Logical characterizations of neural architectures via floats and reals

Antti Kuusisto

Mathematics Research Centre

Tampere University, Finland

In this talk we discuss logical characterizations for:

- Graph neural networks with direct links to distributed computing
- Graph transformers
- Ordinary neural networks, concentrating on recurrent NNs.

Logical characterizations of graph neural networks (GNNs)

- P. Barceló et al. (ICLR 2020) characterize a class of GNNs in restriction to first-order definable properties via
  1. FOC^2 in the scenario where the GNNs contain *global readouts*,
  2. graded modal logic in the scenario without global readouts.
- M. Grohe (LICS 2023) characterizes a class of GNNs via Guarded Fragment + counting quantifiers + built-in relations.

Both works restrict to constant-time GNNs.

Logical characterizations of graph neural networks (GNNs)

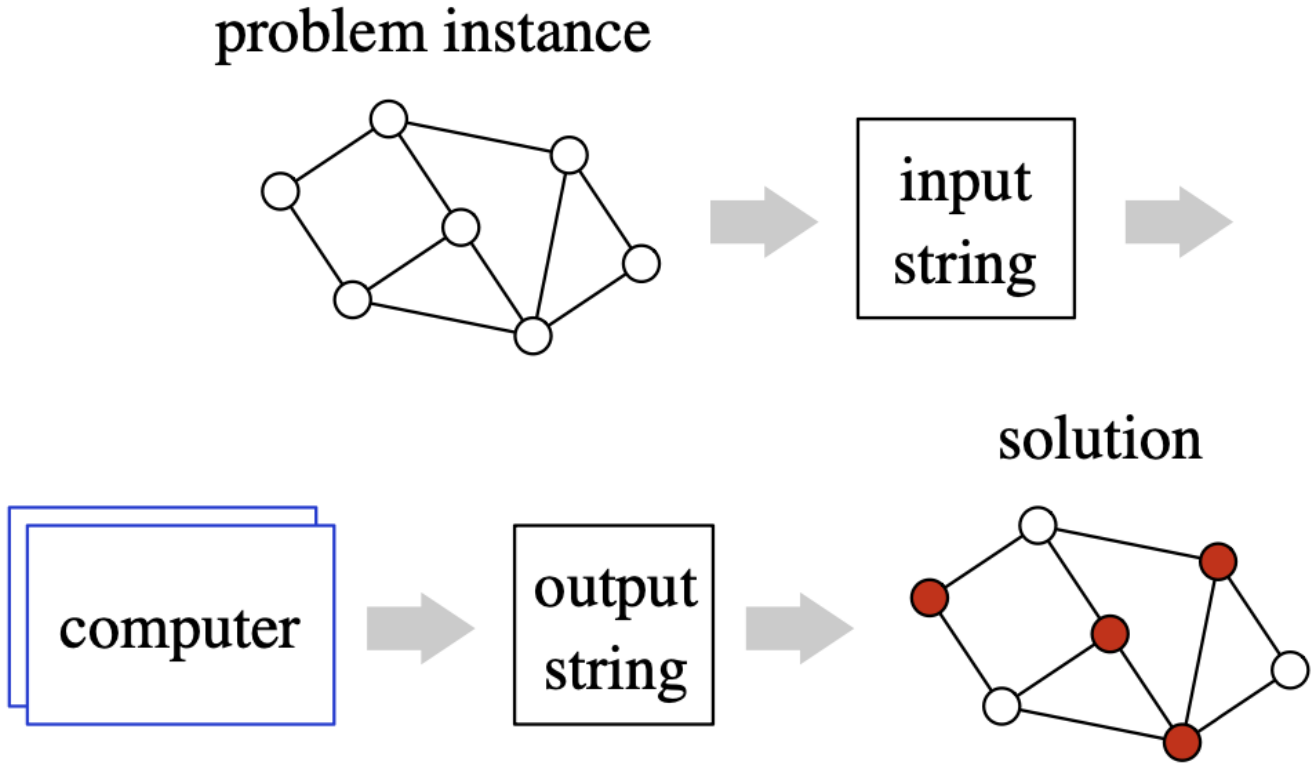
- M. Pflüger et al. (AAAI 2024) characterize a class of recurrent (i.e., non-constant-time) GNNs in restriction to LocMMFP-definable properties via graded μ-calculus.

The logic LocMMFP extends first-order logic with a least fixed-point operator over unary monotone formulas. Here no syntactic restriction is imposed on the formulas to be iterated, so LocMMFP has undecidable syntax (which is also noted by the authors).

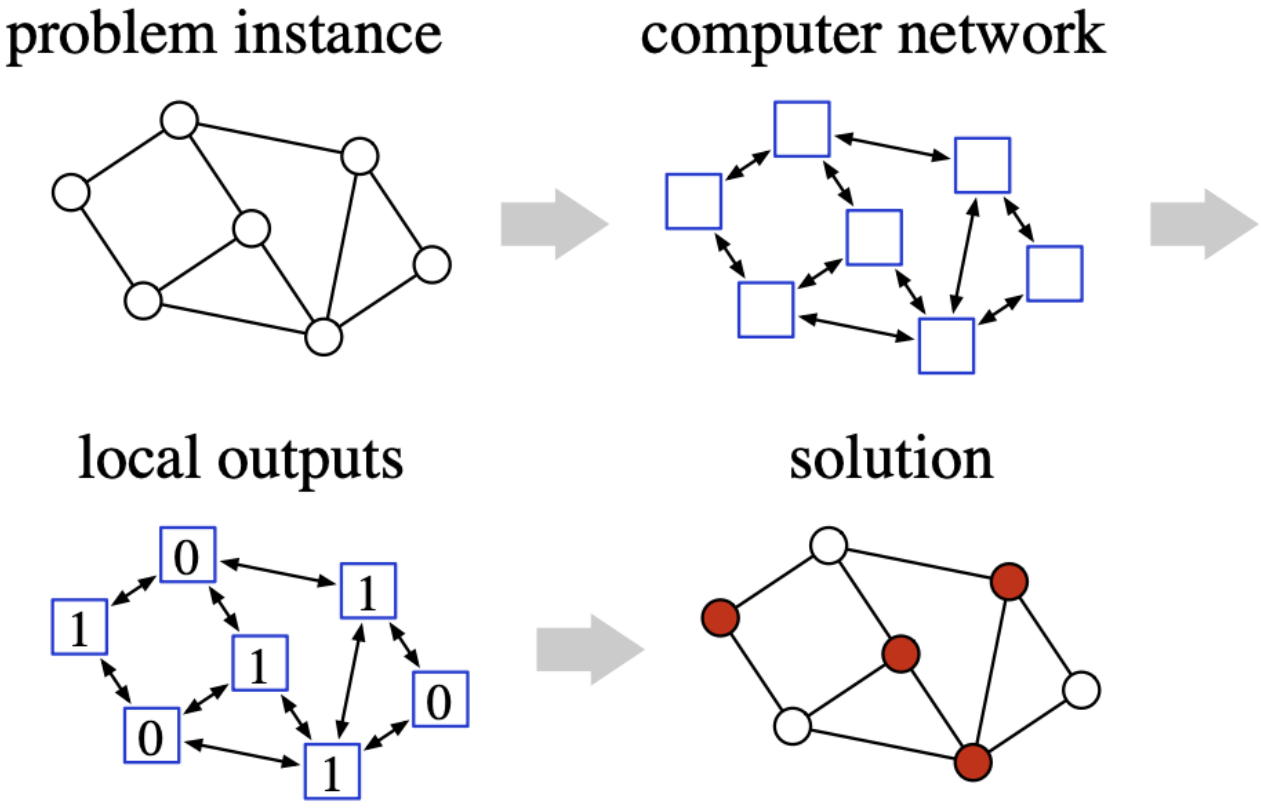
Descriptive complexity of distributed computing

- Hella et al. (PODC 2012) characterize constant-time distributed automata classes via modal logics.
- K. (CSL 2013) lifts these characterizations to recurrent (i.e., non-constant-time) distributed automata via Modal Substitution Calculus (MSC), a rule-based modal logic with a recursion mechanism.

Classical computing:



Distributed computing:



A distributed system is defined by a labelled directed graph

$$(W, R, p_1, ..., p_k)$$

together with an automaton A.

- Each node $w \in W$ contains a copy (A, w) of the automaton A.
- $R \subseteq W \times W$ is a collection of communication channels.
- Predicates $p_i \subseteq W$ encode a local input at each node.

The ith input bit at node w is 1 iff $w \in p_i$.

We could allow for more communication channels $R_1, \ldots, R_m$ without changing any results to be presented.

Computation proceeds in synchronous steps.

In one time step, each machine (A, w)

- Receives messages from its neighbours and sends messages to its neighbours,
- Updates its own state based on
  - its previous state, and
  - the received messages.

A distributed automaton A is a tuple (Q, π, δ, F).

1. Q is a finite set of states.
2. At each node, $\pi : \mathcal{P}(\{p_1, \ldots, p_k\}) \to Q$ determines the initial state of the automaton based on the proposition symbols satisfied at the node.
3. $\delta : Q \times \mathcal{P}(Q) \to Q$ gives each node a new state based on
   - the state at the node in the previous round and
   - the set of states sent by the neighbours, i.e., the set $\{q \in Q \mid q \text{ is some neighbour's state in the previous round}\}$.
4. $F \subseteq Q$ is a set of accepting states.

A node w accepts if it visits some accepting state $q \in F$ at least once.

More formally:

Each communication round $n \in \mathbb{N}$ defines a **global configuration** $f_n : W \to Q$.

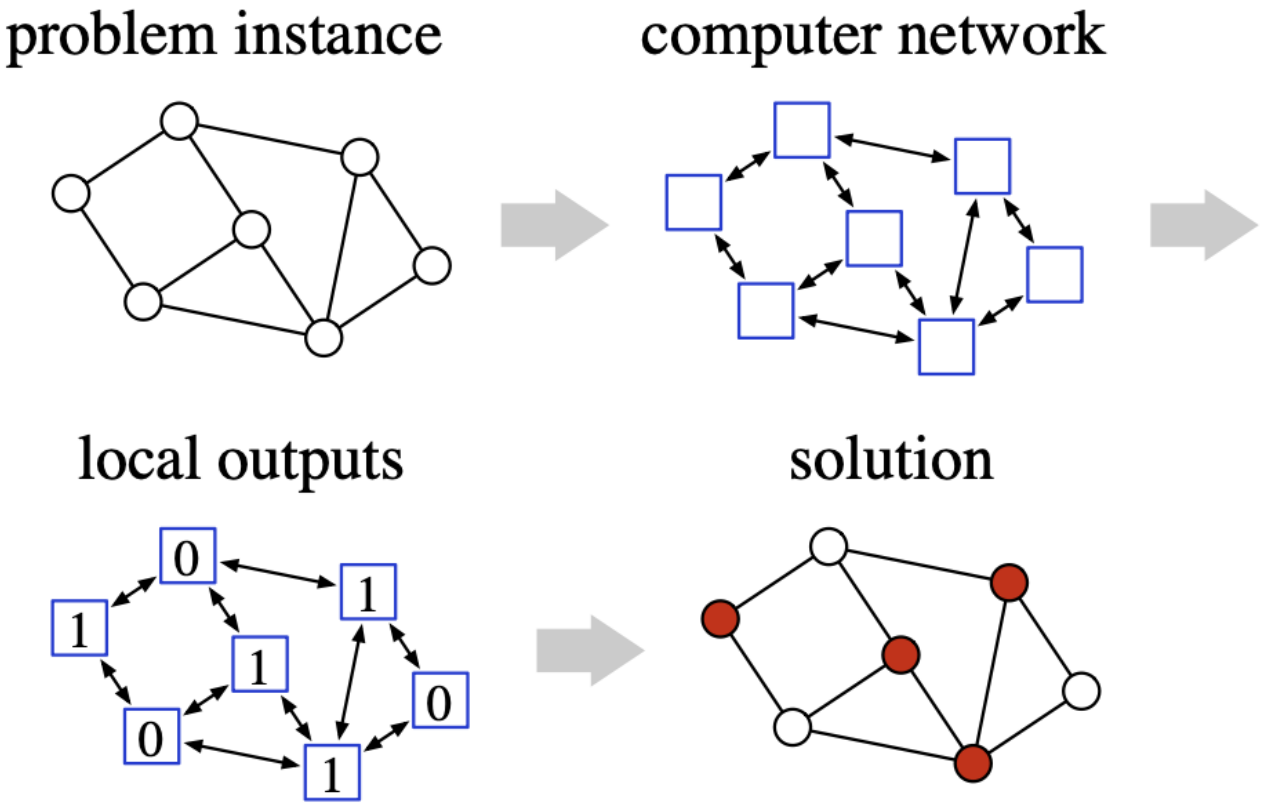
$f_0(w) :=$ initial state at w obtained by the initialization function π.

Define $Q' :=$ the set of states realized by the neigbours of w in round n. Then $f_{n+1}(w) :=$ the new state $\delta(f_n(w), Q'))$ at w.

Strictly speaking, by "*neighbours of* w" we here mean out-neigbours. This means messages (i.e., states) are sent from the head of an arrow towards the tail. This is simply a convention and could be reversed.

Node w accepts iff $f_k(w) \in F$ for some $k \in \mathbb{N}$.

The automaton A therefore computes a subset $S \subseteq W$ (the set of accepting nodes) of the distributed system $(W, R, p_1, \ldots, p_k)$.



These automata are called finite message passing automata (FMPAs).

If the state space Q is allowed to be infinite, we have a message passing automaton (MPA). The transition function δ of an MPA is allowed to be non-computable.

Modal Substitution Calculus (MSC)

$$\begin{array}{ll} X_1 \;:\!-\; p \vee \Diamond q & X_1 \;:\!-\; p \vee X_n \vee \Diamond X_1 \\ \vdots & \vdots \\ X_n \;:\!-\; \Box(p \wedge q) & X_n \;:\!-\; X_1 \vee \Diamond \neg X_n \end{array}$$

A **program** of MSC consists of two lists of **rules**:

1. base rules on the left,
2. induction rules on the right.

- $\varphi \;::=\; p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi$
- $\varphi \;::=\; p \mid X \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi$

$$\begin{array}{lll} X_1 & :- & \psi_1 \\ \vdots & & \\ X_n & :- & \psi_n \end{array} \qquad \begin{array}{lll} X_1 & :- & \varphi_1 \\ \vdots & & \\ X_n & :- & \varphi_n \end{array}$$

Define $X_i^0 := \psi_i$

Define X_i^{n+1} to be the formula obtained from φ_i by replacing each X_j by X_j^n.

$M, w \models \text{program}$ iff $M, w \models X_1^n$ for some $n \in \mathbb{N}$.

Here $M = (W, R, p_1, \ldots, p_k)$ is our distributed system and $w \in W$ a node.
Recall $M, w \models \Diamond\varphi$ iff $M, u \models \varphi$ for some u such that $R(w, u)$.

Theorem. *Modal substitution calculus* MSC *is precisely as expressive as finite message passing automata* FMPAs.

Theorem. *Co-theories of modal logic are precisely as expressive as message passing automata* MPAs.

A co-theory is just an infinite disjunction of formulas.

Proposition. MSC *is orthogonal in expressive power to* MSO.

Proof. MSC easily defines the **centre point property**, i.e., the node property stating that for some k, no matter which way you walk, you will end up in a dead-end in precisely k steps. An easy Ehrenfeucht-Fraïssé game argument shows the centre point property is not definable in MSO.

On the other hand, MSC cannot define non-reachability. □

Proposition. *The satisfiability problem of the* 1*-variable fragment of* MSC *is* PSPACE*-complete.*

Proposition. MSC *contains the* μ*-fragment of the modal* μ*-calculus.*

An FMPA whose transitions are independent of the current state is called forgetful. In other words,

$$\delta(q, S) = \delta(q', S)$$

for all $q, q' \in Q$ and all $S \subseteq Q$.

Theorem. (K. and Reiter, *Inf. Comput.* 2020)
The class of forgetful FMPAs is equiexpressive with MSO on word models. On trees, forgetful FMPAs are more expressive than MSO.

Theorem. (K. and Reiter, *Inf. Comput.* 2020)
The emptiness problem for forgetful FMPAs is in LogSpace*, while for general FMPAs it is undecidable.*

Graph neural networks (GNNs)

Let $\mathrm{mult}(S)$ denote the set of multisets over S, and define $\Pi := \{p_1, \ldots, p_k\}$. Let $d \in \mathbb{N}$.

A **recurrent graph neural network** is a tuple (π, δ, F) that computes in a labeled directed graph $(W, R, p_1, \ldots, p_k)$ as follows.

- The **initialization function** $\pi : \mathcal{P}(\Pi) \to \mathbb{R}^d$ assigns to each node w an initial **feature vector** $x_w^0 = \pi(P)$, where P is the set of proposition symbols true at w.
- In round $t = 1, 2, \ldots$, each node w computes a **new feature vector**

$$x_w^t = \mathrm{COM}\left(x_w^{t-1}, \mathrm{AGG}\left(\{\!\{\, x_u^{t-1} \mid (w,u) \in R \,\}\!\}\right)\right)$$

  where

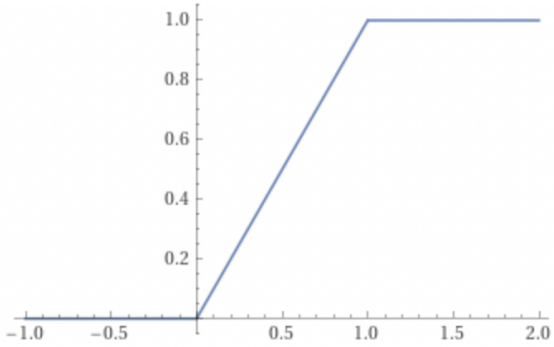
  1. $\mathrm{AGG} : \mathit{mult}(\mathbb{R}^d) \to \mathbb{R}^d$ is an **aggregation function** (typically sum),
  2. $\mathrm{COM} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is a **combination function**.

- GNNFs are exactly like GNNs but use floats and sum as an aggregation function.
- Simple GNNFs are GNNFs with

$$\mathrm{COM}\left(x_v^{t-1}, \mathrm{AGG}\left(\{\{\, x_u^{t-1} | (v,u) \in R \,\}\}\right)\right)$$
$$= f\Big(C\, x_v^{t-1} + \sum_{(v,u)\in R} A\, x_u^{t-1} \;+\; \mathbf{b}\Big)$$

where

1. A and C are matrices and $\mathbf{b} \in \mathbb{R}^d$ a bias factor,
2. $f : \mathbb{R}^d \mapsto \mathbb{R}^d$ is the pointwise applied truncated ReLU, that is the function $\mathrm{ReLU}^*(x) = min(max(0, x), 1)$):



Simple GNNFs are based on the GNN model of Barceló et al. 2020.

In which order do we sum over neighbours' feature vectors?

A typical solution would be to sum in some order given implicitly by the input graph.

But this would lead to graph neural networks not being invariant under isomorphism because **floating point sum is not associative**.

We sum in the order of the floats in the sum. This a common choice also based on numerical analysis as it gives small rounding errors:

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

Nicholas J. Higham. The accuracy of floating point summation. SIAM J. Sci. Comput., 14(4):783-799, 1993.

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.

Theorem. (Ahvonen, Heiman, K., Lutz; 2024)
The following have the same expressive power:

- GNNFs
- Simple GNNFs
- GMSC

GMSC, or **graded** modal substitution calculus, is obtained by allowing diamonds $\Diamond^{\geq k}$ in MSC in addition to standard diamonds $\Diamond$.

Note that every GMSC-program has a natural **counting threshold** given by the maximum k in the diamonds $\Diamond^{\geq k}$.

The input graphs have unbounded degree, and GNNFs sum over unbounded collections of feature vectors of successors, so it may be puzzling how GMSC could suffice to characterize GNNFs.

The key is that sums over multisets of floats have the following property:

Proposition. *There exists a* ***saturation threshold*** $\ell \in \mathbb{Z}_+$ *such if a float* f *occurs at least* ℓ *times in a multiset* S*, then the sum of floats in* S *does not change if we increase the multiplicity of* f *in* S*.*

Theorem. (Ahvonen, Heiman, K., Lutz; NeurIPS 2024)
GNNs are equiexpressive with infinite disjunctions of formulas of GML.

GML:
$\varphi ::= \; p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond^{\geq k}\varphi$

Theorem. (Ahvonen, Heiman, K., Lutz; NeurIPS 2024)
In restriction to MSO-definable node properties, the following have equal expressive power:

1. GNNs
2. GNNFs
3. Simple GNNFs
4. GMSC

Given the earlier results, it suffices to show equivalence of 1 and 4. The non-trivial step is showing $\mathrm{GNN}\boldsymbol{s} \leq \mathrm{GMSC}$. This can be done by translating MSO-formulas into GMSC in a way resembling the construction of tree automata. However, note that MSO can easily express non-GMSC-definable properties, so there are further ingredients dealt with by invariances relating to tree structures.

We note that the acceptance condition we gave for GNNs can be varied and the characterizations still hold. We simply need to vary the acceptance condition of GMSC in the "same way."

The results also hold in restriction to the constant-time case. The infinite disjunctions of GML in this case must contain only modal logic formulas of bounded modal depth.

Graph Transformers

GPS-networks are a powerful Graph Transformer architecture based essentially on the seminal attention mechanism of Vasvani et al., NIPS 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin: Attention is All you Need. NIPS 2017: 5998-6008

Consider a graph with n nodes, each labeled with a vector in $\mathbb{R}^d$. This labeled domain of the graph gives rise to a matrix $D \in \mathbb{R}^{n \times d}$.

Attention updates the label vectors:

$$\mathrm{softmax}\left(\frac{(DW_Q)(DW_K)^{\mathsf{Transpose}}}{\sqrt{d_k}}\right)(DW_V)$$

which uses three constant matrices of the GPS-network:

- $W_Q \in \mathbb{R}^{d \times d_k}$ is the "query weight matrix"
- $W_K \in \mathbb{R}^{d \times d_k}$ is the "key weight matrix"
- $W_V \in \mathbb{R}^{d \times d_v}$ is the "value weight matrix"

In addition to the attention mechanism, a GPS-network runs a constant-iteration GNN.

Theorem. (Ahvonen, Funk, Heiman, K., Lutz, AAAI 2026.) In restriction to properties definable in first-order logic, GPS-networks are equally expressive with GML + global modality $\langle G \rangle$.

$$\varphi \ ::= \ \ p \mid \neg\varphi \mid \varphi \vee \varphi \mid \ \Diamond^{\geq k}\varphi \mid \langle G \rangle \varphi$$

$M, w \models \langle G \rangle \varphi$

$\Leftrightarrow$

$M, u \models \varphi$ for some u in the model.

Theorem. (Ahvonen, Funk, Heiman, K., Lutz, AAAI 2026.) When computing with floats, GPS-networks **with floats** are equally expressive with GML + counting global modality $\langle G \rangle^{\leq k}$.

$$\varphi \ ::= \ \ p \mid \neg\varphi \mid \varphi \vee \varphi \mid \ \Diamond^{\geq k}\varphi \mid \langle G \rangle^{\geq k}\varphi$$

$M, w \models \langle G \rangle^{\geq k} \varphi$

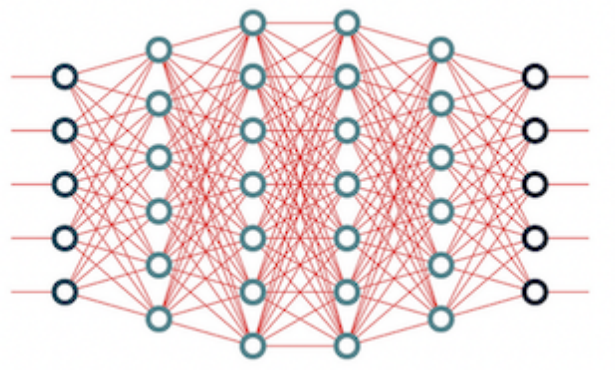
$\Leftrightarrow$

$M, u \models \varphi$ for at least k elements u of the model.

The reason floats increase the expressive power is explained by the so-called underflow phenomenon: calculations that should give something very close to zero can give precisely zero over floats.

Underflow enables discretizations that increase the expressive power so that counting becomes possible.

Basic Neural Networks



When considering recurrent neural networks based on a floating point system, the following holds.

Theorem (Ahvonen, Heiman, K, CSL 2024). *Given a recurrent neural network NN with piecewise polynomial activation functions, we can construct an equivalent Diamond-free GMSC-program* Π *such that* the size of Π is

$$\mathcal{O}\Big(N(\Delta + P\Omega^2)\Big)$$

where

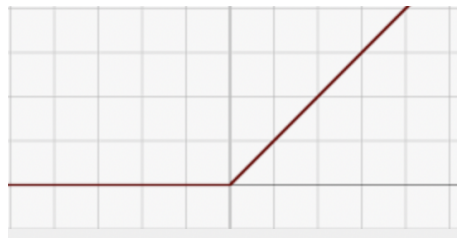
- N is the number of nodes of the neural network
- Δ is the maximum in-degree of the neural network
- P is the number of polynomials in the piecewise polynomial activation function
- Ω is the maximum order of the polynomials used

The **computation delay factor** of the simulating diamond-free GMSC-program is $\mathcal{O}\Big(\log(\Omega)\log(\Delta)\Big)$.

Theorem. *Given a diamond-free GMSC-program Π of size s, we can construct a neural network (for any floating-point system) with*

- at most s nodes,
- ReLU activation,
- computation delay $\mathcal{O}(d)$ where d is the maximum nesting depth of Boolean operators in the rule bodies.

Corollary. (Ahvonen, Heiman, K. CSL 2024). Recurrent NNs *can be translated—with a polynomial blow-up in size—to equivalent NNs with ReLU as the activation function.*



Corollary. (Ahvonen, Heiman, K., JLC 2026). Recurrent NNs can be translated—with a polynomial blow-up in size—to equivalent NNs with no activation function, i.e., linear NNs. Floats explain this.

Similar results are also proved for the non-recurrent, feedforward neural networks in the JLC 2026 article.

Thanks!