Logical Foundations for Modern Computational Models

Veeti Ahvonen

Tampere University

Mathematics Research Centre

28.1.2026

Motivation

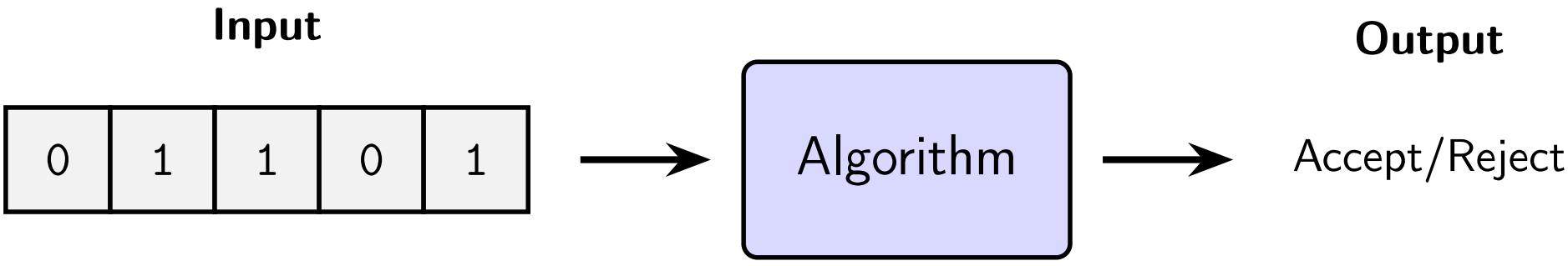
Modern computational models (e.g. neural networks) have been proven to be highly successful in solving a wide variety of problems.

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

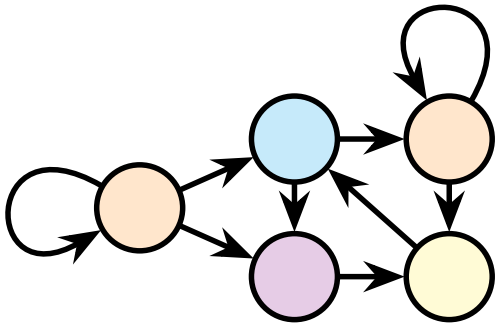
To address this weakness we use logic to study the expressive power of

1. distributed computing models, and
2. machine learning frameworks.

Typical sequential computing framework

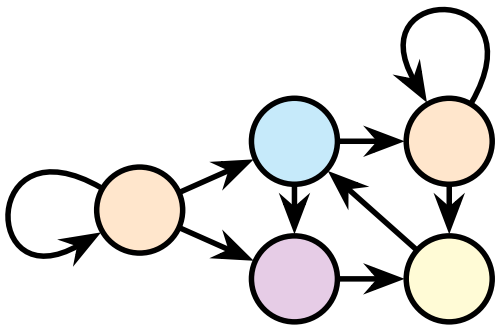


Distributed computing



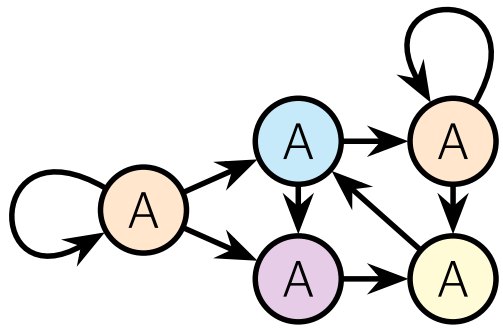
Input graph

Distributed computing



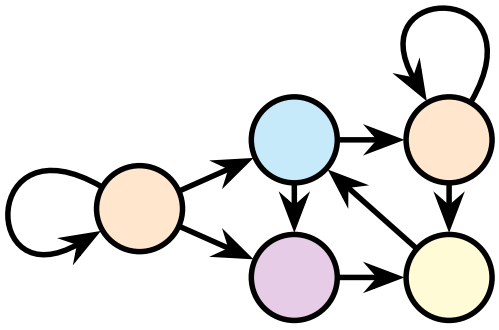
Input graph

Place same
algorithm A
on each node



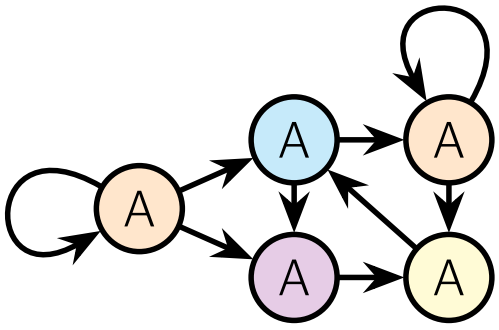
Distributed System

Distributed computing



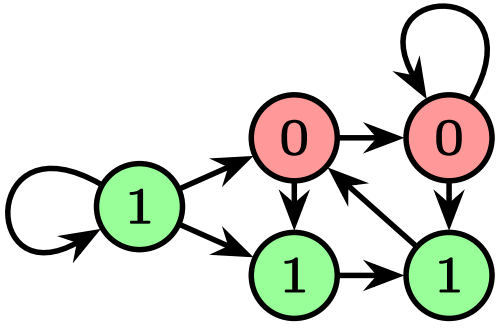
Input graph

Place same algorithm A on each node



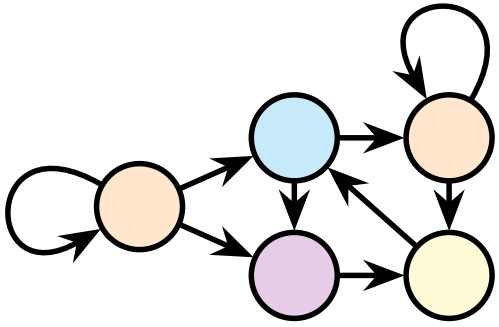
Distributed System

Local message passing

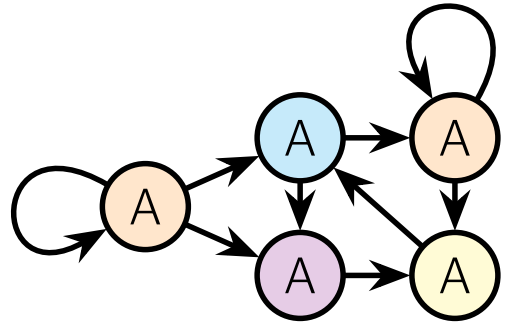


Local Outputs

Distributed computing



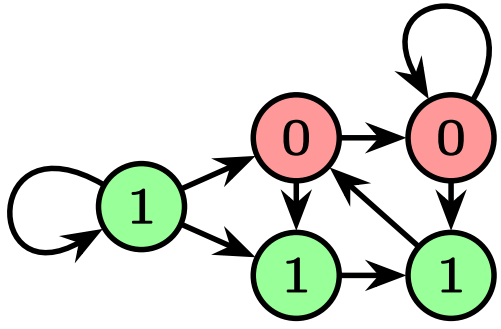
Place same algorithm A on each node



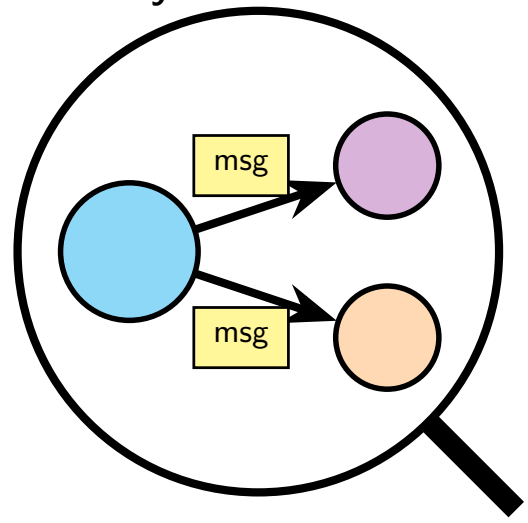
Input graph

Distributed System

Local message passing



Local Outputs

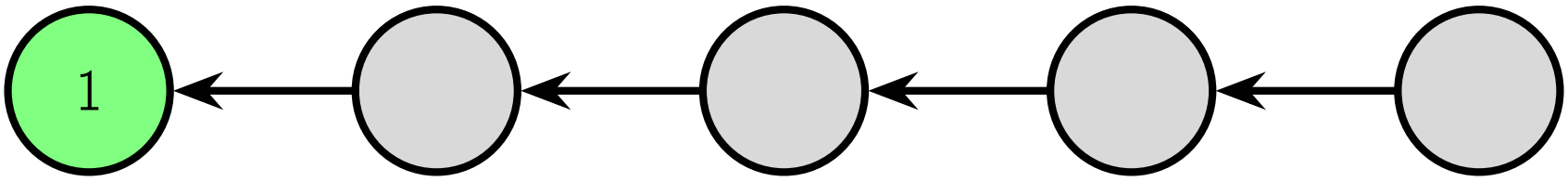


Local View

Each node updates its state according to the received messages and node's own state.

Example: Reachability

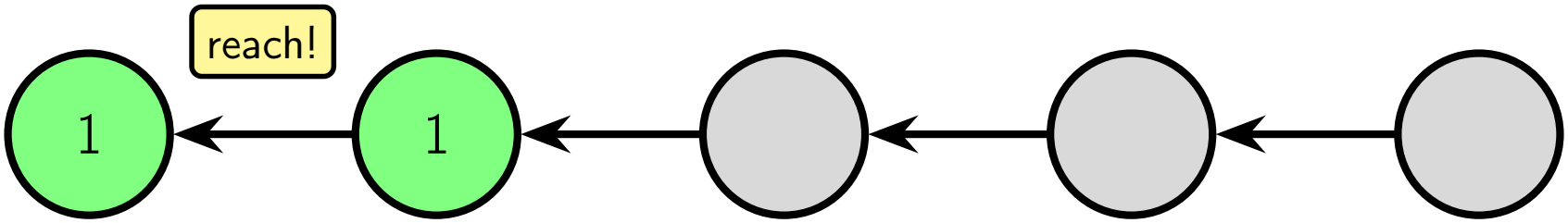
Initial State



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

Example: Reachability

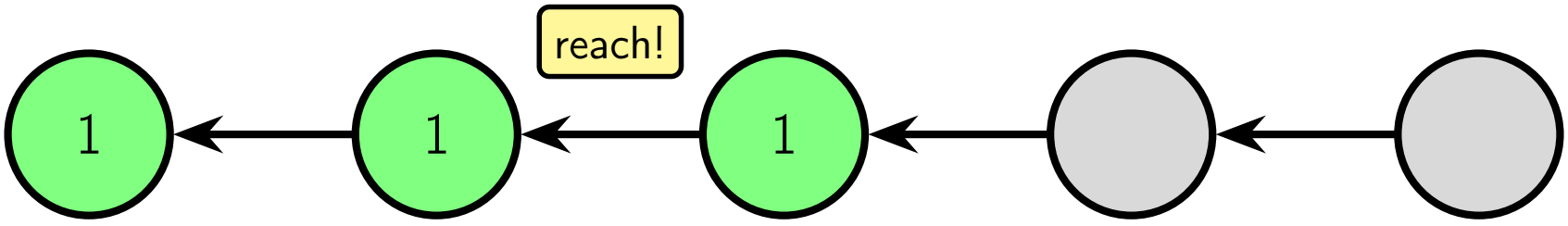
Round 1



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

Example: Reachability

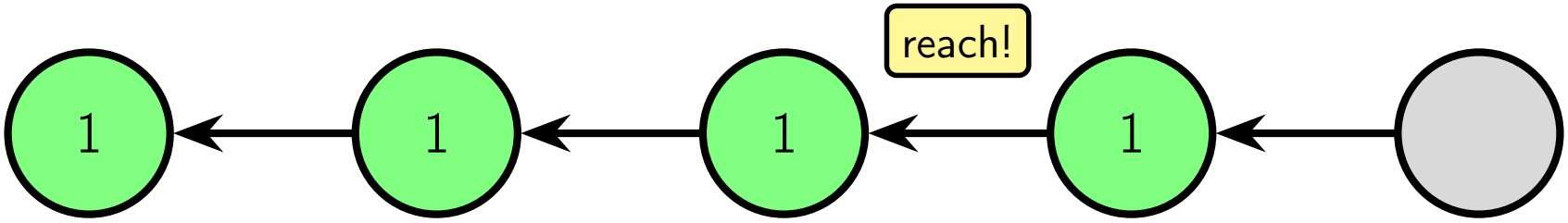
Round 2



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

Example: Reachability

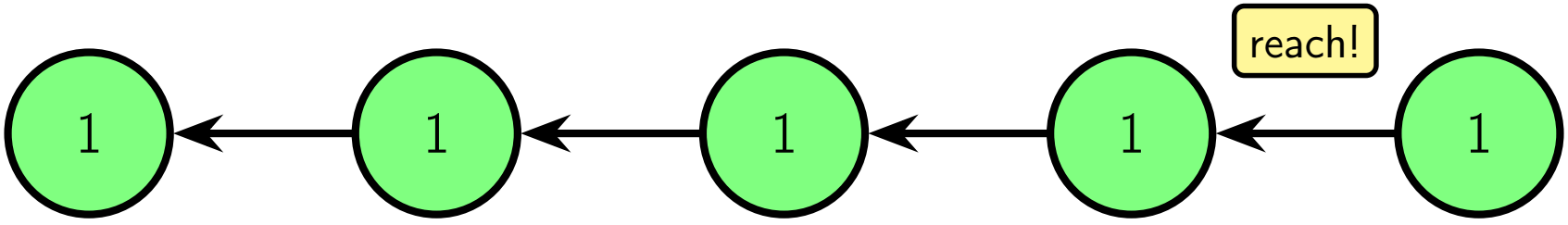
Round 3



Information continues toward the root.

Example: Reachability

Round 4



Root receives the message.

Related work on distributed computing

Hella et al. (PODC 2012) built links between variants of modal logic and **constant-iteration** distributed automata.

Kuusisto (CSL 2013) used Modal substitution calculus (or MSC)—a rule-based extension of modal logic—to characterize a class of distributed automata that may run for an **unbounded** number of rounds.

Reiter (ICALP 2017) used the μ-fragment of the modal μ-calculus to characterize distributed automata in the **asynchronous setting**.

Distributed automata

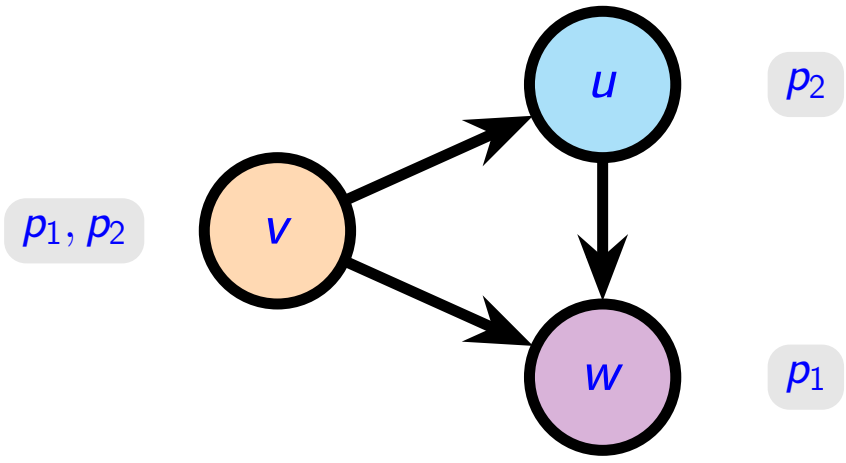
A **finite distributed automaton** (or FDA) is a tuple (Q, δ, π, F)

- 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
- $\pi\colon \mathcal{P}(\Pi) \to Q$ 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.

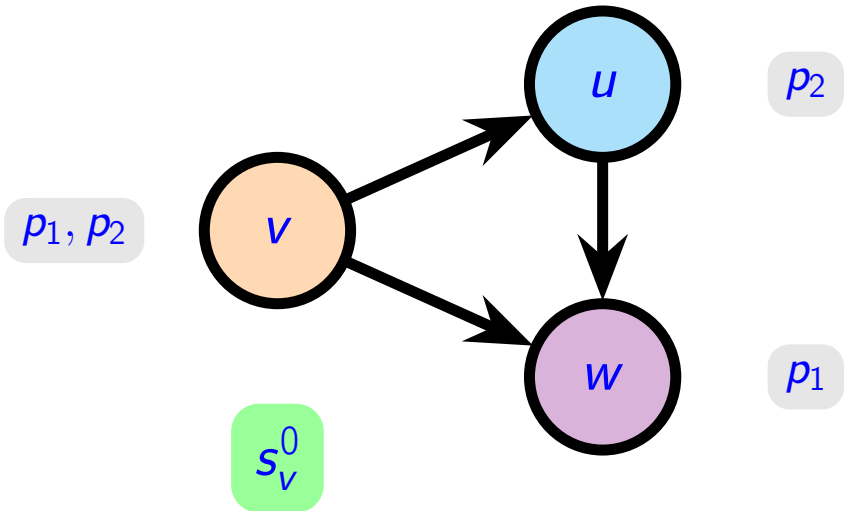
A DA accepts a node of a graph if it enters an accepting state during the run.

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.



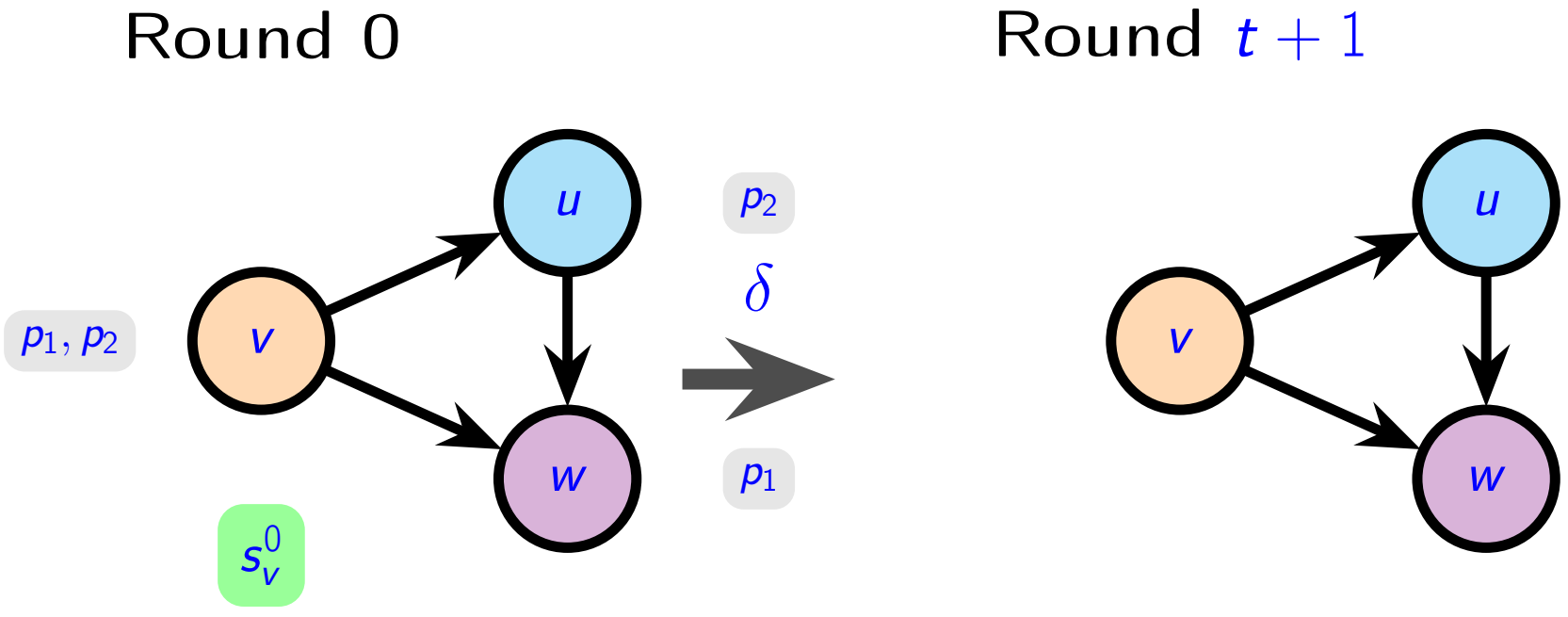
Formally a graph is a tuple $(V, E, p_1, \ldots, p_n)$, where V is a set of nodes, E is a set of edges and node-labels $p_i \subseteq V$ encode a local input at each node.

Round 0



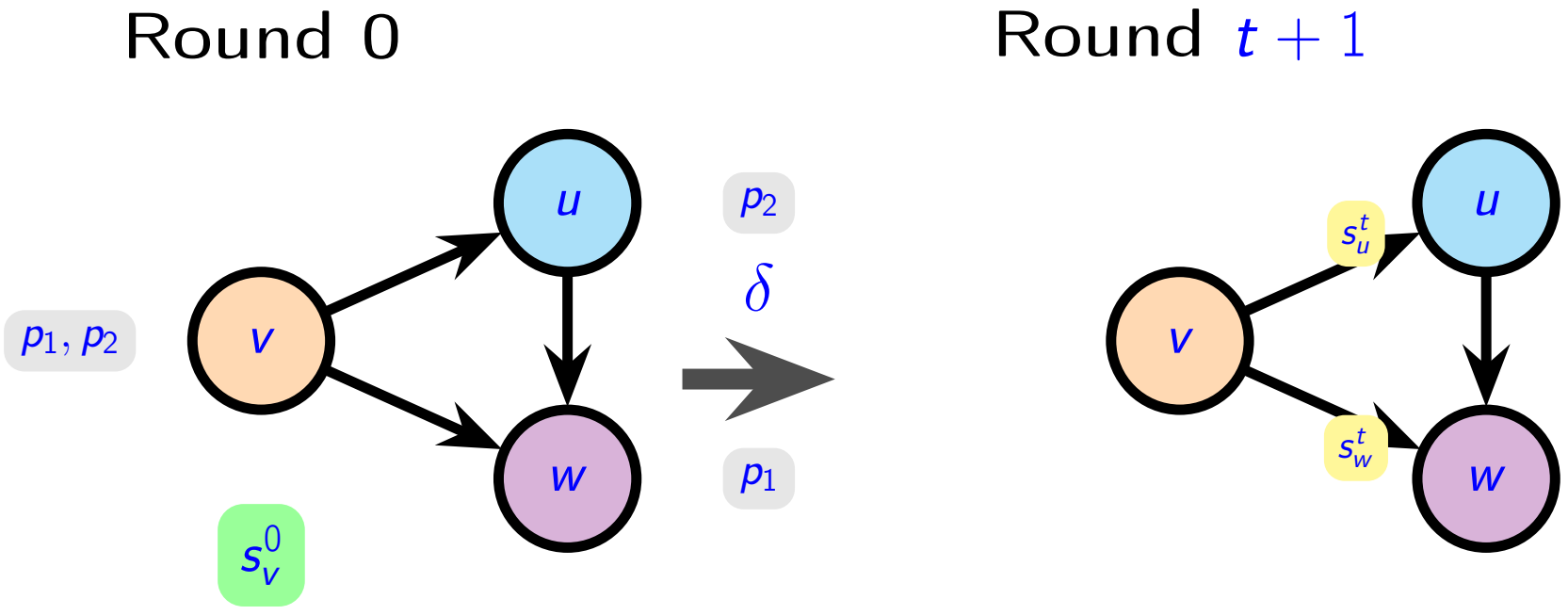
$s_v^0 = \pi(p_1, p_2)$

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.



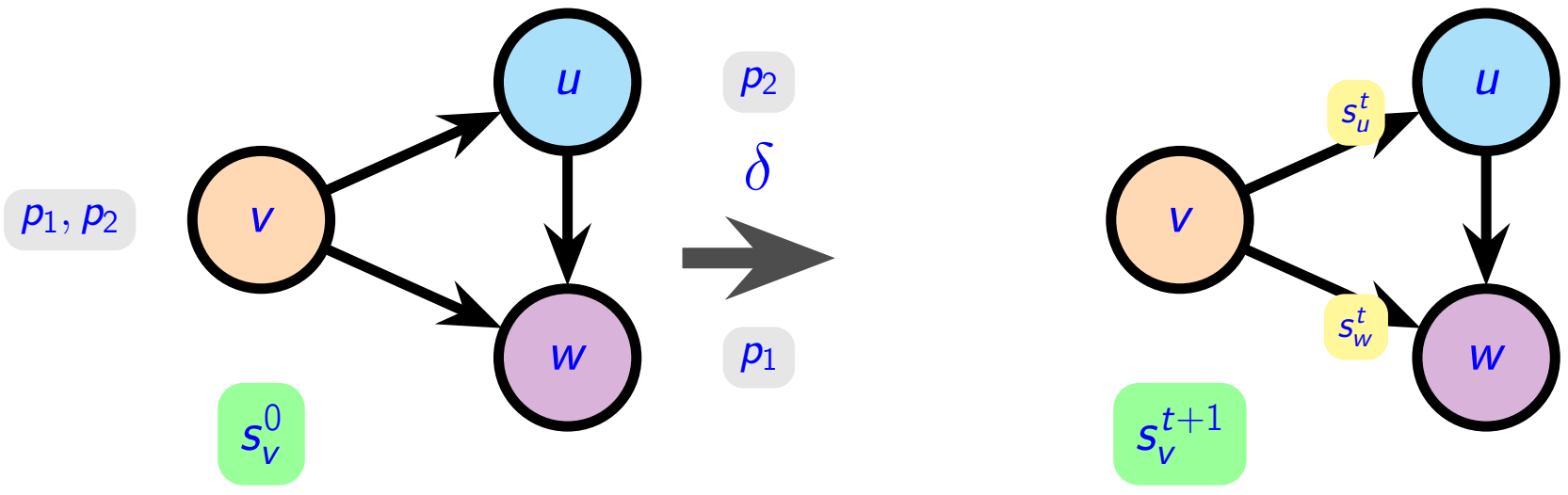
$s_v^0 = \pi(p_1, p_2)$

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.



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

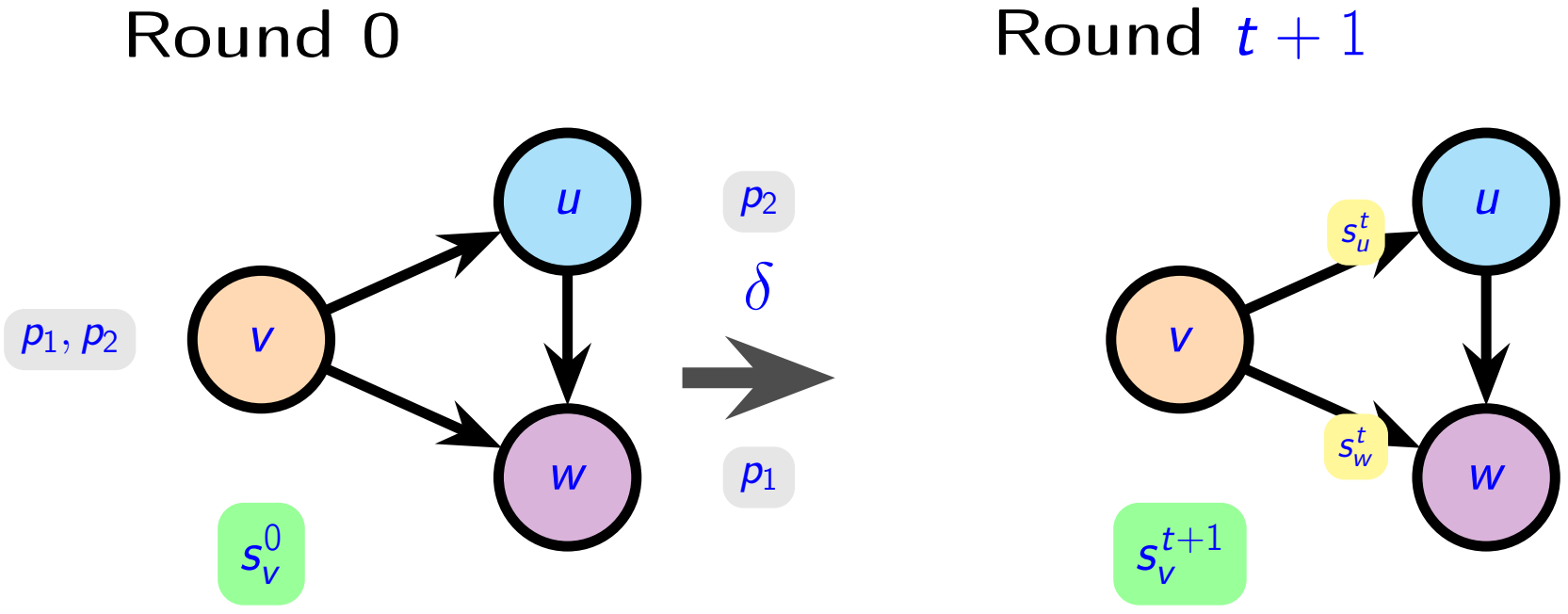
Formally a graph is a tuple $(V, E, p_1, \ldots, p_n)$, where V is a set of nodes, E is a set of edges and node-labels $p_i \subseteq V$ encode a local input at each node.



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

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

Formally 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.



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

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

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

Modal substitution calculus

base rule

$$X_1(0) :- p \wedge q$$

$$\vdots$$

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

induction rule

$$X_1 :- X_3 \vee \Box p$$

$$\vdots$$

$$X_n :- \Diamond X_1$$

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

Semantics:

- X_i^0 denotes the base formula of X_i
- X_i^{t+1} is obtained from the induction formula of X_j by substituting each X_i by X_j^t.

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.

MSC example

The reachability problem for p:

$$X(0) :- p, \qquad X :- \Diamond X.$$

p, $\Diamond p$, $\Diamond\Diamond p$, $\ldots$

If p is reachable from the studied node, it will eventually be accepted.

Some results on DAs

Theorem 1 (Kuusisto CSL13)

MSC and finite distributed automata have the same expressive power.

We can further extend this for bounded counting finite distributed automata. A k-bounded counting FDA is defined analogously to an FDA, but its transition functions are of the form

$$\delta\colon Q \times \mathit{mult}_k(Q) \to Q$$

$\mathit{mult}_k(Q)$ denotes the multisets of Q whose multiplicities are at most k.

For example, the 2-multiset of $\{\{1,1,1,2,2\}\}$ is $\{\{1,1,2,2\}\}$.

Theorem 2 (A., Heiman, Kuusisto 24)

GMSC and bounded counting FDAs have the same expressive power.

Here GMSC is the extension of MSC with counting modalities $\Diamond_{\geq k}$.

Some results on DAs

Define ω-GML to be the logic obtained from graded modal logic (or GML) by extending its syntax with countably infinite disjunctions.

Theorem 3 (A., Heiman, Kuusisto 24)

ω-GML and counting distributed automata have the same expressive power.

Notice that here an automaton can have countably infinitely many states.

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 $\mathrm{COM}\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ (typically an NN)
- 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.

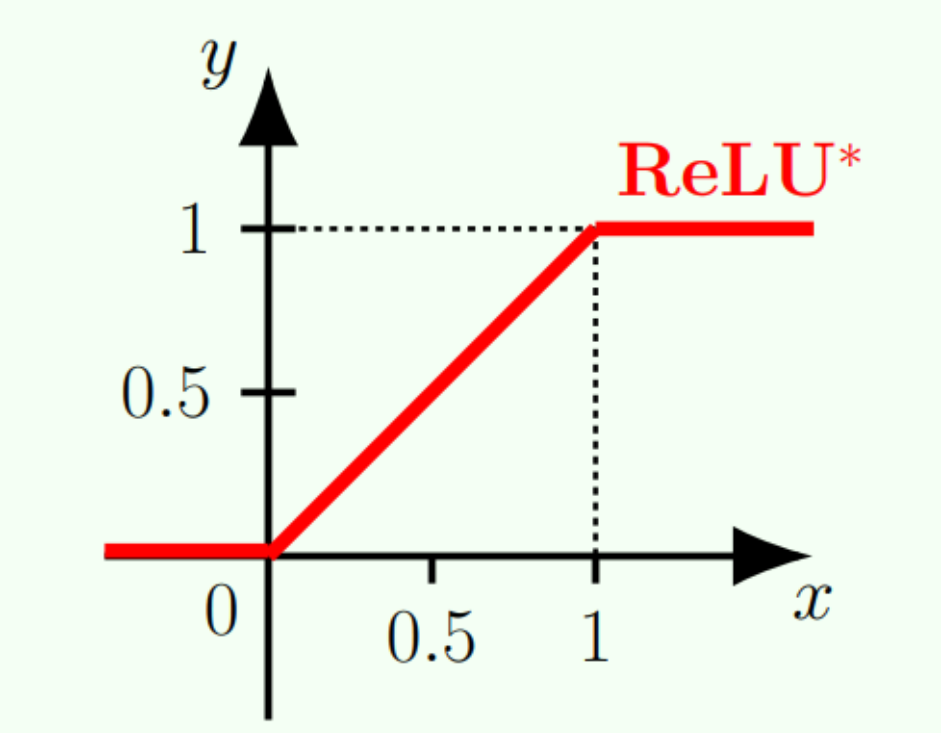
Simple GNNs

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

$$\boldsymbol{x}_v^t = \mathrm{ReLU}^*\left(\boldsymbol{x}_v^{t-1} \cdot \boldsymbol{C} + \Big(\sum_{(v,u)\in E} \boldsymbol{x}_u^{t-1}\Big) \cdot \boldsymbol{A} + \boldsymbol{b}\right).$$

$\boldsymbol{C}, \boldsymbol{A} \in \mathbb{R}^{d\times d}$ are matrices and $\boldsymbol{b} \in \mathbb{R}^d$ is a bias vector.

$\mathrm{ReLU}^* = \min(\max(0, x), 1)$ is applied pointwise.



Related work on GNNs

Theorem 4 (Barceló et al. ICLR20)

In restriction to first-order logic, the following have the same expressive power:

- *constant-iteration GNNs with reals*
- *Simple constant-iteration GNNs with reals*
- *Graded modal logic*

Also many other relevant results on constant-iteration GNNs: Grohe (LICS21, LICS23), Benedikt et al. (ICALP24), Grau et al. (AAAI26), etc.

Also Pflüger et al. (AAAI24) gave a logical characterization in *restriction to a background logic*.

Floating-point GNNs

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

A GNN with floats is fixed with a finite set of floating-point numbers and real arithmetic operations are approximated.

However, float sum as an aggregation function is **bounded**, because

- Floating-point sum is not associative
- We cannot sum over a random node order since otherwise isomorphism invariance is violated
- We use (increasing) order of floats instead

Results on GNNs

Theorem 5 (A., Heiman, Kuusisto, Lutz, 24)

Same expressive power:

- *GMSC*
- *floating-point GNNs*
- *Simple floating-point GNNs*
- *bounded counting FDAs*

Theorem 6 (A., Heiman, Kuusisto, Lutz, 24)

Same expressive power:

- ω*-GML*
- *GNNs with reals*
- *counting DAs*

Results on GNNs

Theorem 7 (A., Heiman, Kuusisto, Lutz, 24)

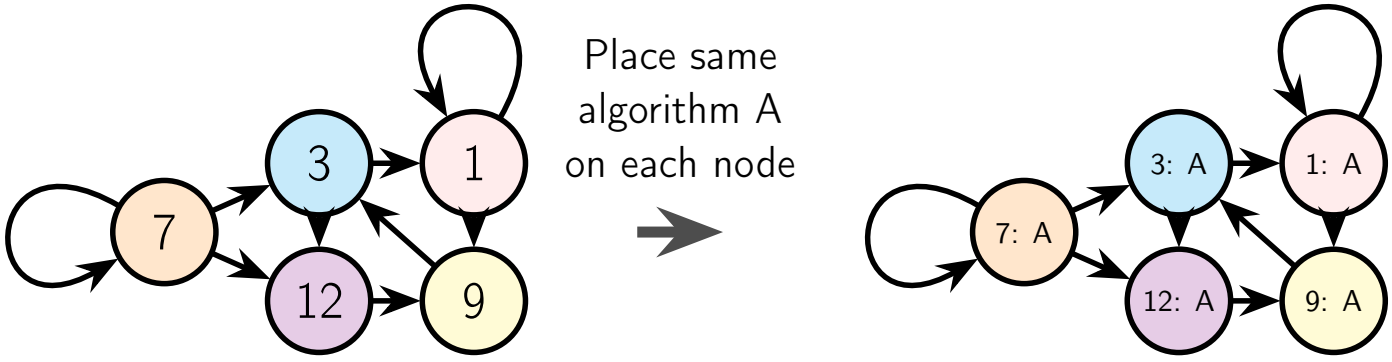
In restriction to monadic second-order logic, the following have the same expressive power:

- *floating-point GNNs*
- *GNNs with reals*
- *GMSC*
- ω*-GML*
- *bounded counting FDAs*
- *counting DAs*

Proof.

The proof is based on properties of monadic second-order logic on tree-shaped graphs, and so-called "tree automata". □

Distributed computing with identifiers



Input
Each node has a
unique identifier

Distributed System
Algorithm also uses
identifiers as input

Identifiers (or IDs) enable:

- Symmetry and similarity breaking (e.g. self-loops can be detected),
- Leader election, and
- addressing a message to specific nodes.

Bollig, Bouyer and Reiter (FoSSaCS 2019) studied distributed register machines with identifiers via a fragment of partial fixed-point logic.

Our contributions on distributed computing with identifiers

We characterize the expressive power of circuit-based distributed algorithms with identifiers via modal substitution calculus.

The obtained translations are highly efficient in relation to size (and computation time).

Graphs with identifiers

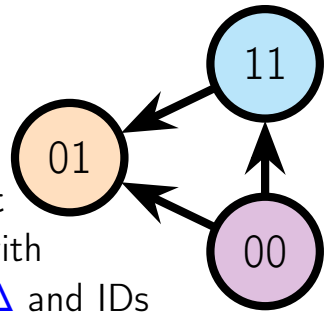
Formally, a graph with identifiers is a tuple

$$(V, E, p_1, \ldots, p_n, q_1, \ldots, q_m)$$

where

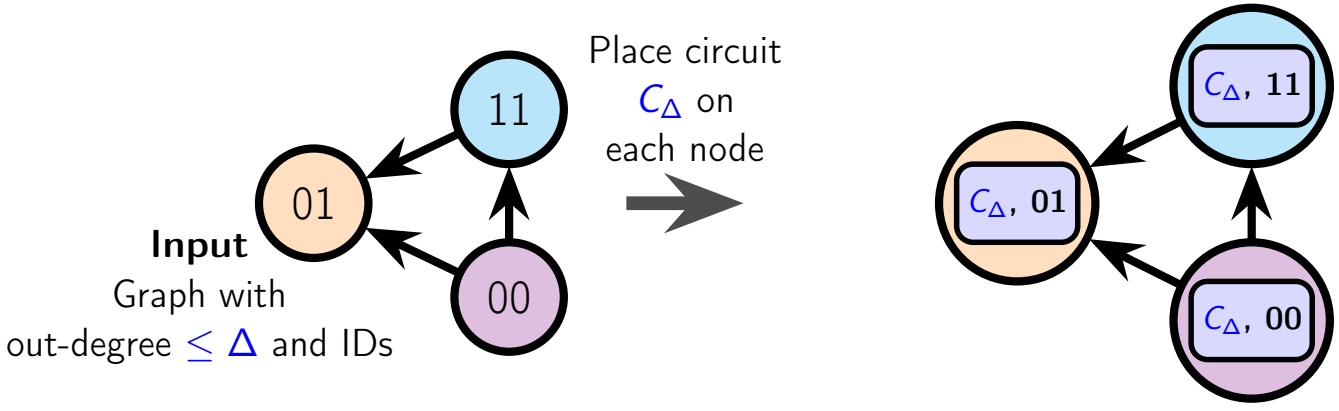
- V is a set of nodes,
- E is a set of edges,
- node labels (or proposition symbols) $p_1, \ldots, p_n$ encode unique identifiers (thus labels are ordered),
- node labels $q_1, \ldots, q_m$ are additional labels (act as ordinary proposition symbols in Kripke models).

Distributed computing with circuits and IDs

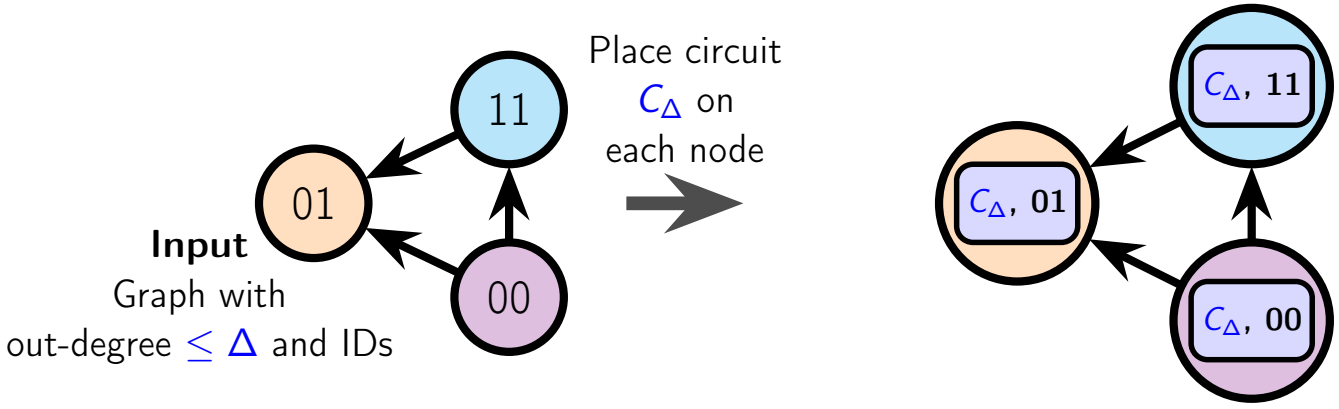


Input
Graph with
out-degree $\leq \Delta$ and IDs

Distributed computing with circuits and IDs



Distributed computing with circuits and IDs

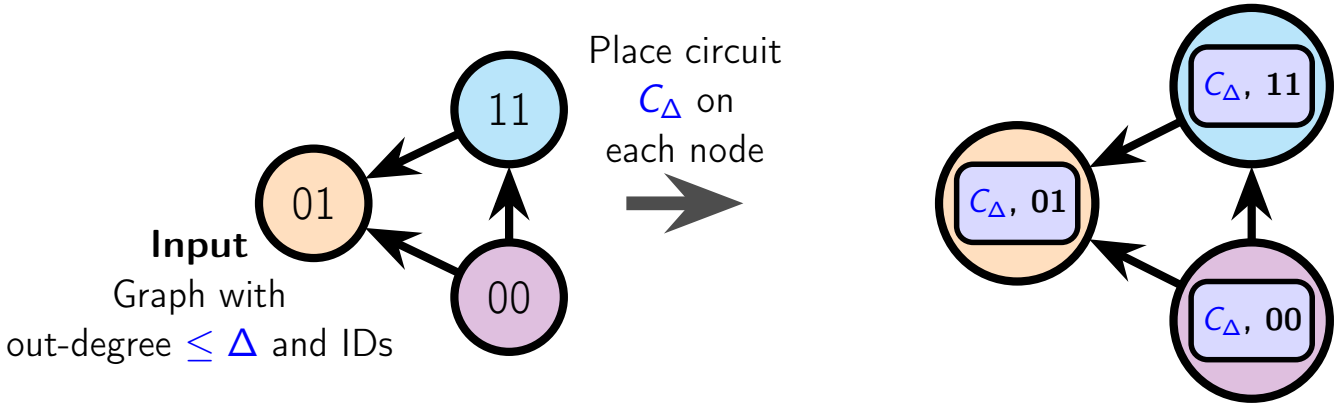


How each node updates its state (one round)

Circuit C_Δ computes: $f : \{0,1\}^{k+k\cdot\Delta} \to \{0,1\}^k$

k = state length (bits per node), Δ = max out-degree

Distributed computing with circuits and IDs



How each node updates its state (one round)

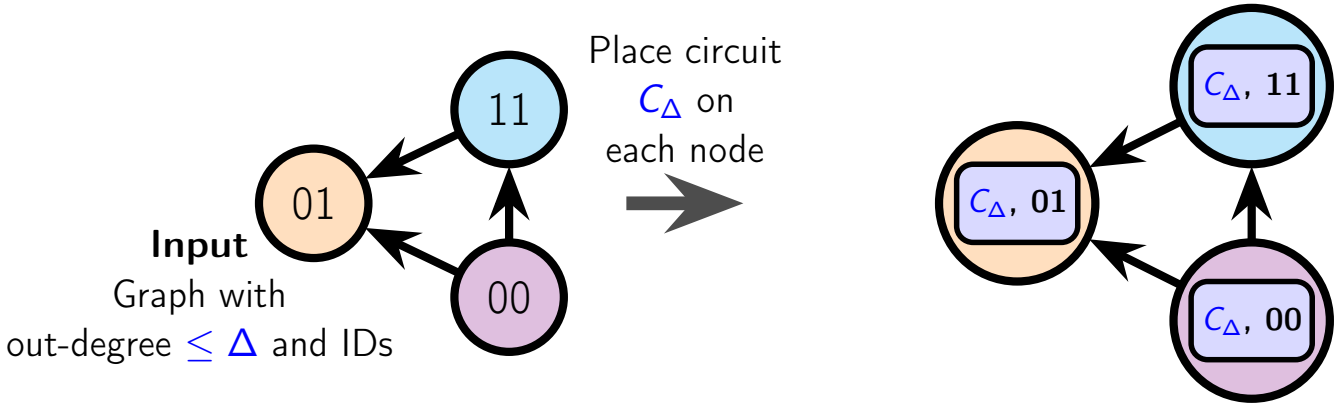
Circuit C_Δ computes: $f : \{0,1\}^{k+k\cdot\Delta} \to \{0,1\}^k$

k = state length (bits per node), Δ = max out-degree

Step 1: Gather

Own state:	0	1	1
From out-nbr 1:	0	0	0
From out-nbr 2:	0	1	0

Distributed computing with circuits and IDs



How each node updates its state (one round)

Circuit C_Δ computes: $f : \{0,1\}^{k+k\cdot\Delta} \to \{0,1\}^k$

k = state length (bits per node), Δ = max out-degree

Step 1: Gather

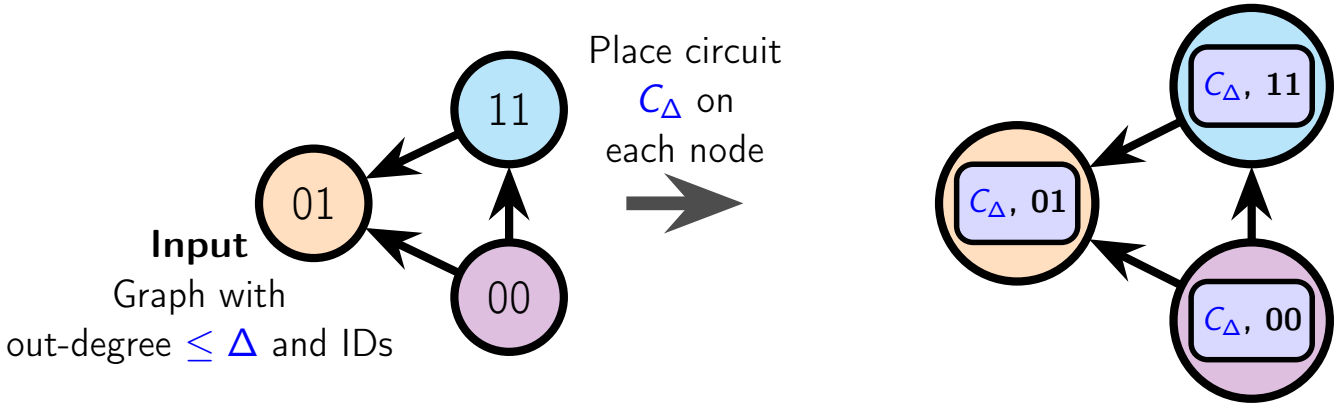
Own state:	0	1	1
From out-nbr 1:	0	0	0
From out-nbr 2:	0	1	0

Step 2: Concatenate

Input to circuit:

0	1	1	0	0	0	⋯

Distributed computing with circuits and IDs

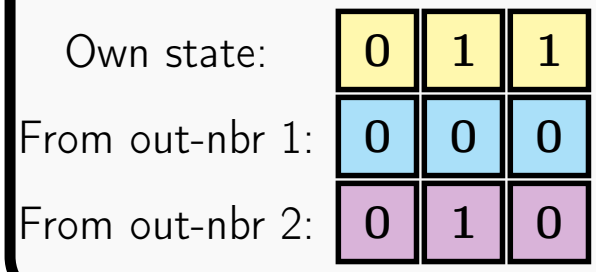


How each node updates its state (one round)

Circuit C_Δ computes: $f : \{0,1\}^{k+k\cdot\Delta} \to \{0,1\}^k$

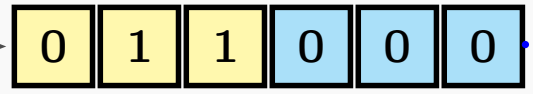
k = state length (bits per node), Δ = max out-degree

Step 1: Gather

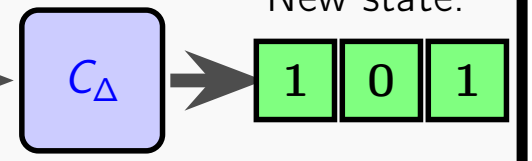


Step 2: Concatenate

Input to circuit:



Step 3: Compute



Results on distributed computing with identifiers

Theorem 8 (A., Heiman, Hella, Kuusisto 23)

For a given out-degree Δ and for a given program of MSC, we can construct an equivalent distributed circuit whose size is linear in the size of the program.

Theorem 9 (A., Heiman, Hella, Kuusisto 23)

Given a bounded fan-in distributed circuit, we can construct an equivalent program of MSC whose size is linear in the size of the circuit.

Final remarks

Some of our other results (informally).

Theorem 10 (A., Heiman, Kuusisto 23)

Given a floating-point neural network with a piecewise polynomial activation function, we can construct an equivalent diamond-free GMSC program, and vice versa. Size blow-ups in both directions are small.

Theorem 11 (A., Funk, Heiman, Kuusisto, Lutz 25)

Floating-point graph transformers with message-passing and GML with global counting modality have the same expressive power.

Theorem 12 (A., Funk, Heiman, Kuusisto, Lutz 25)

In restriction to first-order logic, real-based graph transformers with message-passing and GML with global non-counting modality have the same expressive power.

Thank you!