# On the Expressive Power of Graph Transformers

Veeti Ahvonen

Mathematics Research Centre, Tampere University

3.10.2025

This talk is based on the joint work "Expressive power of Graph Transformers via Logic" with Maurice Funk, Damian Heiman, Antti Kuusisto and Carsten Lutz.

# Background & contributions related to graph learning

Transformers are the basis of modern large language models (Vaswani et al. NIPS, ChatGPT, Copilot, etc.), but relatively little is known about their precise expressive power on graphs.

For example, Yang et al. (NeurIPS), Chiang et al. (ICML), Li and Cotterel, Jerad et al., studied ordinary transformers via temporal logics and built links between the classes of formal languages.

We give a unrestricted logical characterization of float-based GTs.

Restricted to first-order logic, we characterize real-based GTs.

We also consider transformers over words.

# Preliminaries: Feedforward neural networks

A **perceptron layer** of dimension $(i, o)$ is a tuple $P = (\alpha, W, b)$, where

- $\alpha \colon \mathbb{R} \to \mathbb{R}$ is an activation function (e.g. $\mathrm{ReLU}(x) = \max(0, x)$),
- $W \in \mathbb{R}^{o \times i}$ is a weight matrix
- and $b \in \mathbb{R}^o$ is a bias term.

Given an $\mathrm{x} \in \mathbb{R}^i$, we let $\alpha(W\mathrm{x} + b) =: P(\mathrm{x}) \in \mathbb{R}^o$, where $\alpha$ is applied pointwise.

A **feedforward neural network** (or FNN) is a finite sequence of perceptron layers, where each layer propagates its output to the next layer.

Given an $\mathrm{x} \in \mathbb{R}^i$, we let $F(\mathrm{x}) := P^{(n)}(\cdots(P^{(1)}(\mathrm{x}))\cdots)$.

**Fact:** "FNNs can approximate any continuous function to an arbitrary degree of accuracy."

# Graph Transformers

An **attention head** $H$ of dim. $(i, h)$ is defined w.r.t. three matrices in $\mathbb{R}^{i \times h}$: the **query-matrix** $W_Q$, the **key-matrix** $W_K$ and the **value-matrix** $W_V$.

For a given $X \in \mathbb{R}^{n \times i}$ it computes the following $n \times h$-matrix:

$$H(X) := \mathrm{softmax}\left(\frac{(XW_Q)(XW_K)^\top}{\sqrt{h}}\right)(XW_V).$$

The function $\mathrm{softmax}\colon \mathbb{R}^+ \to \mathbb{R}^+$ is length preserving and defined by

$$\mathrm{softmax}(\mathrm{x})_i := \frac{e^{\mathrm{x}_i - b}}{\sum_j e^{\mathrm{x}_j - b}}, \text{ where } b = \mathrm{argmax}(\mathrm{x}).$$

An **attention module** of dimension $d$ is a tuple $A = (H^{(1)}, \ldots, H^{(k)}, W_O)$, where each $H^{(j)}$ is an attention head of dimension $(d, h)$, and $W_O \in \mathbb{R}^{kh \times d}$. Then we define $A(X) := concat(H^{(1)}(X), \ldots, H^{(k)}(X))W_O$.

# Graph Transformers

Let $\Pi = \{p_1, \ldots, p_\ell\}$ be a set of labels. Fix an arbitrary labeled graph $\mathcal{G} = (V, E, p_1, \ldots, p_\ell)$ with $n$ vertices and an ordering $<^V$ over $V$. Its labeling induces a Boolean **feature matrix** $M^{\mathcal{G}} \in \{0,1\}^{n \times \ell}$, where each row is the one-hot representation of the labels at a vertex.

A **graph transformer** (GT) consists of an FNN $I$ of dim. $(\ell, d)$, a **classification function** $C \colon \mathbb{R}^d \to \{0,1\}$, and a finite sequence of **transformer layers** $L^{(1)}, \ldots, L^{(k)}$, each consisting of an attention module $A^{(j)}$ and an FNN $F^{(j)}$, both of dimension $d$.

Now, $GT$ computes over $\mathcal{G}$ a sequence of feature matrices: $M^{(0)} := I(M^{\mathcal{G}})$,

$$M_A^{(j)} := M^{(j-1)} + A^{(j)}(M^{(j-1)}),$$
$$M^{(j)} := M_A^{(j)} + F^{(j)}(M_A^{(j)}).$$

Finally $GT(\mathcal{G}) := C(M^{(k)})$. Note that each GT defines a vertex property!

# Graph neural networks and GPS-networks

**Graph neural networks** are defined analogously to GTs but each layer is a **message passing layer** $MP^{(j)} = (\mathrm{COM}^{(j)}, \mathrm{AGG}^{(j)})$. For each node $v$ in the studied graph $\mathcal{G}$ it computes $M^{(j)} := MP^{(j)}(M^{(j-1)})$ as follows:

$$M_v^{(j)} := \mathrm{COM}^{(j)}\left(M_v^{(j-1)}, \mathrm{AGG}^{(j)}(\{\{M_u^{(j-1)} \mid (v,u) \in E\}\})\right).$$

- $\mathrm{COM}$ is an FNN of dim. $(2d, d)$
- $\mathrm{AGG}\colon \mathit{multiset}(\mathbb{R}^d) \to \mathbb{R}^d$ is an aggregation function (typically sum!)

A **GPS-network** uses **GPS**-layers $(MP^{(j)}, A^{(j)}, F^{(j)})$ which combines the message passing layer $MP^{(j)}$ of GNNs and transformer layer $(A^{(j)}, F^{(j)})$ as follows. A GPS-layer computes similarly to GTs, but we merge $MP^{(j)}(M^{(j-1)})$ to $M_A^{(j)}$ by using sum before applying $F$.

# Logics

**Graded modal logic with counting global modality** (or $\mathrm{GML} + \mathrm{GC}$):

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond_{\geq k}\varphi \mid \langle G \rangle_{\geq k}\varphi$$

Let $(\mathcal{G}, v)$ be a pointed labeled graph with vertices $V$ and edges $E$.

$$\begin{aligned} \mathcal{G}, v \models \Diamond_{\geq k}\varphi &\iff |\{\, u \mid (v,u) \in E, \mathcal{G}, u \models \varphi \,\}| \geq k \\ \mathcal{G}, v \models \langle G \rangle_{\geq k}\varphi &\iff |\{\, u \mid \mathcal{G}, u \models \varphi \,\}| \geq k \end{aligned}$$

The logic $\mathrm{GML} + \mathrm{G}$ is the fragment of $\mathrm{GML} + \mathrm{GC}$, where diamonds $\langle G \rangle_{\geq k}$ are allowed only if $k = 1$. The logics $\mathrm{PL} + \mathrm{GC}$ and $\mathrm{PL} + \mathrm{G}$ are defined analogously, but do not include diamonds $\Diamond_{\geq k}$ for any $k$.

# Real-based Characterizations

## Theorem 1

*Relative to* $\mathrm{FO}$*, the following pairs have the same expressive power:*

$$\mathrm{GML} + \mathrm{G} \equiv \textit{GPS-networks}$$

$$\mathrm{PL} + \mathrm{G} \equiv \textit{Graph Transformers}$$

## Proof.

"$\Rightarrow$" By induction on the structure of a formula. FNNs can be used to simulate Boolean connctives. Attention heads can be used to handle subformulae of the form $\langle G \rangle \varphi$, and message passing modules can be used to simulate ordinary counting diamonds $\Diamond_{\geq k}$.

"$\Leftarrow$" For GPS-networks, we introduce a new type of bisimilarity called **global-ratio graded bisimilarity** that takes into account the multiplicities with which graded bisimulation types are realized. Then we prove a corresponding van Bethem/Rosen theorem: Every $\mathrm{FO}$-formula invariant under global-ratio graded bisimilation is equivalent to a formula of $\mathrm{GML} + \mathrm{G}$. An analogous result can be obtained for GTs. $\square$

# Float-based characterizations

Consider graph transformers that use floating-point numbers instead of reals.

We notice that the floating-point sum $+$ is not associative due the rounding. Rounding errors also include

- **overflow** (a resulting float exceeds the largest representable value)
- **underflow** (a resulting float is too close to $0$ and rounds to $0$).

*In which order, we should perform the sum?* We cannot sum over a linear order of the nodes since that would break the isomorphism invariance of GTs!

We use the order of the floats instead, and a common choice is summing in the increasing order of floats due to its numerical accuracy.

# Float-based characterizations

## Theorem 2

*The following pairs have the same expressive power:*

$$\mathrm{GML} + \mathrm{GC} \equiv \textit{GPS-networks with floats}$$

$$\mathrm{PL} + \mathrm{GC} \equiv \textit{Graph Transformers with floats}$$

## Proof.

"$\Rightarrow$" As with reals, by induction on the structure of a formula. However, we need to pay attention on the properties of the floats. The most hardest part is to simulate global diamonds $\langle G \rangle_{\geq k}$ by using attention heads, but this can be done by using the underflow phenomenon.
"$\Leftarrow$" All local steps (e.g. FNNs) can be simulated by using $\mathrm{PL}$, since it is Boolean complete. Message passing modules are easy to handle by using diamonds $\Diamond_{\geq k}$. The hardest part is to simulate attention heads. This can be done by carefully simulating each matrix operation one-by-one, then the softmax function can simulated by using diamonds $\langle G \rangle_{\geq k}$ due the overflow that occurs in the sum operations with large graphs. $\square$

# Follow-up results

- We could use average hard function instead of the softmax function and the results would still hold.
- In the case of floats, our results also hold when restricted to word-shaped graphs.
- In the case floats, we do not have to restrict ourselves to Boolean classification. We could consider transformers whose classification functions maps each vertex to a floating-point vector.

Thank you!