Logic meets LLMs: Attention has a logic

Miguel Moreno
Tampere University
(joint work with Veeti Ahvonen, Damian Heiman, Antti Kuusisto, and Matias Selin)

UN Encuentro 70

18 - 22 August, 2026

Motivation

Modern computational models (e.g. neural networks) have been proven very useful for a variety of problems.

In particular, the transformers architecture in the LLMs.

However, many of these models are only understood at a heuristic level.

To address this weakness we use logic to study their expressive power.

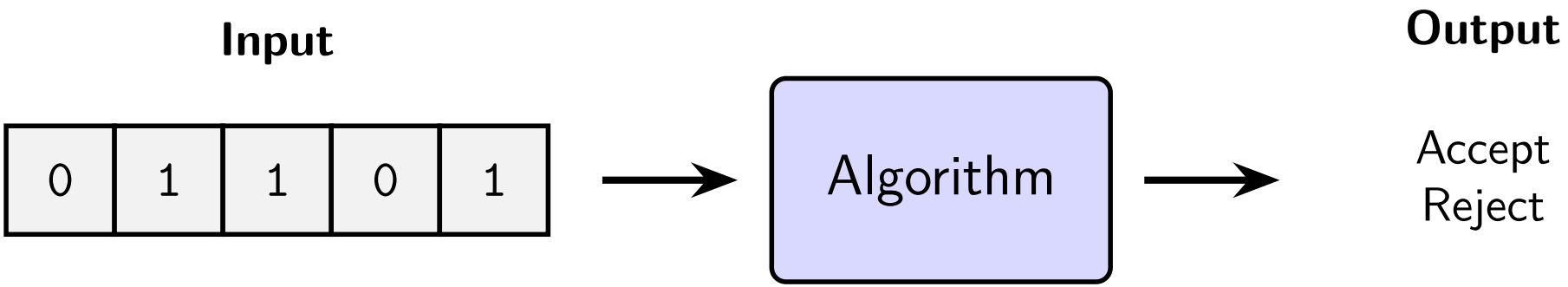
Big picture

One of the major goals in the theory of transformers is to answer a simple question:

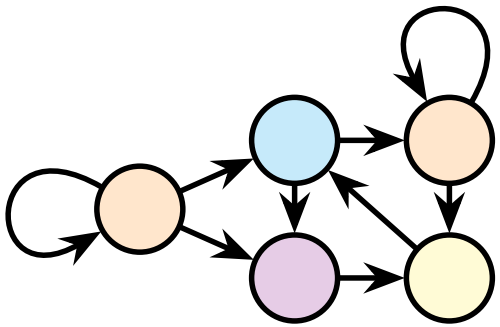
What can different transformer architectures compute?

We study the original encoder–decoder architecture, revealing a remarkably clean correspondence with temporal logic and distributed automata.

Typical sequential computing framework

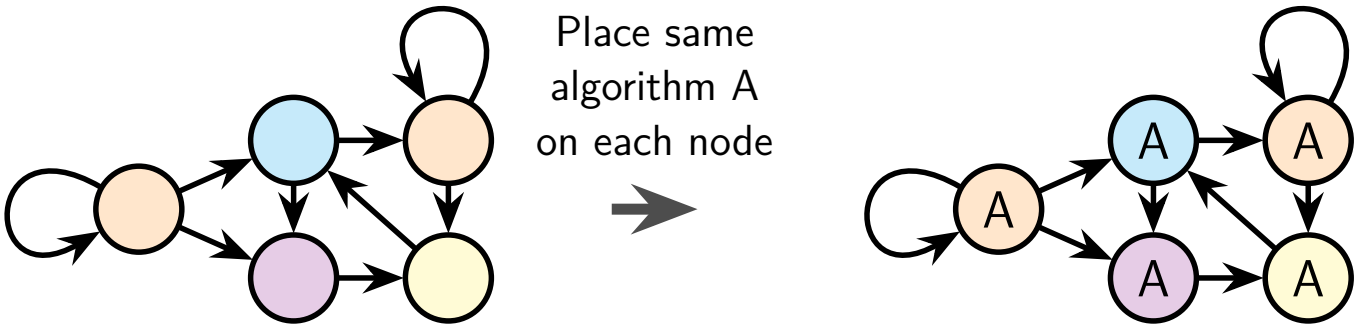


Distributed computing



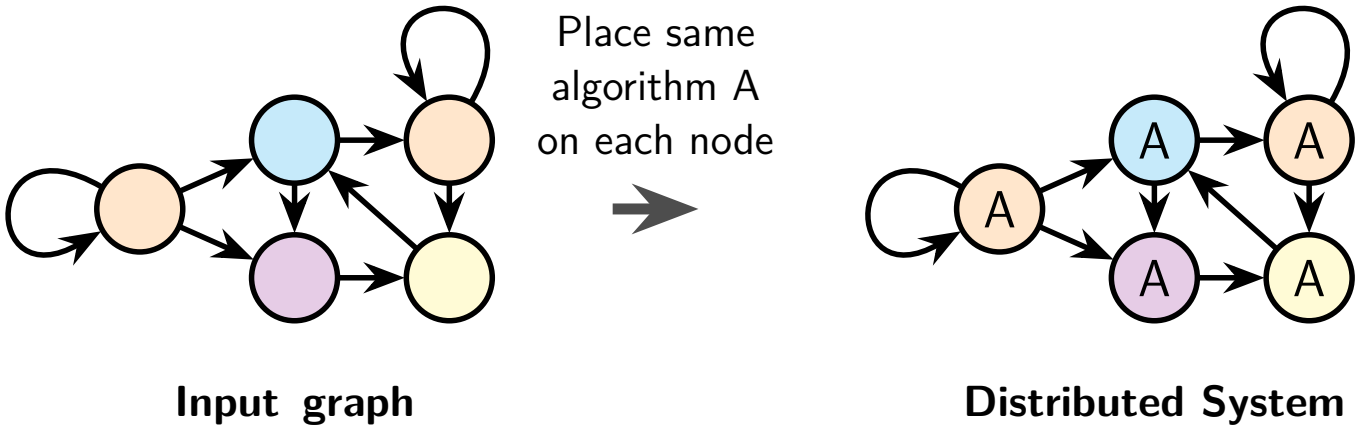
Input graph

Distributed computing

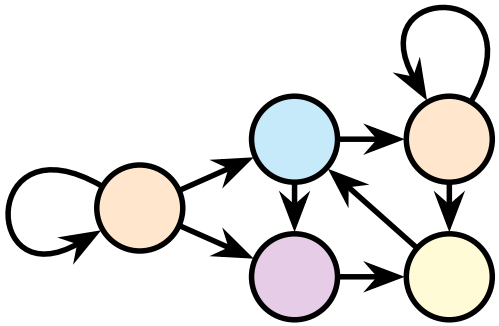


Input graph **Distributed System**

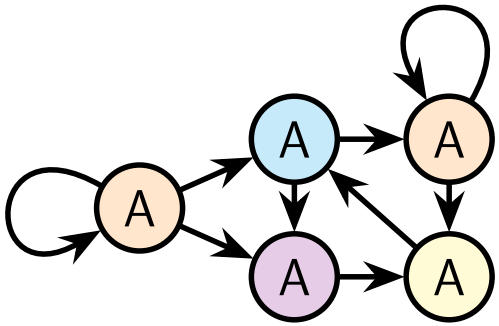
Distributed computing



Distributed computing



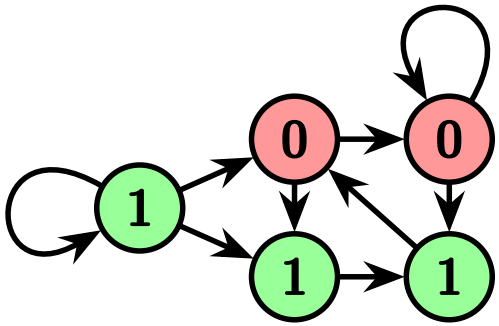
Place same algorithm A on each node



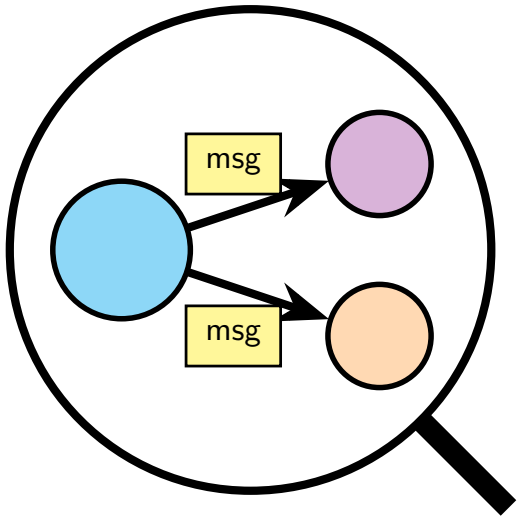
Input graph

Distributed System

Local message passing



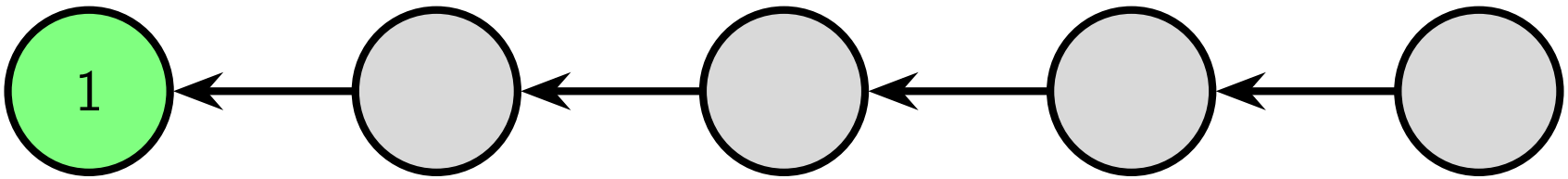
Local Outputs



Local View

Example: Reachability

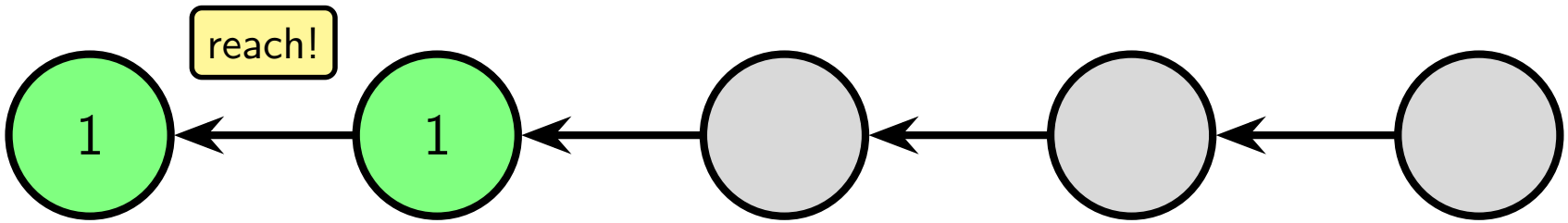
Initial State



Source (green) is reachable. Others are **unknown** (gray).

Example: Reachability

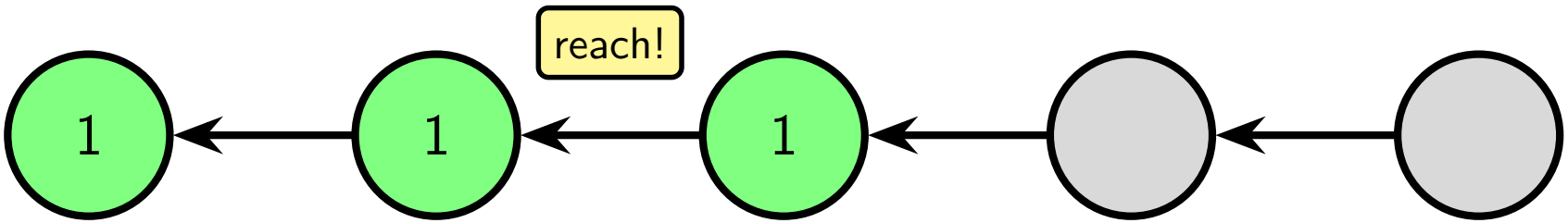
Round 1



Source sends “reach” — Node 2 receives from its out-neighbour.

Example: Reachability

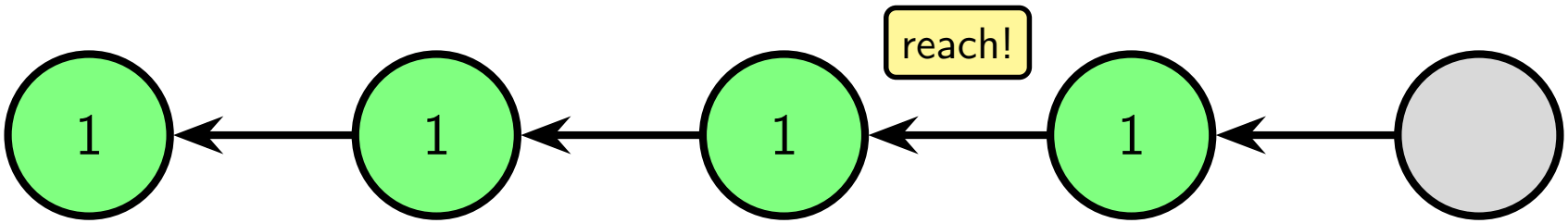
Round 2



Message propagates: each node receives from its out-neighbour.

Example: Reachability

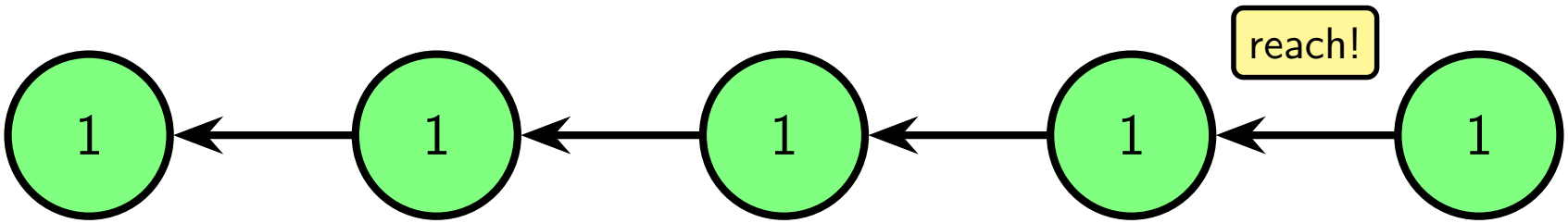
Round 3



Information continues toward the root.

Example: Reachability

Round 4



Root receives the message.

Distributed system

A distributed system is defined by a labelled directed graph $(W, R, p_1, \ldots, p_k)$ 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 i-th input bit at node w is 1 iff $w \in p_i$.

Distributed automata

A **finite distributed automaton** (or FDA) is a tuple (Q, δ, π, F)

- Q is a finite non-empty set of states,
- $\delta\colon Q \times \mathcal{P}(Q) \to Q$ is a transition function that gives a new state to each node according to
  - the node's own current state
  - and the set of node's neighbours' states
- $\pi\colon \mathcal{P}(\{p_1, \ldots, p_n\}) \to Q$ is an initialization function from node-labels to an initial state,
- $F \subseteq Q$ is a set of accepting states.

If Q is countably infinite, we simply call such an automaton a DA. A DA accepts a node of a graph if it enters an accepting state during the run.

Distributed automata counting

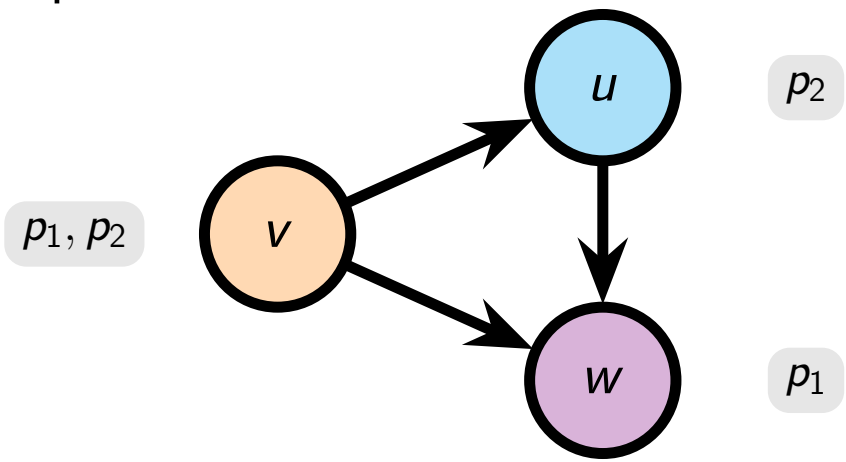
A **finite distributed automaton** (or FDA) is a tuple (Q, δ, π, F)

- Q is a finite non-empty set of states,
- $\delta\colon Q \times \mathcal{M}_k(Q) \times \mathcal{M}_k(Q) \to Q$ is a transition function that gives a new state to each node according to
  - the node's own current state
  - the k-capped multiset of node's neighbours' states
  - the k-capped multiset of all node's states
- $\pi\colon \mathcal{P}(\{p_1, \ldots, p_n\}) \to Q$ is an initialization function from node-labels to an initial state,
- $F \subseteq Q$ is a set of accepting states.

If Q is countably infinite, we simply call such an automaton a DA. A DA accepts a node of a graph if it enters an accepting state during the run.

Overview

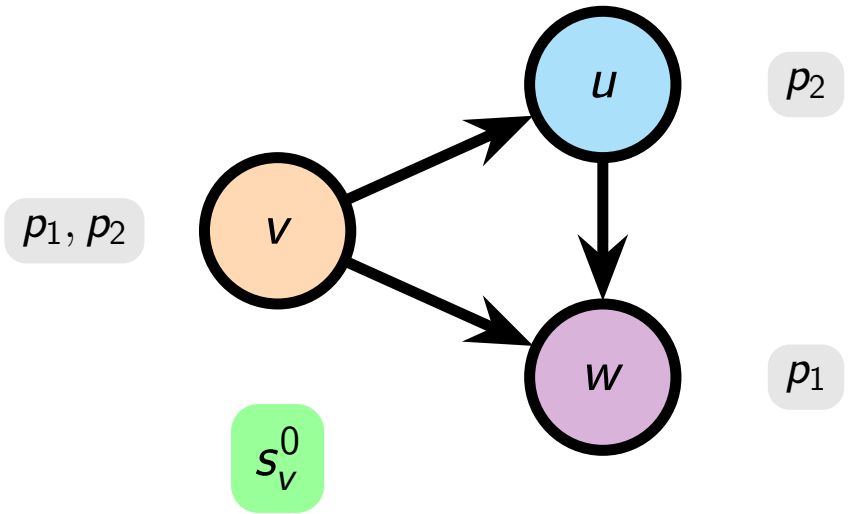
Formally a graph is a tuple $(V, E, p_1, \ldots, p_n)$, where V is a set of nodes, E is a set of edges and node-labels $p_i \subseteq V$ encode a local input at each node.



Overview

Formally a graph is a tuple $(V, E, p_1, \ldots, p_n)$, where V is a set of nodes, E is a set of edges and node-labels $p_i \subseteq V$ encode a local input at each node.

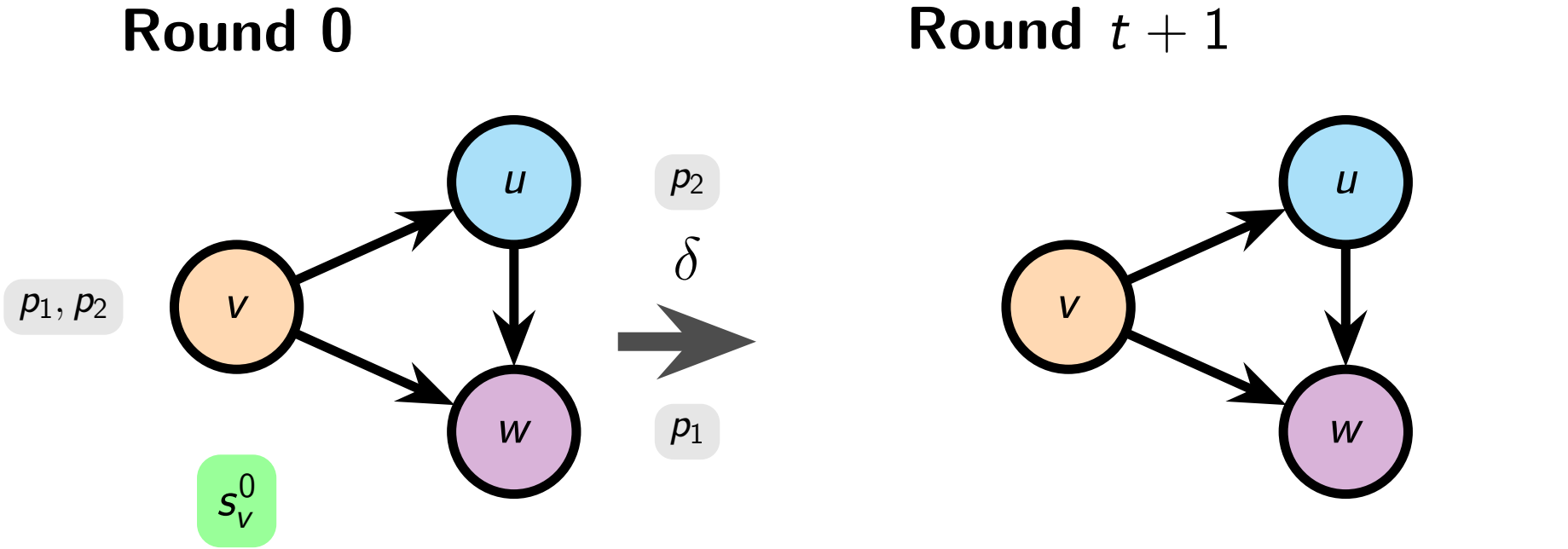
Round 0



$$s_v^0 = \pi(p_1, p_2)$$

Overview

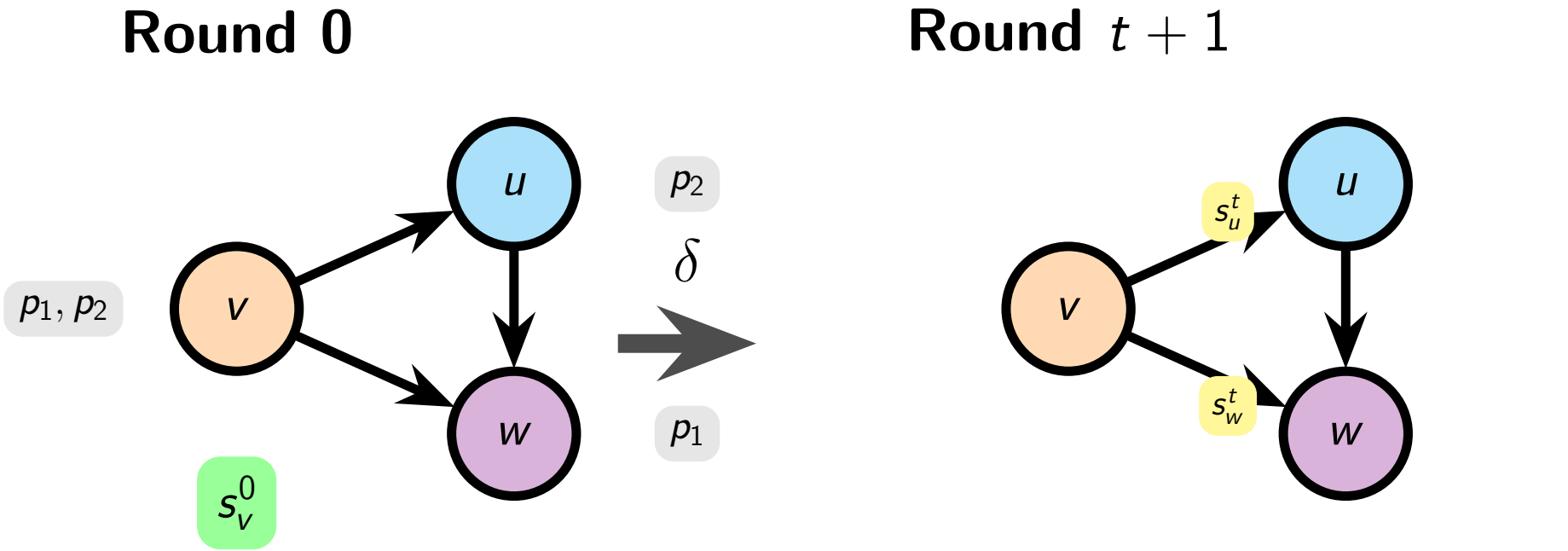
Formally a graph is a tuple $(V, E, p_1, \ldots, p_n)$, where V is a set of nodes, E is a set of edges and node-labels $p_i \subseteq V$ encode a local input at each node.



$$s_v^0 = \pi(p_1, p_2)$$

Overview

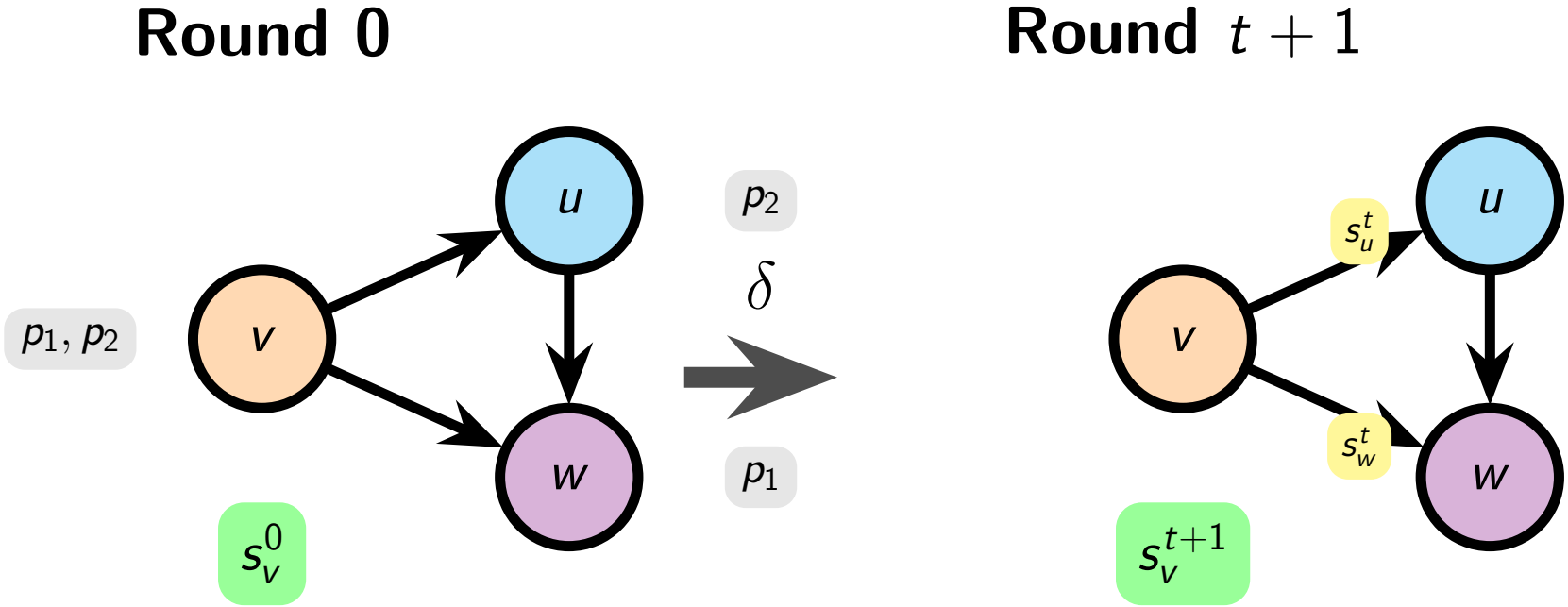
Formally a graph is a tuple $(V, E, p_1, \ldots, p_n)$, where V is a set of nodes, E is a set of edges and node-labels $p_i \subseteq V$ encode a local input at each node.



$$s_v^0 = \pi(p_1, p_2)$$

Overview

Formally a graph is a tuple $(V, E, p_1, \ldots, p_n)$, where V is a set of nodes, E is a set of edges and node-labels $p_i \subseteq V$ encode a local input at each node.



$$s_v^0 = \pi(p_1, p_2)$$

$$s_v^{t+1} = \delta\left(s_v^t, \{s_u^t, s_w^t\}\right)$$

Formally

Each communication round $t \in N$ defines a global configuration $S^t : V \to Q$ ($S^t_v = S^t(v)$).

$S^0_v :=$ initial state at v obtained by the initialization function π.

Define $Q' :=$ the set of states realized by the neigbours of v in round t. Then $S^{t+1}_v :=$ the new state $\delta(S^t_v, Q')$ at v.

Strictly speaking, by "neighbours of v " we here mean out-neigbours.

A node is accepted if it enters an accepting state during the run.

Related work on distributed computing

Hella et al. (PODC 2012) characterize **constant-iteration** distributed automata classes via modal logics.

Modal substitution calculus

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

$$\overbrace{X_1(0) :- p \wedge q}^{\text{base rule}} \qquad \overbrace{X_1 :- X_3 \vee \Box p}^{\text{induction rule}}$$

$$\vdots \qquad\qquad \vdots$$

$$X_n(0) :- \Diamond q \vee \neg q \qquad X_n :- \Diamond X_1$$

- $\varphi ::= \bot \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi$ (ordinary modal logic)
- $\varphi ::= \bot \mid p \mid X \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi$

Semantics

$$\begin{array}{ll} X_1(0) :- \psi_1 & X_1 :- \varphi_1 \\ \vdots & \vdots \\ X_n(0) :- \psi_n & X_n :- \varphi_n \end{array}$$

- $X_i^0 := \psi_i$
- X_i^{t+1} is obtained from the induction formula φ_i by substituting each X_j by X_j^t.

Semantics

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

A program **accepts** a node v of a graph if for some $t \in \mathbb{N}$, the formula X_1^t is true at v.

Here $M = (V, R, p_1, \ldots, p_k)$ is our distributed system and $v \in V$ a node.

Recall $M, v \models \Diamond\varphi$ iff $M, u \models \varphi$ for some u such that $R(v, u)$.

MSC and DAs

Theorem (Kuusisto CSL13)

MSC and finite distributed automata have the same expressive power.

Graph neural network

A **graph neural network** (or GNN) is a counting distributed automaton (δ, π, F) that is obtained by a learning process.
The initialization function $\pi\colon \mathcal{P}(\Pi) \to \mathbb{R}^d$ gives an initial feature vector s_v^0 for each node v.
The transition function δ is induced by

- a combination function $\mathrm{COM}\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$
- an aggregation function $\mathrm{AGG}\colon \mathit{mult}(\mathbb{R}^d) \to \mathbb{R}^d$ (typically sum)

In round $t = 1, 2, \ldots$, every node v computes a new state

$$s_v^{t+1} := \mathrm{COM}(s_v^t, \mathrm{AGG}(\{\{s_u^t \mid (v,u) \in E\}\}))$$

Similarly to DAs, a GNN accepts a node v of a graph if in some round $t \in \mathbb{N}$, v visits an accepting feature vector.

GNN and GMSC

A GNN with **floating-point numbers** acts similarly to a GNN with real numbers.

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

Theorem (Ahvonen, Heiman, Kuusisto, Lutz, 24)

The following have the same expressive power:

- *GMSC*
- *floating-point GNNs*

CPG Automaton

A **counting past-global distributed automaton** (CPG-automaton) is a tuple (Q, π, δ, n, b)

- $Q \subseteq \{0,1\}^m$ is a finite non-empty set of states,
- $\delta \colon Q \times \mathcal{M}_k(Q) \times \mathcal{M}_k(Q) \to Q$ is a transition function, where $\mathcal{M}_k(Q)$ is the set of k-**capped multisets** over Q,
- $\pi \colon \Sigma \to Q$ is an initialization function,
- $n \in \mathbb{N}$ is the output round,
- $b \in \mathbb{Z}_+$ is the output length.

GPTL^{-} Logic

The Σ-formulae of the temporal logic GPTL^{-} are constructed by the grammar

$$\varphi ::= \bot \mid p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \langle G \rangle^{\mathsf{pre}}_{\geq k} \varphi \mid \langle P \rangle^{\mathsf{suf}} \varphi,$$

Word-shaped graphs

Let Σ be a finite set of **tokens**.

A Σ-**labelled word-shaped graph** is a tuple (V, E, λ):

- $V = [n]$
- $(i, j) \in E$ iff $j = i + 1$
- $\lambda : V \to \Sigma$ (**labelling function**)

Word-shaped graphs

A Σ-**labelled two-sorted word-shaped graph** is a $\Sigma \cup \{\mathrm{BOS}\}$-labelled word-shaped graph such that:

Only one vertex v has BOS assigned.

The **prefix** is the subgraph induced by $[v-1]$.

The **suffix** is the subgraph induced by $[n] - [v-1]$.

Masking and softmax

The **strict future mask** is the function on square matrices given by

$$\mathrm{mask}(S)_{ij} := \begin{cases} -\infty & \text{when } j \geq i, \\ S_{ij} & \text{otherwise.} \end{cases}$$

Softmax is the function on vectors $\mathbf{x}$ given by

$$\mathrm{softmax}(\mathbf{x})_i := \frac{e^{x_i - b_{\mathbf{x}}}}{\sum_{j=1}^{n} e^{x_j - b_{\mathbf{x}}}}.$$

Where $b_{\mathbf{x}} := \max\{x_1, \ldots, x_{|\mathbf{x}|}\}$.

Attention

An **attention head** is a function defined by:

$$H_{\text{mask}}(X, Y) := \text{softmax}\left(\text{mask}\left(\frac{(YW^Q)(XW^K)^\top}{\sqrt{d_k}}\right)\right)(XW^V).$$

Where W^Q, W^K, W^V are parameter matrices.

Attention

A **masked multi-head self-attention layer** is a function defined by:

$$\mathsf{MHSA}_{\mathsf{mask}}(X, X) := \mathrm{concat}(H^{(1)}_{\mathsf{mask}}(X, X), \ldots, H^{(h)}_{\mathsf{mask}}(X, X))W^{O}.$$

Where W^{O} is parameter matrix.

Multi-head cross-attention layers, MHCA, are defined similarly.

Multi-layer perceptrons

A **multi-layer perceptron** is a function defined by:

$$M(\mathbf{x}) := \mathrm{ReLU}(\mathbf{x}W_1 + b_1)W_2 + b_2.$$

Where W_1 and W_2 are parameter matrices, and b_1 and b_2 are bias vectors.

ReLU is applied element-wise. M is applied to a matrix separately to each row.

Encoder

An **encoder layer** is a function given by:

$$\mathrm{EL}(X) = M(X') + X'$$

where $X' = \mathrm{MHSA}(X) + X$.

An **encoder of length** L is a composition of encoder layers:

$$\mathrm{E}(X) = (\mathrm{EL}_L \circ \cdots \circ \mathrm{EL}_1)(X).$$

Dencoder

A **decoder layer** is a function given by:

$$\mathrm{DL}(X, Y) = M(Y'') + Y''$$

where $Y'' = \mathrm{MHCA}(X, Y') + Y'$ and $Y' = \mathrm{MHSA}_{\mathrm{mask}}(Y) + Y$.

A **decoder of length** L is a composition of decoder layers:

$$\mathrm{D}(X, Y) = \mathrm{DL}_L(X, \mathrm{DL}_{L-1}(X, \cdots \mathrm{DL}_2(X, \mathrm{DL}_1(X, Y)) \cdots)).$$

Output

An **output head** is a function given by:

$$\mathrm{Out}(Y) = Y \cdot W^{\mathrm{out}} + b^{\mathrm{out}},$$

where W^{out} and b^{out} are learned parameters.

Encoder–decoder transformer

An **encoder–decoder transformer** maps Σ-labelled two-sorted graph $\mathcal{G}$ to matrices by:

$$\mathsf{T}(\mathcal{G}) = \mathsf{Out}(\mathsf{D}(\mathsf{E}(\mathrm{em}(\mathcal{G}_p)), \mathrm{em}(\mathcal{G}_s))),$$

where $\mathcal{G}_p$ is the prefix and $\mathcal{G}_s$ the suffix of $\mathcal{G}$.

Equivalence

Theorem (Ahvonen, Heiman, Kuusisto, M., Selin, 26)

The following have the same expressive power:

- *Encoder–decoder transformers without the final softmax*
- GPTL^{-} *Logic*
- CPG *Automata*

Thank you

Article at: https://arxiv.org/abs/2605.07705

References

- V. Ahvonen, D. Heiman, A. Kuusisto, and C. Lutz, *Logical Characterizations of Recurrent Graph Neural Networks with Reals and Floats.*, In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advancesin Neural Information Processing Systems* **37**, 104205–104249 (2024).
- L. Hella, M. Järvisalo, A. Kuusisto, J. Laurinharju, T. Lempiäinen, K. Luosto, J. Suomela, and J. Virtema, *Weak Models of Distributed Computing, with Connections to Modal Logic*, Distrib. Comput. **28**, 31–53 (2015).

References

- A. Kuusisto, *Modal Logic and Distributed Message Passing Automata*, In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013)* of Leibniz International Proceedings in Informatics (LIPIcs), **23**, 452–468 (1913).
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. 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*, Curran Associates **30**, (2017).