

Descriptive complexity for distributed computing and neural networks

Antti Kuusisto
Tampere University, Finland

Descriptive complexity 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.

Descriptive complexity 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**.

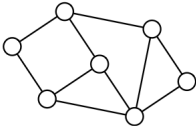
The logic **LocMMFP** extends first-order logic with a least fixed-point operator over unary monotone formulas. Here no syntactic restriction is imposed on the formulas to be iterated, so **LocMMFP** has undecidable syntax (which is also noted by the authors).

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:

problem instance



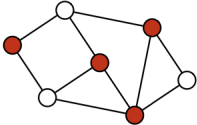
input string



solution

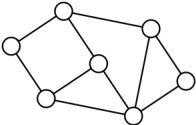


output string

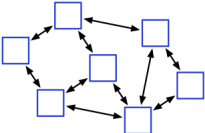


Distributed computing:

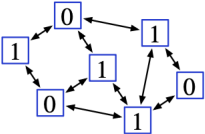
problem instance



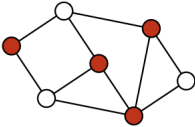
computer network



local outputs



solution



A distributed system is defined by a labelled directed graph

$$(W, R, p_1, \dots, p_k)$$

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 $w \in p_i$.

We could allow for more communication channels R_1, \dots, 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 the received messages and previous state.

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

1. Q is a finite set of states.
2. At each node, $\pi : \mathcal{P}(\{p_1, \dots, p_k\}) \rightarrow Q$ determines the **initial state** of the automaton based on the proposition symbols satisfied at the node.
3. $\delta : Q \times \mathcal{P}(Q) \rightarrow Q$ 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** $f_n : W \rightarrow Q$.

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

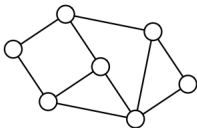
Define $Q' :=$ the set of states realized by the neighbours of w in round n . Then $f_{n+1}(w) :=$ the new state $\delta(f_n(w), Q')$ at w .

Strictly speaking, by “*neighbours of w* ” we here mean out-neighbours. 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.

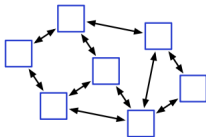
Node w accepts iff $f_k(w) \in F$ for some $k \in \mathbb{N}$.

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

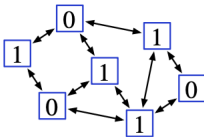
problem instance



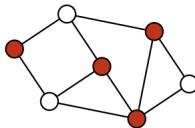
computer network



local outputs



solution



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)

$$\begin{array}{ll} X_1 & :- \quad p \vee \diamond q & X_1 & :- \quad p \vee X_n \vee \diamond X_1 \\ \vdots & & \vdots & \\ X_n & :- \quad \square(p \wedge q) & X_n & :- \quad X_1 \vee \diamond \neg X_n \end{array}$$

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

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

▶ $\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \diamond\varphi$

▶ $\varphi ::= p \mid X \mid \neg\varphi \mid \varphi \wedge \varphi \mid \diamond\varphi$

$$\begin{array}{ll}
 X_1 := \psi_1 & X_1 := \varphi_1 \\
 \vdots & \vdots \\
 X_n := \psi_n & X_n := \varphi_n
 \end{array}$$

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 $M = (W, R, p_1, \dots, p_k)$ 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 $\text{mult}(S)$ denote the set of multisets over S , and define $\Pi := \{p_1, \dots, p_k\}$. 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, \dots, p_k) as follows.

- ▶ The **initialization function** $\pi : \mathcal{P}(\Pi) \rightarrow \mathbb{R}^d$ 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 $t = 1, 2, \dots$, each node w computes a **new feature vector**

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

where

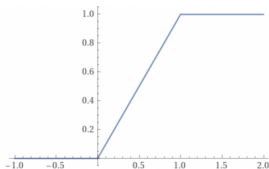
1. **AGG** : $\text{mult}(\mathbb{R}^d) \rightarrow \mathbb{R}^d$ is an **aggregation function** (typically sum),
2. **COM** : $\mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^d$ is a **combination function**.

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

$$\begin{aligned} \text{COM} \left(x_v^{t-1}, \text{AGG} \left(\{ \{ x_u^{t-1} \mid (v, u) \in R \} \} \right) \right) \\ = f \left(C x_v^{t-1} + \sum_{(v,u) \in R} A x_u^{t-1} + \mathbf{b} \right) \end{aligned}$$

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 $\text{ReLU}^*(x) = \min(\max(0, x), 1)$:



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 is a common choice also based on numerical analysis as it gives small rounding errors.

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:

There exists an ℓ such if a float f occurs in a multiset S at least ℓ times, then, if we **alter the multiplicity** of f in S so that the multiplicity stays above ℓ , the sum over the new multiset gives the same value as the sum over the original multiset S .

Theorem. (Ahvonen, Heiman, K., Lutz; 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; 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 $\text{GNNs} \leq \text{GMSC}$. This can be done by translating MSO-formulas into GMSC in a way resembling the construction of bottom-up 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.”

Game-theoretic semantics for GMSC

A **semantic game** to check if a program Π is true in (M, w) is played between **Eloise** and **Abelard**. Intuitively, Eloise is trying to prove the formula true and Abelard opposes this.

The game begins by Eloise choosing a number $k \in \mathbb{N}$. This determines the **beginning position** $(\text{Eloise}, \varphi, w, k)$, where φ is the body of the induction rule of X_1 . (If $k = 0$, φ is the body of the base rule of X_1).

After that, positions in the game are of type $(\mathbf{V}, \psi, u, \ell)$ where

- ▶ $\mathbf{V} \in \{\text{Eloise}, \text{Abelard}\}$ is the current **verifier** in the game. The other player is the **falsifier**.
- ▶ ψ is a subformula of the body of some rule in the program.
- ▶ u is a node.
- ▶ $\ell \in \{0, \dots, k\}$.

Intuitively, a position $(\mathbf{V}, \psi, u, \ell)$ corresponds to the following claim:

“The player $\mathbf{V} \in \{\text{Eloise}, \text{Abelard}\}$ can verify ψ at u in ℓ cycles of recursion.”

1. In a position (\mathbf{V}, p, u, ℓ) , the game ends. The verifier $\mathbf{V} \in \{\text{Eloise}, \text{Abelard}\}$ wins if $u \in p$. Otherwise the falsifier wins.
2. In $(\mathbf{V}, \neg\psi, u, \ell)$, the game continues from $(\mathbf{V}', \psi, u, \ell)$, where $\mathbf{V}' \neq \mathbf{V}$, so the verifier takes the role of the falsifier.
3. In $(\mathbf{V}, \psi \wedge \theta, u, \ell)$, the falsifier chooses a conjunct $\chi \in \{\psi, \theta\}$ and the game continues from the position $(\mathbf{V}, \chi, u, \ell)$.

4. In $(\mathbf{V}, \diamond^{\geq k}\psi, u, \ell)$, the verifier chooses a set $\{u_1, \dots, u_k\}$ of k distinct out-neighbours of u . Then the falsifier chooses some node $v \in \{u_1, \dots, u_k\}$ and the game continues from $(\mathbf{V}, \psi, v, \ell)$. If the verifier cannot choose k out-neighbours, the falsifier wins.
5. In (\mathbf{V}, X, u, ℓ) with $\ell > 0$, the game continues from the position $(\mathbf{V}, \psi, u, \ell - 1)$, where ψ is the body of the induction rule of X . If $\ell = 1$, then the game continues from $(\mathbf{V}, \theta, u, 0)$, where θ is the body of the base rule of X .

A note on “ordinary” neural networks (CSL24)

Theorem. *Given a neural network NN with piecewise polynomial activation, we can construct an equivalent Diamond-free GMSC-program Π such that the size of Π is*

$$\mathcal{O}\left(N(\Delta + P\Omega^2)(r^4 + r^3B^2 + rB^4)\right)$$

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
- ▶ r and B are the **floating point precision and base**:

$$\pm 0 . d_1 \cdots d_p \times \mathbf{B}^{\pm e_1 \cdots e_q}$$

where $r = \max(p, q)$.

The **computation delay factor** of the simulating diamond-free GMSC-program is $\mathcal{O}\left((\log(\Omega) + 1)(\log(r) + \log(B)) + \log(\Delta)\right)$.

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. NNs can be translated—with a polynomial blow-up in size—to equivalent NNs with ReLU as the activation function.

Thanks.

antti.kuusisto@tuni.fi