+$ is defined by

$$\mathrm{UH}(\mathbf{x})_i := \begin{cases} 1, & \text{if } i \in \min(\mathcal{I}_\mathbf{x}) \\ 0, & \text{otherwise.} \end{cases}$$

For example, given $\mathbf{x} = (-1, 5, 10, 0, 10, 5)$, we have $\mathrm{UH}(\mathbf{x}) = (0, 0, 1, 0, 0, 0)$. For the implementation of UH with floats over a floating-point format $\mathcal{F}$, we assume the function is exactly the same, i.e., the leftmost component with the maximum value $\max(\mathbf{x})$ is given the floating-point value $1 \in \mathcal{F}$ and others are given $0 \in \mathcal{F}$. Analogously to the average hard function and the softmax function, attention heads that use the unique hard function are called unique hard-attention heads. The same naming applies to modules, graph transformers and GPS-networks.

Based on the literature, graph transformers and GPS-networks that use unique hard-attention are not really used in real-life applications. One of the reasons is that they are not invariant under isomorphism.

Proposition 17. *There is a unique hard-attention* GT *(and* GT[F]*) and a unique hard-attention* GPS*-network (and* GPS[F]*-network) that is not isomorphism invariant.*

Proof. Let $\Pi = \{p\}$. Consider the following isomorphic Π-labeled graphs:

- $\mathcal{G}_1 = ([2], \emptyset, \lambda_1)$, where $\lambda_1(1) = \{p\}$ and $\lambda_1(2) = \emptyset$,
- $\mathcal{G}_2 = ([2], \emptyset, \lambda_2)$, where $\lambda_2(1) = \emptyset$ and $\lambda_2(2) = \{p\}$.

Now, consider a unique hard-attention head H with $W_Q = W_K = [0]$ and $W_V = [1]$. Now, $H(X) = X$ (with reals and floats). By using H, it is easy to design a GT G such that $G(\mathcal{G}_1) = \lambda_1$ and $G(\mathcal{G}_2) = \lambda_2$ even though $\mathcal{G}_1$ and $\mathcal{G}_2$ are isomorphic. An analogous result applies for GT[F]s, GPS-networks and GPS[F]-networks. □

However, over word-shaped graphs (or simply over words), transformers based on unique hard-attention preserve isomorphism invariance, and their theoretical properties, such as expressive power, have been studied, for example, in (Hao et al., 2022; Barceló et al., 2024; Yang et al., 2024). Moreover, the unique hard-attention function could be used with vertex-ordered graphs (but note that we do not consider such graphs in this paper).

A.3 Graph transformers and graph neural networks

In the main body of this paper (i.e., outside of the appendix), we defined our graph neural networks and graph transformers as Boolean vertex classifiers. In a more general sense, we could allow our computing models to have an arbitrary (finite) output dimension. A graph transformer $(P, L^{(1)}, \ldots, L^{(k)}, C)$ **of I/H/O dimension** (p, d, q) (over the set Π of vertex label symbols) refers to a graph transformer where P is an MLP of I/O dimension (p, d) and each $L^{(i)}$ is a transformer layer d as defined in the main body of this paper and C is an MLP (possibly not ReLU-activated) of I/O dimension (d, q). Analogously, we define GPS-networks **of I/H/O dimension** (p, d, q) over Π and GNNs **of I/H/O dimension** (p, d, q) over Π. For brevity, we can exclude Π and leave it implicit when it is clear from the context. For example, when comparing two GPS-networks with the same input dimension, we can assume that they are defined over the same set of vertex label symbols.

B Proofs for Section 3

B.1 Proof of Lemma 6

A **basic** GPS**-layer** of dimension d is a tuple $G = (\sigma, A, C, b, H)$, where $\sigma\colon \mathbb{R} \to \mathbb{R}$ is an activation function, $A, C \in \mathbb{R}^{d \times d}$ are matrices, b is a $d \times 1$ matrix, and H is a self-attention head of I/H dimension (d, d). Analogously to GPS-layers as defined in Section 2, G computes the output feature map λ' based on an input feature map λ as follows: for each $v \in V$,

$$\lambda'(v) := \sigma\Big(\lambda(v)C + \big(\sum_{v' \in \mathrm{Neigh}_{\mathcal{G}}(v)} \lambda(v')\big)A + H(\lambda)_{v,*} + b\Big).$$

A **basic** GPS-network is a tuple $(P, L^{(1)}, \ldots, L^{(k)}, C)$ that is defined like a GPS-network, except all layers are basic GPS-layers. Below, we assume that both GPS-networks are defined over the same set Π of vertex label symbols.

Lemma 18. *For every basic* GPS*-network G of I/H/O dimension $(p, d, 1)$, there is a (non-basic)* GPS*-network $\hat{G}$ of I/H/O dimension $(p, 2d, 1)$ such that for all labeled graphs $\mathcal{G}$, $G(\mathcal{G}) = \hat{G}(\mathcal{G})$.*

Proof. Let $G = (P, L^{(1)}, \ldots, L^{(k)}, C)$ be a basic GPS-network of I/H/O dimension $(p, d, 1)$. In constructing an equivalent non-basic GPS-network $\hat{G}$, the main challenge is dealing with the skip-connections. For this, we use d additional hidden dimensions and maintain that the feature maps computed by $\hat{G}$ contain the feature maps computed by G in the first d dimensions, and 0 in the other d dimensions. The equivalent non-basic GPS-network is then $\hat{G} = (\hat{P}, \hat{L}^{(1)}, \ldots, \hat{L}^{(k)}, \hat{C})$, where

- $\hat{P}$ is obtained from P by adding d output dimensions that have the value 0,
- $\hat{L}^{(i)}$ is obtained from $L^{(i)}$ by
  - Constructing $\mathrm{MP}^{(i)}$ to output 0 in the first d dimensions and also to output $\lambda^{(i)}(v)C + \sum_{v' \in \mathrm{Neigh}_{\mathcal{G}}(v)} \lambda^{(i)}(v')A + b$ in the second d dimensions, by choosing sum as the aggregation and combination functions.
  - Constructing $\mathrm{SA}^{(i)}$ to output 0 in the first d dimensions and $H(\lambda^{(i)})_v$ in the second d dimensions by using H as a single attention head.
  - Constructing $\mathrm{FF}^{(i)}$ such that, on input $\mathbf{x}_1$ in the first d dimension and $\mathbf{x}_2$ in the second d dimensions, outputs $\sigma(\mathbf{x}_2) - \mathbf{x}_1$ in the first d dimensions and 0 in the second d dimensions.
- $\hat{C}$ is obtained from C by adding d input dimensions, which are ignored. □

Lemma 6. *For every* GML + G*-formula, there is an equivalent* GPS*-network. This applies to both soft-attention and average hard-attention.*

Proof. Let Π be a finite set of vertex label symbols and let φ be a GML + G formula over Π. For constructing a GPS-network that is equivalent to φ, we extend the construction of (Barceló et al., 2020) to GML + G, which will result in a basic GPS-network that uses step-activation. Lemma 18 then shows that an equivalent GPS-network exists.

Let $\varphi_1, \ldots, \varphi_d$ be the subformulae of φ ordered such that if φ_i is a subformula of φ_j, then $i \leq j$. Hence, $\varphi_d = \varphi$. Let $p = |\Pi|$. We use a GPS-network $N = (P, L^{(1)}, \ldots, L^{(d)}, C)$ over (Π, d), where $L^{(1)}, \ldots, L^{(d)}$ are d basic GPS-layers. In fact, all basic GPS-layers of N will be exactly identical.[6] We aim to achieve that, for every pointed Π-labeled graph $(\mathcal{G}, v)$ with $\mathcal{G} = (V, E, \lambda)$, on which N computes feature maps $\lambda^{(0)}, \ldots, \lambda^{(d)}$,

(∗) $\lambda^{(i)}(u)_j = 1$ if $\mathcal{G}, u \models \varphi_j$ and $\lambda^{(i)}(u)_j = 0$ otherwise, for all i, j with $1 \leq j \leq i \leq d$ and all $u \in V$.

The final classification layer C will then accept only vertices u with $\lambda^{(d)}(u)_d = 1$.

The initial MLP P is of dimension (p, d) and is a projection such that, for every vertex label symbol $s \in \Pi$, $1 \leq j \leq d$ and $u \in V$, $\lambda^{(0)}(u)_j = 1$ if $\varphi_j = s$ and $s \in \lambda(u)$, and $\lambda^{(0)}(u)_j = 0$ otherwise.

The basic GPS-layers are constructed as follows. The activation function σ is the Heaviside

[6] A GNN with all layers identical is called *homogeneous* in (Barceló et al., 2020).

step function, that is,

$$\sigma(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0. \end{cases}$$

In the self-attention head H, choose $\alpha = \text{softmax}$, and set all entries in W_Q and W_K to 0. The latter ensures that in the computation of $H(X)$, every vertex pays attention $\frac{1}{|V|}$ to every vertex. Set the entries of A, C, b, and W_V, depending on the subformulae of φ, as follows. In column j,

1. if $\varphi_j \in \Pi$, then set $C_{jj} = 1$,
2. if $\varphi_j = \varphi_k \wedge \varphi_{k'}$, set $C_{k,j} = C_{k',j} = 1$ and $b_j = -1$,
3. if $\varphi_j = \neg\varphi_k$, set $C_{k,j} = -1$ and $b_j = 1$,
4. if $\varphi_j = \Diamond_{\geq c}\varphi_k$, set $A_{k,j} = 1$ and $b_j = -c + 1$,
5. if $\varphi_j = \langle \mathsf{G} \rangle \varphi_k$, set $(W_V)_{k,j} = 1$,

and set all other values of column j in A, C, W_V and b to 0. Using induction on j, one can now show that $(*)$ is satisfied. The arguments for Cases 1 to 4 can be found in (Barceló et al., 2020). Thus, we only explicitly treat Case 5.

Thus, let $\varphi_j = \langle \mathsf{G} \rangle \varphi_k$ and assume that $(*)$ holds for φ_k, that is, $\lambda^{(i)}(u)_k = 1$ if $\mathcal{G}, u \models \varphi_k$ and $\lambda^{(i)}(u)_k = 0$ otherwise, for all i with $k \leq i \leq d$ and all $u \in V$. Now consider any $u \in V$. Recall that

$$\lambda^{(i+1)}(u) := \sigma\Big(\lambda^{(i)}(u)C + \sum_{u' \in \text{Neigh}_{\mathcal{G}}(v)} \lambda^{(i)}(u')A + H(\lambda^{(i)})_{u,*} + b\Big).$$

We are interested in $\lambda^{(i+1)}(u)_j$. As $C_{k,j} = A_{k,j} = b_j = 0$ for all k,

$$\lambda^{(i+1)}(u)_j := \sigma\big(H(\lambda^{(i)})_{u,j}\big).$$

By choice of W_K and W_Q, $H(\lambda^{(i)})$ computes

$$\frac{\sum_{v \in V} \lambda^{(i)}(v)}{|V|} W_V.$$

By $(*)$ all $\lambda^{(i)}(v)$ are either 0 or 1. Hence, by choice of W_V, $H(\lambda^{(i)})_{u,j} > 0$ if and only if there is a vertex $u' \in V$ with $\lambda^{(i)}(u')_k = 1$. By choice of σ and $(*)$ it thus follows that $\lambda^{(i+1)}(u)_j = 1$ if $\mathcal{G}, u \models \langle \mathsf{G} \rangle \varphi_k$ and $\lambda^{(i+1)}(u)_j = 0$ otherwise, as required.

Observe that since all entries of W_Q and W_K are 0, the same argument also applies if one chooses AH as the attention function. □

B.2 Proof of Lemma 8

Lemma 8. *Let N be a soft-attention or average hard-attention* GPS*-network. Then N is invariant under $\sim_{G\%}$.*

Proof. Let $N = (P, L^{(1)}, \ldots, L^{(k)}, C)$ be a GPS-network, and $(\mathcal{G}_1, v_1), (\mathcal{G}_2, v_2)$ Π-labeled pointed graphs with $\mathcal{G}_1 = (V_1, E_1, \lambda_1)$, $\mathcal{G}_2 = (V_2, E_2, \lambda_2)$, such that $(\mathcal{G}_1, v_1) \sim_{G\%} (\mathcal{G}_2, v_2)$.

For $j \in \{1, 2\}$, let $\lambda_j^{(0)}, \ldots, \lambda_j^{(k)}$ be the feature maps of dimension d computed by the layers of N on $\mathcal{G}_j$. To prove the lemma, it suffices to show the following for $0 \leq i \leq k$:

$(*)$ for all $u \in V_1$ and $v \in V_2$, $\mathsf{tp}_{\mathcal{G}_1}(u) = \mathsf{tp}_{\mathcal{G}_2}(v)$ implies $\lambda_1^{(i)}(u) = \lambda_2^{(i)}(v)$.

We prove $(*)$ by induction on i.

For the induction start, where $i = 0$, recall that $\mathsf{tp}_{\mathcal{G}_1}(u) = \mathsf{tp}_{\mathcal{G}_2}(v)$ implies that $(\mathcal{G}_1, u) \sim (\mathcal{G}_2, v)$. Then $(*)$ follows from Condition **atom** of graded bisimulations and the definition of $\lambda_j^{(0)}$, $j \in \{1, 2\}$.

For the induction step, assume that $(*)$ holds for i and let $u \in V_1$ and $v \in V_2$ such that $\mathsf{tp}_{\mathcal{G}_1}(u) = \mathsf{tp}_{\mathcal{G}_2}(v)$. We have to show that $\lambda_1^{(i+1)}(u) = \lambda_2^{(i+1)}(v)$. Recall that, for $j \in \{1, 2\}$,

$$\begin{aligned}
\lambda_{j,B}^{(i+1)} &:= \lambda_j^{(i)} + B^{(i+1)}\big(\lambda_j^{(i)}\big), \text{ where } B \in \{\mathrm{SA}, \mathrm{MP}\},\\
\lambda_{j,\mathrm{SA+MP}}^{(i+1)} &:= \lambda_{j,\mathrm{SA}}^{(i+1)} + \lambda_{j,\mathrm{MP}}^{(i+1)}, \text{ and}\\
\lambda_j^{(i+1)} &:= \lambda_{j,\mathrm{SA+MP}}^{(i+1)} + \mathrm{FF}^{(i+1)}\big(\lambda_{j,\mathrm{SA+MP}}^{(i+1)}\big).
\end{aligned}$$

As $+$ and $\mathrm{FF}^{(i+1)}$ are applied row-wise and $\lambda_1^{(i)}(u) = \lambda_2^{(i)}(v)$ by the induction hypothesis, it suffices to show that for all $u \in V_1$ and $v \in V_2$, $\mathsf{tp}_{\mathcal{G}_1}(u) = \mathsf{tp}_{\mathcal{G}_2}(v)$ implies $\mathrm{MP}^{(i+1)}\big(\lambda_1^{(i)}\big)(u) = \mathrm{MP}^{(i+1)}\big(\lambda_2^{(i)}\big)(v)$ and $\mathrm{SA}^{(i+1)}\big(\lambda_1^{(i)}\big)(u) = \mathrm{SA}^{(i+1)}\big(\lambda_2^{(i)}\big)(v)$.

We begin with $\mathrm{MP}^{(i+1)}$. Recall that, for $j \in \{1, 2\}$,

$$\begin{aligned}
&\mathrm{MP}^{(i+1)}\big(\lambda_j^{(i)}\big)(u) = \mathrm{COM}^{(i+1)}\Big(\lambda_j^{(i)}(u), \gamma_j^{(i)}(u)\Big), \text{ where}\\
&\gamma_j^{(i)}(u) := \mathrm{AGG}^{(i+1)}\big(\{\!\{\lambda_j^{(i)}(u') \mid (u, u') \in E_j\}\!\}\big).
\end{aligned}$$

From the induction hypothesis and the Conditions **graded forth** and **graded back** of graded bisimulations, it follows that

$$\{\!\{\lambda_1^{(i)}(u') \mid (u, u') \in E_1\}\!\} = \{\!\{\lambda_2^{(i)}(v') \mid (v, v') \in E_2\}\!\}.$$

Thus, as $\lambda_1^{(i)}(u) = \lambda_2^{(i)}(v)$ by induction hypothesis, it further follows that

$$\mathrm{MP}^{(i+1)}\big(\lambda_1^{(i)}\big)(u) = \mathrm{MP}^{(i+1)}\big(\lambda_2^{(i)}\big)(v).$$

Now consider the self-attention module $\mathrm{SA}^{(i+1)} = (H^{(1)}, \ldots, H^{(k')}, W_O)$ and recall that, for $j \in \{1, 2\}$, $\mathrm{SA}^{(i+1)}\big(\lambda_j^{(i)}\big) = \mathcal{H}\big(\lambda_j^{(i)}\big)W_O$ where $\mathcal{H}\big(\lambda_j^{(i)}\big)$ is the concatenation of the matrices $H^{(1)}\big(\lambda_j^{(i)}\big), \ldots, H^{(k')}\big(\lambda_j^{(i)}\big)$. It thus suffices to show that for all $\ell \in [k']$, all $u \in V_1$ and $v \in V_2$,

$$\mathsf{tp}_{\mathcal{G}_1}(u) = \mathsf{tp}_{\mathcal{G}_2}(v) \text{ implies } H^{(\ell)}(\lambda_1^{(i)})_{u,*} = H^{(\ell)}(\lambda_2^{(i)})_{v,*}.$$

Thus, let $\ell \in [k']$ and

$$H^{(\ell)}(X) := \alpha\left(\frac{(XW_Q)(XW_K)^{\mathsf{T}}}{\sqrt{d_h}}\right)(XW_V).$$

Consider the matrices

$$A = \frac{(\lambda_1^{(i)}W_Q)(\lambda_1^{(i)}W_K)^{\mathsf{T}}}{\sqrt{d_h}}$$

and

$$B = \frac{(\lambda_2^{(i)}W_Q)(\lambda_2^{(i)}W_K)^{\mathsf{T}}}{\sqrt{d_h}}.$$

To show that $H^{(\ell)}(\lambda_1^{(i)})_{u,*} = H^{(\ell)}(\lambda_2^{(i)})_{v,*}$, we must show that $\big(\alpha(A)\lambda_1^{(i)}\big)_{u,*} = \big(\alpha(B)\lambda_2^{(i)}\big)_{v,*}$, for $\alpha \in \{\mathrm{softmax}, \mathrm{AH}\}$.

For $q > 0$ the rational number that shows global-ratio graded bisimilarity of $(\mathcal{G}_1, v_1)$ and $(\mathcal{G}_2, v_2)$, the induction hypothesis implies that for all $\mathbf{x} \in \mathbb{R}^d$:

$$|\{u' \in V_1 \mid \lambda_1^{(i)}(u') = \mathbf{x}\}| = q \cdot |\{v' \in V_2 \mid \lambda_2^{(i)}(v') = \mathbf{x}\}|. \tag{$\dagger_1$}$$

Furthermore, observe that computing an entry $A_{u,u'}$ depends only on $\lambda_1^{(i)}(u)$ and $\lambda_1^{(i)}(u')$ (and the same holds for B and $\lambda_2^{(i)}$). It then follows from the induction hypothesis that, for all $u' \in V_1$ and $v' \in V_2$, $\mathsf{tp}_{\mathcal{G}_1}(u') = \mathsf{tp}_{\mathcal{G}_2}(v')$ implies $A_{u,u'} = B_{v,v'}$. Hence, for all $a \in \mathbb{R}$,

$$|\{u' \in V_1 \mid A_{u,u'} = a\}| = q \cdot |\{v' \in V_2 \mid B_{v,v'} = a\}|.$$

Therefore, if $\alpha = \mathrm{softmax}$,

$$\sum_{u' \in V_1} e^{A_{u,u'} - b} = q \cdot \sum_{v' \in V_2} e^{B_{v,v'} - b},$$

and, for all vertices $u' \in V_1$ and $v' \in V_2$ with $\mathsf{tp}_{\mathcal{G}_1}(u') = \mathsf{tp}_{\mathcal{G}_2}(v')$,

$$\begin{aligned}
\mathrm{softmax}(A)_{u,u'} &= \frac{e^{A_{u,u'} - b}}{\sum_{u'' \in V_1} e^{A_{u,u''} - b}} \\
&= \frac{e^{B_{v,v'} - b}}{q \cdot \sum_{v'' \in V_2} e^{B_{v,v''} - b}} \\
&= \frac{1}{q} \cdot \mathrm{softmax}(B)_{v,v'}.
\end{aligned} \tag{$\dagger_2$}$$

Let $\mathcal{T}$ be the collection of all graded bisimulation types over Π. We obtain that

$$\begin{aligned}
&(\mathrm{softmax}(A)\lambda_1^{(i)})_{u,*} \\
&\quad = \sum_{u' \in V_1} \mathrm{softmax}(A)_{u,u'} \lambda_1^{(i)}(u') \\
&\quad = \sum_{t \in \mathcal{T}} \sum_{\substack{u' \in V_1 \\ \mathsf{tp}_{\mathcal{G}_1}(u') = t}} \mathrm{softmax}(A)_{u,u'} \lambda_1^{(i)}(u') \\
&\quad = \sum_{t \in \mathcal{T}} \Big(q \cdot \sum_{\substack{v' \in V_2 \\ \mathsf{tp}_{\mathcal{G}_2}(v') = t}} \frac{1}{q} \cdot \mathrm{softmax}(B)_{v,v'} \lambda_2^{(i)}(v') \Big) \\
&\quad = q \cdot \sum_{v' \in V_2} \frac{1}{q} \cdot \mathrm{softmax}(B)_{v,v'} \lambda_2^{(i)}(v') \\
&\quad = \sum_{v' \in V_2} \mathrm{softmax}(B)_{v,v'} \lambda_2^{(i)}(v') \\
&\quad = (\mathrm{softmax}(B)\lambda_2^{(i)})_{v,*}.
\end{aligned}$$

Note that the third equation holds by ($\dagger_1$) and ($\dagger_2$).

For $\alpha = \mathrm{AH}$, consider the sets of indices with maximal value $\mathcal{I}_{A_{u,*}}$ and $\mathcal{I}_{B_{v,*}}$. By ($\dagger_1$), $|\mathcal{I}_{A_{u,*}}| = q \cdot |\mathcal{I}_{B_{v,*}}|$, and thus, for $u' \in \mathcal{I}_{A_{u,*}}$ and $v' \in \mathcal{I}_{B_{v,*}}$,

$$\mathrm{AH}(A)_{u,u'} = \frac{1}{|\mathcal{I}_{A_{u,*}}|} = \frac{1}{q \cdot |\mathcal{I}_{B_{v,*}}|} = \frac{1}{q} \cdot \mathrm{AH}(B)_{v,v'}.$$

Hence, using the same argument as for softmax, we obtain $(\mathrm{AH}(A)\lambda_1^{(i)})_{u,*} = (\mathrm{AH}(B)\lambda_2^{(i)})_{v,*}$. □

B.3 Proof of Theorem 9

We begin by introducing another central notion of bisimulation, which is associated with GML + G.

Definition 19 (Global Graded Bisimulation)**.** A **global graded bisimulation** Z between Π-labeled graphs $\mathcal{G}_1 = (V_1, E_1, \lambda_1)$ and $\mathcal{G}_2 = (V_2, E_2, \lambda_2)$ is a graded bisimulation such that the following additional conditions are satisfied:

global forth for all $v_1 \in V_1$ there is a $v_2 \in V_2$ with $(v_1, v_2) \in Z$.

global back for all $v_2 \in V_2$ there is a $v_1 \in V_1$ with $(v_1, v_2) \in Z$.

If there is a global graded bisimulation Z between Π-labeled graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ with $(v_1, v_2) \in Z$, we write $(\mathcal{G}_1, v_1) \sim_G (\mathcal{G}_2, v_2)$.

In this section, we aim to show that every FO-formula that is invariant under $\sim_{G\%}$ is equivalent to a GML + G-formula. This implies that these FO-formulae are also invariant under $\sim_G$, hence, relative to FO, $\sim_{G\%}$ and $\sim_G$ are the same. To show this, we apply techniques by Otto (Otto, 2004, 2019).

The **graded bisimulation game** that is associated with GML is played between two players, spoiler and duplicator over two Π-labeled graphs $\mathcal{G}_1 = (V_1, E_1, \lambda_1)$ and $\mathcal{G}_2 = (V_2, E_2, \lambda_2)$. Positions are pairs $(v_1, v_2) \in V_1 \times V_2$ and a single round played from this position allows spoiler to challenge duplicator. The **c-graded ℓ-round bisimulation game**, consists of ℓ rounds. Each round consists of the following moves:

graded down spoiler chooses a non-empty subset of $\mathrm{Neigh}_{\mathcal{G}_1}(v_1)$ or $\mathrm{Neigh}_{\mathcal{G}_2}(v_2)$ of size at most c. duplicator must respond with a matching subset of $\mathrm{Neigh}_{\mathcal{G}_2}(v_2)$ or $\mathrm{Neigh}_{\mathcal{G}_1}(v_1)$ on the opposite side of the same size.

spoiler then picks a vertex in the set proposed by duplicator and duplicator must respond by picking a vertex in the set proposed by spoiler.

Either player loses in this round if stuck, and duplicator loses as soon as the current position (v_1, v_2) violates $\lambda_1(v_1) = \lambda_2(v_2)$.

If duplicator has a winning strategy for the c-graded ℓ-round bisimulation game on Π-labeled graphs $\mathcal{G}_1$, $\mathcal{G}_2$ on starting position (v_1, v_2), we say that $(\mathcal{G}_1, v_1), (\mathcal{G}_2, v_2)$ are **c-graded ℓ-bisimilar**, written $(\mathcal{G}_1, v_1) \sim^{c,\ell} (\mathcal{G}_2, v_2)$. We say that $(\mathcal{G}_1, v_1), (\mathcal{G}_2, v_2)$ are **globally c-graded ℓ-bisimilar**, written $(\mathcal{G}_1, v_1) \sim_G^{c,\ell} (\mathcal{G}_2, v_2)$, if (i) $(\mathcal{G}_1, v_1) \sim^{c,\ell} (\mathcal{G}_2, v_2)$ and (ii) for every $u_1 \in V(\mathcal{G}_1)$ there is a $u_2 \in V(\mathcal{G}_2)$ such that $(\mathcal{G}_1, u_1) \sim^{c,\ell} (\mathcal{G}_2, u_2)$ and vice versa.

Global c-graded ℓ-bisimilarity is closely connected to GML + G as the following observations show.

Lemma 20. *Let $(\mathcal{G}_1, v_1)$, $(\mathcal{G}_2, v_2)$ be Π-labeled pointed graphs. If $(\mathcal{G}_1, v_1) \sim (\mathcal{G}_2, v_2)$, then, for all $c, \ell \geq 0$, $(\mathcal{G}_1, v_1) \sim^{c,\ell} (\mathcal{G}_2, v_2)$.*

Lemma 21. *Let φ be a* GML + G*-formula. Then, there are $c, \ell \geq 0$ such that φ is invariant under $\sim_G^{c,\ell}$. If φ is a* PL + G*-formula, then φ is invariant under $\sim_G^{0,0}$.*

Lemma 22. *For every $c, \ell \geq 0$, the equivalence relation $\sim_G^{c,\ell}$ has finite index and for each equivalence class of $\sim_G^{c,\ell}$, there is a* GML + G*-formula φ that defines this equivalence class. If $\ell = 0$, then φ is a* PL + G*-formula.*

For the first step of our proof, we now introduce the up-and-down variant $\approx$ of graded bisimulation, and its global variant $\approx_G$. In a Π-labeled graph $\mathcal{G} = (V, E, \lambda)$, we use $\mathrm{Pred}_{\mathcal{G}}(v)$ to denote the set of **predecessors** of $v \in V$, that is, $\{v' \mid (v', v) \in E\}$.

Definition 23 (Up-Down Graded Bisimulation)**.** An **up-down graded bisimulation** Z between Π-labeled graphs $\mathcal{G}_1 = (V_1, E_1, \lambda_1)$ and $\mathcal{G}_2 = (V_2, E_2, \lambda_2)$ is a graded bisimulation that satisfies the following additional conditions:

graded up forth for all $(u_1, u_2) \in Z$, for all $k \geq 1$: for pairwise distinct $v_1, \ldots v_k \in \mathrm{Pred}_{\mathcal{G}_1}(u_1)$ there are pairwise distinct $v'_1, \ldots, v'_k \in \mathrm{Pred}_{\mathcal{G}_2}(u_2)$ such that $(v_1, v'_1), \ldots, (v_k, v'_k) \in Z$.

graded up back for all $(u_1, u_2) \in Z$, for all $k \geq 1$: for pairwise distinct $v'_1, \ldots, v'_k \in \mathrm{Pred}_{\mathcal{G}_2}(u_2)$ there are pairwise distinct $v_1, \ldots, v_k \in \mathrm{Pred}_{\mathcal{G}_1}(u_1)$ such that $(v_1, v'_1), \ldots, (v_k, v'_k) \in Z$.

If there is an up-down graded bisimulation Z between Π-labeled graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ with $(v_1, v_2) \in Z$, we write $(\mathcal{G}_1, v_1) \approx (\mathcal{G}_2, v_2)$.

In analogy to global graded bisimulations, we also define **global up-down graded bisimulations** $\approx_G$ in the obvious way using the **global forth** and **global back** conditions.

Furthermore, we also define an up-down variant of graded bisimulation games. The **up-down c-graded ℓ-round bisimulation game** extends the c-graded ℓ-round bisimulation game by equipping spoiler with the following additional move:

graded up spoiler chooses a non-empty subset of $\mathrm{Pred}_{\mathcal{G}_1}(v_1)$ or $\mathrm{Pred}_{\mathcal{G}_2}(v_2)$ of size at most c. duplicator must respond with a matching subset of $\mathrm{Pred}_{\mathcal{G}_2}(v_2)$ or $\mathrm{Pred}_{\mathcal{G}_1}(v_1)$ on the opposite side of the same size.

spoiler then picks a vertex in the set proposed by duplicator and duplicator must respond by picking a vertex in the set proposed by spoiler.

If duplicator has a winning strategy for the up-down c-graded ℓ-round bisimulation game on Π-labeled graphs $\mathcal{G}_1$, $\mathcal{G}_2$ on starting position (v_1, v_2), we say that $(\mathcal{G}_1, v_1), (\mathcal{G}_2, v_2)$ are **up-down c-graded ℓ-bisimilar**, written $(\mathcal{G}_1, v_1) \approx^{c,\ell} (\mathcal{G}_2, v_2)$. We say that $(\mathcal{G}_1, v_1), (\mathcal{G}_2, v_2)$ are **globally up-down c-graded ℓ-bisimilar**, written $(\mathcal{G}_1, v_1) \approx_G^{c,\ell} (\mathcal{G}_2, v_2)$, if (i) $(\mathcal{G}_1, v_1) \approx^{c,\ell} (\mathcal{G}_2, v_2)$ and (ii) for every $u_1 \in V(\mathcal{G}_1)$ there is a $u_2 \in V(\mathcal{G}_2)$ such that $(\mathcal{G}_1, u_1) \approx^{c,\ell} (\mathcal{G}_2, u_2)$ and vice versa.

We now start with a small but useful observation about $\sim_{G\%}$. For a graph $\mathcal{G} = (V, E, \lambda)$, let $q \cdot \mathcal{G}$, for $q \geq 1$ denote the q-fold disjoint union of $\mathcal{G}$ with itself, that is,

$$
\begin{aligned}
V(q \cdot \mathcal{G}) &= V \times [q], \\
E(q \cdot \mathcal{G}) &= \{((v, i), (v', i)) \mid (v, v') \in E, 1 \leq i \leq q\}, \\
\lambda(q \cdot \mathcal{G}) &= \{(v, i) \mapsto \lambda(v) \mid (v, i) \in V(q \cdot \mathcal{G})\}.
\end{aligned}
$$

Proposition 24. *$\sim_{G\%}$ is preserved under disjoint copies, that is, for all $q \geq 1$ and i with $1 \leq i \leq q$,*

$$(\mathcal{G}, v) \sim_{G\%} (q \cdot \mathcal{G}, (v, i)).$$

This observation allows us to show the following first lemma.

Lemma 25. *For every* FO*-formula $\varphi(x)$ that is invariant under $\sim_{G\%}$, there are $c, \ell \geq 0$ such that φ is invariant under $\approx_G^{c,\ell}$.*

Proof. This is essentially already proved by Otto in (Otto, 2004). What is shown there is that if an FO-formula $\varphi(x)$ is invariant under $\hat{\sim}_G$, then there are $c, \ell \geq 0$ such that φ is invariant under $\approx_G^{c,\ell}$, where $\hat{\sim}_G$ is defined like $\sim_G$, but with **graded back** and **graded forth** replaced by non-graded versions. Note that this is a stronger precondition since $(\mathcal{G}, v) \sim_{G\%} (\mathcal{G}', v')$ implies $(\mathcal{G}, v) \hat{\sim}_G (\mathcal{G}', v')$. Nevertheless, exactly the same constructions also establish Lemma 25. Below, we describe Otto's proof in some more detail, and discuss why it yields Lemma 25. For full details, we refer the reader to (Otto, 2004).

We start with introducing some relevant notions. Let $\mathcal{G} = (V, E, \lambda)$ be a Π-labeled graph. Then

- the **distance** between vertices $v, v' \in V$, denoted $d(v, v')$ is the length of the shortest path between v and v' in $\mathcal{G}$ viewed as an undirected graph, and ∞ if no such path exists;
- the **neighborhood of radius** $\ell \geq 0$ around a vertex v in a graph $\mathcal{G}$ is

$$N^\ell(v) = \{v' \in V \mid d(v, v') \leq \ell\}.$$

We next define FO-formulae in Gaifman form and their dimensions:

- an FO-formula $\varphi(x)$ is **ℓ-local** if it is equivalent to its relativization to $N^\ell(v)$;
- an FO-formula $\varphi(x)$ is a **simple ℓ-local Gaifman formula** if it is a Boolean combination of (i) ℓ-local formulae $\psi(x)$ and (ii) formulae $\exists y\, \vartheta(y)$ with ϑ an ℓ-local formula; the **local quantifier rank** of φ is the maximum quantifier rank of all constituting formulae ψ, ϑ.

Let $\varphi(x)$ be an FO-formula that is invariant under $\sim_{G\%}$. Then by Proposition 24 $\varphi(x)$ is invariant under disjoint copies. By Proposition 19 of (Otto, 2004), this implies that $\varphi(x)$ is equivalent to a simple ℓ-local Gaifman formula $\psi(x)$, for some $\ell \geq 0$.

We can then use the proof of Lemma 35 of (Otto, 2004), without modifying any constructions, to show that together with invariance under $\sim_{G\%}$, this implies that $\varphi(x)$ is invariant under $\approx_G^{c,\ell}$ where c is the local quantifier rank of ψ. This implies Lemma 25. We next give some details. Assume to the contrary of what we have to show that there are Π-labeled pointed graphs $(\mathcal{G}_1, v_1)$ and $(\mathcal{G}_2, v_2)$ such that $(\mathcal{G}_1, v_1) \approx_G^{c,\ell} (\mathcal{G}_2, v_2)$, $\mathcal{G}_1 \models \varphi(v_1)$, and $\mathcal{G}_2 \not\models \varphi(v_2)$. We show how to transform $(\mathcal{G}_1, v_1)$ and $(\mathcal{G}_2, v_2)$ into Π-labeled pointed graphs $(\mathcal{H}_1, u_1)$ and $(\mathcal{H}_2, u_2)$ such that the following conditions are satisfied:

1. $(\mathcal{G}_i, v_i) \sim_{G\%} (\mathcal{H}_i, u_i)$ for $i \in \{1, 2\}$;
2. there is no simple ℓ-local Gaifman formula $\chi(x)$ such that $\mathcal{H}_1 \models \chi(u_1)$ and $\mathcal{H}_2 \not\models \chi(u_2)$.

This clearly yields the desired contradiction: from $\mathcal{G}_1 \models \varphi(v_1)$ and $\mathcal{G}_2 \not\models \varphi(v_2)$ and Point 1, we obtain $\mathcal{H}_1 \models \varphi(u_1)$, and $\mathcal{H}_2 \not\models \varphi(u_2)$, in contradiction to Point 2 and φ being equivalent to ψ.

In (Otto, 2004), $(\mathcal{H}_1, u_1)$ and $(\mathcal{H}_2, u_2)$ are obtained from $(\mathcal{G}_1, v_1)$ and $(\mathcal{G}_2, v_2)$ in three steps, which sufficiently increase the length of all cycles in $\mathcal{G}_1$ and $\mathcal{G}_2$. Recall that the **girth** of a graph is the minimal length of cycles in that graph. The first step introduces extra vertices to ensure that all cycles have at least length 3, the second step takes the product with the Caley graph of a group that has high girth which increases the length of all cycles of length at least 3, and the last step removes the extra vertices introduced by the first step.

Formally, the first step is to transform $(\mathcal{G}_1, v_1)$ and $(\mathcal{G}_2, v_2)$ into $\Pi \cup \{X, Y\}$-labeled pointed graphs $(\mathcal{I}_1, v_1)$ and $(\mathcal{I}_2, v_2)$ where X, Y are fresh vertex labels. For $i \in \{1, 2\}$, define[7]

$$\begin{array}{rcl} V(\mathcal{I}_i) & = & V(\mathcal{G}_i) \cup \{v_e, u_e \mid e \in E(\mathcal{G}_i)\} \\ E(\mathcal{I}_i) & = & \{(v, v_e), (v_e, u_e), (u_e, u) \mid \\ & & \qquad e = (v, u) \in E(\mathcal{G}_i)\} \\ \lambda(\mathcal{I}_i) & = & \{v \mapsto L \mid v \mapsto L \text{ in } \lambda(\mathcal{G}_i)\} \cup \\ & & \{v_e \mapsto \{X\}, u_e \mapsto \{Y\} \mid e \in E(\mathcal{G}_i)\}. \end{array}$$

We next take the product with (the Cayley graph of) a finite group of high girth. Let $(G, \circ)$ be a finite group of sufficiently high girth and $g \colon E(\mathcal{I}_i) \to G$ an embedding such that $\{g(e) \mid e \in E(\mathcal{I}_i)\} \cap \{g(e)^{-1} \mid e \in E(\mathcal{I}_i)\} = \emptyset$. We set

$$\begin{array}{rcl} V(\mathcal{I}'_i) & = & V(\mathcal{I}_i) \times G \\ E(\mathcal{I}'_i) & = & \{((v, a), (u, a \circ g(e))) \mid \\ & & \qquad e = (v, u) \in E(\mathcal{I}_i),\ a \in G\} \\ \lambda(\mathcal{I}'_i) & = & \{(v, a) \mapsto L \mid v \mapsto L \text{ in } \lambda(\mathcal{I}_i),\ a \in G\}. \end{array}$$

In the third step, we 'reverse' the effect of the first step and move back to Π-labeled pointed graphs by setting

$$\begin{array}{rcl} V(\mathcal{H}_i) & = & \{(v, a) \mid (v, a) \in V(\mathcal{I}'_i) \text{ with } v \in V(\mathcal{G}_i)\} \\ E(\mathcal{H}_i) & = & \{((v, a_1), (u, a_4)) \mid \\ & & \qquad ((v, a_1), (v_e, a_2)), ((v_e, a_2), (u_e, a_3)), \\ & & \qquad\quad ((u_e, a_3), (u, a_4)) \in E(\mathcal{I}'_i), \\ & & X \in \lambda(\mathcal{I}'_i)((v_e, a_2)),\ \text{and } Y \in \lambda(\mathcal{I}'_i)((u_e, a_3))\} \end{array}$$

and taking $\lambda(\mathcal{H}_i)$ to be the restriction of $\lambda(\mathcal{I}'_i)$ to the vertices in $\mathcal{H}_i$. We choose an $a \in G$ and set $u_1 = (v_1, a)$ and $u_2 = (v_2, a)$.

It is proved in (Otto, 2004) (Lemma 35) that $(\mathcal{G}_1, v_1) \approx_G^{c,\ell} (\mathcal{G}_2, v_2)$ implies Point 2 above. It thus remains to prove Point 1, that is, for $i \in \{1, 2\}$ (i) $(\mathcal{G}_i, v_i) \sim (\mathcal{H}_i, u_i)$ and (ii) there exists a rational $q > 0$ such that for each graded bisimulation type t over Π,

$$\begin{array}{l} |\{v \in V(\mathcal{G}_i) \mid \mathsf{tp}_{\mathcal{G}_i}(v) = t\}| \\ \qquad = q \cdot |\{v \in V(\mathcal{H}_i) \mid \mathsf{tp}_{\mathcal{H}_i}(v) = t\}|. \end{array}$$

It is easy to verify that

$$Z = \{(v, (v, a)) \mid (v, a) \in V(\mathcal{H}_i)\}$$

is a graded bisimulation. Together with the choice of u_1 and u_2, this yields Point (i). It also implies $\mathsf{tp}_{\mathcal{G}_i}(v) = \mathsf{tp}_{\mathcal{H}_i}(v, a)$ for all $(v, a) \in V(\mathcal{H}_i)$ and thus yields Point (ii) for $q = \frac{1}{|G|}$. $\square$

[7] We replace every edge with a path of length 3. In (Otto, 2004), a path of length 2 is used. We believe that this is a mistake as a reflexive edge in $\mathcal{G}_i$ then results in two symmetric edges in $\mathcal{I}_i$, but this is not allowed in a graph that is simple according to Definition 27 in (Otto, 2004).

In the next step of the proof, we show invariance under an up-down bisimulation relation that can count out-neighbors, but not predecessors.

Definition 26 (Up-Ungraded Down-Graded Bisimulation)**.** An **up-ungraded down-graded bisimulation** Z between Π-labeled graphs $\mathcal{G}_1 = (V_1, E_1, \lambda_1)$ and $\mathcal{G}_2 = (V_2, E_2, \lambda_2)$ is a graded bisimulation that satisfies the following additional conditions:

ungraded up forth for all $(u_1, u_2) \in Z$ and $v_1 \in \mathrm{Pred}_{\mathcal{G}_1}(u_1)$, there is a $v_2 \in \mathrm{Pred}_{\mathcal{G}_2}(u_2)$ such that $(v_1, v_2) \in Z$.

ungraded up back for all $(u_1, u_2) \in Z$ and $v_2 \in \mathrm{Pred}_{\mathcal{G}_2}(u_2)$, there is a $v_1 \in \mathrm{Pred}_{\mathcal{G}_1}(u_1)$ such that $(v_1, v_2) \in Z$.

If there is an up-ungraded down-graded bisimulation Z between Π-labeled graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ with $(v_1, v_2) \in Z$, we write $(\mathcal{G}_1, v_1) \approx^{\downarrow} (\mathcal{G}_2, v_2)$.

In analogy to global graded bisimulations, we also define **global up-ungraded down-graded bisimulations** $\approx^{\downarrow}_{G}$ in the obvious way using the conditions **global forth** and **global back**.

Furthermore, we define a variant of graded bisimulation games that corresponds to $\approx^{\downarrow}$. The **up-ungraded down-c-graded ℓ-round bisimulation game** extends the c-graded ℓ-round bisimulation game by equipping spoiler with the following additional move:

ungraded up spoiler chooses an element of $\mathrm{Pred}_{\mathcal{G}_1}(v_1)$ or $\mathrm{Pred}_{\mathcal{G}_2}(v_2)$. duplicator must respond with an element of $\mathrm{Pred}_{\mathcal{G}_2}(v_2)$ or $\mathrm{Pred}_{\mathcal{G}_1}(v_1)$ on the opposite side.

If duplicator has a winning strategy for the up-ungraded down-c-graded ℓ-round bisimulation game on Π-labeled graphs $\mathcal{G}_1$, $\mathcal{G}_2$ on starting position (v_1, v_2), we say that $(\mathcal{G}_1, v_1), (\mathcal{G}_2, v_2)$ are **up-ungraded down-c-graded ℓ-bisimilar**, written $(\mathcal{G}_1, v_1) \approx^{c\downarrow,\ell} (\mathcal{G}_2, v_2)$. We say that $(\mathcal{G}_1, v_1), (\mathcal{G}_2, v_2)$ are **globally up-ungraded down-c-graded ℓ-bisimilar**, written $(\mathcal{G}_1, v_1) \approx^{c\downarrow,\ell}_{G} (\mathcal{G}_2, v_2)$, if (i) $(\mathcal{G}_1, v_1) \approx^{c\downarrow,\ell} (\mathcal{G}_2, v_2)$ and (ii) for every $u_1 \in V(\mathcal{G}_1)$ there is a $u_2 \in V(\mathcal{G}_2)$ such that $(\mathcal{G}_1, u_1) \approx^{c\downarrow,\ell} (\mathcal{G}_2, u_2)$ and vice versa.

Lemma 27. *Every* FO*-formula $\varphi(x)$ that is invariant under $\sim_{G\%}$ and under $\approx^{c,\ell}_{G}$, with $c, \ell \geq 0$, is also invariant under $\approx^{c\downarrow,\ell}_{G}$.*

Proof. Let $\varphi(x)$ be an FO-formula that is invariant under $\sim_{G\%}$ and $\approx^{c,\ell}_{G}$ with $c, \ell \geq 0$. Assume to the contrary of what we have to show that there exist Π-labeled pointed graphs $(\mathcal{G}_1, v_1)$ and $(\mathcal{G}_2, v_2)$ such that $(\mathcal{G}_1, v_1) \approx^{c\downarrow,\ell}_{G} (\mathcal{G}_2, v_2)$, $\mathcal{G}_1 \models \varphi(v_1)$ and $\mathcal{G}_2 \not\models \varphi(v_2)$. We construct new graphs $\mathcal{H}_1, \mathcal{H}_2$ as follows.

Let $W_{c,\ell}$ be the finite set of all words over the alphabet $\{1, \ldots, c\}$ that are of length at most ℓ, and let $\cdot$ denote concatenation. For $i \in \{1, 2\}$ and $\mathcal{G}_i = (V_i, E_i, \lambda_i)$, set $\mathcal{H}_i = (V_i', E_i', \lambda_i')$, where

$$
\begin{aligned}
V_i' &= \{(v, w) \mid v \in V_i, w \in W_{c,\ell}\},\\
E_i' &= \{((v, w \cdot b), (u, w)) \mid (v, u) \in E_i,\\
&\qquad\qquad b \in [c], w \cdot b \in W_{c,\ell}\} \cup\\
&\quad \{((v, \varepsilon), (u, \varepsilon)) \mid (v, u) \in E_i\},\\
\lambda_i' &= \{(v, w) \mapsto \lambda_i(v) \mid (v, w) \in V_i'\}.
\end{aligned}
$$

We show that $\mathcal{H}_1, \mathcal{H}_2$ satisfy the following conditions:

1. $(\mathcal{G}_i, v_i) \sim_{G\%} (\mathcal{H}_i, (v_i, w))$ for all $w \in W_{c,\ell}$ and $i \in \{1, 2\}$;
2. $(\mathcal{H}_1, (v, w_1)) \approx^{c,k} (\mathcal{H}_2, (u, w_2))$ for all $k \leq \ell$ and $w_1, w_2 \in W_{c,\ell}$ such that
   $(*)$ if $|w_1| \neq |w_2|$, then $\max(|w_1|, |w_2|) \leq \ell - k$,
   and all v, u such that $(\mathcal{G}_1, v) \approx^{c\downarrow,k} (\mathcal{G}_2, u)$;
3. $(\mathcal{H}_1, (v_1, \varepsilon)) \approx_G^{c,\ell} (\mathcal{H}_2, (v_2, \varepsilon))$.

This yields the desired contradiction: from Point 1 we can conclude that $\mathcal{H}_1 \models \varphi(v_1, \varepsilon)$ and $\mathcal{H}_2 \not\models \varphi(v_2, \varepsilon)$, but this together with Point 3 contradicts that φ is invariant under $\approx_G^{c,\ell}$. We use Point 2 to show Point 3.

Proof of Point 1. Observe that for all $i \in \{1, 2\}$,

$$\{(v, (v, w)) \mid (v, w) \in V_i'\}$$

is a graded (downwards-only) bisimulation between $\mathcal{G}_i$ and $\mathcal{H}_i$. Hence, for all $(v, w) \in V_i'$, $\mathsf{tp}_{\mathcal{G}_i}(v) = \mathsf{tp}_{\mathcal{H}_i}(v, w)$. Note that, as before, tp refers to the *downwards-only* graded bisimulation type. Point 1 then follows from the fact that for every vertex $v \in V_i$, there are exactly $|W_{c,\ell}|$ vertices $(v, w) \in V_i'$.

Proof of Point 2. We show Point 2 by induction on k. In the case $k = 0$, $(\mathcal{H}_1, (v, w_1)) \approx^{c,k} (\mathcal{H}_2, (u, w_2))$ is immediate by definition of λ_i'. Now let $k > 0$, and assume that the up-down c-graded k-round bisimulation game starts in position $((v, w_1), (u, w_2))$ such that $(*)$ is satisfied.

If $\mathsf{spoiler}$ makes a **graded down** move in $\mathcal{H}_1$, all vertices they select must be of shape (v', w_1') with either $w_1 = w_1' = \varepsilon$, or $w_1 = w_1' \cdot b$ for some $b \in [c]$. As $(\mathcal{G}_1, v) \approx^{c\downarrow,k} (\mathcal{G}_2, u)$ and by construction of $\mathcal{H}_1$ and $\mathcal{H}_2$, for each (v', w_1') selected by $\mathsf{spoiler}$, there must be a distinct $(u', w_2') \in V_2'$ with $w_2 = w_2' = \varepsilon$ or $w_2 = w_2' \cdot b$ for some $b \in [c]$, such that $(\mathcal{G}_1, v') \approx^{c\downarrow,k-1} (\mathcal{G}_2, u')$. Note that $\max(|w_1'|, |w_2'|) \leq \max(|w_1|, |w_2|) \leq \ell - k$ and therefore $(*)$ holds for every possible new position. The induction hypothesis then yields that $\mathsf{duplicator}$ has a winning strategy for the $k-1$-round game from any possible new position, as required. The symmetric argument applies if $\mathsf{spoiler}$ makes a **graded down** move in $\mathcal{H}_2$.

If $\mathsf{spoiler}$ makes a **graded up** move in $\mathcal{H}_1$, they select at most c distinct predecessors of (v, w_1), which are of the form $(v', w_1 \cdot b)$ with $b \in [c]$, or possibly (v', ε) if $w_1 = \varepsilon$. As $(\mathcal{G}_1, v) \approx^{c\downarrow,k} (\mathcal{G}_2, u)$, there must be, for each such v' that occurs in the set selected by $\mathsf{spoiler}$, a corresponding predecessor u' of u such that $(\mathcal{G}_1, v') \approx^{c\downarrow,k-1} (\mathcal{G}_2, u')$. Now, for each $(v', w_1 \cdot b)$ or (v', ε) in the set selected by $\mathsf{spoiler}$, $\mathsf{duplicator}$ can select an answer $(u', w_2 \cdot a)$, starting with $a = 1$ and increasing a such that all choices are distinct. This is possible because w_1 and w_2 satisfy $(*)$: If $|w_1| = |w_2|$, then the existence of predecessors of (v, w_1) implies that $|w_1| = |w_2| < \ell$ by construction of $\mathcal{H}_1$, and thus (u, w_2) has predecessors of the shape $(u', w_2 \cdot a)$ by construction of $\mathcal{H}_2$. If $|w_1| \neq |w_2|$, then $(*)$ implies that $\max(|w_1|, |w_2|) \leq \ell - k \leq \ell - 1$ and therefore (u, w_2) also has vertices of the shape $(u', w_2 \cdot a)$ as predecessors by construction of $\mathcal{H}_2$. Furthermore, $\mathsf{spoiler}$ selects at most c vertices and $(u', w_2 \cdot a) \in V_2'$ for $a \in [c]$.

Now $\mathsf{spoiler}$ selects one vertex from $\mathsf{duplicator}$'s set, and $\mathsf{duplicator}$ can respond accordingly. Let $((v', w_1'), (u', w_2'))$ be the resulting position. If $|w_1'| = |w_2'|$, then $(*)$ holds. If $|w_1'| \neq |w_2'|$, then $\max(|w_1'|, |w_2'|) \leq \max(|w_1|, |w_2|) + 1 \leq \ell - (k - 1)$. Thus $(*)$ also holds for $k - 1$. As furthermore $(\mathcal{G}_1, v') \approx^{c\downarrow,k-1} (\mathcal{G}_2, u')$ the induction hypothesis implies that $\mathsf{duplicator}$ has a winning strategy for the $k - 1$ round game from this position. A symmetric argument applies if $\mathsf{spoiler}$ makes a **graded up** move in $\mathcal{H}_2$.

Proof of Point 3. $(\mathcal{H}_1,(v_1,\varepsilon)) \approx^{c,\ell} (\mathcal{H}_2,(v_2,\varepsilon))$ follows from Point 2 and the fact that $(\mathcal{G}_1,v_1) \approx^{c\downarrow,\ell} (\mathcal{G}_2,v_2)$. Let (v,w) be any vertex in $\mathcal{H}_1$. As $(\mathcal{G}_1,v_1) \approx_G^{c\downarrow,\ell} (\mathcal{G}_2,v_2)$, there must be a vertex u in $\mathcal{G}_2$ such that $(\mathcal{G}_1,v) \approx^{c\downarrow,\ell} (\mathcal{G}_2,u)$. Consider the vertex (u,w) in $\mathcal{H}_2$. As the second components of (v,w) and (u,w) are identical, $(*)$ of Point 2 is satisfied. Thus, $(\mathcal{H}_1,(v,w)) \approx^{c,\ell} (\mathcal{H}_2,(u,w))$ as required. The symmetric argument applies to any vertex (u,w) in $\mathcal{H}_2$. □

In the last step of our proof, we finally show invariance under $\sim_G^{c,\ell}$.

Lemma 28. *Every* FO*-formula $\varphi(x)$ that is invariant under $\sim_{G\%}$ and under $\approx_G^{c\downarrow,\ell}$, with $c,\ell \geq 0$, is also invariant under $\sim_G^{c,2\ell}$.*

Proof. The proof is inspired by the unique history construction from Otto's proof of Lemma 40 in (Otto, 2004). We can, however, not use that construction as is because given a Π-labeled pointed graph $(\mathcal{G},v)$ we want to produce a Π-labeled pointed graph $(\mathcal{H},u)$ such that $(\mathcal{G},v) \sim_{G\%} (\mathcal{H},u)$, and Otto's construction does not achieve this. We thus adapt it in a suitable way.

Let $\varphi(x)$ be an FO-formula that is invariant under $\sim_{G\%}$ and $\approx_G^{c\downarrow,\ell}$, with $c,\ell \geq 0$. Assume to the contrary of what we have to show that there exist pointed graphs $(\mathcal{G}_1,v_1)$ and $(\mathcal{G}_2,v_2)$ such that $(\mathcal{G}_1,v_1) \sim_G^{c,2\ell} (\mathcal{G}_2,v_2)$, $\mathcal{G}_1 \models \varphi(v_1)$, and $\mathcal{G}_2 \not\models \varphi(v_2)$. We construct new graphs $\mathcal{H}_1,\mathcal{H}_2$ as follows.

Let Π be a finite set of vertex label symbols. A **c-graded ℓ-bisimulation type over Π is a maximal set t of Π-labeled pointed graphs such that $(\mathcal{G},v) \sim^{c,\ell} (\mathcal{G}',v')$ for all $(\mathcal{G},v),(\mathcal{G}',v') \in t$.**

For a Π-labeled pointed graph $(\mathcal{G},v)$, we use $\mathsf{tp}_{\mathcal{G}}^{c,\ell}(v)$ to denote the unique c-graded ℓ-bisimulation type t over Π such that $(\mathcal{G},v) \in t$. If $u_1,\ldots,u_k$ is a path in $\mathcal{G}$ with $k \leq \ell$, then we call the sequence $h = \mathsf{tp}_{\mathcal{G}}^{c,\ell}(u_1),\ldots,\mathsf{tp}_{\mathcal{G}}^{c,\ell}(u_k)$ an **ℓ-history** of u_k in $\mathcal{G}$. We say that h is **maximal** if $k=\ell$ or u_1 does not have any predecessors in $\mathcal{G}$.[8] We use $\mathsf{tail}(h)$ to denote $\mathsf{tp}_{\mathcal{G}}^{c,\ell}(u_k)$. Note that a vertex may have multiple ℓ-histories, of varying lengths, and that every vertex has at least one maximal ℓ-history. We say that an ℓ-history h_2 is a **continuation** of an ℓ-history $h_1 = t_1,\ldots,t_k$ if one of the following conditions holds:

- $k < \ell$ and $h_2 = t_1,\ldots,t_k,t$ for some t;
- $k = \ell$ and $h_2 = t_2,\ldots,t_k,t$ for some t.

An **ℓ-history choice** for $\mathcal{G}$ is a function χ that assigns to each vertex v in $\mathcal{G}$ a (not necessarily maximal) ℓ-history of v in $\mathcal{G}$.

For $i \in \{1,2\}$, the graph $\mathcal{H}_i$ is defined as follows:

- $V(\mathcal{H}_i)$ consists of all pairs (v,χ) with $v \in V(\mathcal{G}_i)$ and χ a history choice for $\mathcal{G}_i$;
- for every edge $(v,u) \in E(\mathcal{G}_i)$ and every vertex $(v,\chi) \in V(\mathcal{H}_i)$, we find an edge $((v,\chi),(u,\chi')) \in E(\mathcal{H}_i)$ where $\chi' = \chi$ except that $\chi'(u)$ is the unique continuation of $\chi(v)$ with $\mathsf{tail}(\chi'(u)) = \mathsf{tp}_{\mathcal{G}_i}^{c,\ell}(u)$;
- each vertex $(v,\chi) \in V(\mathcal{H}_i)$ carries the same vertex labels as the vertex $v \in V(\mathcal{G}_i)$.

Informally, a main purpose of ℓ-history choices is to ensure that every vertex from $\mathcal{G}_i$ gets duplicated exactly the same number of times in $\mathcal{H}_i$. This number is n_H, the number of ℓ-history choices for $\mathcal{G}_i$.

[8] Our histories are defined in a more liberal way than in (Otto, 2004), where they must be maximal.

For $i \in \{1,2\}$, choose u_i to be some vertex $(v_i, \chi) \in V(\mathcal{H}_i)$ such that $\chi(v_i) = \mathsf{tp}^{c,\ell}_{\mathcal{G}_i}(v_i)$. We show the following, for all $i \in \{1,2\}$:

1. $\mathsf{tp}_{\mathcal{G}_i}(v) = \mathsf{tp}_{\mathcal{H}_i}(v,\chi)$ for all $(v,\chi) \in V(\mathcal{H}_i)$;
2. $(\mathcal{G}_i, v_i) \sim_{G\%} (\mathcal{H}_i, u_i)$;
3. $(\mathcal{G}_1, v) \sim^{c,2\ell} (\mathcal{G}_2, v')$ implies $(\mathcal{H}_1, (v,\chi)) \sim^{c,2\ell} (\mathcal{H}_2, (v',\chi'))$ for all $(v,\chi) \in V(\mathcal{H}_1)$ and $(v',\chi') \in V(\mathcal{H}_2)$;
4. each vertex $(v,\chi) \in V(\mathcal{H}_i)$ has unique maximal ℓ-history $\chi(v)$;
5. $(\mathcal{H}_1, (v_1,\chi)) \approx^{c\downarrow,\ell}_G (\mathcal{H}_2, (v_2,\chi'))$ whenever $\chi(v_1) = \chi'(v_2) = \mathsf{tp}^{c,\ell}_{\mathcal{G}_1}(v_1) = \mathsf{tp}^{c,\ell}_{\mathcal{G}_2}(v_2)$ for $i \in \{1,2\}$.

Note that the equality $\mathsf{tp}^{c,\ell}_{\mathcal{G}_1}(v_1) = \mathsf{tp}^{c,\ell}_{\mathcal{G}_2}(v_2)$ in Point 5 holds since $(\mathcal{G}_1, v_1) \sim^{c,2\ell}_G (\mathcal{G}_2, v_2)$.

The above gives the desired contradiction. From Point 2 and $\varphi(x)$ being invariant under $\sim_{G\%}$ it follows that $\mathcal{H}_1 \models \varphi(u_1)$ and $\mathcal{H}_2 \not\models \varphi(u_2)$. But this is a contradiction to Point 5 because φ is invariant under $\approx^{c\downarrow,\ell}_G$. The purpose of Points 1, 3 and 4 is to support the proof of Points 2 and 5.

Proof of Point 1. Point 1 follows from the observation that

$$\{(v,(v,\chi)) \mid (v,\chi) \in V(\mathcal{H}_i)\}$$

is a graded bisimulation between $\mathcal{G}_i$ and $\mathcal{H}_i$.

Proof of Point 2. Point 2 follows from Point 1 and the fact that for every vertex $v \in V(\mathcal{G}_i)$, there are exactly n_H vertices $(v,\chi) \in V(\mathcal{H}_i)$.

Proof of Point 3. Let $v \in V(\mathcal{G}_1)$ and $v' \in V(\mathcal{G}_2)$ such that $(\mathcal{G}_1, v) \sim^{c,2\ell} (\mathcal{G}_2, v')$. Then duplicator has a winning strategy S for the c-graded 2ℓ-round bisimulation game on $\mathcal{G}_1, \mathcal{G}_2$ from starting position (v,v'). Let $(v,\chi) \in V(\mathcal{H}_1)$ and $(v',\chi') \in V(\mathcal{H}_2)$. We can use S to identify a winning strategy S' for duplicator in the c-graded 2ℓ-round bisimulation game on $\mathcal{H}_1, \mathcal{H}_2$ from starting position $((v,\chi),(v',\chi'))$. In fact, S' simply mimics the responses of duplicator in S.

Let the current game position be $((u,\xi),(u',\xi'))$ in the second game and assume that spoiler plays $X \subseteq \mathrm{Neigh}_{\mathcal{H}_1}((u,\xi))$ with $|X| \leq c$ in a **graded down** move. Consider the corresponding position (u,u') in the first game and set $X_0 = \{w \mid (w,\rho) \in X\}$. By construction of $\mathcal{H}_1$, we have $X_0 \subseteq \mathrm{Neigh}_{\mathcal{G}_1}(u)$ and $|X| = |X_0|$. Thus spoiler can play X_0 in the first game using the same kind of move, and duplicator has a response $Y_0 \subseteq \mathrm{Neigh}_{\mathcal{G}_2}(u')$ with $|Y_0| = |X_0|$. In the second game, duplicator then plays the set Y that consists of all elements (w,ρ) such that $w \in Y_0$ and $\rho = \xi'$ except that $\rho(w)$ is the unique continuation of $\xi'(u')$ with $\mathsf{tail}(\rho(w)) = \mathsf{tp}^{c,\ell}_{\mathcal{G}_2}(w)$. Clearly, $|Y| = |X|$.

As part of the same move, spoiler then chooses an element $(w',\xi'') \in Y$. In the first game, they may choose $w' \in Y_0$, and duplicator has a response $w \in X_0$. By definition of X_0 there is a $(w,\rho) \in X$. By construction of $\mathcal{H}_1$ and choice of X, there is in fact a unique such (w,ρ) with $\rho = \xi$ except that $\rho(w)$ is the unique continuation of $\xi(u)$ with $\mathsf{tail}(\rho(w)) = \mathsf{tp}^{c,\ell}_{\mathcal{G}_1}(w)$. In the second game, duplicator replies with choosing (w,ρ). The case where spoiler plays a set $X \subseteq \mathrm{Neigh}_{\mathcal{H}_2}((u',\xi'))$ is symmetric. It can be verified that the described strategy S' is indeed winning.

Proof of Point 4. We have to argue that if $(v,\chi) \in V(\mathcal{H}_i)$ and h is a maximal ℓ-history of (v,χ) in $\mathcal{H}_i$, then $h = \chi(v)$. Thus let $(v,\chi) \in V(\mathcal{H}_i)$, and let $\chi(v) = t_1, \ldots, t_k$. Take

any path $(u_1, \chi_1), \ldots, (u_m, \chi_m) = (v, \chi)$ in $\mathcal{H}_i$ that gives rise to a maximal ℓ-history $h = t'_1, \ldots, t'_m$ of (v, χ). Note that $\chi_m(u_m) = \chi(v)$ and, by construction of $\mathcal{H}_i$, $\mathsf{tp}^{c,\ell}_{\mathcal{H}_i}((v, \chi)) = \mathsf{tail}(\chi(v))$.

For $0 \leq z \leq k$, let $\mathsf{head}_z(\chi(v))$ denote the result of dropping from $\chi(v)$ all but the trailing z types and let $\mathsf{tail}_z(\chi_j(u_j))$ denote the result of dropping from $\chi_j(u_j)$ all but the leading z types. Using the construction of $E(\mathcal{H}_i)$ and the fact that $\chi_m(u_m) = \chi(v)$, one may verify that one of the following must hold:

(a) $m < \ell$ and $\chi_z(u_z) = \mathsf{head}_z(\chi(v))$ for $1 \leq z \leq m$;

(b) $m = \ell$ and $\mathsf{tail}_z(\chi_z(u_z)) = \mathsf{head}_z(\chi(v))$ for $1 \leq z \leq m$.

In both cases, by definition of $V(\mathcal{H}_i)$ and Point 1, this implies $t_{k-z} = t'_{m-z}$ for $0 \leq z < \min\{k, m\}$. It thus remains to show that $m = k$.

First assume to the contrary that $m > k$. Then we must clearly have $k < \ell$. Since $\chi_m(u_m) = \chi(v)$, the length of $\chi_m(u_m)$ is strictly smaller than ℓ. But then the length of $\chi_j(u_j)$ is strictly smaller than that of $\chi_{j+1}(u_{j+1})$ for $1 \leq j < m$. As a consequence, $m < \ell$. We are thus in Case (a) above and therefore $\chi_{m-(k-1)}(u_{m-(k-1)})$ must be a sequence of length 1. By construction of $E(\mathcal{H}_i)$, the vertex $(u_{m-(k-1)}, \chi_{m-(k-1)})$ therefore has no predecessor in $\mathcal{H}_i$, in contradiction to the fact that (u_{m-k}, χ_{m-k}) is such a predecessor.

Now assume that $k > m$. Then clearly $m < \ell$. We first observe that since h is a maximal ℓ-history, $\chi_1(u_1)$ is a sequence of length 1. Assume otherwise. By definition of $V(\mathcal{H}_i)$, this implies that u_1 has a predecessor w in $\mathcal{G}_i$ which has $\mathsf{head}_1(\chi_1(u_1))$ as an ℓ-history. But then by definition of $E(\mathcal{H}_i)$, (w, ρ) is a predecessor of (u_1, χ_1) in $\mathcal{H}_1$ for every ρ such that $\rho = \chi_1$ except that $\chi_1(u_1)$ is the unique continuation of $\rho(w)$ with $\mathsf{tail}(\chi_1(u_1)) = \mathsf{tp}^{c,\ell}_{\mathcal{G}_i}(u_1)$. There is clearly at least one such ρ. But since $m < \ell$ this implies that $\mathsf{tp}^{c,\ell}_{\mathcal{G}_i}(w), h$ is an ℓ-history of (v, χ) in $\mathcal{H}_i$, in contradiction to h being maximal. We have thus shown that $\chi_1(u_1)$ is a sequence of length 1. But then in both Case (a) and Case (b) above, $\mathsf{head}_{m-1}(\chi(v))$ is a sequence of length 1, implying that $k = m$. Contradiction.

Proof of Point 5. Let $i \in \{1, 2\}$. We show that

(a) for every vertex $(v, \chi) \in V(\mathcal{H}_i)$, there is a vertex $(v', \chi') \in V(\mathcal{H}_{3-i})$ such that $\mathsf{duplicator}$ has a winning strategy in the up-ungraded down-c-graded ℓ-round bisimulation game on $\mathcal{H}_1, \mathcal{H}_2$ starting from position $((v, \chi), (v', \chi'))$, and

(b) if $v = v_1$ and $\chi(v_1) = \mathsf{tp}^{c,\ell}_{\mathcal{G}_1}(v_1)$, then we can choose (v', χ') such that $v' = v_2$ and $\chi'(v_2) = \mathsf{tp}^{c,\ell}_{\mathcal{G}_2}(v_2)$.

This clearly yields Point 5. Thus let $(v, \chi) \in V(\mathcal{H}_i)$. We want to choose a vertex $(v', \chi') \in V(\mathcal{H}_{3-i})$ that satisfies the conditions in Points (a) and (b). Importantly, we want to choose (v', χ') so that

(c) $(\mathcal{G}_i, v) \sim^{c,\ell} (\mathcal{G}_{3-i}, v')$ and

(d) $\chi(v) = \chi'(v')$.

By Point 4, (v, χ) has unique maximal ℓ-history $\chi(v)$ in $\mathcal{H}_i$. Let $\chi(v) = t_1, \ldots, t_k$. We thus find a path $x_1, \ldots, x_k = (v, \chi)$ in $\mathcal{H}_i$ such that $\mathsf{tp}^{c,\ell}_{\mathcal{H}_i}(x_r) = t_r$ for $1 \leq r \leq k$, and $k = \ell$ or x_1 has no predecessor in $\mathcal{H}_i$. Let $x_r = (u_r, \chi_r)$ for $1 \leq r \leq k$. Since $(\mathcal{G}_1, v_1) \sim^{c,2\ell}_G (\mathcal{G}_2, v_2)$, we can find in $\mathcal{G}_{3-i}$ a vertex w such that $(\mathcal{G}_i, u_1) \sim^{c,2\ell} (\mathcal{G}_{3-i}, w)$. Note that by definition of ℓ-histories, $h' = \mathsf{tp}^{\ell}_{\mathcal{G}_{3-i}}(w)$ is a (possibly not maximal) ℓ-history of w in $\mathcal{G}_{3-i}$. By definition of $\mathcal{H}_{3-i}$, we thus find a vertex $y_1 = (w, \chi')$ in $\mathcal{H}_{3-i}$ such that $\chi'(w) = h'$. From Point 3, we obtain $(\mathcal{H}_1, x_1) \sim^{c,2\ell} (\mathcal{H}_2, y_1)$. Exercising the **graded forth** property k

times, following the path $x_1, \dots, x_k$ in $\mathcal{H}_i$, we find in $\mathcal{H}_{3-i}$ a path $y_1, \dots, y_k$ such that $(\mathcal{G}_i, x_r) \sim^{c,2\ell-r} (\mathcal{G}_{3-i}, y_r)$ for $1 \leq r \leq k$. As the vertex (v', χ') in Point (a) above, we choose y_k. Note that Point (b) above is satisfied automatically. Also note that Points (c) and (d) are satisfied. For Point (d), this is the case because, by choice of (v', χ'), this vertex has ℓ-history $\chi(v)$ in $\mathcal{H}_{3-i}$, no matter whether $k = \ell$ or x_1 has no predecessor in $\mathcal{H}_i$. By Point (4), this implies that $\chi'(v') = \chi(v)$, as required.

It remains to argue that duplicator has a winning strategy S' in the up-ungraded down-c-graded ℓ-round bisimulation game on $\mathcal{H}_i, \mathcal{H}_{3-i}$ from position $(v, \chi), (v', \chi')$. Key to this strategy is to ensure the following invariants: if $(u_1, \xi_1), (u_2, \xi_2)$ is the position reached after $m \in \{0, \dots, \ell\}$ rounds, then

(i) $(\mathcal{G}_1, u_1) \sim^{c,\ell-m} (\mathcal{G}_2, u_2)$ and

(ii) the length $\ell - m$-suffixes of $\xi_1(u_1)$ and $\xi_2(u_2)$ are identical.

Note that these invariants are satisfied at the initial position due to Points (c) and (d). We now describe S'. Let the current game position be $(u_1, \xi_1), (u_2, \xi_2)$ and assume that spoiler plays $X \subseteq \mathrm{Neigh}_{\mathcal{H}_i}((u_i, \xi_i))$ with $|X| \leq c$ in a **graded down** move, $i \in \{1, 2\}$. By Invariant (i), duplicator has a winning strategy S in the down-c-graded $\ell - m$-bisimulation game starting in position (u_1, u_2). Consider the set $X_0 = \{w \mid (w, \xi) \in X\}$. By construction of $\mathcal{H}_i$, we have $X_0 \subseteq \mathrm{Neigh}_{\mathcal{G}_i}(u_i)$ and $|X_0| \leq c$ and thus spoiler can play X_0 in S using the same kind of move. duplicator has a response $Y_0 \subseteq \mathrm{Neigh}_{\mathcal{G}_{3-i}}(u_{3-i})$. For S', duplicator then plays the set Y that consists of all elements (w, ρ) such that $w \in Y_0$ and $\rho = \xi_{3-i}$ except that $\rho(w)$ is the unique continuation of $\xi_{3-i}(u_{3-i})$ with $\mathsf{tail}(\rho(w)) = \mathsf{tp}^{c,\ell}_{\mathcal{G}_{3-i}}(w)$. As part of the same move, spoiler then chooses an element $(w', \xi') \in Y$. In S, they may choose $w' \in Y_0$, and duplicator has a response $w \in X_0$. By definition of X_0 there is a $(w, \rho) \in X$. In S', duplicator replies with choosing such a (w, ρ).

The second case is that spoiler plays $(w_i, \rho_i) \in V(\mathcal{H}_i)$, in an **ungraded up** move, $i \in \{1, 2\}$. By Invariant (ii), the length $\ell - m$-suffixes of $\xi_1(u_1)$ and $\xi_2(u_2)$ are identical. By Point 4 and since the choice of (w_i, ρ_i) shows that (u_i, ξ_i) has a predecessor in $\mathcal{H}_i$, also (u_{3-i}, ξ_{3-i}) has a predecessor (w_{3-i}, ρ_{3-i}) in $\mathcal{H}_{3-i}$. Moreover, by (ii) and Point 4, the length $\ell - (m+1)$-suffix of $\rho_{3-i}(w_{3-i})$ is independent of which of the possibly many predecessors we choose and in particular, Invariant (ii) is again satisfied. It is also easy to see that Invariant (ii) being satisfied implies that so is Invariant (i): as the 0-suffixes of $\rho_1(w_1)$ and $\rho_2(w_2)$ must be identical, it follows from Point 4 that $\mathsf{tp}^{c,\ell}_{\mathcal{H}_1}((w_1, \rho_1)) = \mathsf{tp}^{c,\ell}_{\mathcal{H}_2}((w_2, \rho_2))$ and from Point 1 and Lemma 20 that $\mathsf{tp}^{c,\ell}_{\mathcal{G}_1}(w_1) = \mathsf{tp}^{c,\ell}_{\mathcal{G}_2}(w_2)$.

At this point, it is not hard to verify that the described strategy S' is indeed winning. □

Using Lemmas 25, 27, and 28, we can now complete the proof of Theorem 9.

Theorem 9. *For every* FO*-formula* $\varphi(x)$ *over* Π*, the following are equivalent:*

1. φ *is invariant under* $\sim_{G\%}$*;*
2. φ *is equivalent to a* GML + G*-formula over all (finite!)* Π*-labeled pointed graphs.*

Proof. "1 ⇒ 2". If φ is invariant under $\sim_{G\%}$, then by Lemma 25, there are c, ℓ such that φ is invariant under $\approx^{c,\ell}_G$. By Lemma 27, φ is then also invariant under $\approx^{c\downarrow,\ell}_G$. It then follows from Lemma 28 that there are also c', ℓ' such that φ is invariant under $\sim^{c',\ell'}_G$.

By Lemma 22, $\sim^{c',\ell'}_G$ has a finite number of equivalence classes, and each equivalence class can be defined using a GML + G-formula. Thus, a $\varphi' \in$ GML + G with $\varphi \equiv \varphi'$ can

be obtained by taking the disjunction of the formulae that define the equivalence classes which are models of φ.

"2 $\Rightarrow$ 1": If $\varphi \equiv \varphi'$ for some $\varphi' \in \mathrm{GML} + \mathrm{G}$, then by Lemma 21, there are c, ℓ (which can be determined from φ') such that φ is invariant under $\sim_G^{c,\ell}$. Thus, by Lemma 20, φ is also invariant under $\sim_G$ and $\sim_{G\%}$. □

A slight variation of the above proof also shows the following.

Corollary 29. *For every* FO*-formula* $\varphi(x)$ *over* Π*, the following are equivalent:*

1. φ *is invariant under* $\sim_G$*;*
2. φ *is equivalent to a* GML + G*-formula over all (finite!)* Π*-labeled pointed graphs.*

To see this, note that every FO-formula that is invariant under $\sim_G$ is also invariant under $\sim_{G\%}$, so the "1 $\Rightarrow$ 2" direction follows from Theorem 9. The "2 $\Rightarrow$ 1" direction simply follows from the fact that all GML + G-formulae are invariant under $\sim_G$.

We note that Theorem 9 and Corollary 29 also have interesting applications regarding GNNs with global readout. In particular, Barceló et al. (2020) left open the non-trivial question of the precise expressive power of GNNs with global readout, relative to FO; see also Hauke and Walega (2025). It is easy to see that GNN+Gs are invariant under $\sim_G$ (recall that GNN+G refers to GNNs with *non-counting* global readout), and thus Corollary 29 together with a straightforward translation of GML + G into GNN + G in the style of Barceló et al. (2020) (see also the proof of Lemma 6) yields the following.

Corollary 30. *Relative to* FO*,* GNN + G *and* GML + G *have the same expressive power.*

Another relevant variation is to consider arithmetic mean as the aggregation function for global readout and to stick with sum for local aggregation. Let us use GNN+GM to refer to this version of GNNs. One can verify that GNN+GM is invariant under $\sim_{G\%}$. Thus, Theorem 9 together with a straightforward variation of the proof of Lemma 6 yields the following.

Corollary 31. *Relative to* FO*,* GNN+GM *and* GML + G *have the same expressive power.*

The expressive power of GNN + GC remains an interesting open problem.

B.4 Proof of Theorem 10

Theorem 10. *Relative to* FO*, the following have the same expressive power:* PL + G*, soft-attention* GT*s, average hard-attention* GT*s.*

To capture the FO vertex properties that can be expressed by GTs, we define an equivalence relation on graphs under which GTs are invariant.

Definition 32 (Label-ratio equivalence)**.** Two Π-labeled pointed graphs $(\mathcal{G}_1, v_1)$, $(\mathcal{G}_2, v_2)$ are **label-ratio equivalent**, written $(\mathcal{G}_1, v_1) \sim_{\lambda\%} (\mathcal{G}_2, v_2)$, if $\lambda_1(v_1) = \lambda_2(v_2)$ and there exists a rational number $q > 0$ such that for all $t \subseteq \Pi$,

$$|\{v \in V(\mathcal{G}_1) \mid \lambda_1(v) = t\}| = q \cdot |\{v \in V(\mathcal{G}_2) \mid \lambda_2(v) = t\}|.$$

Intuitively, Definition 32 is that of global-ratio graded bisimilarity without the **graded forth** and **graded back** conditions of graded bisimulations. Directly from the definitions of the two relations, one can show that invariance under $\sim_{\lambda\%}$ implies invariance under $\sim_{G\%}$.

In the proof of Lemma 8, the **graded forth** and **graded back** conditions are only used in the part of the proof that is concerned with message passing modules. As GTs do not contain message passing modules, the proof of Lemma 8 also yields the following.

Proposition 33. *Let T be a soft-attention or average hard-attention* GT*. Then T is invariant under $\sim_{\lambda\%}$.*

We now show a counterpart of Theorem 9 for $\sim_{\lambda\%}$.

Lemma 34. *Every* FO*-formula $\varphi(x)$ that is invariant under $\sim_{\lambda\%}$ is also invariant under $\sim_G^{0,0}$.*

Proof. Let $\varphi(x)$ be an FO-formula that is invariant under $\sim_{\lambda\%}$. Further assume, to the contrary of what we have to show, that there are Π-labeled pointed graphs $(\mathcal{G}_1, v_1)$, $(\mathcal{G}_2, v_2)$ such that $(\mathcal{G}_1, v_1) \sim_G^{0,0} (\mathcal{G}_2, v_2)$, $\mathcal{G}_1 \models \varphi(v_1)$, and $\mathcal{G}_2 \not\models \varphi(v_2)$.

Since φ is invariant under $\sim_{\lambda\%}$, it is also invariant under $\sim_{G\%}$. We can thus first apply Lemma 25 to conclude that there are c, ℓ such that φ is invariant under $\approx_G^{c,\ell}$, Lemma 27 to conclude that φ is invariant under $\approx_G^{c\downarrow,\ell}$ and then Lemma 28 to conclude that there are c', ℓ' such that φ is invariant under $\sim_G^{c',\ell'}$.

From $\mathcal{G}_1$ and $\mathcal{G}_2$, we now construct graphs $\mathcal{H}_1$, $\mathcal{H}_2$ such that

1. $(\mathcal{G}_i, v_i) \sim_{\lambda\%} (\mathcal{H}_i, v_i)$ for all $i \in \{1, 2\}$;
2. $(\mathcal{H}_1, v_1) \sim_G^{c',\ell'} (\mathcal{H}_2, v_2)$.

This shows the desired contradiction. Point 1 and the fact that $\varphi(x)$ is invariant under $\sim_{\lambda\%}$ implies that $\mathcal{H}_1 \models \varphi(v_1)$ and $\mathcal{H}_2 \not\models \varphi(v_2)$. This then contradicts that φ is invariant under $\sim_G^{c',\ell'}$ and Point 2.

For $\mathcal{G}_1 = (V_1, E_1, \lambda_1)$ and $\mathcal{G}_2 = (V_2, E_2, \lambda_2)$, the graphs $\mathcal{H}_1$ and $\mathcal{H}_2$ can simply be obtained by setting $\mathcal{H}_1 = (V_1, \emptyset, \lambda_1)$ and $\mathcal{H}_2 = (V_2, \emptyset, \lambda_2)$, that is, removing all edges from $\mathcal{G}_1$ and $\mathcal{G}_2$. As $\sim_{\lambda\%}$ only considers λ_1 and λ_2, one can then verify that Point 1 holds, via the rational number $q = 1$. For Point 2, observe that no vertex in $\mathcal{H}_1$ and $\mathcal{H}_2$ has any successors, which means that there is no position in which spoiler can make any moves. As in addition $(\mathcal{G}_1, v_1) \sim_G^{0,0} (\mathcal{G}_2, v_2)$ implies that $(\mathcal{H}_1, v_1) \sim_G^{0,0} (\mathcal{H}_2, v_2)$, Point 2 follows. □

Lemma 35. *For any* FO*-formula $\varphi(x)$ over Π, the following are equivalent:*

1. *φ is invariant under $\sim_{\lambda\%}$;*
2. *φ is equivalent to a* PL + G*-formula φ' over all Π-labeled pointed graphs.*

Proof. "1 ⇒ 2". If φ is invariant under $\sim_{\lambda\%}$, then by Lemma 34, φ is also invariant under $\sim_G^{0,0}$. By Lemma 22, $\sim_G^{0,0}$ has a finite number of equivalence classes, and each equivalence class can be defined using a PL + G-formula. Thus, a PL + G-formula φ' with $\varphi \equiv \varphi'$ can be obtained by taking the disjunction of the formulae that define the equivalence classes that contain pointed graphs which satisfy φ.

"2 ⇒ 1". If $\varphi \equiv \varphi'$ for some $\varphi' \in$ PL + G, then by Lemma 21, φ is invariant under $\sim_G^{0,0}$. Thus, φ is also invariant under $\sim_{\lambda\%}$. □

The first direction of Theorem 10 can now be proved by combining Proposition 33 with Lemma 35. For the second direction, observe that in the proof of Lemma 6, message passing layers are only required to express subformulae of the form $\Diamond_{\geq k}\varphi$. All other types of subformulae can be expressed solely with MLPs and (soft-attention or average hard-attention) self-attention heads. Hence, for every PL + G-formula, there is also an equivalent GT.

C Floating-point preliminaries

C.1 Floating-point numbers and arithmetic

Here we define more formally the floating-point arithmetic operations used in this paper.

Let $\mathcal{F}$ be a floating-point format. The arithmetic operations $+$, $-$, $\cdot$ and $\div$ over a floating point format $\mathcal{F}$ are functions of the form $\mathcal{F} \times \mathcal{F} \to \mathcal{F}$ and they are computed as follows. Let $\star$ be one of these operations. First, as discussed in the main section, if one of the inputs is NaN, the output is also NaN. Otherwise, we take the precise operation of $\star$ w.r.t. the real arithmetic extended with ∞ and $-\infty$ and then round the precise result by using the "round to nearest, ties to even" method, which means that we round to the nearest number in the format $\mathcal{F}$ as though there was no upper bound for the exponent (i.e., we allow numbers that exceed the maximum exponent in $\mathcal{F}$). With ties we round to the number with an even least significant digit. If the maximum exponent is exceeded, we round to ∞ or $-\infty$ depending on the sign. Analogously, we define the arithmetic operation $\sqrt{x}$ over $\mathcal{F}$ which is a function of the type $\mathcal{F} \to \mathcal{F}$. Moreover, in the case where the operation leads to an undefined number, i.e., $\frac{\pm\infty}{\pm\infty}$, $\frac{\pm\infty}{\mp\infty}$, $0 \cdot \pm\infty$, $\pm\infty \mp \infty$, the output is NaN. In the IEEE754 standard, it is suggested that these operations, $+$, $-$, $\cdot$, $\div$ and $\sqrt{x}$, are taken as basic operations, and other operations can be defined in terms of these operations (or taken directly as basic operations).

We already discussed an implementation of the average hard-attention function with floats, where the denominator is obtained via the floating-point sum. Another possible technique would be to calculate the denominator by rounding the real size of the set into $\mathcal{F}$ directly, but it is not clear whether integers greater than any float in $\mathcal{F}$ should round down to the greatest non-infinite float in $\mathcal{F}$ or result in overflow. Our characterizations hold in the former case, which we will discuss in the proofs, and we leave the latter as an interesting open question.

We note that our definition of floating-point formats includes two floats that represent the number zero: one with a positive sign and one with a negative sign. For the operations, we preserve the sign in the preceding calculations if possible, and otherwise default to positive zero. Whether we include one or both zeros in formats does not change our results on expressive power.

We assume that the exponential function $\exp(x)$ over $\mathcal{F}$ is defined by using range reductions and polynomial approximations; this is done in a similar way as in (Tang, 1989) and in the popular math library fdlibm as follows.[9] Informally, the algorithm consists of four steps:

1. **Check for exceptions:** If x is too large, the output is ∞, and if x is too small, the output is 0. When x is too close to zero, the output is 1. If x is NaN, the output is NaN.

[9]Moreover, the exponent function was analyzed from the circuit complexity perspective and defined in an analogous way in (Hesse et al., 2002; Chiang, 2025).

2. **Range reduction:** Given an $x \in \mathcal{F}$, we first compute an integer $k = \lfloor \frac{x}{\ln 2} \rfloor \in \mathcal{F}$ and $r = x - k \log 2$.
3. **Polynomial approximation:** We approximate $\exp(r)$ by using the Taylor approximation of $\exp(x)$ of a small degree. As shown in (Nilsson et al., 2014), even the Taylor series of $\exp(x)$ of degree 6 suffices for accurate results with small input values. Another popular choice is the Remes algorithm to find a small polynomial that approximates $\exp(x)$ with small input values. Thus, we can assume that the polynomial used for the approximation is fixed and has some constant degree $c \in \mathbb{N}$.
4. **Combination:** We set that $\exp(x) = 2^k \cdot \exp(r)$.

Step 3 is often evaluated by using addition, multiplication and Horner's rule, i.e., a polynomial

$$a_0 + a_1 x + \cdots + a_n x^n,$$

over $\mathcal{F}$, where $a_0, \ldots, a_n \in \mathcal{F}$, is evaluated as

$$a_0 + x\big(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n) \cdots)\big).$$

In the last step of the algorithm, 2^k is trivial to compute, since k is an integer and the base of the format is 2. More complicated implementations are also possible for us, e.g., we could directly implement the source code of the exponent function of the math library fdlibm.

C.2 Interpreting labeling functions as float feature maps

In this section, we discuss another type of translation for floats, where the labeling of a graph may be interpreted in a different way.

Given a floating-point format $\mathcal{F}(p, q)$, an $(\mathcal{F}^d, \Pi)$**-labeled graph** (V, E, λ) refers to a Π-labeled graph with $|\Pi| = d(p+q+1)$; in this case, the labeling function λ can be identified with a floating-point feature map $\lambda_{\mathcal{F}} \colon V \to \mathcal{F}^d$ in the following natural way. We first split Π into d subsets $P_1, \ldots, P_d$ of equal size such that P_1 contains the $p+q+1$ least elements of Π (with respect to $<^{\Pi}$), P_2 contains the next $p+q+1$ elements, and so forth. For each P_i, we can order its elements with respect to $<^{\Pi}$ into a sequence s_i and interpret this sequence in each vertex as a floating-point number over p and q based on which symbols the vertex is labeled with. For each such float (which may be neither normalized, subnormalized, ∞, $-\infty$ nor NaN), there is a corresponding float in $\mathcal{F}$ that is interpreted as the same real value. Thus, $(s_1, \ldots, s_d)$ can be interpreted as a vector in $\mathcal{F}^d$ depending on which symbols a given vertex is labeled with. We can leave Π implicit and refer to $(\mathcal{F}^d, \Pi)$-labeled graphs as $\mathcal{F}^d$-labeled graphs. Thus, a computing model over $\mathcal{F}$ (e.g., a GPS[F]-network or a GT[F]) can run over such a graph by interpreting its labeling function as a floating-point feature map, instead of transforming the labeling function into the corresponding binary valued feature map.

C.3 Other classification heads

Here we discuss definitions for types of classification other than Boolean vertex classification. In particular, we consider Boolean graph classification, non-Boolean vertex classification and non-Boolean graph classification.

First, we consider graph classification as opposed to vertex classification. A **Boolean graph classification head** is a readout gadget where the MLP is a Boolean vertex classification head. As with Boolean vertex classifiers, note that the MLP of the readout gadget

is not assumed to be ReLU-activated, meaning that it can use, e.g., step functions. For example, (Rosenbluth et al., 2024) use readout gadgets as final classification heads when graph classification tasks are considered.

Both Boolean vertex classifiers and Boolean graph classifiers can be generalized further. Any MLP can be considered a **(general) vertex classifier**. Likewise, any readout gadget can be considered a **(general) graph classifier**. Here instead of 0 or 1, we classify vertices and graphs using feature vectors, i.e., the output dimension can be any positive integer.

Now we can modify all the variants of GPS-networks, GTs and GNNs by replacing the Boolean vertex classifiers with any of the classifiers discussed here with the same input dimension as the Boolean vertex classifier. The above classifiers are defined analogously for floats, and we can modify GT[F]s, GPS[F]-networks and GNN[F]s analogously.

We also define fragments of our logics that exclusively define graph properties. Let $\mathcal{L}$ be one of the logics discussed in Section 2.3. The set of **Π-formulae ψ of $\mathcal{L}^*$** is defined according to the following grammar:

$$\psi ::= \langle \mathsf{G} \rangle_{\geq k} \varphi \mid \neg \psi \mid \psi \wedge \psi \mid \langle \mathsf{G} \rangle_{\geq k} \psi,$$

where φ is a Π-formula of $\mathcal{L}$. The semantics of $\mathcal{L}^*$ is defined in the natural way.

C.4 Equivalence of graph classifiers and general classifiers

Here we introduce concepts of equivalence and expressive power that account for graph classification and non-Boolean classification.

A **graph property** is simply a vertex property $\lambda \colon V \to \{0,1\}$ such that for some $b \in \{0,1\}$, $\lambda(v) = b$ for all $v \in V$. The concepts of equivalence between learning models and logics from Section 2.4 extend in a natural way for graph classification by considering graph properties and graph classifiers instead of vertex properties and vertex classifiers.

An **m-ary feature update over** Π is simply an isomorphism invariant mapping U that takes a Π-labeled graph $\mathcal{G} = (V, E, \lambda)$ as input and outputs a new feature map $\lambda' \colon V \to \{0,1\}^m$. In the case of floats, given a floating-point format $\mathcal{F}$ and $p, q \in \mathbb{N}$, an **m-ary feature update over** $\mathcal{F}^p$ is simply an isomorphism invariant mapping $U^p_{\mathcal{F}}$ that takes an $\mathcal{F}^p$-labeled graph $\mathcal{G} = (V, E, \lambda)$ as input and outputs a new feature map $\lambda' \colon V \to \mathcal{F}^m$. In the case with floats we can leave the set of vertex labels implicit and omit it. From the perspective of logics, a feature update is just a query.

Note that our computing models with general classification heads (e.g., GPS-networks and GTs with non-Boolean vertex classification heads) are essentially just feature updates. Analogously, sequences of formulae of our logics can be seen as classes of feature updates, i.e., a sequence $(\varphi_1, \ldots, \varphi_k)$ of Π-formulae of a logic $\mathcal{L}$ defines a k-ary feature update over Π.

Let $\mathrm{id}_{\mathcal{F}} \colon \{0,1\}^* \to \mathcal{F}^*$ be a function that maps each binary string $\mathbf{b}$ to a floating point string $\mathbf{f}$ of equal length such that $\mathbf{f}(i) = b_{\mathcal{F}}$ iff $\mathbf{b}(i) = b$, where $b_{\mathcal{F}}$ denotes the corresponding float string of b. Given a feature update U_1 over Π and a feature update U_2 over $\mathcal{F}^d$, we say that U_2 is **equivalent to** U_1 (w.r.t. $\mathrm{id}_{\mathcal{F}}$), if for each Π-labeled graph (V, E, λ), the feature map $U_2(V, E, \mathrm{id}_{\mathcal{F}}(\lambda))$ is the same as $\mathrm{id}_{\mathcal{F}}(U_1(V, E, \lambda))$. Respectively, a feature update U_1 over Π is **equivalent to** a feature update U_2 over $\mathcal{F}^d$, if U_1 defines the same feature update as U_2 over Π-labeled graphs (recall that floats are just binary strings).

Given a class $\mathcal{L}$ of feature updates over Π (e.g. sequences of formulae of a logic) and a class $\mathcal{C}$ of feature updates over $\mathcal{F}^d$ for any d (e.g. GTs with non-Boolean vertex classification

heads), we say that $\mathcal{L}$ and $\mathcal{C}$ **have the same expressive power** (w.r.t. the general classification), if for each feature update $U \in \mathcal{L}$ there is an equivalent feature update $U' \in \mathcal{C}$, and vice versa. Analogously, two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ of computing models **have the same expressive power**, if for each feature update $U_1 \in \mathcal{C}_1$ there is a feature update $U_2 \in \mathcal{C}_2$ that defines the same feature update, and vice versa.

D Proofs for Section 4

In this section, we give the full proofs of the results in Section 4. The translations in this section that are given in terms of vertex classification also generalize for graph classification. In the case of logics, this means replacing the logic $\mathcal{L}$ appearing in a result with the logic $\mathcal{L}^*$[10] (see Appendix C.3 for the definition of $\mathcal{L}^*$ and Appendix D.4 for the formal analysis how our results generalize for graph classification). The translations that are given in terms of Boolean classification also generalize for general classifiers (see Appendix D.4). In the case of logics, this means considering sequences of formulae instead of a single formula.

Regarding the results of this section where we translate logic formulae into GT[F]s, GPS[F]-networks, etc., we make one very important assumption, namely that that the inputs of GT[F]s, transformer layers, GPS[F]-networks, GPS-layers, GNN[F]s, message-passing layers, MLP[F]s, perceptron layers, etc. *are always non-negative*. Their outputs may contain negative values, but such values are always turned non-negative before they are given as input to anything else. Note that this does not restrict our translations from logics to computing models, since when simulating formulae, our architectures start with floats corresponding to 1s and 0s.

D.1 Proof of Theorem 14

In this section, we give the proof of Theorem 14.

Theorem 14. *The following have the same expressive power:* $\mathrm{PL} + \mathrm{GC}$*, soft-attention* $\mathrm{GT[F]}$*s and average hard-attention* $\mathrm{GT[F]}$*s (and* $\mathrm{GNN} + \mathrm{GC[F]}$*s with constant local aggregation functions). This also holds when the* $\mathrm{GT[F]}$*s and* $\mathrm{GNN} + \mathrm{GC[F]}$*s are simple.*

We start by proving the following lemma.

Lemma 36. *The following have the same expressive power (w.r.t. general vertex classification): sequences of* PL*-formulae,* $\mathrm{MLP[F]}$*s and* ReLU*-activated* $\mathrm{MLP[F]}$*s.*

We first show the translation from MLP[F]s to PL. This translation is given in terms of general classification, as described in Appendix C.3, as MLPs do not give Boolean vertex classifications by default.

Lemma 37. *For each* $\mathrm{MLP[F]}$*, we can construct an equivalent sequence of* PL*-formulae. For each Boolean vertex classifier on floats, we can define an equivalent* PL*-formula.*

Proof. This follows from the Boolean completeness of PL. First, consider that an MLP only performs floating-point operations locally, i.e., it does not involve communication between

[10] Alternatively, we could simply restrict the concept of equivalence to those formulae of $\mathcal{L}$ that define graph properties, but $\mathcal{L}^*$ gives us a proper syntax.

vertices. Thus, the MLP can be expressed as a function $f_{\mathcal{F}} \colon \mathcal{F}^n \to \mathcal{F}^m$, where n is the input dimension and m the output dimension of the MLP. Since floats are bit strings, this means we can interpret $f_{\mathcal{F}}$ as a function $f_{\mathbb{B}} \colon \mathbb{B}^{kn} \to \mathbb{B}^{km}$, where k is the number of bits in the floats in $\mathcal{F}$. Since each bit in the output of $f_{\mathbb{B}}$ can be expressed as a Boolean combination of the bits in the input, it follows that a sequence of PL-formulae can simulate the MLP. In the case of Boolean vertex classifiers, a single formula is enough. □

We next show the translation from PL to ReLU-activated MLPs, which can also be derived from Theorem 15 in (Ahvonen et al., 2024a).

Lemma 38. *For each* Π*-formula* φ *of* PL*, we can construct an equivalent real or floating-point* ReLU*-activated Boolean vertex classification head. Moreover, for each sequence of* PL*-formulae, we can construct an equivalent* ReLU*-activated* MLP*.*

Proof. The idea is to split φ into its subformulae and calculate them one at a time in successive layers of the MLP.

Let $\psi_1, \ldots, \psi_d$ be an enumeration of the subformulae of φ such that $\psi_1 = \varphi$. We construct the MLP M as follows. All hidden dimensions of M are d; the ith component intuitively corresponds to ψ_i. The number of hidden layers is the formula depth of φ and we calculate one formula depth per layer.

The first layer is constructed as follows. If ψ_i is a proposition symbol, then M performs the identity transformation to that proposition symbol while placing it in the correct component. If $\psi_i = \top$, then the ith component becomes 1. All other components become zero.

Next, consider the ℓth hidden layer. If the formula depth of ψ_i is not ℓ, then the ith component is copied from the previous layer. Otherwise, if $\psi_i = \neg\psi_j$ for some j, then the ith component is obtained by multiplying the jth component by -1 and adding bias 1. If $\psi_i = \psi_j \wedge \psi_k$ for some j and k, then the ith component is obtained by multiplying the jth and kth components and adding them together with the bias -1. It is simple to prove by induction that the vector component corresponding to the truth value of a particular formula becomes correctly calculated after a particular layer of the MLP, with the first component (which corresponds to φ) becoming correctly calculated after the second-to-last layer.

For the final layer, we copy only the value of φ from the previous layer (i.e., we copy the first component and ignore the rest). □

Now we move our focus to transformers. We start by establishing that the softmax function saturates over floats in a similar way as the sum of a multiset of floats (recall Proposition 12). For a vector $\mathbf{v}$, let $\mathbf{v}^+_k$ denote the set of all vectors obtained from $\mathbf{v}$ by adding additional components (to the right of $\mathbf{v}$) containing elements that already appear at least k times in $\mathbf{v}$. For example, if $\mathbf{v} = (2, 3, 2, 1, 3, 3)$ and $k = 2$, then $(2, 3, 2, 1, 3, 3, 2, 2)$ and $(2, 3, 2, 1, 3, 3, 3, 3, 2)$ are in $\mathbf{v}^+_2$, but $(2, 3, 2, 1, 3, 3, 1, 2, 2)$ is not in $\mathbf{v}^+_2$. We let $\mathrm{softmax}_{\mathcal{F}}$ and $\mathrm{AH}_{\mathcal{F}}$ denote the implementations of softmax and AH with floats in the floating-point format $\mathcal{F}$ as described in Section 4.2.

Proposition 39. *For all floating-point formats* $\mathcal{F}$*, there exists a* $k \in \mathbb{N}$ *such that for all vectors* $\mathbf{v}$ *over floats in* $\mathcal{F}$*, we have* $\mathrm{softmax}_{\mathcal{F}}(\mathbf{v})_i = \mathrm{softmax}_{\mathcal{F}}(\mathbf{u})_i$ *and* $\mathrm{AH}_{\mathcal{F}}(\mathbf{v})_i = \mathrm{AH}_{\mathcal{F}}(\mathbf{u})_i$ *for all* $\mathbf{u} \in \mathbf{v}^+_k$*, where* i *ranges over the components of* $\mathbf{v}$*.*

Proof. This follows in a straightforward way from Proposition 12, since adding more elements to $\mathbf{v}$ only affects the denominator in the equation of softmax and AH, and the denominator consists of the saturating sum $\mathrm{SUM}_{\mathcal{F}}$.

Next, we discuss the alternate method of implementing the average hard-attention with floats discussed in Appendix C.1, where the denominator is obtained by rounding the real size of the set $\mathcal{I}_{\mathbf{v}}$ into the floating-point format directly. We only need to know one of each float to determine which of the elements of $\mathbf{v}$ should map to a non-zero value. Furthermore, to determine the values in these positions, only a bounded number of positions in $\mathbf{v}$ can have the largest value in $\mathbf{v}$ before the denominator in the description of AH becomes the largest finite float in $\mathcal{F}$, at which point further instances of that number will not change $\mathrm{AH}_{\mathcal{F}}(\mathbf{v})$. □

As we have already given the translation from MLP[F]s to PL, we next show a translation from attention modules on floats to PL + GC. The translation is given in terms of general vertex classification as defined in Section C.3, since attention modules do not give Boolean vertex classifications.

Lemma 40. *For each floating-point soft or average hard-attention module, there exists an equivalent sequence of* PL + GC*-formulae.*

Proof. Intuitively, to simulate an attention module SA, due to Proposition 12, it is enough for each vertex to know how many of each float feature vector occurs in the graph up to some bound that only depends on the floating-point format, not on the size of the graph. We will analyze the steps of the attention module. During the construction, we will also use FO instead of PL + GC to simulate small steps of a given attention head, but ultimately we will get rid of FO and turn its formulae into PL + GC.

Let $\mathcal{G} = (V, E, \lambda)$ be a graph, let X be the feature matrix of $\mathcal{G}$ and let $\mathcal{F}$ be the floating-point format of SA. Let n be the number of vertices and d the dimension of feature vectors in $\mathcal{G}$, let d_h be the hidden dimension and k the number of attention heads in SA and let W_O be the output matrix as it appears in the definition of attention modules.

We start by considering a single attention head H in SA with weight matrices W_Q, W_K and W_V as they appear in the definition of attention heads. First, consider the matrix products XW_Q, XW_K and XW_V; since floating-point formats are finite and these matrix products do not involve communication between vertices, they can be expressed as PL-formulae by the Boolean completeness of PL by having a single formula per column of the product. For brevity, we let $Q = XW_Q$, $K = (XW_K)^{\mathsf{T}}$, $V = XW_V$ and $O = W_O$. Now, let Y denote $(XW_Q)(XW_K)^{\mathsf{T}}$, let Z denote $\frac{Y}{\sqrt{d_h}}$ and let S denote $\mathrm{softmax}(Y)$.

First, consider the matrix product Y of Q and K. We know that $Y_{i,j} = \sum_{\ell=1}^{d_h} Q_{i,\ell} K_{\ell,j}$. Intuitively, the matrix Y can be viewed as an edge-labeled graph with vertices $[n]$ obtained from Q and K, where the edge (i, j) is weighted with $Y_{i,j}$. In terms of FO, each edge-weight can be expressed as a sequence $(\varphi_1(x, y), \ldots, \varphi_m(x, y))$ of quantifier-free formulae, where m is the length of floating-point numbers in $\mathcal{F}$. Intuitively, $\varphi_i(x, y)$ encodes the ith bit of the float between the vertices x and y. Since quantifier-free FO is Boolean-complete, such a sequence of formulae clearly exists. In later steps, we will turn this sequence of FO-formulae into a sequence of PL + GC-formulae.

Second, consider the matrix Z. We obtain $Z_{i,j}$ as $\frac{Y_{i,j}}{\sqrt{d_h}}$, and this division can be simulated by a sequence of quantifier-free FO-formulae (with two free variables) due to the Boolean completeness of FO.

Third, we consider the matrix S. We note that $S_{i,*}$ is obtained as $\mathrm{softmax}(Z_{i,*})$, which in turn is obtained as follows. Let $b_i = \max\{ Z_{i,j} \mid j \in [n]\}$. We get that $Z_{i,j} = \frac{e^{Z_{i,j}-b_i}}{\sum_{\ell=1}^{n} e^{Z_{i,\ell}-b_i}}$. (Note that due to Proposition 39, only a bounded number of each float in a vector needs to be known to calculate the output of softmax for each vertex.) Due to the Boolean completeness of FO, this step is possible to simulate by a sequence of quantifier-free FO-formulae (with two free variables). This step of the analysis can be done analogously for the case where the attention module uses average hard-attention instead of soft-attention.

Fourth, we consider the output $H(X) = SV$ of the attention head H. We obtain $H(X)_{i,j} = \sum_{\ell=1}^{n} S_{i,\ell} V_{\ell,j}$. Due to Proposition 12, it is enough to know a bounded number of pairs of floats $S_{i,\ell}$ and $V_{\ell,j}$ to compute $H(X)_{i,j}$. In terms of FO, this means that we quantify the sequences of formulae that encode S. However, since the number of floats is bounded, this is also possible by using the counting global modality. Thus, since S can be expressed as a sequence of quantifier-free FO-formulae (with two free variables), we can express $H(X)$ by using a sequence of formulae of $\mathrm{PL} + \mathrm{GC}$ instead of FO.

Fifth, we consider the final output $\mathrm{SA}(X) = \mathcal{H}(X)W_O$ of the attention module, where $\mathcal{H}(X)$ is the concatenation of $H_1(X), \ldots, H_k(X)$ where $H_1, \ldots, H_k$ are the attention heads of SA. Just like with XW_Q, XW_K and XW_V, this is easy to express due to the Boolean completeness of PL. □

Combining the translations from MLPs and attention modules to logics, we obtain a translation from GT[F]s to $\mathrm{PL} + \mathrm{GC}$.

Theorem 41. *For each floating-point soft or average hard-attention graph transformer, there exists an equivalent* $\mathrm{PL} + \mathrm{GC}$*-formula.*

Proof. This follows directly from Lemmas 37 and 40. □

Next, we show our translation from $\mathrm{PL} + \mathrm{GC}$ to GT[F]s. First note that while we have given a translation from PL to MLPs, there is no restriction on the number of layers of the MLP. Typically the MLPs appearing in graph transformers and GPS-networks are assumed to be simple as defined in Section 2.2, and the hidden dimension is restricted to at most twice the input/output dimension, though the dimension restriction is easy to work around by increasing the dimension of the surrounding architecture (for instance, if we want to build a GT[F]-layer with an MLP of hidden dimension d, we simply construct a GT[F] of dimension $2d$). Moreover, while we can translate PL to MLP[F]s by Lemma 38, we also have to give a translation from PL to GT[F]s, as the architecture of the GT[F]s could conceivably ruin the translation. However, we will show that there is no problem, as increasing the inner dimension of the computing models makes it possible to simulate them in the sense defined next.

We start by discussing the notion of *shifting a feature update* (for discussion on feature updates, see Appendix C.4). The intuition is that we construct a feature update of higher dimension that simulates the lower-dimension feature update but moves the result from the first elements of the input vector to the last elements of the output vector (or the other way around); the remaining elements of inputs and outputs are assumed to be zeros. For the formal definition, let $\mathcal{F}$ be a floating-point format, let $\mathfrak{G}[\mathcal{F}, d]$ be the class of $\mathcal{F}^d$-labeled graphs, let $L[\mathcal{F}, d]$ be the class of d-dimensional feature maps over $\mathcal{F}$ and let $d' \geq d$. For each $\mathcal{G} \in \mathfrak{G}[\mathcal{F}, d]$, let $\mathcal{G}_r$ (resp. $\mathcal{G}_\ell$) denote the $\mathcal{F}^{d'}$-featured graph obtained from $\mathcal{G}$ by adding $d' - d$ columns of zeros to the right (resp. left) of the feature matrix of $\mathcal{G}$. For

each $\lambda \in L[\mathcal{F}, d]$, we define λ_r and λ_ℓ analogously. Now, let $f\colon \mathfrak{G}[\mathcal{F}, d] \to L[\mathcal{F}, d]$ and $f'\colon \mathfrak{G}[\mathcal{F}, d'] \to L[\mathcal{F}, d']$ be feature updates. If for each $\mathcal{G} \in \mathfrak{G}[\mathcal{F}, d]$ we have $f'(\mathcal{G}_r) = f(\mathcal{G})_\ell$, then we say that f' **shifts** f **to the right** (by $d' - d$). Likewise if $f'(\mathcal{G}_\ell) = f(\mathcal{G})_r$ for each $\mathcal{G} \in \mathfrak{G}[\mathcal{F}, d]$, then we say that f' **shifts** f **to the left** (by $d' - d$). On the other hand, if $f'(\mathcal{G}_r) = f(\mathcal{G})_r$, then we say that f' is **prefix equivalent** to f.

Now, we show that a transformer layer can shift an MLP.

Lemma 42. *For each* MLP M *of I/O dimension* d*, we can construct a soft-attention or average hard-attention transformer layer* T *of dimension* $2d$ *that shifts* M *to the right (or left). If* M *is simple, then* T *is simple.*

Proof. We "skip" the attention module by using the skip connection wrapped around it and simulate M with the MLP while leveraging the increased dimension to neutralize the effect of the second skip connection.

We construct a transformer layer $T = (\mathrm{SA}, \mathrm{FF})$ that shifts M to the right, as shifting to the left is analogous. The module SA simply outputs a zero matrix (this is possible by setting W_O to be a zero matrix). Due to the skip connection wrapped around the attention module, FF now receives the same input as T. The MLP FF is obtained from M as follows. The number of layers of FF is the same as M, and we add d to the dimensions of each layer. The output of M is computed in identical fashion by FF, but the output is placed in the last d components of the output vector (which is possible via simple manipulations of the weight matrices). The extra dimensions in each layer are used to remember the first d components of the input of FF in each layer (i.e., we perform an identity transformation to them in each layer, which is possible because we have assumed that all inputs are non-negative and they are thus unaffected by ReLU). In the final layer these remembered values are multiplied by -1 and placed in the first d components of the output vector (there is no ReLU on the final layer, so these values of the output are non-positive). Now, if the last d columns of the feature matrix of the input were zero columns, then the skip connection wrapped around the MLP does not affect the last d components of the output and cancels out the first d components of the output. □

Now we can translate PL to GT[F]s, but not yet to simple GT[F]s. To translate to this simpler architecture, we need a way of breaking an MLP down into multiple simple MLPs, which can be carried out in a sequence. This simply means that the output of an MLP is given as input to the next MLP in the sequence.

Lemma 43. *For each* ReLU*-activated* MLP M *with* k *layers, I/O-dimension* d *and maximum hidden dimension* d_h*, we can construct a sequence* $(M_1, \ldots, M_{k-1})$ *of simple* MLP*s with I/O-dimension* $d' := \max\{d, d_h\}$ *that are, as a sequence, prefix equivalent to* M*.*

Proof. We simply separate the layers of the MLP and transform each one into a simple MLP.

The first layer of M_i performs the same transformation from the prefix of its input vector to the prefix of its output vector as the ith layer of M would (the I/O dimensions of the ith layer of M matching the lengths of the prefixes). The remaining components of the output vector of the first layer are zeros. The second layer then simply performs an identity transformation to each component, with the exception of M_{k-1}, where the second layer instead performs the same operation as the final layer of M (again w.r.t. prefixes). □

Lemmas 42 and 43 together mean that a k-layer MLP can be simulated by $k-1$ simple transformer layers. This is achieved through alternation by the odd layers shifting the simple MLPs to the right and even layers shifting them to the left or vice versa.

Now we are almost ready to show our translation from PL + GC to GT[F]s. Before this, we require a couple of lemmas showing that MLPs can check some simple binary conditions on inputs, i.e., whether an element of the input vector is greater than or equal to some specific float.

Lemma 44. *Let $\mathcal{F}$ be a floating-point format and let $F \in \mathcal{F}$. We can construct a 4-layer Boolean vertex classifier of I/O dimension $(d, 1)$ that for an input vector $(x_1, \ldots, x_d)$ outputs (1) if $x_i \geq F$ and (0) otherwise.*

Proof. We use ReLU and negative weights to flatten all values $x \geq F$ to 1 and all values $x < F$ to 0. With $\mathbb{R}$, this would not be possible, since there is no greatest $x \in \mathbb{R}$ such that $x < F$, but such a float exists in $\mathcal{F}$.

For the first layer of the MLP, we want to squish all values at least F to a single value. Working with $\mathbb{R}$ we could use weight -1 and bias F for the ith component (and 0 for others) to squish all values greater than F to 0, while all values less than F would become positive values. However, with floats this only works for some values of F, as performing this operation for the smallest number in the format (or greatest if F is negative) may result in overflow[11] for large enough values of F. If F is a large enough value to cause overflow, then we divide both the weight and bias by 2; since floats are in base 2, the resulting numbers are precisely representable in the format. After this first layer, the value is 0 if $x_i \geq F$ and otherwise some positive value.

Next, let f be the smallest positive floating-point number in $\mathcal{F}$. For the second layer, we want to squish positive values (i.e. the case where $x_i < F$) to a single value. We use the weight -1 and bias f. After this second layer, the value is f if $x_i \geq F$ and 0 otherwise.

It is now easy to define two more successive layers transforming f into 1 by using positive weights and biases 0. For example, if $e_{\max}$ denotes the greatest possible (non-biased) exponent in $\mathcal{F}$, then the weight of the first layer might be $0.10\cdots0 \times 2^{e_{\max}}$ transforming f into $0.10\cdots0 \times 2^{-p}$. The weight of the second layer could then be $0.10\cdots0 \times 2^{p+2}$ transforming $0.10\cdots0 \times 2^{-p}$ to 1. □

Lemma 45. *Let $\mathcal{F}$ be a floating-point format and let $F \in \mathcal{F}$. We can construct a 6-layer Boolean vertex classifier of I/H/O-dimension $(d, 2, 1)$ that for an input vector $(x_1, \ldots, x_d)$ outputs (1) if $x_i = F$ and (0) otherwise.*

Proof. We use the construction from the proof of Lemma 44 in the two hidden components. The first checks if $x_i \geq F$ and the second checks if $x_i \leq F$. We add one more layer to the MLP that takes the sum of these two components with bias -1 (and one more layer with an identity transformation because there is no ReLU on the final layer). □

We ease into our translation from PL + GC to GT[F]s by first considering a translation from PL + G to unique hard-attention GT[F]s as it uses a similar general strategy as the case for soft or average hard-attention, but is also both simpler and shorter. Note that, in the below theorem (and others like it), the final simple Boolean vertex classifier does not have to use any step functions, as the ReLU will suffice.

[11] Overflow means that the maximum exponent of the floating-point format is exceeded. In these cases, the operation will output ∞ or $-\infty$.

Theorem 46. *For each* PL + G*-formula, we can construct an equivalent simple unique hard-attention* GT[F].

Proof. The idea is to use a single transformer layer to simulate a single subformula of the PL + G-formula. We use the MLPs to simulate the operators of PL and transformer layers to simulate the non-counting global modality.

Let φ be a PL + G-formula and let $\psi_1, \ldots, \psi_d$ be an enumeration of the subformulae of φ including proposition symbols and $\top$ such that $\psi_1 = \varphi$. We construct an $\mathcal{O}(d)$-layer graph transformer over any floating-point format $\mathcal{F}$.

First, we give a general idea of the construction. The initial MLP transforms the feature vectors into vectors that have two components i and $2i$ for each subformula ψ_i. For the first of these, the MLP preserves the truth values of proposition symbols, setting the component corresponding to $\top$ to 1. All other components are set to 0 (including the last d components). The transformer layers calculate the truth values of the subformulae starting from simple subformulae and moving to more complex ones. The layer we construct depends on the subformula ψ_i under evaluation, i.e., on whether ψ_i is a PL-formula or of the type $\langle \mathsf{G} \rangle \psi_j$ for some $j \in [d]$. Both before and after each evaluation of a subformula, the feature matrix is a binary matrix where subformulae not yet calculated have a corresponding column of 0s and those already calculated have a corresponding column of 1s and 0s. (Additionally, either the first or last d columns are zero columns, as the evaluated values may shift from left to right and vice versa.) Finally, the classification head just copies the first element of the feature vector.

Now, we start the construction. The operators of PL are handled by the MLPs; this is possible due to Lemmas 38, 42 and 43 and is also what necessitates the hidden dimension $2d$ as the values shift from left to right or vice versa. All that is left is to define a transformer layer that simulates the non-counting universal modality. Assume that we have to simulate a formula of type $\psi_i := \langle \mathsf{G} \rangle \psi_j$. Given a matrix $X \in \mathbb{B}^{n \times d}$, assume that column k contains the truth values of $\top$ (i.e., column k is a column of only 1s). Now, let W_Q be the Boolean-valued $(d \times 1)$-matrix where only the kth row is 1. Likewise, W_K is the Boolean-valued matrix where only the jth row is 1. We obtain that

$$Y = \frac{XW_Q(XW_K)^\mathsf{T}}{\sqrt{1}}$$

is an $(n \times n)$-matrix, where each row contains the transpose of column j of X. Now, $Z = \mathrm{UH}(Y)$ gives an $(n \times n)$-matrix, where a single column contains only ones and others are zero columns; if there is at least one vertex where ψ_j is true, then the column of 1s corresponds to one such vertex. Let $W_V = W_K$. Now, $Z(XW_V)$ gives an $(n \times 1)$-matrix that contains only ones if ψ_j is true in at least one vertex and otherwise the vector is a zero vector. Finally, the matrix W_O is the $(1 \times d)$-matrix where exactly the ith element is 1 and others are 0s. Thus, the attention module outputs a matrix where the ith column contains the truth values of ψ_i and other columns are zero columns. The skip connection is used to recover all previously calculated columns. Finally we "skip" the MLP of the layer, i.e., the MLP multiplies every component by 0 and thus outputs a zero vector; the so-far calculated columns are then recovered via the skip connection of the MLP. □

Next, we show our translation from PL + GC to soft and average hard-attention GT[F]s. This case is naturally more involved than the case with unique hard-attention; the key insight involves leveraging floating-point underflow. We first recall the related Proposition 13.

Proposition 13. *Let $\mathcal{F}$ be a floating-point format, let f be the smallest positive float in $\mathcal{F}$ and let k be some even integer such that $\frac{k}{2}$ is accurately representable in $\mathcal{F}$. For all $F \in \mathcal{F}$, $|F| \leq |\frac{1}{k}|$ if and only if $F \cdot (\frac{k}{2} f) = 0$.*

Using the above proposition, we now give the translation from PL + GC to GT[F]s.

Theorem 47. *Given a formula of* PL + GC*, we can construct an equivalent simple soft-attention or simple average hard-attention* GT[F].

Proof. First, we describe the proof strategy informally. The idea is to use the MLP to handle Boolean connectives and self-attention to simulate counting global modalities $\langle \mathsf{G} \rangle_{\geq k}$. Due to Proposition 13, we can handle half the possible values of k by simply checking if underflow occurs in the output of a particular attention head; if the output is 0 (i.e., underflow occurs), it signals that the column of the input matrix has at least k 1s. For the other half of the possible values of k, the above attention head cannot identify if there are at least k 1s in a column, but it can check a nearby upper and lower bound for the number of 1s. For the two or three values falling between the bounds, we can use some additional numerical analysis to identify the number of 1s.

Now, we may start the formal proof. In this proof, addition, subtraction, multiplication and division are always assumed to be exact (i.e., $\frac{1}{k}$ refers to the precise value even if k is a floating-point number), and the rounding operations inherent in floating-point arithmetic are always made explicit wherever they need to be performed. Thus, for each $x \in \mathbb{R}$ we let $\mathit{round}(x)$ denote the rounded value of x in $\mathcal{F}$.

General architecture of the graph transformer

Let φ be a PL + GC-formula. We mostly follow the same general architecture as in the proof of Lemma 46 but with the following distinctions. We choose $\mathcal{F}$ to be a floating-point format such that for the maximum grade K appearing in the global modalities of φ, all integers $k \in [K]$ can be represented precisely in $\mathcal{F}$ and the number $\frac{1}{k}$ rounds to a different value for each $k \in [K]$. The hidden dimension is $2(d+4)$ (where d is the number of subformulae of φ including $\top$). The factor 2 is because we use Lemma 42 to simulate MLPs, which requires shifting. However, this means that there is always one half of the input which consists of 0s that do not cause problems in attention heads, so we may treat the construction as having dimension $d+4$ except where Lemma 42 is applied. The last four of the $d+4$ columns are auxiliary and used to help compute some of the other columns. The initial MLP sets these auxiliary positions to 0.

As in the proof of Lemma 46, each transformer layer again only focuses on a single subformula ψ_i of φ, but we may require more than a single transformer layer (in a row) per subformula of the type $\langle \mathsf{G} \rangle_{\geq k} \psi_j$. Again the transformer layer depends on whether the subformula ψ_i under evaluation is a PL-formula or of the form $\langle \mathsf{G} \rangle_{\geq k} \psi_j$ for some j, k. The operators of PL are again handled by the MLPs, made possible by Lemmas 38, 42 and 43, so all that is left is to define transformer layers that simulate the counting universal modality.

Simulating counting universal modalities:

If $\psi_i := \langle \mathsf{G} \rangle_{\geq k} \psi_j$ for some previously computed subformula ψ_j, then we make use of multiple transformer layers in a row. The argument has two main cases: the simple case is the one where k is an even number and $\mathit{round}(\frac{1}{k}) \leq \frac{1}{k}$ (i.e., $\frac{1}{k}$ does not round upward in $\mathcal{F}$), and the more complicated case is the one where $\mathit{round}(\frac{1}{k}) > \frac{1}{k}$. The cases where k is an odd number can be reduced to the two cases for even numbers, so we will only consider them

briefly.

The case $round(\frac{1}{k}) \leq \frac{1}{k}$:

First, consider the case where k is an even number other than zero and $round(\frac{1}{k}) \leq \frac{1}{k}$. We start by constructing a single transformer layer consisting of two self-attention heads. The first of these checks if there are at least k 1s in column j of the feature matrix, and it is constructed as follows.

1. Let $F \in \mathcal{F}$ be the greatest floating-point number in $\mathcal{F}$ such that $round(F^2) \neq \infty$. The query matrix W_Q and key matrix W_K are identical $((d+4) \times 1)$-matrices (i.e., vectors), where only the jth element is F and others are 0s. Thus, before softmax, we have an $(n \times n)$-matrix where each row is either a zero vector or the jth column of the Boolean input matrix X (multiplied by $round(F^2)$).
2. Next, before softmax, the rows are biased according to the maximum element on the row. For the zero rows, there is no change. For the other rows, each 0 is replaced with $-round(F^2)$ and each $round(F^2)$ is replaced with 0.
3. Now, applying softmax to a row of 0s gives a row of $round(\frac{1}{n'})$ where n' is the sum of a multiset of n 1s, where n is the number of rows in X. For the other rows, let ℓ be the number of 1s in column j of X, let M be a multiset of ℓ 1s and let ℓ' denote $\mathrm{SUM}_{\mathcal{F}}(M)$ where $\mathrm{SUM}_{\mathcal{F}}$ is the saturating sum from Proposition 12. The application of softmax will then give a row where each $-round(F^2)$ is replaced with 0 and each 0 is replaced with $round(\frac{1}{\ell'})$. This is because $e^{-round(F^2)}$ rounds to 0 and thus softmax gives an even probability distribution for the remaining positions, calculated by first taking the saturating sum of values $e^0 = 1$.
    - The analysis in step 3 above is identical for average hard-attention, when the denominator of AH is obtained as a multiset of 1s. On the other hand, consider the implementation of AH discussed in Appendix C.1, where the size of the set in the denominator is rounded into the floating-point format directly. In this case, the analysis is also identical except that $n' = \min\{round(n), F_{\max}\}$ and $\ell' = \min\{round(\ell), F_{\max}\}$, where n and ℓ are the same as above and $F_{\max}$ is the largest non-infinite floating-point number in the format. This is because, according to the implementation, n and ℓ are rounded into the format, but not to ∞.
4. Let f be the smallest positive floating-point number in $\mathcal{F}$. The value matrix W_V is defined in the same way as W_Q and W_K, except that we use $\frac{k}{2}f$ instead of F. Thus, the matrix product $V = XW_V$ gives the jth column of X where each 1 is replaced with $\frac{k}{2}f$.
5. The product of the $(n \times n)$-matrix N resulting from step 3 and the $(n \times 1)$-matrix V from step 4 thus multiplies each column where $round(\frac{1}{\ell'})$ appears with $\frac{k}{2}f$ and then takes the sum of each row. Now the matrix product gives us $round(\frac{1}{\ell'})\frac{k}{2}f$ which by Proposition 13 rounds to 0 exactly when $round(\frac{1}{\ell'}) \leq \frac{1}{k}$. Recall that $round(\frac{1}{k}) \leq \frac{1}{k}$ and since $\frac{1}{h}$ rounds to a different value for all $h \in [K]$ we must have $round(\frac{1}{k-1}) > \frac{1}{k}$. This means that $round(\frac{1}{\ell'}) \leq \frac{1}{k}$ is equivalent to $\ell' \geq k$. Now NV is a zero vector if and only if the formula ψ_j is true in at least k vertices of the graph (or if the formula is true in zero of the n vertices and $n \geq k$).

From the above constructed attention head, we see that we still have to check if zero vertices satisfy ψ_j; we do this with a second attention head. This second head will output a zero vector if and only if there are zero formulae that satisfy ψ_j; otherwise, the elements of the vector will all be some other singular value. For this attention head, we define the query

and key matrices as zero matrices, which results in the output of softmax being a matrix where each element is $round(\frac{1}{n'})$ for some $n' \leq n$ (this is because softmax calculates n' as a sum of a multiset of n 1s, and the sum of a multiset of floats saturates by Proposition 12). The value matrix W'_V is the $((d+4)\times 1)$-matrix where the jth row is 1 and others are zero; the matrix XW'_V is thus the jth column of X. The output of the attention head is thus a vector where each element is $\mathrm{SUM}_{\mathcal{F}}(M)$ where M is a multiset of ℓ copies of $round(\frac{1}{n'})$, where ℓ is the number of vertices satisfying ψ_j. Since $round(\frac{1}{n'})$ is never zero, this means that the output is a vector of 0s if $\ell = 0$ and otherwise each component is some positive value. By the same analysis, this construction works for average hard-attention. For the alternate implementation of AH discussed in Appendix C.1, the same analysis holds with the difference that $n' = \min\{round(n), F_{\max}\}$ where $F_{\max}$ is the greatest float in the format other than ∞.

Now, to check if $\ell \geq k$ (where ℓ is the number of 1s in column j of X), we simply have to check if the output of the first attention head is a zero vector and the output of the second attention head is a non-zero vector. Unfortunately, if $0 < \ell < k \leq n$, then the output of the first attention head may contain both 0s and non-zero values, meaning that some rows may currently be under a false impression that $\ell \geq k$. We will use a second transformer layer to distribute the necessary information between all vertices. First, we finish the current layer by defining the weight matrix W_O to store the output vectors of the two attention heads in two separate columns $d+1$ and $d+2$. Next, the MLP of this layer normalizes these two columns such that all positive values are replaced with 1s (this is possible by Lemma 44; if we desire a simple graph transformer, then we can simply use multiple consecutive layers to accomplish this task by Lemmas 42 and 43.)

The second layer uses the second attention head of the first layer, except that it now checks column $d+1$ instead of column j, i.e., the $(d+1)$th row of the value matrix is 1 and others are 0. If the column $d+1$ has at least one 1, then all elements of the output vector of the attention head become non-zero. The weight matrix W'_O places these values in the column $d+3$. Now, the MLP once again normalizes positive values in column $d+3$ to 1. Then, we use a third layer that does not do anything in the self-attention module, but the MLP checks that the values in the columns $d+2$ and $d+3$ are both 1. A simple sum of the values is enough; if $\ell \geq k$, then the sum is 1. Otherwise the sum is 0 or 2, because it is not possible that $\ell = 0$ and $0 < \ell < k$. This result is placed in the ith component. The MLP also resets columns $d+1$ through $d+4$ by multiplying them in the final layer of the MLP with -1 (whence they will be eliminated by the skip connection) such that these columns can be used again in later layers.

On the other hand, assume that k is an odd number and $round(\frac{1}{k-1}) > \frac{1}{k-1}$. Then $round(\frac{1}{k-1})\frac{k-1}{2}f \geq \frac{1}{2}f$ but $round(\frac{1}{k})\frac{k-1}{2}f < \frac{1}{2}f$ because we assumed that $\frac{1}{k-1}$ and $\frac{1}{k}$ round to different values in $\mathcal{F}$ and $\frac{1}{k} < \frac{1}{k-1}$. Thus, the above construction can now be used to check if at least k vertices satisfy ψ_j.

The case $round(\frac{1}{k}) > \frac{1}{k}$:

Now assume that k is an even number such that $round(\frac{1}{k}) > \frac{1}{k}$ (the case where k is an odd number such that $round(\frac{1}{k-1}) \leq \frac{1}{k-1}$ is similar). We can use the above construction to calculate $\langle \mathsf{G} \rangle_{\geq k+1}\psi_j$. We can also calculate either $\langle \mathsf{G} \rangle_{\geq k-1}\psi_j$ or $\langle \mathsf{G} \rangle_{\geq k-2}\psi_j$ depending on whether $round(\frac{1}{k-2}) > \frac{1}{k-2}$. Thus, in order to calculate $\langle \mathsf{G} \rangle_{\geq k}\psi_j$, we can simply calculate the upper bound $(k+1)$ and the lower bound $(k-1$ or $k-2)$, and then separate the remaining cases, i.e., we need a way of distinguishing the cases where the formula is satisfied in k, $k-1$ and $k-2$ vertices. To do this, we simply use the above construction for k. Recall that $round(\frac{1}{k}) > \frac{1}{k}$ and $\frac{1}{h}$ rounds to a different value for all $h \in [K]$. If $k-1$ is the lower

bound, then we have $\frac{1}{k-1} < round(\frac{1}{k-1}) < \frac{1}{k-2}$. Thus for all $k > 3$,

$$\begin{aligned}\frac{1}{2} = \frac{k}{2} \cdot \frac{1}{k} &< \frac{k}{2} \cdot round\left(\frac{1}{k}\right) \\ &< \frac{k}{2} \cdot round\left(\frac{1}{k-1}\right) < \frac{k}{2} \cdot \frac{1}{k-2} < \frac{3}{2}.\end{aligned}$$

Likewise, if $k-2$ is the lower bound, then we have $round(\frac{1}{k-2}) \leq \frac{1}{k-2}$ and thus for all $k > 3$,

$$\begin{aligned}\frac{1}{2} = \frac{k}{2} \cdot \frac{1}{k} &< \frac{k}{2} \cdot round\left(\frac{1}{k}\right) \\ &< \frac{k}{2} \cdot round\left(\frac{1}{k-1}\right) \\ &< \frac{k}{2} \cdot round\left(\frac{1}{k-2}\right) \leq \frac{k}{2} \cdot \frac{1}{k-2} < \frac{3}{2}.\end{aligned}$$

Thus, multiplying $round(\frac{1}{k-1})$ or $round(\frac{1}{k})$ with $\frac{k}{2}f$ gives f in each case (because we round to the nearest number) and likewise for $round(\frac{1}{k-2})$ when $k-2$ is the lower bound. Accordingly, the output vector is then $(k-2)f$, $(k-1)f$ or kf. The only even values of k this analysis does not account for are $k = 0$ and $k = 2$. Because $round(\frac{1}{2}) = \frac{1}{2}$, the case $k = 2$ does not need to be examined here, and we already showed in the previous case how to construct an attention head that checks for $k = 0$.

For this last case, we thus end up with four separate columns $d+1$ through $d+4$ that need to be combined; column $d+1$ determines if zero vertices satisfy ψ_j,column $d+2$ checks that at least $k-2$ or $k-1$ vertices satisfy ψ_j, column $d+3$ checks that at least $k+1$ vertices satisfy ψ_j and column $d+4$ gives a distinct value for the cases $k-2$, $k-1$ and k. To output 1, we thus want the column $d+1$ to be positive, the column $d+2$ to be 0, and either the column $d+3$ to be 0 or the column $d+4$ to be the distinguished value corresponding to k. Checking each of these conditions individually can be done by MLPs that only use the ReLU activation function by Lemmas 44 and 45 (i.e., for each condition the output will be 1 if it is cleared and 0 otherwise). Now, one more MLP layer can check that the conditions are met. by taking the sum of the components and adding the bias -2. We also add one more transformer layer that, as in the easier case, distributes the information (whether $\ell \geq k$) to all vertices, since only vertices where ψ_j is true know this. Finally, the MLP again resets the columns $d+1$ through $d+4$. This concludes the description for handling the counting global modality as all cases have been covered. □

As the final piece of Theorem 14, we characterize GNN + GC[F]s with constant local aggregation functions. We utilize Theorem 15 and though Theorem 15 is only proven in the next appendix, the proof of Theorem 15 is independent of the result below.

Theorem 48. GNN + GC[F]*s with constant local aggregation functions have the same expressive power as* PL + GC*. This also applies when the* GNN + GC[F]*s are simple.*

Proof. Theorem 15 states that GNN[F]s with counting global readout have the same expressive power as GML + GC. If the GNN[F]s have constant aggregation functions, then the translation in (Ahvonen et al., 2024b) (which we apply for Theorem 15) results in a PL + GC-formula. Likewise, the translation from GML + GC to GNN[F]s results in a GNN[F] with a constant local aggregation function if the formula is from PL + GC. □

D.2 Proof of Theorem 15

In this section, we provide the proof of Theorem 15.

Theorem 15. *The following pairs have the same expressive power (denoted by $\equiv$):*

$$\text{GNN[F]} \equiv \text{GML}, \text{GNN} + \text{G[F]} \equiv \text{GML} + \text{G} \text{ and } \text{GNN} + \text{GC[F]} \equiv \text{GML} + \text{GC}.$$

This also holds when each type of GNN[F] *is simple.*

First, we note that Theorem 3.2 in (Ahvonen et al., 2024b) showed that recurrent GNN[F]s and the logic GMSC have the same expressive power. A simple modification of the proofs of (Ahvonen et al., 2024b) gives us the following result.

Lemma 49. GNN[F]*s, simple* GNN[F]*s and* GML *have the same expressive power.*

Proof. Note that some of the concepts used in this proof are taken from (Ahvonen et al., 2024b). First, we translate a GNN[F] into GML as follows. First, we use Proposition B.17 in (Ahvonen et al., 2024b) which states that a GNN with n layers can be modified into a recurrent GNN that gives the same output after exactly n rounds of iteration. The construction in the proposition is for reals, but it is easy to see that it also works for floating-point GNNs (i.e., we can turn an n-layer floating-point GNN into an equivalent recurrent floating-point GNN that uses the same layer repeatedly). We must note that the GNN[F]s in (Ahvonen et al., 2024b) do not include skip connections, but they do allow arbitrary combination functions, which means that we can simply treat the skip connections as part of the combination function when using Proposition B.17. By Lemma B.3 and Theorem 3.2 of (Ahvonen et al., 2024b), a recurrent GNN[F] can be translated into a GMSC-program, which are a recursive generalization of GML. We can run this program for exactly n rounds, which gives us an equivalent GML-formula, obtained as the disjunction of the nth iteration formulae of the appointed predicates of the program.

For the translation from GML to GNN[F]s, we can simply use the proof of Proposition 4.1 in (Barceló et al., 2020), as floating-point numbers and the saturating floating-point sum are sufficient for the construction given there. This direction also follows from Lemma B.5 in (Ahvonen et al., 2024b). Note that the MLPs of the GNN[F] use the truncated ReLU as the activation function, i.e., the function $\text{ReLU}^*(x) = \min\{1, \max\{0, x\}\}$. We could also use the ordinary ReLU by adding a couple of additional layers to the MLPs, and we could obtain a GNN[F] using simple MLPs by splitting the computation of the non-simple ReLU-activated MLP into multiple message-passing layers.

One important thing to note is that the accepting condition for the GNN[F]s in (Ahvonen et al., 2024b) specifies a set of so-called accepting feature vectors. Trivially this accepting condition can simulate any MLP that gives a Boolean vertex classification by simply listing all the feature vectors that are accepted by the MLP. For the other direction, note that the constructions in Proposition 4.1 in (Barceló et al., 2020) and Lemma B.5 in (Ahvonen et al., 2024b) are made such that it is possible to determine whether a feature vector is accepting or not simply by checking if a single element of the feature vector is 0 or 1, which means that the Boolean vertex classifier of our GNN[F]s only needs to project this single element. □

Next we show the generalization of Lemma 49 for counting global readouts and counting global modalities.

Lemma 50. GNN + GC[F]*s, simple* GNN + GC[F]*s and* GML + GC *have the same expressive power.*

Proof. This follows via a simple modification to the proof of Theorem 3.2 in (Ahvonen et al., 2024b). We simply have to include a counting global readout in the recurrent float GNNs and a counting global modality in the logic GMSC, and the two can be used to simulate one another in the proofs. In the case of the R-simple recurrent GNNs in (Ahvonen et al., 2024b), this simply involves an additional sum aggregation of all feature vectors in the graph and an additional weight matrix in the combination functions. □

Finally, we prove the last part of Theorem 15. In the lemma below, note that simple GNN + G[F]s use a version of the floating-point sum that only sums one of each feature vector in the multiset.

Lemma 51. GNN + G[F]*s, simple* GNN + G[F]*s and* GML + G *have the same expressive power.*

Proof. When modifying the constructions in (Ahvonen et al., 2024b), we notice that if the aggregation function AGG in the global readout is set-based, i.e., $\text{AGG}(M) = \text{AGG}(M_{|1})$ for all multisets M, then only the non-counting global modality is required to simulate it and vice versa, only a set-based aggregation is required to simulate a non-counting global modality. □

D.3 Proof of Theorem 16

In this section, we give the proof of Theorem 16.

Theorem 16. *The following have the same expressive power:* GML + GC*, soft-attention* GPS[F]*-networks, average hard-attention* GPS[F]*-networks and* GNN + GC[F]*s. This also holds in the case where the* GPS[F]*-networks and* GNN + GC[F]*s are simple.*

For the direction from GPS[F]-networks to logic, we only need to append the analysis of GT[F]s to account for the message-passing layers. This is rather trivial since GNN[F]s translate into GML and the skip connection and architecture do not get in the way.

Theorem 52. *For each floating-point soft or average hard-attention* GPS*-network, there exists an equivalent sequence of* GML + GC*-formulae.*

Proof. This follows directly from Theorem 41 and Lemma 49. □

Because we have already translated PL + GC to GT[F]s, PL to MLP[F]s and GML to GNN[F]s, our proof will consist of translating transformer layers, MLP[F]s and message-passing layers into GPS[F]-networks. We start with the translation from MLP[F]s to GPS[F]-networks. Note that the lemmas of this section apply for both reals and floats.

Lemma 53. *For each* MLP M *of I/O dimension* d*, we can construct a* GPS*-layer* G *of dimension* $3d$ *that shifts* M *to the right (or left). If* M *is simple, then* G *is simple.*

Proof. The proof is very similar to the proof of Lemma 42. We construct a GPS-layer $G = (\text{SA}, \text{MP}, \text{FF})$ that shifts M to the right as shifting to the left is analogous. We

construct MP to copy the first d elements of a vertex's feature vector and place them in the middle d components of the output vector, other components being 0. The module SA again outputs a zero matrix. Now the middle d components of the input of FF are the same as the first d components of the input of G. The MLP FF is constructed the same as before, placing the output of M (w.r.t. the middle d components of the input) into the last d vector components of the output vector, while the other components of the output cancel out the input (which is achieved by remembering these inputs in each layer). □

Lemmas 43 and 53 together mean that a k-layer MLP can be simulated by $k-1$ simple GPS-layers.

Next, we show how GPS-layers can simulate a message-passing layer.

Lemma 54. *For each message-passing layer* M *of dimension d, we can construct a* GPS*-layer G of dimension $3d$ that shifts* M *to the right (or left). If* M *uses sum aggregation, then G is simple.*

Proof. At this point the proof is almost routine by similarity to the proofs of Lemmas 42 and 53. We construct a GPS-layer $G = (\mathrm{SA}, \mathrm{MP}, \mathrm{FF})$ that shifts M to the right, as shifting to the left is analogous. We define MP to place the output of M into the middle d components, the others being 0; this is possible by trivial matrix manipulations in the simple case. Meanwhile, SA simply outputs a zero matrix as in previous proofs. Finally, FF places the middle d components of the input into the last d components of the output while the other components of the output cancel out the input. □

As the last result of this section, we show how a GPS-layer can simulate a transformer layer.

Lemma 55. *For each transformer layer T of dimension d, we can construct a* GPS*-layer G' of dimension $3d$ that shifts T to the right (or left) (with the same attention mechanism). If T is simple, so is G'.*

Proof. Let $T = (\mathrm{SA}, \mathrm{FF})$ be a transformer layer of dimension d. We construct an equivalent GPS-layer $G' = (\mathrm{SA}', \mathrm{MP}', \mathrm{FF}')$. The proof is similar to previous proofs, but special attention is paid to the attention module.

We define the attention module SA' by modifying SA as follows. We use the usual notation W_Q, W_K, W_V and W_O for the involved matrices of SA. For each $W \in \{W_Q, W_K, W_V\}$, we construct W' by adding rows of zeros to the bottom of W. Likewise, we obtain W'_O by adding the same number of columns of zeros to the left and right of W_O (meaning that W_O is in the middle d columns of W'_O). Now, if $\mathrm{SA}(\mathcal{G}) = \mathcal{G}'$, then $\mathrm{SA}'(\mathcal{G}_r)$ is the graph obtained from $\mathcal{G}'$ by adding d zero columns to the left and right of the feature matrix (recall the meaning of $\mathcal{G}_r$ from Appendix D.1).

The message-passing layer places the first d components of a feature vector to the middle d elements in the output vector. The first and last d elements of the output vector are simply 0s.

Now (assuming that the last $2d$ columns of input feature matrices are zero columns) the skip connections of both SA' and MP' only affect the first d columns of their respective output matrices. When the outputs of SA' and MP' are added together in the GPS-layer, the middle d columns are the same as they would be after the skip connection is applied to SA in the transformer layer T.

Now the middle d elements of the input vectors of FF$'$ are the same as the input vectors of FF. As in earlier proofs, we define FF$'$ to do the same transformations as FF to the middle d elements, but placing the output to the last d elements of the output vector, while the other elements of the output vector are defined to cancel out the input after the skip connection is applied. □

Now Theorem 16 follows directly from the results of this section. As a final note, we give a one-sided translation from GML + G to unique hard-attention GPS[F]-networks.

Theorem 56. *For each* GML + G*-formula, we can construct an equivalent simple unique hard-attention* GPS[F]*-network.*

Proof. This follows from Theorem 46, Lemma 49 and Lemma 55. □

D.4 Characterizations with graph classification and non-Boolean classification

In this section, we explain in more detail how Theorems 14, 15 and 16 also generalize for graph classification and non-Boolean classification.

Note that our results generalize, in a sense, trivially for graph classification if we simply restrict our analysis to those learning models and logic formulae that define graph properties. What makes the generalization to graph classification interesting is that we have fixed the syntax of the logics and we have also fixed learning models that exclusively define graph properties via a graph classification head.

For graph classification, we replace the logic $\mathcal{L}$ appearing in the result with $\mathcal{L}^*$ and Boolean vertex classifiers with Boolean graph classifiers. Recall that the syntax of the logic $\mathcal{L}^*$ is defined as

$$\psi ::= \langle \mathsf{G} \rangle_{\geq k} \varphi \mid \neg \psi \mid \psi \wedge \psi \mid \langle \mathsf{G} \rangle_{\geq k} \psi,$$

where φ is a Π-formula of $\mathcal{L}$. When translating from a logic $\mathcal{L}$ to, e.g., a GPS[F]-network, note first that each formula of $\mathcal{L}^*$ is also a formula of $\mathcal{L}$,[12] and we can use the same construction as we do for vertex classification. Recall that a computing model that is a graph classifier uses a readout gadget instead of an MLP as a final classification head as defined in Appendix C.3. Thus, the only difference is that the final classifier is a readout gadget instead of an MLP, and so we use a readout gadget that uses the max aggregation function (this also works for the sum aggregation function by Lemma 44) and where the MLP is the same as the Boolean vertex classifier in the construction. Since the final MLP in the construction is simply a projection of a single feature of the feature vector, and since this component of the vector has the same value regardless of vertex, the max aggregation suffices. For the other direction, note that prior to the classification head, we have an equivalent formula of $\mathcal{L}$ for each bit of a feature vector of a vertex. After this, we can simulate the aggregation function of the readout gadget with the global diamond $\langle \mathsf{G} \rangle_{\geq k}$ and the MLP via the connectives $\neg$ and $\wedge$, which gives us a formula of $\mathcal{L}^*$.

For non-Boolean classification, we simply use a sequence of formulae instead of a single formula and a general classification head instead of a Boolean classification head. From logics to learning models, we simply have a single component of the output feature vector

[12] Note that this is not the case logics that do not have the counting global modality; for these logics, we can simply remove $\langle \mathsf{G} \rangle_{\geq k} \psi$ from the syntax and handle the remaining counting global modalities with the readout gadget.

for each formula in the sequence. From learning models to logics, we have a formula in the sequence per each bit of each float in the output feature vector of the learning model.

E Transformers on Words and Positional Encodings

In this section, we go through the follow-up results explained in the conclusion section 5 for floating-point based positional encodings and transformers over words.

E.1 Transformers on Words

We start by considering transformers over word-shaped graphs (see Appendix A.1 for the definition of word-shaped graphs). As already noted, a GT over word-shaped graphs is just an 'encoder-only transformer without causal masking'. For example, the popular BERT (Devlin et al., 2019) is such a model inspired by the work of Vaswani et al. (2017).

It is straightforward to see that our proofs of Theorem 14 and Theorem 16 generalize when restricted to word-shaped graphs. Our results hold in all four cases: Boolean and general vertex classification and Boolean and general graph classification. Boolean vertex classification is perhaps the most commonly used for word-shaped graphs, or more precisely, a word $\mathbf{w}$ is said to be accepted if its pointed word-shaped graph $(\mathcal{G}_{\mathbf{w}}, 1)$ is classified to 1. Note that GTs and GPS-networks in the corollaries below do not use causal masking.

Corollary 57. *When restricted to word-shaped graphs, the following have the same expressive power:* $\mathrm{PL} + \mathrm{GC}$*, soft-attention* $\mathrm{GT[F]}$*s and average hard-attention* $\mathrm{GT[F]}$*s (and* $\mathrm{GNN} + \mathrm{GC[F]}$*s with constant local aggregation functions). This also holds when the* $\mathrm{GT[F]}$*s and* $\mathrm{GNN} + \mathrm{GC[F]}$*s are simple.*

Corollary 58. *When restricted to word-shaped graphs, the following have the same expressive power:* $\mathrm{GML} + \mathrm{GC}$*, soft-attention* $\mathrm{GPS[F]}$*-networks and average hard-attention* $\mathrm{GPS[F]}$*-networks (and* $\mathrm{GNN} + \mathrm{GC[F]}$*s). This also holds when the* $\mathrm{GPS[F]}$*-networks and* $\mathrm{GNN} + \mathrm{GC[F]}$*s are simple.*

In order to characterize floating-point unique hard-attention graph transformers on words, we need to introduce a new logic. We start by discussing a known fragment of linear-time temporal logic (LTL). The set of Π-formulae of **linear-time temporal logic with past** (LTLP) is defined by the grammar

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{P}\varphi,$$

where $p \in \Pi$. The (full) linear-time temporal logic also includes the "since" operator S, the "until" operator U and the "future" operator F, but we do not consider them here. The semantics of LTLP is defined over words as follows. Given a Π-labeled pointed word-shaped graph $(\mathcal{G}_{\mathbf{w}}, i)$ and a Π-formula φ of LTLP, we define the truth of φ in $(\mathcal{G}_{\mathbf{w}}, i)$ as follows.

- $\mathcal{G}_{\mathbf{w}}, i \models p$ iff $p \in \lambda(\mathcal{G}_{\mathbf{w}})_i$.
- The semantics of Boolean operators are defined in the usual way.
- $\mathcal{G}_{\mathbf{w}}, i \models \mathsf{P}\varphi$ iff there is some $j < i$ such that $\mathcal{G}_{\mathbf{w}}, j \models \varphi$.

Now, since we have defined LTLP, we can define the logic $\mathcal{L}$ used for the characterization. The logic $\mathcal{L}$ is a mix of $\mathrm{PL} + \mathrm{G}$ and LTLP, and obtained from the logic PL as follows: we

extend PL with the 2-ary operator GB called **global-before**. The semantics of the formula $\varphi \,\mathsf{GB}\, \psi$ is the same as that of the formula

$$\langle \mathsf{G} \rangle (\psi \wedge \neg \mathsf{P} \varphi).$$

Intuitively, the formula states that "there is a state where ψ is true and φ does not precede ψ". We call this logic **PL with global-before** (or PL+GB).

We start with the following observation. Below, we let LTLP + G denote the logic obtained from LTLP by extending its syntax with the global non-counting modality. Below, "$\mathcal{C}_1$ is strictly more expressive than $\mathcal{C}_2$" means that for every $x \in \mathcal{C}_2$, there is an equivalent object $y \in \mathcal{C}_1$, but there is an object $z \in \mathcal{C}_1$ that is not equivalent to any object in $\mathcal{C}_2$. Moreover, "$\mathcal{C}_1$ and $\mathcal{C}_2$ have orthogonal expressive power" means that there is an object in $\mathcal{C}_1$ that is not equivalent to any object in $\mathcal{C}_2$, and vice versa.

Proposition 59. *Over word-shaped graphs, the following holds.*

- PL + GB *is strictly more expressive than* PL + G.
- LTLP *and* PL + GB *have orthogonal expressive power.*
- LTLP + G *is strictly more expressive than* PL + GB.

Proof. First, we show that PL + GB is strictly more expressive than PL + G. We notice that any formula of the type $\langle \mathsf{G} \rangle \varphi$ can be expressed as $\bot \,\mathsf{GB}\, \varphi$, which means that each formula of PL + G can be expressed in PL + GB. Clearly, PL + GB is strictly more expressive than PL + G because, e.g., the formula $\top \mathsf{GB} p$, which intuitively states that "the first letter in a word is p", is not expressible in PL + G.

Secondly, it is easy to see that LTLP and PL + GB have orthogonal expressive power by considering the following invariance. LTLP is invariant over word-shaped graphs under extending a word from the right with a unary string, i.e., given a pointed Π-word $(\mathbf{w}, i)$, for every Π-formula φ in LTLP, we have $\mathcal{G}_{\mathbf{w}}, i \models \varphi$ iff $\mathcal{G}_{\mathbf{wu}}, i \models \varphi$, where $\mathbf{u}$ is a unary word (i.e. $\mathbf{u} \in \{p\}$ for some letter p) and $\mathbf{wu}$ denotes the concatenation of $\mathbf{w}$ with $\mathbf{u}$. However, clearly, PL + GB is not invariant under extending a word from the right. On the other hand, PL + GB can express formulae of the type $\langle \mathsf{G} \rangle \varphi$ which LTLP cannot.

Lastly, it is trivial that PL + GB is contained in LTLP+G. To see that LTLP+G is strictly more expressive, note that LTLP + G contains LTLP that can already express properties not expressible in PL + GB. □

Before we characterize unique hard-attention, we prove the following helpful lemma.

Lemma 60. *Let $\mathcal{F}$ be a float format, let $X \in \mathcal{F}^{n \times d}$ and $W_Q, W_K \in \mathcal{F}^{d \times d_h}$. If the ith and the jth row of X are identical, then the ith row (resp. column) and the jth row (resp. column) of the matrix*

$$\frac{(XW_Q)(XW_K)^{\mathsf{T}}}{\sqrt{d_h}},$$

are identical.

Proof. Since the ith row and the jth row of X are identical, the ith row and the jth row of XW_Q are identical and the same is true for XW_K. To see that the ith row and the jth row of $Y := (XW_Q)(XW_K)^{\mathsf{T}}$ are identical, note that for each column k, we have $Y_{i,k} = (XW_Q)_{i,*}((XW_K)_{k,*})^{\mathsf{T}}$ and $Y_{j,k} = (XW_Q)_{j,*}((XW_K)_{k,*})^{\mathsf{T}}$. By an analogous argument,

we can also see that the ith column and the jth column of Y are identical. Dividing all elements in Y with $\sqrt{d_h}$ naturally does not change which rows or columns are identical. $\square$

We will show that, over words, PL+GB has the same expressive power as float-based graph transformers with unique hard-attention (without causal masking).

Theorem 61. *Over word-shaped graphs,* PL + GB *has the same expressive power as unique hard-attention* GT[F]*s. This also applies when* GT[F]*s are simple.*

Proof. Let T be a GT[F] with unique hard-attention over (Π, d) over the floating-point format $\mathcal{F}$. We construct an equivalent Π-formula of PL + GB as follows. The construction is almost the same as the case with soft-attention and average hard-attention. The hardest part is to simulate the unique hard-attention function in each attention head. We first intuitively describe the simulation of an attention head

$$H(X) = \mathrm{UH}\Big(\frac{(XW_Q)(XW_K)^\mathsf{T}}{\sqrt{d_h}}\Big)(XW_V).$$

First, we observe that the unique hard function UH in H outputs a square matrix S that is just a selection matrix, i.e., each row contains one 1 and all other elements of the row are 0s. Secondly, by Lemma 60 and the former observation we notice that if the ith row and the jth row of the input matrix X are identical, then in H the selection matrix S will select the same row from XW_V to be the ith row and the jth row of $H(X)$. To find out which row of XW_V will be selected by the selection matrix S of $H(X)$, we have to look at the values of XW_Q and $(XW_K)^\mathsf{T}$ as follows. For each row $\vec{q}$ of XW_Q, we find the leftmost column $\vec{k}$ of $(XW_K)^\mathsf{T}$ that maximizes the value $\vec{q}\vec{k}$ across all the columns of $(XW_K)^\mathsf{T}$. If the leftmost column of $(XW_K)^\mathsf{T}$ that gives the maximum value with the ith row of XW_Q is the jth column, then the selection matrix S selects the jth row of XW_V for the ith row of $H(X)$. In terms of words, if X represents the feature matrix of a word, this means that for each node v (in the word) we have to find the leftmost node w that maximizes the value of

$$\frac{(XW_Q)_{v,*}((XW_K)^\mathsf{T})_{*,w}}{\sqrt{d_h}}.$$

Now, with this intuition, we start the formal construction. Let $\mathcal{G}$ be the studied graph. The construction is similar to the case with GT[F]s with soft-attention and average hard-attention except for the simulation of unique hard-attention heads. Assume that for each node u the current feature vector $\vec{f}_u$ of u is encoded by the formula sequence $\vec{\psi}$ as in the proof of Theorem 41. Then we define the formulae $\vec{\psi}^Q$, $\vec{\psi}^K$ and $\vec{\psi}^V$ which encode for each node u the vectors $\vec{q}_u := \vec{f}_u W_Q$, $\vec{k}_u := \vec{f}_u W_K$ and $\vec{v}_u := \vec{f}_u W_V$, respectively. By using these formulae, we can write down the formula sequence $\vec{\theta}$ which encodes for each node u the feature vector $\vec{g}_u$ that is obtained from XW_V by using the selection matrix S described above. For each node u, we define the floating-point number

$$M_u := \max\left\{ \frac{(\vec{q}_u)(\vec{k}_w)^\mathsf{T}}{\sqrt{d_h}} \,\middle|\, w \in V(\mathcal{G}) \right\}.$$

For each feature vector pair $(\vec{f}, \vec{g}) \in \mathcal{F}^d \times \mathcal{F}^d$, we can define a formula $\psi_{\vec{f},\vec{g}}$ (by using the operator GB and the auxiliary formulae defined above), which intuitively checks two things when evaluated at a given node u:

1. First, the formula checks that there is a node w such that 1) $\vec{k}_w = \vec{f}$, 2) $\vec{q}_u\vec{k}_w = M_u$ and 3) $\vec{v}_w = \vec{g}$.
2. At the same time the formula checks that there is no node w' on the left of w with $\vec{q}_u\vec{k}_{w'} = M_u$ (this part requires the operator GB).

By using these formulae, it is easy to define the formula sequence $\vec{\theta}$ as follows. At each node, precisely one of the formulae $\psi_{\vec{f},\vec{g}}$ is true. We construct $\vec{\theta}$ such that for each feature vector pair $(\vec{f},\vec{g}) \in \mathcal{F}^d \times \mathcal{F}^d$, if $\psi_{\vec{f},\vec{g}}$ is the true formula, then $\vec{\theta}$ encodes the feature vector $\vec{g}$, which is the feature vector selected by the selection matrix for that node.

For the converse, let θ be a Π-formula of $\mathrm{PL} + \mathrm{GB}$. We show by induction on the construction of a formula how to simulate θ. Proposition symbols and Boolean connectives are analogous to the case with soft-attention and average hard-attention. To simulate the operator $\varphi\,\mathsf{GB}\,\psi$, we use the unique hard-attention module. Assume that the truth values of φ and ψ are encoded into a matrix $X \in \mathbb{B}^{n\times d}$ in the columns i and j, respectively. Moreover, we assume that the truth of $\top$ is encoded into the column k. Now, we build an attention head H as follows. The matrix W_Q is just the $d \times 2$ Boolean-valued matrix, where in both columns the kth row is 1 and others are 0. The matrix W_K is the $d \times 2$ matrix where in the first column the ith row is ∞ and in the second column the jth row is ∞, and other entries are 0. Now, the resulting matrix $\frac{(XW_Q)(XW_K)^\mathsf{T}}{\sqrt{2}}$ is a matrix where each column is filled with some float f that represents either ∞ of 0. Therefore, UH will assign 1 to the same column in all rows. Now, we define that W_V is the Boolean-valued $(d\times 2)$-matrix, where in the first column the jth row is 1 and other entries in both columns are zero. Now, it is easy to verify that $H(X)$ is the $(n\times 2)$-matrix where each row contains the same pair $(x,0)$, where x is either 0 or 1. Now, we have $x = 1$ if $\langle\mathsf{G}\rangle(\psi \wedge \neg\mathsf{P}\varphi)$ is true and $x = 0$ otherwise. The rest of the construction is easy to complete. □

By using the result above and Lemma 49, it is straightforward to obtain the following theorem for GPS-networks on word-shaped graphs (without causal masking).

Theorem 62. *Over word-shaped graphs,* $\mathrm{GML} + \mathrm{GB}$ *has the same expressive power as unique hard-attention* GPS[F]*-networks. This also applies when* GPS[F]*-networks are simple.*

E.2 Positional Encodings

In this section, we shall consider our computing models with positional encodings. In the end of Section 4.3, we had a preliminary discussion about positional encodings, but let us go through the related concepts on positional encodings in a little bit more detail.

Often each GNN, GT or GPS-network $A = (P, L^{(1)}, \ldots, L^{(k)}, C)$ based on reals (with input dimension ℓ) is associated with a **positional encoding** (or PE) π, that is, a mapping that gives for a graph $\mathcal{G}$ a function $\pi(\mathcal{G})\colon V(\mathcal{G}) \to \mathbb{R}^\ell$.[13] A popular PE is LapPE (Rampásek et al., 2022). Now, A with π computes over $\mathcal{G}$ a sequence of feature maps similarly to A, but for each vertex v in $\mathcal{G}$, we define $\lambda_v^{(0)} := P(\lambda)_v + \pi(\mathcal{G})_v$. Analogously to computing models based on reals, computing models based on floats can be associated with a positional encoding: Let $\mathcal{F}(p,q)$ be a floating-point format and let π be a positional encoding of dimension d over $\mathcal{F}$, i.e., for each graph $\mathcal{G}$, π gives a mapping $\pi(\mathcal{G})\colon V(\mathcal{G}) \to \mathcal{F}^d$. Our

[13] In the preliminary draft of this paper (Ahvonen et al., 2025), there is a typo stating that positional encodings are isomorphism-invariant. For example, Rosenbluth et al. (2024) notes that real-valued LapPE is not isomorphism-invariant.

positional encodings are "absolute", but one could also study "relative" positional encodings that give a weight between the vertices of the studied graph $\mathcal{G}$, i.e., a relative positional encoding over $\mathcal{G}$ is a function $V(\mathcal{G}) \times V(\mathcal{G}) \to \mathcal{F}^d$. If a GNN[F], GNN + GC[F], GNN + G[F], GT[F] or GPS[F]-network T is paired with a positional encoding π, we assume that T is defined over the same floating-point format $\mathcal{F}$ as π. Moreover, if $\mathcal{C}$ is a class of computing models (over floats) we let $\mathcal{C}[\pi]$ denote the class of computing models, where each computing model over a floating-point format $\mathcal{F}$ is paired with the positional encoding π over $\mathcal{F}$.

Let Π be a set of vertex labels. Given a logic $\mathcal{L}$, a Π-formula of the logic $\mathcal{L}[\pi]$ is a $\Pi \cup \Pi_\pi$-formula of the logic $\mathcal{L}$, where $\Pi_\pi = \{ \ell_i^\pi \mid i \in [d(p+q+1)] \}$ is a fresh set of vertex label symbols (i.e. $\Pi \cap \Pi_\pi = \emptyset$). Now, a Π-formula $\varphi \in \mathcal{L}[\pi]$ is interpreted over $\Pi \cup \Pi_\pi$-labeled graphs $\mathcal{G}$ such that for each vertex v in $\mathcal{G}$, we have $\mathcal{G}, v \models \ell_i^\pi$ iff the ith bit of $\pi(\mathcal{G})_v$ is 1. That is, each Π-formula of $\mathcal{L}[\pi]$ is interpreted over graphs where each graph includes vertex label symbols such that in each vertex v, the vertex label symbols of v encode $\pi(\mathcal{G})_v$ in binary. In terms of FO this would mean adding an additional set of predicates to the logic.

Now, it is straightforward to see that Theorem 14 and Theorem 16 apply for graph transformers and GPS-networks with positional encoding π when the logic is extended with predicates that encode positional encodings. Also Theorem 61 and Theorem 62 can be generalized in an analogous way.

Corollary 63. *Let π be a positional encoding. The following have the same expressive power:* $\mathrm{PL} + \mathrm{GC}[\pi]$*, soft-attention* $\mathrm{GT}[\mathrm{F}]$*s with π and average hard-attention* $\mathrm{GT}[\mathrm{F}]$*s with π (and* $\mathrm{GNN} + \mathrm{GC}[\mathrm{F}]$*s with constant local aggregation functions and π). Moreover, the same holds when restricted to word-shaped graphs, and when the* $\mathrm{GT}[\mathrm{F}]$*s and* $\mathrm{GNN} + \mathrm{GC}[\mathrm{F}]$*s are simple.*

Corollary 64. *Let π be a positional encoding. The following have the same expressive power:* $\mathrm{GML} + \mathrm{GC}[\pi]$*, soft-attention* $\mathrm{GPS}[\mathrm{F}]$*-networks with π, average hard-attention* $\mathrm{GPS}[\mathrm{F}]$*-networks with π and* $\mathrm{GNN} + \mathrm{GC}[\mathrm{F}]$*s with π. Moreover, the same holds when restricted to word-shaped graphs, and when the* $\mathrm{GPS}[\mathrm{F}]$*-networks and* $\mathrm{GNN} + \mathrm{GC}[\mathrm{F}]$*s are simple.*

Corollary 65. *Let π be a positional encoding. The following have the same expressive power when restricted to word-shaped graphs:* $\mathrm{PL} + \mathrm{GB}[\pi]$ *and unique hard-attention* $\mathrm{GT}[\mathrm{F}]$*s with π. This also applies when* $\mathrm{GT}[\mathrm{F}]$*s are simple.*

Corollary 66. *Let π be a positional encoding. The following have the same expressive power when restricted to word-shaped graphs:* $\mathrm{GML} + \mathrm{GB}[\pi]$ *and unique hard-attention* $\mathrm{GPS}[\mathrm{F}]$*-networks with π. This also applies when* $\mathrm{GPS}[\mathrm{F}]$*-networks are simple.*

While we primarily consider graph transformers and GPS-networks, Theorem 15 concerning GNNs can be similarly generalized to cover positional encodings.

Corollary 67. *Let π be a positional encoding. The following pairs have the same expressive power:* $\mathrm{GNN}[\mathrm{F}]$*s with π and* $\mathrm{GML}[\pi]$*;* $\mathrm{GNN} + \mathrm{G}[\mathrm{F}]$*s with π and* $\mathrm{GML} + \mathrm{G}[\pi]$*;* $\mathrm{GNN} + \mathrm{GC}[\mathrm{F}]$*s with π and* $\mathrm{GML} + \mathrm{GC}[\pi]$*. This also applies when each type of* $\mathrm{GNN}[\mathrm{F}]$ *is simple.*

One could also characterize a positional encoding directly and combine that characterization with any of our Theorems 14, 15, 16, 61 and 62. We leave this for future work.