Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Halting Recurrent GNNs and the Graded μ-Calculus

Jeroen Bollen[1] , **Jan Van den Bussche**[1] , **Stijn Vansummeren**[1] , **Jonni Virtema**[2]
[1]Data Science Institute, Hasselt University, Belgium
[2]School of Computer Science, University of Sheffield, UK.
{first, second}@uhasselt.be, j.t.virtema@sheffield.ac.uk

Abstract

Graph Neural Networks (GNNs) are machine-learning models that operate on graph-structured data. Their expressive power is intimately related to logics that are invariant under graded bisimilarity. Current proposals for recurrent GNNs either assume that the graph size is given to the model, or suffer from a lack of termination guarantees. Here, we propose a halting mechanism for recurrent GNNs. We prove that our model can express all node classifiers definable in graded modal μ-calculus, even for the standard GNN variant that is oblivious to the graph size. To prove our main result, we develop a new approximate semantics for graded μ-calculus, which we believe to be of independent interest. We leverage this new semantics into a new model-checking algorithm, called the counting algorithm, which is oblivious to the graph size. In a final step we show that the counting algorithm can be implemented on a halting-classifier recurrent GNN.

1 Introduction

Graph neural networks (GNNs) represent a popular class of machine-learning models on graphs (Hamilton 2020). A multitude of GNN variants have been proposed (Sato 2020; Wu et al. 2021; Zhang 2025), but in their basic form, a GNN updates a vector of numerical features in every node of a graph by combining the node's own feature vector with the sum of those of its neighbors. The combination is usually expressed by a feedforward neural network with ReLU activation functions (Goodfellow, Bengio, and Courville 2016). The parameters of this network are typically learned, but in this paper we are concerned with the intrinsic expressiveness of the GNN model, and not on how GNNs can be concretely obtained from training data.

The "message passing" (Gilmer, Schoenholz, and others 2017) between neighbors in the graph, just described, starts from an initial feature map and can go on for a fixed or variable number of rounds (referred to as *fixed-depth* or *recurrent* GNNs, respectively). Ultimately, of course, we want to do something with the feature vectors computed for the nodes. We focus on the task of *node classification* where in the end a boolean-valued classification function is applied to the feature vector of each node. In this way, GNNs express unary (i.e., node-selecting) queries on graphs. In graph learning, unary queries on graphs are known as node classifiers.

It is natural to ask about the power of GNNs in expressing node classifiers. Interestingly, this question can be approached through logic. Initial results focused on the question of *distinguishability:* given two finite pointed graphs (G, v) and (H, w), does there exist a GNN $\mathcal{N}$ such that $\mathcal{N}(G, v)$ is true but $\mathcal{N}(H, w)$ is false? Distinguishability by GNNs was found to be closely related to distinguishability by color refinement. Specifically, it is easy to see that when (G, v) and (H, w) are graded bisimilar (see, e.g., (Otto 2023) for a definition), which is equivalent to indistinguishability by color refinement, then (G, v) and (H, w) are indistinguishable by GNNs. It turns out that the converse implication holds as well (Grohe 2021).

For distinguishing finite graphs, it does not matter whether we work with fixed-depth or recurrent GNNs. This changes when considering *uniform expressiveness;* that is, the question which unary graph queries are expressible by GNNs? We saw above that all expressible graph queries are invariant under graded bisimilarity. An important related logic is *graded modal logic* GML (de Rijke 2000), for it can express all so-called *logical classifiers* (i.e., unary queries that are expressible in first-order logic) that are *invariant* under graded bisimulation (Otto 2023). Interestingly, it has been shown that every GML formula is expressible by a fixed-depth GNN (Barceló et al. 2020).

What about recurrent GNNs? A logical step is to gauge their expressiveness through the extension of GML with recursion, i.e., the *graded μ-calculus* μGML (Kupferman, Sattler, and Vardi 2002). This logic is not only fundamental in computer-aided verification, but also lies at the basis of expressive description logics. Our main result shows that every μGML formula is expressible by a recurrent GNN. Two important remarks are in order here. First, we work with the plain-vanilla variety of GNNs: the combination function is a feedforward network employing ReLU. Second, our recurrent GNNs are *halting*. By this we mean that the GNN includes a boolean-valued halting function, defined on feature vectors. The GNN we construct for any μGML formula φ comes with the following halting guarantee: on every finite graph, there exists an iteration where the halting function is true in every node. At that point, the classification function can be applied at every node and will be correct, i.e., agree with φ. This global halting condition will be reached in a number of iterations that is polynomial in the graph size.

Getting such a halting guarantee is quite challenging, because a GNN operating on a graph G cannot know the precise

number N of nodes of G. Therefore, we cannot simply iterate fixpoint formulas N times to obtain the correct result. Extensions of GNNs that know the graph size, e.g., by global readout layers (Barceló et al. 2020), or simply by setting the graph size at initialisation (Pflueger, Tena Cucala, and Kostylev 2024), have been considered. Our contribution is to show that global readouts are not necessary for expressing $\mu\mathsf{GML}$, if one adopts a global halting condition instead. This result is not only interesting from the perspective of fundamental understanding; simpler neural network architectures also tend to be easier to train, although experimental follow-up work is necessary to confirm this.

We prove our result in several steps. We consider, for any natural number k, the approximate semantics for $\mu\mathsf{GML}$ that iterates all fixpoints exactly k times. Of independent interest, we define a notion of when the k-approximation semantics is *stable* on a graph G, and show that when a k-approximation is stable, it coincides with the true semantics of $\mu\mathsf{GML}$. The challenge is to show that a recurrent GNN can express the k-approximation semantics for increasing values of k as well as track stability of the current approximation. We overcome this in two steps. We first define an algorithm, called the *counting algorithm*, that incrementally computes k-approximations and their stability. The correctness of the algorithm, which is oblivious to the graph size, is nontrivial. In a second, equally crucial step, we show how to implement the counting algorithm in a halting-classifier recurrent GNN.

This paper is organized as follows. Section 2 discusses related work. Section 3 provides preliminaries. Section 4 introduces halting-classifier recurrent GNNs. Section 5 presents the translation of $\mu\mathsf{GML}$ into halting-classifier recurrent GNNs. Section 6 offers concluding remarks.

Proofs are omitted due to space constraints, but may be found in the technical report (Bollen et al. 2025).

2 Related Work

In comparison with related work, the innovative aspect of our recurrent GNN model is the halting aspect. We are aware of two prior works relating recurrent GNNs to μ-calculus.

Pflueger et al. (2024) consider a very general, abstract recurrent GNN model, where the combination function can be an arbitrary function, not necessarily a feedforward network, and likewise an arbitrary aggregation function can be used instead of summing of neighbors. Such a general model is still invariant under graded bisimulation. Moreover, they do not guarantee termination. Instead, they classify a node n as true if, during the infinite sequence of iterated message passing, the classification function applied to n eventually stabilizes to true. There is no apparent way to test this effectively.

In the setting of Pflueger et al., *every* node classifier invariant under graded bisimilarity (in particular, every $\mu\mathsf{GML}$ formula) is expressible by a recurrent GNN. Indeed, the contribution of their work lies much more in the converse direction. Establishing a preservation theorem regarding "local" monadic fixpoint logic (LocMMFP) formulas invariant under graded bisimilarity, they obtain that every recurrent GNN that expresses a node classifier expressible in LocMMFP is actually expressible in $\mu\mathsf{GML}$.

Ahvonen et al. (2024a) investigate recurrent GNNs with yet another acceptance condition, which is neither a global halting condition (as ours) nor a local stabilization condition (as in Pflueger et al.). Their notion of GNN does not employ a classification function, but instead includes a set of accepting feature vectors. A node is classified as true if, during the iteration, it obtains such an accepting feature vector. Like acceptance by stabilization a la Pflueger et al., this semantics offers no practical termination guarantee; we can see when a node has "decided to be true", but there is no clear-cut way to know that this will eventually happen, and hence no way to know that a node will be classified as false.

The acceptance semantics of Ahvonen et al. is a double-edged sword. On the one hand, it allows the expression of node classifiers such as the centre point property (Kuusisto 2013) that are not expressible in $\mu\mathsf{GML}$. On the other hand, by the lack of a global halting condition, nested or alternating fixpoints cannot be expressed; the most we can get using accepting feature vectors are countably infinite disjunctions of GML formulas (denoted by ωGML). Using an instantiation of their model over the real numbers, with arbitrary aggregation and combination functions, Ahvonen et al. show that recurrent GNNs capture all of ωGML. They also propose a realistic instantiation where every GNN comes with its own finite floating-point domain, aggregation is sum, and combination functions are truncated-ReLU layers. These floating-point recurrent GNNs are shown to coincide with GMSC, a logic that iterates GML formulas. In a technical report (Ahvonen et al. 2024b), they show that this continues to hold when using acceptance by stabilization a la Pflueger et al. Interestingly, they show the real and the float instantiations to express exactly the same node classifiers in MSO.

In parallel with our own work, Rosenbluth and Grohe (2025) have studied the computational completeness of recurrent GNNs. In contrast to ours, their model provides the graph size as input to the GNN, and operates on graphs with bitstring features. Their focus then is on showing that their model is Turing-complete for feature transformers invariant under the colorings produced by color refinement.

3 Preliminaries

Notation. We denote by $\mathbb{N}$ and $\mathbb{R}$ the sets of natural numbers and real numbers, respectively. We denote by $\mathbb{B} = \{0, 1\}$ the set of booleans, where we identify false with 0 and true with 1. We denote the cardinality of a set X by $|X|$, the powerset of X by $\mathcal{P}(X)$ and by $\mathcal{M}(X)$ the set of finite multisets over X, i.e., the set of functions $M : X \to \mathbb{N}$ whose *support* $supp(M) := \{x \in X \mid M(x) > 0\}$ is finite. Intuitively, $M(x) = n$ means M contains n copies of x. We use doubly curly braces $\{\!\{\ldots\}\!\}$ to denote multiset comprehension.

Graphs. We work with finite, node-labeled, directed graphs. Given a set X, an *X-labeled graph* is a triple $G = (N, E, g)$ where N is a finite set of nodes; $E \subseteq N \times N$ is the edge relation, and $g\colon N \to X$ is the *node labeling function*. When $X = \mathbb{R}^d$ for some $d \in \mathbb{N}$, we call g a *feature map*. To ease notation, we write N_G for the node set of G; $n \in G$ to indicate that $n \in N_G$; $G(n)$ instead of $g(n)$ to denote the label of node n in G; and $E_G(n)$ for the set $\{m \in N \mid$

$(n, m) \in E\}$ of all out-neighbours of n in G. Our results easily extend to graphs that also admit labels on edges as well as nodes.

Label transformers and node classifiers. We write $\mathcal{G}[X]$ for the set of all X-labeled graphs. A *label transformer* is a function $f\colon \mathcal{G}[X] \to \mathcal{G}[Y]$ such that G and $f(G)$ have the same nodes and edges for all $G \in \mathcal{G}[X]$, but may differ in the node labels. If $h\colon X \to Y$ is a function, then we write $h{\uparrow}\colon \mathcal{G}[X] \to \mathcal{G}[Y]$ for the label transformer that replaces the label of each node n in the input graph G by $h(n)$. We say that $h{\uparrow}$ is obtained by *lifting* h. A *node classifier* (on X) is a label transformer $f\colon \mathcal{G}[X] \to \mathcal{G}[\mathbb{B}]$.

Graded modal μ-calculus. We are interested in comparing the expressive power of recurrent GNNs to that of the *graded modal μ-calculus* $\mu\mathsf{GML}$. To define $\mu\mathsf{GML}$, we assume given a finite set of *proposition symbols* $\mathbb{P} = \{p, q, \dots\}$ and a countable set $\mathbb{X} = \{X, Y, \dots\}$ of *variables*, disjoint with $\mathbb{P}$.

The formulae of $\mu\mathsf{GML}$ have the following syntax, where $p \in \mathbb{P}$, $X \in \mathbb{X}$, and $k \in \mathbb{N}$ with $k \geq 1$.

$$\varphi ::= p \mid \neg p \mid X \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Diamond_k \varphi \mid \Box_k \varphi \mid \mu X.\varphi \mid \nu X.\varphi.$$

Without loss of generality, we hence adopt formulae in negation normal form, where the negation operator $\neg$ can only be applied to proposition symbols. The modal operator $\Diamond_k\varphi$ expresses that there are at least k neighbours where φ holds while $\Box_k\varphi$ expresses that there are strictly less than k neighbours where φ does not hold. We also write $\Diamond\varphi$ and $\Box\varphi$ for $\Diamond_1\varphi$ and $\Box_1\varphi$. The fixed-point operators $\mu X.\varphi$ and $\nu X.\varphi$ are respectively used to define least and greatest fixed points, allowing for the expression of recursive properties.

Semantically, $\mu\mathsf{GML}$ formulae operate on $\mathcal{P}(\mathbb{P})$-labeled graphs. A *valuation* on such a graph G is a function V assigning a subset of vertices in G to each variable. The valuation $V[X \mapsto S]$ is defined as the function that is equal to V on all variables except X, which is mapped to S. Given a $\mathcal{P}(\mathbb{P})$-labeled graph G and a valuation V, a μ-calculus formula φ evaluates to a set $[\![\varphi]\!]^G_V$ of nodes in G, inductively defined as follows.

$$\begin{aligned}
[\![p]\!]^G_V &:= \{n \in G \mid p \in G(n)\} \\
[\![\neg p]\!]^G_V &:= \{n \in G \mid p \notin G(n)\} \\
[\![X]\!]^G_V &:= V(X) \\
[\![\varphi \wedge \psi]\!]^G_V &:= [\![\varphi]\!]^G_V \cap [\![\psi]\!]^G_V \\
[\![\varphi \vee \psi]\!]^G_V &:= [\![\varphi]\!]^G_V \cup [\![\psi]\!]^G_V \\
[\![\Diamond_k\varphi]\!]^G_V &:= \left\{n \in G \,\middle|\, |E_G(n) \cap [\![\varphi]\!]^G_V| \geq k\right\} \\
[\![\Box_k\varphi]\!]^G_V &:= \left\{n \in G \,\middle|\, |E_G(n) \setminus [\![\varphi]\!]^G_V| < k\right\} \\
[\![\mu X.\varphi]\!]^G_V &:= \bigcap \left\{S \subseteq N_G \,\middle|\, [\![\varphi]\!]^G_{V[X \mapsto S]} \subseteq S\right\} \\
[\![\nu X.\varphi]\!]^G_V &:= \bigcup \left\{S \subseteq N_G \,\middle|\, S \subseteq [\![\varphi]\!]^G_{V[X \mapsto S]}\right\}
\end{aligned}$$

The notions of free and bound variables are defined as usual: $\mu X.\varphi$ binds X in φ and similarly for $\nu X.\varphi$. We use *free* (φ), and *vars* (φ) to denote the sets of free and all variables occurring in φ, respectively. A formula is *well-named* if its free variables are distinct from its bound variables, and every variable is bound at most once in the formula. Throughout the paper we assume that all considered formulae are well-named. This is without loss of generality since ill-named formulae may always be made well-named by suitably renaming bound variables if necessary.

A *sentence* is a formula without free variables. Note that to evaluate a sentence, the valuation V is actually not needed, as the semantics depends only the valuation of the free variables. We can then write $[\![\varphi]\!]^G$ to denote $[\![\varphi]\!]^G_V$ for any valuation V.

A node classifier C on $\mathcal{P}(\mathbb{P})$-graphs is *definable* in μ-calculus if there exists a μ-calculus sentence φ such that $[\![\varphi]\!]^G = \{n \in G \mid C(G)(n) = 1\}$ for all $\mathcal{P}(\mathbb{P})$-graphs G. We also say that φ defines C in this case.

4 Halting Recurrent Graph Neural Networks

In this section we introduce our recurrent GNN model and show invariance under total surjective graded bisimulations.

An *aggregate-combine (AC) layer* (Barceló et al. 2020; Grohe 2021; Geerts, Steegmans, and Van den Bussche 2022) of input dimension p and output dimension q is a pair $L = (\text{AGG}, \text{COMB})$ of functions where $\text{AGG}\colon \mathcal{M}(\mathbb{R}^p) \to \mathbb{R}^h$ with $h \in \mathbb{N}$ is an *aggregation function* and $\text{COMB}\colon \mathbb{R}^p \times \mathbb{R}^h \to \mathbb{R}^q$ is a *combination function*. Semantically, such a layer is a label transformer: when executed on $\mathbb{R}^p$-labeled graph $G = (N, E, g)$, it returns the $\mathbb{R}^q$-labeled graph $G' = (N, E, g')$ with g' defined by

$$g'\colon n \mapsto \text{COMB}\big(g(n), \text{AGG}\,\{\!\{g(m) \mid m \in E_G(n)\}\!\}\big).$$

We abuse notation, and indicate by L both the AC layer and the label transformer that it defines.

Definition 4.1 (Recurrent GNN)**.** Given a finite set X, a *halting-classifier-based recurrent GNN over X of dimension* d is a tuple $\mathcal{N} = (\text{IN}, L, \text{HLT}, \text{OUT})$ where $\text{IN}\colon X \to \mathbb{R}^d$ is the *initialisation function*; L is an AC layer with input and output dimension d; $\text{HLT}\colon \mathbb{R}^d \to \mathbb{B}$ is the *halting function*; and $\text{OUT}\colon \mathbb{R}^d \to \mathbb{B}$ is the *readout function*.

For parsimony we simply say "recurrent GNN" instead of "halting-classifier-based recurrent GNN" in what follows.

We next define the semantics of recurrent GNNs. A *run* of $\mathcal{N}$ over X-labeled graph G is a finite sequence $H_0, H_1, \dots, H_k$ of $\mathbb{R}^d$-labeled graphs such that $H_0 = \text{IN}{\uparrow}(G)$ and $H_{i+1} = L(H_i)$ for every $1 \leq i < k$. A run is *complete* if every node in $\text{HLT}{\uparrow}(H_k)$ is labeled 1, and k is the first index for which this condition holds. The *output* of a complete run is the $\mathbb{B}$-labeled graph $\text{OUT}{\uparrow}(H_k)$.

Definition 4.2 (Halting recurrent GNN)**.** A recurrent GNN $\mathcal{N}$ over X is *halting* if there exists a complete run for every X-labeled input graph. If $\mathcal{N}$ is halting, then we write $\mathcal{N}(G)$ for the output of the complete run of $\mathcal{N}$ on G.

Halting recurrent GNNs hence define node classifiers whereas arbitrary recurrent GNNs may define only partial node classifiers, as some of their computations (i.e., runs) may never terminate. For completeness sake, we note that the problem of determining whether a given recurrent GNN is halting, is undecidable. However, the GNNs that we construct in this paper are always halting.

Proposition 4.3 (Undecidability of Halting)**.** *The problem of determining whether a given recurrent GNN is halting is undecidable.*

The proof is based on a reduction from the (undecidable) halting problem for 3-counter machines (Minsky 1967).

Simple recurrent GNNs. A simple but practically relevant choice for the aggregation and combination functions of an AC layer, which is commonly used in the literature (Barceló et al. 2020; Ahvonen et al. 2024a; Geerts, Steegmans, and Van den Bussche 2022), is to take

$$\textsc{Agg}\colon \mathcal{M}(\mathbb{R}^p) \to \mathbb{R}^p\colon M \mapsto \sum_{\mathbf{x}\in \mathit{supp}(M)} M(\mathbf{x})\cdot \mathbf{x}$$

$$\textsc{Comb}\colon \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}^q\colon (\mathbf{x},\mathbf{y}) \mapsto f(\mathbf{x} \mid \mathbf{y})$$

where $M(\mathbf{x})\cdot\mathbf{x}$ is the scalar multiplication of $\mathbf{x}$ by its multiplicity in M; $\mathbf{x} \mid \mathbf{y}$ denotes vector concatenation; and f is a *ReLU-based feedforward neural network* (RFNN). That is, $f\colon \mathbb{R}^{2p} \to \mathbb{R}^q$ is of the form

$$A_\ell \circ \mathrm{ReLU} \circ A_{\ell-1} \circ \cdots \circ \mathrm{ReLU} \circ A_1$$

where $A_i\colon \mathbb{R}^{p_i} \to \mathbb{R}^{p_{i+1}}, 1 \le i \le \ell$ are affine transformations with $p_1 = 2p$ and $p_{\ell+1} = q$, and ReLU is the Rectified Linear Unit, applying $\mathrm{ReLU}(x_i) = \max(0, x_i)$ to each vector element x_i of its input vector $\mathbf{x}$. If L is of this form then we call L *simple*.

We say that the halting function HLT and readout function OUT of a recurrent GNN are *simple* if there is some $1 \le i \le d$ such that for all $\mathbf{x} \in \mathbb{R}^d$, $\textsc{Hlt}(\mathbf{x}) = 1$ (resp. $\textsc{Out}(\mathbf{x}) = 1$) if, and only if, the i-th element of $\mathbf{x}$ is > 0.

Definition 4.4. A recurrent GNN is *simple* if L, HLT and OUT are all simple.

Invariance under graded bisimulation. As discussed in the Introduction and Section 2, a multitude of recurrent and non-recurrent GNN variants have been proposed in the literature. An important property in all of these variants, however, is that they are invariant under a notion of graded bisimulation. Note that our halting condition is global and this needs to be reflected in the notion of bisimulation. As a sanity check, therefore, we next establish invariance of our recurrent GNNs under total surjective graded bisimulations.

Definition 4.5 (Graded bisimulation)**.** Let G and H be X-labeled graphs over a set of labels X. A relation $Z \subseteq N_G \times N_H$ is a *graded bisimulation* (or *g-bisimulation*) between G and H if for every $(n, m) \in Z$ the following hold:

1. $G(n) = H(m)$,
2. there is a bijection $f\colon E_G(n) \to E_H(m)$ such that $(i, f(i)) \in Z$, for every $i \in E_G(n)$.

The g-bisimulation is *total* (*surjective*, resp.), if the domain (range, resp.) of Z is N_G (N_H, resp.).

Definition 4.6. A label transformer $f\colon \mathcal{G}[X] \to \mathcal{G}[Y]$ is *invariant under g-bisimulation* if for every pair G and H of X-labeled graphs and every graded bisimulation Z between G and H, it is the case that Z is also a graded bisimulation between $f(G)$ and $f(H)$. (Recall that G and $f(G)$ have the same set of nodes, and similarly for H and $f(H)$.)

Proposition 4.7. *Every node classifier definable by a halting recurrent GNN is invariant under total surjective g-bisimulations.*

Proof. Notice that every aggregate-combine (AC) layer is a label transformer that is g-bisimulation invariant. Likewise the lifted initialisation and lifted readout functions yield g-bisimulation invariant label transformers. Since the composition of g-bisimulation invariant label transformers is a g-bisimulation invariant label transformer, the result follows by observing that the lifted halting function can be seen as a g-bisimulation invariant label transformer as well. Totality and surjectivity of the g-bisimulation quarantees that the readout function is enacted to runs of the same length. □

5 From μ-Calculus to Halting-Classifier Recurrent GNNs

In this section we prove the central result of our paper.

Theorem 5.1. *Every node classifier defined by μGML sentence φ is also definable by a simple halting recurrent GNN.*

Our proof is constructive, but requires us to develop several new concepts and proceeds in multiple steps. First, in Section 5.1 we show how to view the computation of φ as a sequence of *approximations* $\varphi^{(1)}, \varphi^{(2)}, \varphi^{(3)}, \ldots$. We observe that this sequence reaches a fixpoint equaling φ as soon as we reach an approximation $\varphi^{(k)}$ that is *stable* (Def. 5.2), which is guaranteed to happen when k exceeds the graph size, but may also happen earlier (Proposition 5.3). In Section 5.2 we define an algorithm, called the *counting algorithm*, for computing the elements in the sequence of approximations and tracking their stability at the same time. The counting algorithm is expressed as a transition system on *configurations*. We then show in Section 5.3 that configurations can be encoded as labeled graphs, and give a simple recurrent GNN that simulates the counting algorithm's transition system. The GNN's halting classifier tests the stability of the current encoded configuration to decide if φ is fully computed.

We will require the following notation. For a μGML formula φ we write $\mathit{sub}(\varphi)$ for the set of direct subformulae of φ. For example, if $\varphi = \neg p \vee (X \wedge \Diamond q)$, then $\mathit{sub}(\varphi)$ consists of $\neg p$ and $X \wedge \Diamond q$. Note that $\mathit{sub}(p) = \mathit{sub}(\neg p) = \mathit{sub}(X) = \emptyset$. We write $\mathit{sub}^+(\varphi)$ for the set of all strict (not necessarily direct) subformulae of φ, computed recursively. In the example above, $\mathit{sub}^+(\varphi)$ consists of $\neg p$, X, q, $\Diamond q$ and $X \wedge \Diamond q$. We write $\mathit{sub}^*(\varphi)$ for $\mathit{sub}^+(\varphi) \cup \{\varphi\}$. We write $\mathit{sub}_\mu(\varphi)$ (resp. $\mathit{sub}_\nu(\varphi)$) for the subset of $\mathit{sub}(\varphi)$ consisting of those formulae that are of the form $\mu X.\psi$ (resp. $\nu X.\psi$), and let $\mathit{sub}_\pi(\varphi) = \mathit{sub}_\mu(\varphi) \cup \mathit{sub}_\nu(\varphi)$. The notations $\mathit{sub}^+_\mu(\varphi)$, $\mathit{sub}^*_\mu(\varphi)$ etc. are defined similarly as subsets of $\mathit{sub}^+(\varphi)$ and $\mathit{sub}^*(\varphi)$. We abuse notation, and use operators that are defined on formulae also on sets of formulae, unioning the pointwise results. For example, we write $\mathit{sub}^+_\pi(A)$ for $\bigcup_{\varphi\in A} \mathit{sub}^+_\pi(\varphi)$. We will use the notation $\pi X.\psi$ to refer to any fixpoint formula, least or greatest (i.e. $\pi \in \{\mu, \nu\}$).

5.1 Approximations and Stability

We define the syntax of *approximation-adorned* $\mu\mathsf{GML}$ (henceforth simply called adorned $\mu\mathsf{GML}$) to be equal to the syntax of $\mu\mathsf{GML}$, except that all fixpoint-operators are of the form $\mu^i X.\psi$ or $\nu^i X.\psi$, with $i \in \mathbb{N}$ and ψ itself adorned.

$$\varphi ::= p \mid \neg p \mid X \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Diamond_k \varphi \mid \Box_k \varphi \mid \mu^i X.\varphi \mid \nu^i X.\varphi.$$

The semantics of adorned formulae is defined similarly to that of normal formulae, except that

$$[\![\mu^i X.\varphi]\!]^G_V := \begin{cases} \emptyset & \text{if } i = 0, \\ [\![\varphi]\!]^G_{V[X \mapsto [\![\mu^{i-1} X.\varphi]\!]^G_V]} & \text{otherwise.} \end{cases}$$

$$[\![\nu^i X.\varphi]\!]^G_V := \begin{cases} N & \text{if } i = 0, \\ [\![\varphi]\!]^G_{V[X \mapsto [\![\nu^{i-1} X.\varphi]\!]^G_V]} & \text{otherwise.} \end{cases}$$

Intuitively, $\mu^i X.\varphi$ computes an under-approximation of $\mu X.\varphi$, obtained by iterating φ for i iterations, while $\nu^i X.\varphi$ similarly computes an over-approximation of $\nu X.\varphi$.

For a normal $\mu\mathsf{GML}$ formula φ and $i \in \mathbb{N}$, we denote by $\varphi^{(i)}$ the adorned $\mu\mathsf{GML}$ formula that is obtained by adorning every fixpoint operator of the form μX resp νX by $\mu^i X$ resp. $\nu^i X$. For instance:

$$\varphi = \mu Y.\left((p \vee \Diamond Y) \vee (\mu X.\left(q \wedge \Diamond\left(Y \vee \Diamond X\right)\right))\right)$$
$$\varphi^{(i)} = \mu^i Y.\left((p \vee \Diamond Y) \vee \left(\mu^i X.\left(q \wedge \Diamond\left(Y \vee \Diamond X\right)\right)\right)\right)$$

Intuitively, $\varphi^{(i)}$ approximates φ by iterating every fixpoint for i times. We also call $\varphi^{(i)}$ the *i-th uniform approximation* of φ. Note that, because φ may have nested and alternating fixpoints, $\varphi^{(i)}$ itself is not necessarily an under-approximation, nor an over-approximation of φ.

For a fixpoint formula $\varphi = \pi X.\psi$ and $i, k \in \mathbb{N}$ we write $\varphi^{(i,k)}$ for the adorned formula $\pi^i X.\psi^{(k)}$ that iterates the outermost fixpoint i times, and all inner fixpoints k times. For instance:

$$\varphi = \mu Y.\left((p \vee \Diamond Y) \vee (\mu X.\left(q \wedge \Diamond\left(Y \vee \Diamond X\right)\right))\right)$$
$$\varphi^{(i,k)} = \mu^i Y.\left((p \vee \Diamond Y) \vee \left(\mu^k X.\left(q \wedge \Diamond\left(Y \vee \Diamond X\right)\right)\right)\right)$$

Definition 5.2. Let $k \in \mathbb{N}, k \geq 1$. A $\mu\mathsf{GML}$ formula φ is *k-stable on input graph G, valuation V, and node $n \in G$* if

- φ is not a fixpoint formula (i.e., not of the form $\pi X.\psi$) and every direct subformula of φ is k-stable on (G, V, n). In particular p, $\neg p$, and X are always k-stable on (G, V, n) as they do not have direct subformulae.
- Or, φ is a fixpoint formula $\pi X.\psi$ and
  1. $n \in [\![\varphi^{(k,k)}]\!]^G_V$ iff $n \in [\![\varphi^{(k-1,k)}]\!]^G_V$; and
  2. for every $0 \leq i < k$, ψ is k-stable on (G, V_i, n) where $V_i = V[X \mapsto [\![\varphi^{(i,k)}]\!]^G_V]$.

Formula φ is *k-stable on* (G, V) if it is k-stable on (G, V, n) for every $n \in G$.

The following proposition shows that the sequence $\varphi^{(1)}, \varphi^{(2)}, \varphi^{(3)}, \ldots$ of uniform approximations of φ reaches a fixpoint that equals φ as soon as we reach an approximation $\varphi^{(k)}$ that is *k-stable*. This is guaranteed to happen when k exceeds the graph size, but may also happen earlier. Hence, for any $\mu\mathsf{GML}$ formula φ, graph G, and valuation V we may compute $[\![\varphi]\!]^G_V$ by means of the following simple algorithm:

(*) Compute the k-approximation $[\![\varphi^{(k)}]\!]^G_V$ for increasing values of k, and stop as soon as φ becomes *k-stable* on (G, V). Then return $[\![\varphi^{(k)}]\!]^G_V$.

Proposition 5.3. *For all $k \geq 1$, G and V it holds that:*

1. *if φ is k-stable on (G, V) then $[\![\varphi^{(k)}]\!]^G_V = [\![\varphi]\!]^G_V$; and*
2. *if $k > |N_G|$, then φ is k-stable on (G, V).*

We define the following notion related to k-stability.

Definition 5.4. Let $j, k \in \mathbb{N}$ with $k \geq 1$. A $\mu\mathsf{GML}$ fixpoint formula $\varphi = \pi X.\psi$ is *(j,k)-stable on input graph G, valuation V and node $n \in G$* if for every $0 \leq i < j$, ψ is k-stable on (G, V_i, n) where $V_i = V[X \mapsto [\![\varphi^{(i,k)}]\!]^G_V]$.

Note that if $j = 0$, then φ is vacuously $(0, k)$-stable on (G, V, n). Furthermore if φ is k-stable on (G, V, n), then it is also (k, k)-stable on (G, V, n) but the converse does not hold since item (1) of Definition 5.2 is not necessarily guaranteed. In general, (j, k)-stability on fixpoint formulae is hence a weaker notion than k-stability.

5.2 The Counting Algorithm

To prove Theorem 5.1 we next define an algorithm, called the *counting algorithm*, that implements (*). To later allow easy simulation by means of a recurrent GNNs, we define the counting algorithm as a transition system operating on *configurations*. We also take up the important task of proving correctness. How to simulate the counting algorithm by means of a recurrent GNN is shown in Subsection 5.3.

Throughout this section and the next, let φ be a fixed $\mu\mathsf{GML}$ sentence. Our convention that $\mu\mathsf{GML}$ formulae are well-named implies that for every variable $X \in \mathit{vars}(\varphi)$ there exists exactly one fixpoint formula in $\mathit{sub}^*_\pi(\varphi)$ that binds X. We denote this binding formula by $\varphi|_X$.

A *counter* on φ is a mapping $C \colon \mathit{sub}^*_\pi(\varphi) \to \mathbb{N}$. To ease notation, we write $C \leq k$ (resp. $C = k$) to indicate that $C(\alpha) \leq k$ (resp. $C(\alpha) = k$) for all $\alpha \in \mathit{sub}^*_\pi(\varphi)$.

Because the semantics of a formula only depends on its free variables, and because the free variables of φ and all of its subformulae are subsets of $\mathit{vars}(\varphi)$, we can treat valuations on a graph G as finite mappings $V \colon \mathit{vars}(\varphi) \to \mathcal{P}(N_G)$. We refer to such mappings as φ-valuations.

Definition 5.5. Let φ be a $\mu\mathsf{GML}$ sentence. A *configuration* of φ is a tuple $\kappa = (G, k, C, V, R, F, S, T)$ where G is a $\mathcal{P}(\mathbb{P})$-labeled graph; $k \in \mathbb{N}$ with $k \geq 1$ is called the *bound* of κ; C is a counter on φ with $C \leq k - 1$; V is a φ-valuation on G; $R \colon \mathit{sub}^*(\varphi) \to \mathcal{P}(N_G)$; $F \subseteq \mathit{sub}^*(\varphi)$; $S \colon \mathit{sub}^*(\varphi) \to \mathcal{P}(N_G)$; and $T \colon \mathit{sub}^*_\pi(\varphi) \to \mathcal{P}(N_G)$.

Intuitively, the bound k in a configuration will indicate the uniform approximation of $\varphi^{(k)}$ for which we are currently computing $[\![\varphi^{(k)}]\!]^G$ while counter C will record how far we are in this computation. Moreover, V will contain the valuation under which we are currently computing the results of subformulae, while R stores subresults required to continue

computation; F registers for which subformulae of φ the subresults stored in R can be considered valid; S is used to track, for each subformula, on which nodes the subformula is k-stable; and T is used to track, for each fixpoint subformula α, for which nodes n the subformula is $(C(\alpha), k)$ stable on (G, V, n).

Our algorithm does not work on arbitrary configurations, but on configurations for which our intuitive description is coherent. We formalize this as follows. To simplify notation, for a counter C we write $\alpha^{(C,k)}$ for $\alpha^{(C(\alpha),k)}$ and we say that $\alpha \in \mathit{sub}^*_\pi(\varphi)$ is (C, k)-stable on (G, V, n) if α is $(C(\alpha), k)$-stable on (G, V, n).

Definition 5.6. A configuration κ is *coherent* if the following three conditions hold.

1. It is *sound*: $V(X) = [\![\varphi|_X{}^{(C,k)}]\!]^G_V$ for all $X \in \mathit{vars}(\varphi)$.
2. It is *consistent*: for all $\alpha \in F$,
   (a) $R(\alpha) = [\![\alpha^{(k)}]\!]^G_V$;
   (b) $\mathit{sub}(\alpha) \subseteq F$;
   (c) if α is a fixpoint formula, then $C(\alpha) = k - 1$. And
3. It *tracks stability*:
   (a) $S(\alpha) = \{n \in G \mid \alpha \text{ is } k\text{-stable on } (G, V, n)\}$ for every $\alpha \in F$; and
   (b) $T(\alpha) = \{n \in G \mid \alpha \text{ is } (C, k)\text{-stable on } (G, V, n)\}$ for every $\alpha \in \mathit{sub}^*_\pi(\varphi)$.

A configuration κ is *complete* if $\varphi \in F$. It is *stable* if $S(\varphi) = N_G$ with G the graph of κ.

Note that from a coherent and complete configuration we can obtain $[\![\varphi^{(k)}]\!]^G = [\![\varphi^{(k)}]\!]^G_V$ by simply reading $R(\varphi)$. Likewise, k-stability of φ on (G, V) can be obtained by checking that κ is stable.

The goal of the counting algorithm is to compute a coherent and complete configuration for φ on G. Computation starts at the initial configuration w.r.t $k = 1$, defined below.

Definition 5.7. The *initial configuration* of φ on graph G w.r.t. $k \geq 1$ is $\kappa := (G, k, C, V, R, F, S, T)$ where $C = 0$;

$$V(X) = \begin{cases} \emptyset & \text{if } \varphi|_X \in \mathit{sub}^*_\mu(\varphi) \\ N & \text{if } \varphi|_X \in \mathit{sub}^*_\nu(\varphi); \end{cases}$$

R and S map every formula to $\emptyset$; $F = \emptyset$; and T maps every fixpoint formula to $\emptyset$.

The initial configuration is trivially coherent. We complete our definition of the counting algorithm by defining three types of transitions on configurations.

Definition 5.8. Let $\kappa = (G, k, C, V, R, F, S, T)$ be a configuration. A *type-1* transition on κ yields the configuration $\kappa' = (G, k, C, V, R', F', S', T)$ where

$$\begin{aligned} R'(p) &:= \{n \in G \mid p \in G(n)\} \\ R'(\neg p) &:= \{n \in G \mid p \notin G(n)\} \\ R'(X) &:= V(X) \\ R'(\psi \wedge \psi') &:= R(\psi) \cap R(\psi') \\ R'(\psi \vee \psi') &:= R(\psi) \cup R(\psi') \\ R'(\Diamond_\ell \psi) &:= \{n \in G \mid |E_G(n) \cap R(\psi)| \geq \ell\} \\ R'(\Box_\ell \psi) &:= \{n \in G \mid |E_G(n) \setminus R(\psi)| < \ell\} \\ R'(\pi X.\psi) &:= R(\psi) \\ F' &:= \{\alpha \in \mathit{sub}^*(\varphi) \mid \mathit{sub}(\alpha) \subseteq F\} \\ &\quad \setminus \{\alpha \in \mathit{sub}^*_\pi(\varphi) \mid C(\alpha) < k - 1\} \\ S'(\alpha) &:= N_G \cap \bigcap_{\beta \in \mathit{sub}(\alpha)} S(\beta) \qquad \text{if } \alpha \notin \mathit{sub}^*_\pi(\varphi) \\ S'(\pi X.\psi) &:= S(\psi) \cap T(\pi X.\psi) \\ &\quad \cap \{n \in G \mid n \in V(X) \text{ iff } n \in R'(\psi)\} \end{aligned}$$

We denote this by $\kappa \vdash_1 \kappa'$.

Intuitively, if κ is coherent then for any formula $\beta \in F$, R already stores the value of $[\![\beta^{(k)}]\!]^G_V$. Hence, for any α with $\mathit{sub}(\alpha) \subseteq F$ we may read these values to compute $[\![\alpha^{(k)}]\!]^G_V$, cf. the definition of R'. For fixpoint formulae $\alpha = \pi X.\psi$, this is only correct if $C(\alpha) = k - 1$, since we have then already evaluated the body ψ under valuation with $V(X) = [\![\pi^{k-1} X.\psi]\!]^G_V$, and hence

$$\begin{aligned} [\![\alpha^{(k)}]\!]^G_V = [\![\pi^k X.\psi^{(k)}]\!]^G_V &= [\![\psi^{(k)}]\!]^G_{V[X \mapsto [\![\alpha^{(k-1,k)}]\!]^G_V]} \\ &= [\![\psi^{(k)}]\!]^G_V = R(\psi) \end{aligned}$$

Here, the second-to-last equality is by soundness of κ. This reasoning is not correct when $C(\alpha) < k - 1$, which is why all fixpoint formulae with $C(\alpha) < k - 1$ are excluded from F'. Note that $F \subseteq F'$ by consistency of κ.

The construction of S' follows the definition of k-stability. Assume that $\alpha \in F'$. If α is a non-fixpoint formula then it is k-stable on (G, V, n) when every subformula is k-stable on (G, V, n). If α is a fixpoint formula $\alpha = \pi X.\psi$, it is is k-stable on (G, V, n) if ψ is k-stable on (G, V, n); α is itself (C, k) stable on (G, V, n), and $n \in [\![\alpha^{(k)}]\!]^G_V = R'(\alpha)$ iff $n \in [\![\alpha^{(k-1,k)}]\!]^G_V = V(X)$.

Based on this reasoning, we can formally prove:

Lemma 5.9. *If $\kappa \vdash_1 \kappa'$ and κ is coherent then so is κ'.*

To define the second type of transition, we require the notion of a ticking fixpoint formula. We say that $\alpha \in \mathit{sub}^*_\pi(\varphi)$ *ticks in configuration* κ if (1) $\mathit{sub}(\alpha) \subseteq F$; (2) $C(\alpha) < k-1$; and (3) $C(\beta) = k - 1$ for every $\beta \in \mathit{sub}^+_\pi(\alpha)$. We write $\mathit{ticks}(\kappa)$ for the subset of $\mathit{sub}^*_\pi(\varphi)$ that tick in κ. Note that if α ticks, none of its strict fixpoint subformulae can tick. We observe:

Lemma 5.10. *If $\alpha = \pi X.\psi$ ticks in coherent configuration κ then $[\![\alpha^{(C(\alpha)+1,k)}]\!]^G_V = R(\psi)$.*

This lemma shows that if $\alpha = \pi X.\psi$ ticks in a coherent κ then in $R(\psi)$ we have already computed $C(\alpha) + 1$ iterations of α's outermost fixpoint. However, since $C(\alpha) + 1 < k$, we have not yet computed all necessary k fixpoint iterations of $\alpha^{(k)} = \alpha^{(k,k)}$. To ensure that computation can continue, a type-2 transition therefore changes the configuration such that it causes $[\![\alpha^{(C(\alpha)+2,k)}]\!]^G_V$ to be computed in further transition steps. It does so by copying $R(\psi) = [\![\alpha^{(C(\alpha)+1,k)}]\!]^G_V$ to $V(X)$, increasing $C(\alpha)$, and resetting the computation of all subformulae that depend on the value of X, which has now changed.

Formally, define $\mathit{reset}(\kappa)$ to be the smallest subset of $\mathit{vars}(\varphi)$ satisfying

$$\mathit{reset}(\kappa) = \{X \mid \varphi|_X \in \mathit{ticks}(\kappa)\} \cup \{Y \mid \mathit{free}(\varphi|_Y) \cap \mathit{reset}(\kappa) \neq \emptyset\}.$$

Define $\mathit{dep}(\kappa) := \{\varphi|_X \mid X \in \mathit{reset}(\kappa)\} \setminus \mathit{ticks}(\kappa)$. We say that the elements of $\mathit{dep}(\kappa)$ *depend* on a tick in κ.

Definition 5.11. Let $\kappa = (G, k, C, V, R, F, S, T)$ be a configuration. A *type-2* transition on κ yields the configuration $\kappa' = (G, k, C', V', R, F', S, T')$ where

$$C'(\alpha) := \begin{cases} C(\alpha)+1 & \text{if } \alpha \in \mathit{ticks}(\kappa) \\ 0 & \text{if } \alpha \in \mathit{dep}(\kappa) \\ C(\alpha) & \text{otherwise} \end{cases}$$

$$V'(X) := \begin{cases} R(\psi) & \text{if } \varphi|_X \in \mathit{ticks}(\kappa), \varphi|_X = \pi X.\psi \\ \emptyset & \text{if } \varphi|_X \in \mathit{dep}(\kappa) \cap \mathit{sub}^*_\mu(\varphi) \\ N_G & \text{if } \varphi|_X \in \mathit{dep}(\kappa) \cap \mathit{sub}^*_\nu(\varphi) \\ V(X) & \text{otherwise} \end{cases}$$

$$F' := \{\alpha \in F \mid \mathit{free}(\alpha) \cap \mathit{reset}(\kappa) = \emptyset\}$$

$$T'(\pi X.\psi) := \begin{cases} T(\pi X.\psi) \cap S(\psi) & \text{if } \pi X.\psi \in \mathit{ticks}(\kappa) \\ N_G & \text{if } \pi X.\psi \in \mathit{dep}(\kappa) \\ T(\pi X.\psi) & \text{otherwise} \end{cases}$$

We denote this by $\kappa \vdash_2 \kappa'$.

Note that if no fixpoint formula ticks in κ, then $\kappa' = \kappa$, i.e., the transition is a no-op.

Lemma 5.12. *If $\kappa \vdash_2 \kappa'$ and κ is coherent then so is κ'.*

The third kind of transition increases the bound k when κ is complete.

Definition 5.13. Let κ be a configuration with graph G and bound k. A *type-3* transition on κ yields the configuration κ' such that $\kappa = \kappa'$ if κ is not complete. Otherwise, κ' is the initial configuration of φ on G w.r.t. $k+1$. We write $\kappa \vdash_3 \kappa'$ to indicate that κ' is the type-3 transition of κ.

It is straightforward to show:

Lemma 5.14. *If $\kappa \vdash_3 \kappa'$ and κ is coherent, then so is κ'.*

Let us write $\vdash_{1,2}$ for the composition of $\vdash_1$ and $\vdash_2$, with $\vdash_2$ executing after $\vdash_1$. We define $\vdash_{3,1,2}$ similarly. We use $\vdash^*_{1,2}$ to denote the reflexive-transitive closure of $\vdash_{1,2}$ and similarly for $\vdash^*_{3,1,2}$.

Lemma's 5.9, 5.12 and 5.14 show preservation of coherence by $\vdash_{3,1,2}$. We can also show that the three transition types make *progress* when executed in the order $\vdash_{3,1,2}$: $\vdash_3$ transitions to the next bound value when the input is complete and is a no-op otherwise, while $\vdash_{1,2}$ change the configuration to become "strictly more complete", in the following sense.

Proposition 5.15. *Let κ be the initial configuration on φ for G w.r.t. bound k. There exists κ' that is coherent and complete such that $\kappa \vdash^*_{1,2} \kappa'$.*

Combined with Proposition 5.3, this yields correctness of the counting algorithm:

Proposition 5.16. *Let κ be the initial configuration κ on φ for G w.r.t. bound 1. There exists a configuration κ' that is coherent, complete, and stable such that $\kappa \vdash^*_{3,1,2} \kappa'$.*

Proof. Note that $\vdash_3$ is a no-op on configurations that are not complete. Since by Proposition 5.15 $\kappa \vdash^*_{1,2} \kappa''$ with κ'' coherent, complete and having the same bound as κ, there is also a sequence of transitions of the form $\kappa \vdash^*_{3,1,2} \kappa''$: before reaching completion we may vacuously introduce $\vdash_3$ before $\vdash_1$ since this is a no-op. Now two things may happen:

- κ'' is stable, in which case it suffices to take $\kappa' = \kappa''$;
- κ'' is not stable. By executing $\vdash_3$ on κ'' we then obtain the initial configuration κ_2 that has bound 2. By repeating our reasoning, but now starting from κ_2 we know from Proposition 5.3 that will eventually get the desired κ'. □

5.3 Implementing the Counting Algorithm in a Halting-Classifier Recurrent GNN

We next show how to encode configurations as labeled graphs and prove that that there exists a simple recurrent GNNs that simulates $\vdash^*_{3,2,1}$ on such encodings.

To define the encoding of a configuration κ, we first define *local versions* of configurations. Intuitively, if G is the graph of κ then the local configuration of κ at $n \in G$ will contain the information specific to n stored in κ, as well as the information that is common to all nodes, such as k and F. Formally, for a function $M\colon A \to \mathcal{P}(N_G)$ from some domain A to sets of nodes, we define the *local version of M at node n* to be the function $m\colon A \to \mathbb{B}$ such that, for all $a \in A$, $m(a) = 1$ iff $n \in M(a)$.

Definition 5.17. Let $\kappa = (G, k, C, V, R, F, S, T)$ be a configuration and let n be a node in G. The *local version* of (κ, n) is the tuple $(G(n), k, C, v, r, F, s, t)$ where

- $G(n) \subseteq \mathbb{P}$ is the label of n in G;
- $k \in \mathbb{N}$ is the bound of κ;
- F is the validity set of κ; and
- v, r, s, and t are the local versions at n of V, R, S, and T, respectively.

The *encoding of* κ is the graph H that has the same nodes and edges as G, such that $H(n)$ is the local version of (κ, n), for every node $n \in G$.

It is clear that when A is a finite set, we may treat functions $A \to \mathbb{B}$ and $A \to \mathbb{N}$ as boolean resp. natural number vectors. Moreover, we may also treat subsets of a finite universe A as boolean vectors, since such subsets are isomorphic to characteristic functions $A \to \mathbb{B}$. Since all components of local configurations are of this form, it follows that we may treat local configurations as vectors over $\mathbb{N} \cup \mathbb{B}$, and hence also as vectors in $\mathbb{R}^d$ for some large enough value d. For example, for every $p \in \mathbb{P}$ this vector has a component that is 1 if $p \in G(n)$ and is 0 otherwise. In what follows, we hence treat local configurations as vectors in $\mathbb{R}^d$, with the understanding that its components carry a value from either $\mathbb{B}$ or $\mathbb{N}$. To facilitate notation, we will refer to the components of local configuration vector $\mathbf{x}$ using "field access" notation: e.g., $\mathbf{x}[v(X)]$ is the boolean element of $\mathbf{x}$ that stores $v(X)$. Specifically, $\mathbf{x}[\alpha \in F]$ is the boolean element of $\mathbf{x}$ that is 1 iff $\alpha \in F$, and we similarly write $\mathbf{x}[p \in G(n)]$.

In the above sense, the encoding H of G is a $\mathbb{R}^d$-labeled graph.

Let f be a function mapping configurations to configurations. A label transformer $g\colon \mathcal{G}[\mathbb{R}^d] \to \mathcal{G}[\mathbb{R}^d]$ *simulates* such a function f if for all configurations κ, the equality $g(\mathit{enc}(\kappa)) = \mathit{enc}(f(\kappa))$ holds, where $\mathit{enc}(\kappa)$ denotes the encoding of κ as a labeled graph.

It is relatively straightforward to show that there is an AC layer L_i simulating $\vdash_i$, for every $1 \leq i \leq 3$. Hence, their sequential composition $L_3; L_2; L_1$ simulates $\vdash_{3,2,1}$. It immediately follows that there exists a "multi-layer" recurrent GNN that iterates the composition $L_3; L_2; L_1$ to simulate μ-calculus formulae. This is not sufficient to prove Theorem 5.1, however, since (i) we have defined recurrent GNNs to iterate only a single AC layer and (ii) this layer must be simple for Theorem 5.1 to hold.

Unfortunately, we cannot simulate $\vdash_2$ and $\vdash_3$ by means of a simple AC layer: these transitions may cause the counter $C(X)$ of a variable X to be reset to zero if a certain boolean condition b holds. To express this by means of a RFNN in the COMB function of a GNN, we must essentially determine the new value of $C(X)$ by a function of the form "if $b = 1$ then $C(X)$ else 0". This function is non-continuous around $b = 1$, and therefore not expressible by a RFNN, which always expresses a continuous and piecewise-linear transformation.

We hence need to work harder to obtain Theorem 5.1. Our approach is conceptually simple: while we cannot directly express "if $b = 1$ then $C(X)$ else 0" in an AC layer, we may use the iteration capabilities of recurrent GNNs to repeatedly decrement $C(X)$ until it reaches zero. We have to take care, of course, that while we are doing this the state of the other configuration components is not incorrectly altered.

Formally, an *extended configuration* is a pair (κ, D) with κ a configuration, and $D \subseteq \mathit{vars}(\varphi)$. Intuitively, D will hold the variables whose counter value we need to keep decrementing. We also call D the *residual set*.

Given a configuration κ, we define the *partial* transition of type 2 and type 3, denoted $\vdash'_2$ and $\vdash'_3$ as follows. The partial type-2 transition on κ yields the extended configuration (κ', D') where (i) κ' is defined such as in Definition 5.11 except that $C'(X) := C(X)$ for all X with $\varphi|_X \in \mathit{dep}(\kappa)$, and (ii) $D' := \{X \mid \varphi|_X \in \mathit{dep}(\kappa)\}$. The partial type-3 transition on κ yields the extended configuration (κ', D') where (i) κ' is defined such as in Definition 5.13 except that $C'(X) := C(X)$ for all X, and (ii) $D' := \mathit{vars}(\varphi)$. These partial transitions hence delay setting variable counters to zero, but record in D' for which variables this must still happen. For notational convenience, define $\kappa \vdash'_1 (\kappa', \emptyset)$ whenever $\kappa \vdash_1 \kappa'$.

For every $1 \leq i \leq 3$ we then define the *extended type-i transition* on extended configurations, denoted $(\kappa, D) \rightsquigarrow_i (\kappa', D')$. Here, $(\kappa', D') = (\kappa, D)$ if $D \neq \emptyset$; otherwise, (κ', D') is the result of applying $\vdash'_i$ to κ.

Finally, we define a *reset transition* on extended configurations: on (κ, D) the resetting transition $\rightsquigarrow_r$ yields (κ', D') where κ' equals κ on all components except C, and

$$C'(X) := \begin{cases} C(X) - 1 & \text{if } X \in D \text{ and } C(X) > 0 \\ C(X) & \text{otherwise} \end{cases}$$

$$D' := \{X \in D \mid C(X) - 1 > 0\}$$

Note in particular that $(\kappa', D') = (\kappa, D)$ when $D = \emptyset$.

Proposition 5.18. *Let κ, κ' be configurations with κ' complete. Then $\kappa \vdash^*_{3,1,2} \kappa'$ if, and only if, $(\kappa, \emptyset) \rightsquigarrow^*_{3,1,2,r} (\kappa', \emptyset)$.*

Proof sketch. We only illustrate the $\Rightarrow$ direction, the converse direction proceeds similarly but additionally exploits the completeness of κ'. If κ is not complete, then $\vdash_3$ and $\rightsquigarrow_3$ are the identity on κ resp. $(\kappa, \emptyset)$. As such, it is not difficult to see that if κ is not complete we may mimic $\vdash_{3,1,2}$ by executing $\rightsquigarrow_{3,1,2}$ on $(\kappa, \emptyset)$ followed by zero or more executions of $\rightsquigarrow_r$ until the residual set becomes empty. Since each extended transition $\rightsquigarrow_i$ with $1 \leq i \leq 3$ acts as the identity on extended configurations for which the residual set is no-empty, we may also execute $\rightsquigarrow_r$ on extended configurations with non-empty residual set by means of $\rightsquigarrow_{3,1,2,r}$. Consequently, on incomplete configurations we may mimic $\vdash_{3,1,2}$ by executing $\rightsquigarrow^*_{3,1,2,r}$.

If κ is complete, then $\vdash_3$ yields a non-complete configuration, say κ''. Using analogous reasoning as before, we can argue that we may mimic $\vdash_3$ on κ by means of $\rightsquigarrow^*_{3,1,2,r}$. Because κ'' is not-complete, the subsequent execution of $\vdash_{1,2}$ on κ'' is equivalent to execution of $\vdash_{3,1,2}$ on κ''. The latter can be mimicked by means of $\rightsquigarrow^*_{3,1,2,r}$ by our reasoning in the previous case. Consequently, $\vdash_{3,1,2}$ is mimicked by means of two applications of $\rightsquigarrow^*_{3,1,2,r}$. □

We can encode extended configurations as labeled graphs similarly to how we encode configurations as labeled graphs: in the local configuration of (κ, D) at node n we now also include D at every node. The concept of a label transformer simulating a function on extended configurations is defined in the obvious way.

Proposition 5.19. *There exists a simple AC layer simulating $\rightsquigarrow_{3,1}$, as well as RFNNs whose lifting simulate $\rightsquigarrow_2$ and $\rightsquigarrow_r$.*

Proof sketch. The crux is that local versions of extended configurations are vectors whose elements are all in $\mathbb{B}$ or $\mathbb{N}$. It is well-known that, on such vectors, RFNNs can express all functions defined by boolean combinations of (i) the $\mathbb{B}$ input elements and (ii) comparisons on the $\mathbb{N}$ elements. For instance, if $a, b \in \mathbb{B}$ and $c \in \mathbb{N}$ then then function $\phi(a, b, c) = \neg(a \wedge \neg b) \vee (c > 1)$ is definable by an RFNN.

We will only need such functions to define output local configuration vectors of $\rightsquigarrow_{3,1}$, $\rightsquigarrow_2$, and $\rightsquigarrow_r$. For $\rightsquigarrow_{3,1}$ we also need the message passing capability of AC layers.

Let us illustrate how to simulate $\rightsquigarrow_{3,1}$, focusing on a single output element. Assume that $(\kappa, D) \rightsquigarrow_{3,1} (\kappa', D')$. Let H and H' be the encodings of (κ, D) and (κ', D'), respectively. Let $n \in N_H = N_{H'}$. Then $H(n)$ is the input local configuration vector at n, and $H'(n)$ the output vector. We illustrate only how to define the output element $H'(n)[r(\Diamond_\ell \psi)]$ with $\Diamond_\ell \psi$ a subformula of φ.[1] According to the definition of $\rightsquigarrow_{3,1}$:

- if $D' \neq \emptyset$ then $H'(n)[r(\Diamond_\ell \psi] = H(n)[r(\Diamond_\ell \psi]$;

[1] Recall that $H'(n)[r(\Diamond_\ell \psi)]$ stores whether $n \in R'(\Diamond_\ell \psi)$ with R' the result assignment of κ'.

- if $D' = \emptyset$ and κ is complete then $H'(n)[r(\Diamond_\ell \psi] = 0$ since $\vdash_3$ moves to the next initial k-configuraton;
- otherwise, $D' = \emptyset$ and κ is not complete, and $\rightsquigarrow_3$ is the identity on (κ, D) and hence (κ', D') is the result of applying $\rightsquigarrow_1$ on (κ, D). Therefore, according to Definition 5.8, in this case, $H'(n)[r(\Diamond_\ell \psi)] = 1$ if, and only if, $\left(\sum_{m \in E_H(n)} H(m)[r(\psi)]\right) \geq \ell$. Note that in an AC layer, $\sum_{m \in E_H(n)} H(m)[r(\psi)]$ is provided by the AGG function, which aggregates the local vectors of neighboring nodes, so this remains a comparison of an input feature vector element.

In each of these three cases, $H'(n)[r(\Diamond_\ell \psi]$ is hence determined by a boolean combination of input boolean elements and natural number comparisons. The three conditions themselves can also be expressed as boolean combinations: $D' = \emptyset$ is equivalent to $\bigvee_{X \in vars(\varphi)} H(n)[X \in D]$ while completeness of κ is equivalent to $H(n)[\varphi \in F]$. Therefore, the entire computation of $H'(n)[r(\Diamond_\ell \psi]$ is definable by a simple AC layer. □

Corollary 5.20. *There exists a simple AC layer simulating* $\rightsquigarrow_{3,1,2,r}$.

Proof. Observe that simple AC layers are closed under composition with lifted RFNNs: if $L \colon \mathcal{G}[\mathbb{R}^d] \to \mathcal{G}[\mathbb{R}^d]$ is a simple AC layer and $f \colon \mathbb{R}^d \to \mathbb{R}^d$ is a RFNN then $f{\uparrow} \circ L$ is also expressible by a simple AC layer, obtained by taking the RFNN $g \colon \mathbb{R}^d \to \mathbb{R}^d$ in L's COMB function, and replacing it by $f \circ g$. □

We now have all the ingredients to prove Theorem 5.1.

Proof of Theorem 5.1. Let φ be a $\mu\mathsf{GML}$ sentence and G the input to φ. Let d be the dimension necessary to encode local versions of extended configurations as vectors in $\mathbb{R}^d$.

Fix $\mathcal{N} = (\text{IN}, L, \text{HLT}, \text{OUT})$ be the recurrent GNN over $\mathcal{P}(\mathbb{P})$ of dimension d where

- IN$\colon \mathcal{P}(\mathbb{P}) \to \mathbb{R}^d$ maps each finite set P of proposition symbols to the initial local extended configuration of φ w.r.t. $k = 1$, which is $(P, 1, v, \emptyset, r, s, t, \emptyset)$ with v, s, t mapping all formulae to 0, and t mapping all formulae to 1. It is straighforward to see that IN$\uparrow$, when executed on G, returns the encoding of $(\kappa, \emptyset)$ with κ the initial configuration of φ on G w.r.t. $k = 1$.
- L is the simple AC layer simulating $\rightsquigarrow_{3,1,2,r}$, which exists by Corollary 5.20.
- HLT$\colon \mathbb{R}^d \to \mathbb{B}$ is the RFNN that, given the local version $\mathbf{x_n}$ of an extended configuration (κ, D) at node n returns 1 if and only if κ is complete (i.e., $\mathbf{x_n}[\varphi \in F] = 1$), φ is k-stable at (G, V, n), (i.e., $\mathbf{x_n}[s(\varphi)] = 1$), and D is empty (i.e., $\neg(\vee_{X \in vars(\varphi)} \mathbf{x_n}[X \in D]) = 1$). On a graph H that encodes an extended configuration (κ, D), HLT$\uparrow(H)$ has all nodes labeled 1 if, and only if, κ is complete and stable, and $D = \emptyset$.
- OUT$\colon \mathbb{R}^d \to \mathbb{B}$ is the function that on $\mathbf{x_n}$ which is the encoding of (κ, D) at node n, returns the element $\mathbf{x}[r(\varphi)]$ of $\mathbf{x}$ that stores whether $n \in R(\varphi)$.

$\mathcal{N}$ expresses the node classifier defined by φ by the combination of Proposition 5.3, 5.16, 5.18, and Corollary 5.20. Strictly speaking, $\mathcal{N}$ is not simple since HLT need not be simple. This is easily solved, however, by extending local configurations with an extra boolean element that is used to store the result of HLT, and modifying L so that it also computes HLT and stores it in this extra element. Then, HLT can just read this element instead. □

6 Concluding Remarks

We have shown that "bare bones" recurrent GNNs, without any features that allow to determine the graph size, can express μ-calculus, under a terminating semantics. Specifically, the GNN can be halted as soon as we see a stopping bit of one in every node. Morever, it is guaranteed that this will happen after a finite number of iterations (polynomial in the graph size). Proving this possibility result turned out to be surprisingly subtle and intricate.

An interesting question for further research is whether it is possible to make the termination fully convergent, as intended in the original recurrent GNN proposal by Scarselli et al. (2009). Specifically, is it possible to express every $\mu\mathsf{GML}$ formula by a recurrent GNN, with stopping bits and our halting guarantee, and moreover such that the entire feature vector of all nodes will stabilize?

It is also natural to wonder about the converse direction: do recurrent GNN classifiers always fall within $\mu\mathsf{GML}$? The unreserved statement certainly does not hold, even for fixed-depth GNNs (Barceló et al. 2020). As already mentioned, Pflueger et al. obtained a converse result relative to node classifiers expressible in "local" monadic fixpoint logic.

What if we relativize more generally to MSO (monadic second-order logic) node classifiers and strongly connected graphs? In restriction to strongly connected graphs, g-bisimilarity and total surjective g-bisimilarity coincide, and hence here recurrent GNNs are invariant under graded bisimulations (Section 4). Since unary MSO formulas invariant under graded bisimilarity are expressible in $\mu\mathsf{GML}$ (Walukiewicz 2002; Janin and Lenzi 2001), it would then immediately follow that recurrent GNN node classifiers in MSO fall within $\mu\mathsf{GML}$ (when restricted to strongly connected graphs), were it not for the caveat that the cited Janin–Walukiewicz theorem (as well as its graded version) assumes invariance of the MSO formula over all graphs, finite or infinite. In contrast, GNNs are designed to operate on finite graphs only.

There may be a way out of this conundrum, as a breakthrough, solving the open problem of proving the finitary version of the cited Janin–Walukiewicz theorem, has recently been announced (Colcombet, Doumane, and Kuperberg 2024). Another way to sidestep the situation is to agree on a reasonable definition of GNNs working on infinite graphs. This is an interesting line of research and promising work already exists for graphons (Böker, Levie, and others 2023). Of course, the requirement that the recurrent GNN node classifier is expressible in MSO becomes stronger and possibly unnatural in this manner, since MSO cannot express finiteness. The general case where graphs are not necessarily strongly connected remains wide open.

Acknowledgments

This work was supported by the Bijzonder Onderzoeksfonds (BOF) of Hasselt University Grant No. BOF20ZAP02; by the Research Foundation Flanders (FWO) under research project Grant No. G019222N; and by the Flanders AI (FAIR) research program.

References

Ahvonen, V.; Heiman, D.; Kuusisto, A.; and Lutz, C. 2024a. Logical characterizations of recurrent graph neural networks with reals and floats. In Globerson, A.; Mackey, L.; et al., eds., *Proceedings 38th Annual Conference on Neural Information Processing Systems*.

Ahvonen, V.; Heiman, D.; Kuusisto, A.; and Lutz, C. 2024b. Logical characterizations of recurrent graph neural networks with reals and floats. arXiv:2405.14606.

Barceló, P.; Kostylev, E.; Monet, M.; Pérez, J.; Reutter, J.; and Silva, J. 2020. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations*. OpenReview.net.

Böker, J.; Levie, R.; et al. 2023. Fine-grained expressivity of graph neural networks. In *Proceedings 37th Annual Conference on Neural Information Processing Systems*.

Bollen, J.; Van den Bussche, J.; Vansummeren, S.; and Virtema, J. 2025. Halting recurrent GNNs and the graded μ-calculus. Technical report. https://arxiv.org/abs/2505.11050.

Colcombet, T.; Doumane, A.; and Kuperberg, D. 2024. Tree algebras and bisimulation-invariant MSO on finite graphs. arXiv:2407.12677.

de Rijke, M. 2000. A note on graded modal logic. *Studia Logica* 64(2):271–283.

Geerts, F.; Steegmans, J.; and Van den Bussche, J. 2022. On the Expressive Power of Message-Passing Neural Networks as Global Feature Map Transformers. In Varzinczak, I., ed., *Foundations of Information and Knowledge Systems*, LNCS, 20–34. Cham: Springer.

Gilmer, J.; Schoenholz, S.; et al. 2017. Neural message passing for quantum chemistry. In Precup, D., and Teh, Y., eds., *Proceedings 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 1263–1272.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press.

Grohe, M. 2021. The logic of graph neural networks. In *Proceedings 36th Annual ACM/IEEE Symposium on Logic in Computer Science*, 1–17. IEEE.

Hamilton, W. 2020. *Graph Representation Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool.

Janin, D., and Lenzi, G. 2001. Relating levels of the mu-calculus hierarchy and levels of the monadic hierarchy. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, 347–356. IEEE Computer Society.

Kupferman, O.; Sattler, U.; and Vardi, M. 2002. The complexity of the graded μ-calculus. In Voronkov, A., ed., *18th International Conference on Automated Deduction*, volume 2392 of *Lecture Notes in Computer Science*, 423–437. Springer.

Kuusisto, A. 2013. Modal logic and distributed message passing automata. In Rochi Della Rocca, S., ed., *Computer Science Logic*, volume 23 of *Leibniz International Proceedings in Informatics*, 452–468. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Minsky, M. L. 1967. *Computation: Finite and Infinite Machines*. Englewood Cliffs, NJ: Prentice-Hall.

Otto, M. 2023. Graded modal logic and counting bisimulation. arXiv:1910.00039.

Pflueger, M.; Tena Cucala, D.; and Kostylev, E. 2024. Recurrent graph neural networks and their connections to bisimulation and logic. In Woolridge, M., et al., eds., *Proceedings 38th AAAI Conference*, 14608–14616.

Rosenbluth, E., and Grohe, M. 2025. Repetition makes perfect: Recurrens sum-GNNs match message passing limit. arXiv:2505.00291.

Sato, R. 2020. A survey on the expressive power of graph neural networks. arXiv:2003.04078.

Scarselli, F., et al. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20(1):61–80.

Walukiewicz, I. 2002. Monadic second-order logic on tree-like structures. *Theor. Comput. Sci.* 275(1-2):311–346.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32(1):4–24.

Zhang, B. 2025. The expressive power of graph neural networks: A survey. *IEEE Transactions on Knowledge and Data Engineering* 37:1455–1474.