Matias Selin

# DESCRIPTIVE COMPLEXITY AND GRAPH NEURAL NETWORKS

Faculty of Information Technology and Communication Sciences
Master's Thesis
December 2025

# Tiivistelmä

Matias Selin: Descriptive Complexity and Graph Neural Networks
Pro gradu -tutkielma
Tampereen yliopisto
Matematiikan maisteriohjelma
Joulukuu 2025

---


Koneoppimisen sisällä kehittynyt *syväoppiminen* on erityisesti viimeisen vuosikymmenen aikana noussut yhdeksi merkittävimmistä tämänhetkisistä tutkimusaiheista [68]. Suuri osa kehityksestä on kuitenkin perustunut kroonisen epäluotettavaan empiiriseen tutkimukseen samalla kun aiheen matemaattinen teoria on jäänyt pienemmälle huomiolle [40]. Erityistapauksessa, jossa käsiteltävänä on graafimuotoinen data, ovat *graafineuroverkot* [33, 31] saavuttaneet merkittävän suosion huolimatta siitä, että niiden teoreettiset perusominaisuudet ovat pitkään jääneet epämääräisiksi. Esimerkiksi näiden mallien *ilmaisuvoimaa*, eli niiden kykyä ilmaista eri funktioita syötetyllä datalla, on vasta hiljattain ymmärretty paremmin [11]. Yhdeksi lupaavaksi lähestymistavaksi tässä on osoittautunut *kuvaileva vaativuusteoria*, jossa tavoitteena on löytää näitä malleja karakterisoivia *logiikoita* [41].

Tässä tutkielmassa käsittelen kuvailevan vaativuusteorian soveltamista graafineuroverkkoihin. Luvun 1 johdannon jälkeen aloitan tutkielman varsinaisen sisällön luvussa 2 kattavalla kirjallisuuskatsauksella aiheen historiasta, minkä jälkeen kuvailen yksityiskohtaisesti luvussa 3 miten Ahvosen ja muiden [8] *rekurrenttien liukuluvuilla laskevien graafineuroverkkojen* karakterisaatio *laskevalla modaalisubstituutiokalkyylillä* toimii. Kuvailen myös, miten *globaalin lukemisen* lisääminen graafineuroverkkoihin voidaan simuloida logiikan puolella tätä vastaavalla *globaalilla laskevalla modaalitimantilla.*

Luvussa 4 tarkastelen erästä tunnettua graafineuroverkkojen erotteluvoiman rajaa eli *Weisfeilerin ja Lemanin algoritmia* [86, 61] sekä sitä, millä keinoin sen yli voidaan päästä. Eri vaihtoehdoista tehokkain on Saton ja muiden [77] kehittämä *noodirandomointi*, jossa – hieman yksinkertaistettuna – jokaiselle noodille asetetaan yksilöllinen tunniste. Osoitan uutena tuloksena, että Ahvosen ja muiden [8] karakterisaatiotulosta voidaan helposti laajentaa noodirandomoituun tapaukseen sallimalla logiikassa propositiosymbolien kvantifiointi. Tutkielman lopuksi tutkin tätä uutta logiikkaa tarkemmin muun muassa vertailemalla sitä erääseen sen muunnokseen, joka poikkeaa siitä syntaktisesti vain hieman mutta osoittautuu muilta ominaisuuksiltaan varsin erilaiseksi. Tämän luvun tulokset perustuvat yhteistyöhön Antti Kuusiston kanssa.



Avainsanat: syväoppiminen, graafineuroverkot, kuvaileva vaativuusteoria.

# Use of AI in thesis

According to my declaration, I have used the following artificial intelligence applications in my thesis during the research process:

- Free versions of ChatGPT November 2024 – November 2025 (GPT-4o and GPT-4.1).
- Free versions of Claude November 2024 – November 2025 (Claude 3.5 Sonnet, Claude 3.7 Sonnet, and Claude Sonnet 4).
- DeepL translation application November 2024 – November 2025.

Artificial intelligence has been used for the following purposes in all sections of the thesis:

- Grammar checking.
- Assistance with writing style, e.g., coming up with alternative expressions and rephrasing awkward sentences.
- Searching for sources for the literature review.
- Technical assistance in using LaTeX, especially when drawing figures for the thesis with the TikZ library.

I acknowledge that I am fully responsible for the entire content of my thesis, including the parts generated by AI, and accept accountability for any violations of ethical standards in publications.

# Contents

# 1 Introduction

Over the past decade, automation of human reasoning on data has been increasingly dominated by a subfield of machine learning known as *deep learning* [68]. In particular, state-of-the-art methods of processing data in the form of *graphs*—networks of interconnected nodes—revolve around the use of *graph neural networks* (GNNs) [33, 31]. However, theoretical work has lagged behind. Even the basic *expressive power* of many of these models—what kinds of properties of the graphs they can and cannot detect—has remained ambiguous. One framework that can bring clarity to this is *descriptive complexity theory*, where our goal is to find, for a given computational architecture, a *logic* into and from which it can be translated, thus precisely characterizing its expressive power [41]. This thesis introduces the reader to this intersection of computer science and mathematical logic at both intuitive and formal levels, and includes a small original contribution to this project.

Chapter 2 lays out the state of things with an overview of the history of deep learning, distributed computing and descriptive complexity theory, arguing along the way for why the current state of research is in need of more theoretical work. I describe past efforts of a small cluster of research groups working in the intersection of these fields and explain why the recently appeared case of graph neural networks is of particular interest.

The entirety of Chapter 3 stems from a 2024 paper by Ahvonen et al. [8] and begins the mathematical section of the thesis with definitions of graph neural networks, logics and other mathematical formalisms used throughout, including a description of why using actual floating-point systems instead of real numbers in these definitions matters. This is followed by a detailed proof of the characterization of *recurrent floating-point graph neural networks* (GNN[F]s) with a logic called *graded modal substitution calculus* (GMSC). I also demonstrate how a similar characterization can be obtained for GNNs with *global readout*, which allows them to aggregate information across the whole graph.

In Chapter 4, I briefly describe how ordinary graph neural networks are fundamentally limited by the discriminatory power of the *one-dimensional Weisfeiler–Leman graph isomorphism test* [86, 61] and summarize attempts to surpass this barrier. The most promising of these is *random node initialization* [77], of which I give a novel logical characterization in the style of Ahvonen et al. [8] with an extension of GMSC I call *second-order propositional GMSC* (SOPGMSC). The thesis then ends with some remarks related to this logic, as well as how slightly adjusting the placement of the quantifiers produces a variant of it that is surprisingly different. The research in this chapter is based on joint work with Antti Kuusisto.

# 2 The State of Research So Far

The aim of this chapter is to introduce the reader to the recent history of the topics at hand. We begin with a general argument for why the state of research in deep learning requires re-thinking the current balance between empirical and theoretical work. We then follow this up with more detail on the fields of particular interest in this thesis: descriptive complexity theory, distributed computing, and graph neural networks.

## 2.1 The Need for Theory

Traditionally, the development of new computational systems has followed a predictable pattern where mathematical theory and practical implementation work hand in hand. Examples abound. The first digital computers were preceded by developments in Boolean algebra and the Turing machine [79, 83]. Compiler design in the 1960s was fueled by advancements in automata theory and formal languages [3]. Even neural networks have a long history of theory reaching back to the 1940s and 1950s before their recent rise to fame [58, 73]. But this was not to last. The moment this order of things began to shift can perhaps be pinpointed as 2012, when a deep convolutional neural network called AlexNet blew out the competition in the ImageNet image classification challenge and showed that deep neural networks were the future [49]. Thirteen years later, *deep learning*—the use of multilayer networks of *neurons*—is ubiquitous in fields from computer vision and natural language processing to mathematical discovery itself [37].

But how has a field originally described as "esoteric" and "unpromising" [68, Preface] achieved such incredible success? Interestingly, the AlexNet paper contained a prescient remark: "All of our experiments suggest that our results can be improved simply by waiting for faster GPUs and bigger datasets to become available." [49, Chapter 1]. We have waited, and a vast increase in computational resources has indeed driven many of deep learning's most impressive achievements. Such results can be obtained with little gain in theoretical knowledge. *Understanding Deep Learning*, a recent textbook by Simon Prince, an influential figure in the field, concedes that "the study of deep learning is still driven by empirical demonstrations ... [which] are not yet matched by our understanding of deep learning mechanisms." [68, p. 419]. Prince even admits in the first paragraphs of his boldly titled book that "the title is also partly a joke—no-one really understands deep learning at the time of writing." [68, Preface]. Indeed, a large chunk of research in deep learning is based on largely trial-and-error based architectural tweaks, without a theoretical basis that informs why these tweaks work and in what context.

Take a relatively simple-sounding foundational research program: why do some deep learning models generalize well and others do not? In other words, which component of the model architecture or which training protocol decides whether the model fits to the signal or the noise in the training data? In 2017, a research group at Google Brain investigated this question and showed empirically that some deep neural networks could in fact do both: they memorized completely random data while still generalizing well on real tasks, which could not be explained by the well-established statistical learning theory of the time [89].[1] A related observation that also flies in the face of classical theory is the famous *double-descent* phenomenon, first noted in 2019 [14]. Here, as the amount of parameters or "complexity" of a model increases, the test error first falls as the model becomes capable of expressing the underlying structure of the data, then rises as the model overfits to noise, but then, paradoxically, falls again. Though some progress has been made in explaining these observations, formally, the generalization phenomenon remains mysterious, with the Google Brain group stating in a 2021 followup that "a full mathematical characterization of the whole story remains challenging" [90, Section 6.1]. Prince seems to agree, saying that "if the efficient fitting of neural networks is startling, their generalization to new data is dumbfounding" and concluding that "a priori, deep learning shouldn't work" [68, p. 403].

An empirically-minded computer scientist might now interject: "so what?" The results achieved are indeed impressive, and to be sure, the recent deep learning boom must be placed in a context of decades of theoretical work in machine learning and neural networks. But, as we argue, two major concerns are associated with how current research is largely focused on empirical outcomes: poor replicability of results and the growing need for explainable AI.

### 2.1.1 Replicability

The first concern arises when examining our epistemic justifications for believing that progress is actually being made. During the past decade, the *replication crisis* has become a well-recognized phenomenon, most famously in psychology, in which a large fraction of published experimental results turn out not to reproduce [85]. This casts doubt on the credibility of a vast amount of research, erodes public trust in science, and in the worst case, leads to a massive amount of resources wasted on research programs and clinical interventions based on faulty premises.

As an experimental science, machine learning is affected by the same systemic pressures, such as a strong incentive to publish positive results instead of failed replications, as well as a relative lack of openly-accessible and functional data and code. Deep learning methods are also uniquely sensitive to slight changes in experimental setup. As an example, take the field this thesis is mainly concerned with, *graph learning*. It has repeatedly been shown in the case of graph neural networks that older model architectures, which were

[1] More specifically, the authors trained state-of-the-art image classification models on versions of datasets where the image labels were completely randomized, and found that the models could easily achieve zero training error. In other words, the same exact model with the same hyperparameters and the same training process could both intelligently extract the signal from the original dataset and completely overfit the randomized dataset.

considered obsolete, in fact display competitive performance to the state-of-the-art when setting the training and evaluation process to be uniform across all methods [75, 56, 81]. The importance of experimental setup is especially concerning since even high-profile papers make routine errors in separating model *selection* on the validation data and model *evaluation* on the test data, or simply refuse to elaborate, for example, how hyperparameter optimization was performed [22]. In a recent position paper, the state of graph learning was accordingly summarized as a frantic competition for statistically insignificant gains in accuracy on unrealistic benchmark datasets, resulting in the field requiring a paradigm shift to achieve genuinely reliable advances [13].

Graph learning is just one example. Empirical machine learning as a whole is plagued by unreliability: for a comprehensive overview, see, e.g., [40]. Though theory is not immune to all of these issues, it is less vulnerable to them. Especially work that is based on rigorous, formal results can serve as a scientific anchor that is less susceptible to messy real-world factors, some of which are fundamentally unavoidable in empirical work, that can make it hard to discern what is real and what is noise. In addition to serving as a springboard to inspire new work, theoretical work can also give post-hoc explanations of *why* some tweaks work and others do not, thus helping steer research toward model adjustments that genuinely improve performance.

The point of all this is not to denigrate empirical research as a concept. It is very much important and will continue to be so; even most theoreticians are, in principle, concerned with what happens when the systems and algorithms they design and analyze meet the real world. But at this stage, it seems that what deep learning would benefit from the most is thorough theoretical and mathematical work. Ideally, high-quality and honest empirical work would go hand-in-hand with theoretical work, both justifying each other's results and steering each other's research in more fruitful directions. After all, as is argued in a recent article [69], the main goal of science is to produce new *understanding* of how the world we live in works, not just to achieve marginal quantitative improvements in the state-of-the-art.

### 2.1.2 Explainability

The second concern is the fact that AI systems based on deep learning can be *untrustworthy.* Not only is the deep learning process itself mysterious in part, but the massive neural networks it produces are *black boxes.* In other words, we understand neither how we create these models nor what the finished models even do, since rigorously explaining why a certain input makes a deep neural network produce a certain output remains a very challenging task. This is especially concerning at a time when AI systems are being integrated more and more into high-risk and safety-critical applications. These include operating critical infrastructure [30], diagnosing medical conditions [43] and even assisting decision-making in law enforcement [88] and judicial proceedings [48], where they have been shown to produce results systematically biased against certain social groups and to make mistakes with catastrophic consequences. Without a formal understanding of the models, debugging and fixing such behaviour is an ad-hoc exercise that lacks guarantees of success.

These concerns have led to the birth of the field of *explainable AI* (XAI). The goal of XAI is to either algorithmically analyze existing black-box models to create explanations of their behaviour,[2] or to create new models whose internal functioning is inherently human-understandable. The rapid development and adopment of AI models into high-stakes contexts means the need for such work is pressing. This has already been noticed at the level of law; for example, the General Data Protection Regulation (GDPR) that came into force in 2018 in the EU has been argued to provide customers with a "right to explanation" whenever companies use algorithmic decision-making [44]. With current black-box AI systems, such a right does not exist in practice, since even the service provider itself cannot extract an explanation from the algorithm. At this critical time, XAI as a scientific field remains in its infancy, with disagreement over how to evaluate the quality of different explanation methods [54] and even over the precise meaning of basic terms like "explainability" and "interpretability" [65]. More theoretical work—using formal methods based on rigorous mathematics—is thus timely for this important field.

## 2.2 Mapping the Landscape

Deep learning has grown to be a massive field and one could devote an entire academic career purely to summarizing the theoretical work done thus far. To remain within the scope of a master's thesis, we thus focus on the developing subfield of *graph learning*. First, we introduce an intersection of mathematical logic and theoretical computer science known as *descriptive complexity theory* and justify why it is a promising perspective from which to approach modern deep learning in general. We then provide a brief history of an ongoing research program applying descriptive complexity to the field of *distributed computing*. Finally, we describe a model that combines deep learning with distributed computing: the *graph neural network*. We end the section with a short summary of the key theoretical results achieved on this model and a motivation of why more work is necessary. A graphical timeline of the major results in these programs can be found in Figure 2.1.

### 2.2.1 Descriptive Complexity Theory

The traditional approach to measuring the power of computational models is to classify the problems they can solve by the time and space they require. These are certainly important measures in engineering, but as mathematicians, we are not yet satisfied. What about a simpler question: how hard are the problems these systems can solve? In other words, what kinds of functions can they express? What is their *expressive power*? Here, the field of *descriptive*

[2] One approach is *local explanation*, which produces post-hoc explanations that hold in a local neighbourhood of the given input, but whose outputs are sometimes vastly inaccurate when validated [63]. Explanation methods themselves can thus suffer from untrustworthiness.

*complexity theory* comes in. The main goal in this framework is, aptly, to *describe* the inherent mathematical complexity of a computational problem by answering the question "how rich of a *logical language* do I need to specify this problem?" [41].

The first major result in descriptive complexity arose from *automata theory*, when Julius Büchi showed in 1960 that the sets of strings that can be recognized by *finite-state automata* are precisely those that can be characterized by formulas of *monadic second-order logic* (MSO) [16].[3] Perhaps the most famous result in the field was published in 1974, when Ronald Fagin showed that *NP*, the class of problems solvable in polynomial time by a *non-deterministic Turing machine*, is precisely the class of properties expressible in *existential second-order logic* (ESO) [23]. Büchi and Fagin both achieved the "holy grail" of results in descriptive complexity: for a given abstract machine (such as finite-state automata) or class of problems (such as NP), we find a logic $\mathcal{L}$ such that each machine or problem can be translated into an equivalent set of logical formulas $L \subseteq \mathcal{L}$, and vice-versa [72]. By showing that a class of problems or models can be translated to a specific formal logic, we precisely *characterize* their expressive power. Further, analyzing properties of the logic, such as *decidability*—whether or not finding out the truth or falsity of any statement formed in that logic can be automated—can give us information about the models they characterize.

In addition to reaching the mathematical core of the difficulty of a computational task, descriptive complexity has two further attractions as described by Reiter [72]. The first is a practical one: by translating problems of computer science into mathematical logic, we shine a time-tested light on a relatively young field. The study of formal logic has over a hundred years of history, and open questions in the world of computer science often have a natural correspondence with a logical problem solved decades ago. The century-long development of logic provides computer scientists with particularly refined mathematical tools and proof techniques but also grants flexibility: some ideas are simply easier to write and think about by using abstract machines, others by using logical formulas. Second, by showing that two independent mathematical systems characterize the same class of problems, we legitimize them both. Choices made in the mathematical definitions of computational systems, in particular, can at times seem arbitrary. Showing that two seemingly different mathematical devices turn out to describe the same thing, however, gives us strong evidence that the definitions are in fact natural and have captured something essential about the nature of computation.

A final reason to consider this approach is the fact that methods based on mathematical logic have recently emerged as a promising subfield of XAI, namely *formal explainability* in AI (FXAI) [57]. One direction, in the spirit of *formal verification*, is to first give mathematical specifications of these systems, then use them to prove that they satisfy certain properties, i.e., *verify* that they

[3]Interestingly, in the following year, this was independently discovered by both Calvin Elgot [21] and Boris Trakhtenbrot [82]. The result is thus sometimes called the *Büchi–Elgot–Trakhtenbrot theorem*.

do what they are supposed to [78]. The tools used to automate this, such as *Boolean satisfiability solvers* that search the proof space for verifications of the desired properties, rely heavily on work in mathematical logic. Another direction in FXAI is to produce *explanations* of model behaviour also using various tools from mathematical logic—both *local* explanations that post-hoc explain a single prediction, and *global* explanations that uniformly explain an entire class of predictions. Descriptive complexity is a powerful tool in these projects, since identifying models with logics naturally gives us a strong foundation from which to develop these logic-based methods. In addition, the characterizing logic is itself an (often) human-understandable description of the expressive power of the model.

### 2.2.2 Distributed Computing

Both Büchi's and Fagin's results describe machines computing in isolation. When we consider multiple machines and let them communicate with each other during computation, we enter the world of *distributed computing*, where we study *networks* of computation in which separate nodes perform separate computational tasks, pass messages to each other, and thereby work together to solve a single, global problem. Distributed computing originated in engineering challenges involved in the development of parallel hardware and multitasking operating systems in the 1960s, but has since grown to study the large diversity of distributed computation models and their properties [27].

In distributed computing, the network graph is both the input problem and the architecture with which it is solved. A natural question to ask is thus: "what kinds of properties of this graph can these systems detect?" One way of answering this question is to show which logics characterize the expressive power of these models on a given class of input graphs, or in other words, apply *descriptive complexity* to these models. Surprisingly, the descriptive complexity theory of distributed computing is a rather young research program, first pioneered in the early 2010s by a group of Finnish logicians and computer scientists. The work was inspired by a 2012 paper by Hella et al. [38] in which several classes of *synchronous constant-time distributed algorithms* were characterized by various *modal logics*, and explicitly inaugurated in a 2013 paper by Kuusisto [50] in which the constant-time assumption was lifted and distributed algorithms with arbitrary runtimes were characterized by a recursive logic called *modal substitution calculus* (MSC). The essential discovery was that distributed computing and modal logic have an intrinsic connection; for example, the basic modal logic is equivalent to *local* distributed automata, in which each node stops changing its state after a constant number of rounds.

A year later, Kuusisto lifted another restriction with an investigation of distributed algorithms in infinite-size networks [51]. Distributed computing may also be approached from an automata-theoretic perspective by generalizing finite-state automata into *distributed automata*. Accordingly, in 2015, Reiter extended Büchi's result by characterizing MSO on directed graphs with *distributed graph automata* [70], and in 2017, *forgetful distributed automata* were shown to be equivalent to finite-state automata on finite words but strictly more powerful than finite tree automata on ordered directed trees [52].

Especially the logic MSC proved fruitful as a tool in characterizing distributed computation. Recently, MSC has been shown to characterize distributed computing with circuits [5]. *Boolean network logic* (BNL), which is equivalent to the diamond-free fragment of MSC known simply as *substitution calculus* (SC), has been shown to characterize a very general class of neural networks [7]. Finally, a parallel result by Reiter showed that *asynchronous* distributed automata, where nodes can operate at different speeds and messages can be delayed or even lost, are equivalent to a fragment of the *modal $\mu$-calculus*, a logic closely related to MSC [71].

### 2.2.3 Graph Neural Networks

Perhaps one of the most versatile mathematical forms of representing interconnected data is the humble collection of nodes and edges known as a *graph*. Graph data encompasses a large variety of modalities and tasks, which can be at the level of graphs or individual nodes, and can involve classification, regression, or other tasks like *link prediction*, where we try to predict whether or not an edge exists between two nodes. Taking graph classification as an example, these tasks include classifying molecules as toxic or non-toxic based on their chemical structure alone [39], predicting the genre of a film from a collaboration network of its actors [87], or automating the identification of cuneiform signs by representing them as graphs and applying standard methods for graph classification [46].

Graphs have not been spared from the neural revolution. Traditional machine learning methods specialized for graph data, such as *graph kernels* [47], have been largely supplanted in recent years by *graph neural networks* (GNNs). GNNs were originally introduced by Gori et al. in 2005 [33] and have roots going back to the 1990s [80, 29], but their real rise to stardom occurred during the deep learning boom of the 2010s. The "AlexNet moment" for GNNs arrived in 2016 when Kipf and Welling [45] introduced their *graph convolutional network* (GCN) architecture, achieving state-of-the-art performance on datasets from multiple domains. GNNs and graph-based deep learning in general have since emerged as one of the most consistently prominent topics in recent machine learning,[4] partly due to the vast and growing amount of graph data produced by and used to analyze social networks and recommendation systems.

The canonical model of a GNN, also known as a *message-passing neural network* (MPNN) [31], is a deep neural network that receives a graph as a collection of *node embeddings* or *node feature vectors*—floating-point values associated with each node—along with a representation of the geometry of the graph. An ordinary feed-forward neural network is then paired up with each

[4]The 2022 and 2023 editions of the *International Conference on Learning Representations* (ICLR) both had graph neural networks as the third most popular topic [53]. Similarly, the *Conference and Workshop on Neural Information Processing Systems* (NeurIPS) has had graphs as one of the top five topics during the years 2020–2024, falling only slightly in 2023 and 2024 amid the meteoric growth of research in large language models [62].

node, and layer-by-layer, the feature vector of the node is updated by aggregating information about the feature vectors of its neighbours (cf. Figure 3.1). Finally, the refined feature vectors are used to compute a desired function on the input graph, such as predicting whether or not the given molecule is toxic [68, Ch. 13]. In essence, GNNs are simply distributed algorithms whose aggregation and combination functions are learned by training a neural network.

As with other neural methods, years of empirical success and hype were eventually followed by more theoretical study. In the late 2010s, independent work by Morris et al. [61] and Xu et al. [86] found that the *discriminatory power* of GNNs is bounded by the *one-dimensional Weisfeiler–Leman graph isomorphism test*; in other words, GNNs can only distinguish two nodes if the 1-WL-test can (cf. Section 4.1). Afterwards, Sato et al. [76] established a connection between GNNs and *distributed local algorithms* (distributed algorithms that run in constant time rather than being dependent on the size of the input), describing the approximation ratios that GNNs can learn for combinatorial graph problems. These results were promising but indeed described only the discriminatory power of GNNs, which is more of a hint of the precise expressive power of the model. In practice, distinguishing graphs from each other is often just the first step in the task at hand; a natural followup is to then *classify* nodes or graphs, or to perform other operations, such as link prediction—tasks which these results could not comment on.

Another approach is to study the *uniform expressive power* of GNNs with the framework of descriptive complexity. The first such result arrived in 2020 when Barceló et al. [11] showed that, in restriction to first-order definable properties, *constant-iteration* GNNs (ones that have a fixed number of layers) are equivalent to *graded modal logic* (GML), and extending them with *global readout* (allowing them to aggregate information about the entire graph) allows them to express all formulas of the *two-variable fragment of first-order logic* ($\mathrm{FOC}_2$).[5] This work inspired many groups to characterize variations of the basic GNN architecture. *Recurrent* GNNs with a termination condition based on local fixed points were first studied by Pflueger et al. [66] in 2024, who showed an equivalence with *graded two-way modal logic* in restriction to LocMMFP, an extension of first-order logic. Later that same year, Ahvonen et al. [8] published a paper giving an *unrestricted* characterization of recurrent GNNs, with a focus on more realistic restrictions on the combination and aggregation functions used as well as on the effect of finite-accuracy floating-point numbers on the characterization. GNNs with *random initialization* or *rGNNs* [77], have been given a similar treatment by a German group led by Grohe [1, 35, 74], and *graph transformers* have very recently also been characterized [4]. Other groups across Europe remain active in the study of this subfield, with theoretical pre-prints being published at a steady pace [34, 15, 64] alongside applications based on this work for novel graph learning algorithms [42], including explainable graph learning methods [26, 67].

Analysing GNNs through a logical lens has shown promise and accordingly

[5]Interestingly, the converse does not hold: a very recent preprint shows that constant-iteration GNNs with global readout are, in fact, strictly more expressive than $\mathrm{FOC}_2$ [36].

risen in popularity. But the need remains for further work. As recently argued by Morris et al. [60], one key limitation to be addressed in the theory of GNNs is the *coarse-grained* nature of expressivity analysis. The results so far are purely binary, but practical differences between models of the same theoretical expressive power suggests that a more fine-grained analysis measuring *degrees of similarity* between non-isomorphic graphs would be fruitful. Theoretical work has also suffered from an excessive focus on a restricted subset of specific models, while the field consists of an incredibly large, ever-growing family of methods and architectures. At the same time, important practical concerns like generalization to novel graphs as well as the GNN training process are chronically unstudied. Basic theoretical understanding is still of vital importance. The evaluation of different methods of automated reasoning on graphs cannot remain pure accuracy optimization on benchmark datasets—a systematically unreliable measure, as described in the previous section.

GNNs are a special case of the current neural paradigm where empirical progress is evident but theoretical understanding lags behind. Machine learning in general is a complex endeavour with many degrees of freedom: the model architecture, the training process, which relevant subclass of input data we have, which task we are trying to perform, and so on. GNNs complicate matters further by identifying the already high-dimensional input data and the topology of the network with which it is processed. In a more optimistic light, graph learning is thus a fertile field with many perspectives from which to study and many results yet to be found. Mathematical logic, through the last 13 years of applying descriptive complexity to distributed computing, has shown to be a particularly powerful and general tool in this work. This thesis hopes to synthesize these views and contribute a small building block in researching this intersection of logic and computer science.

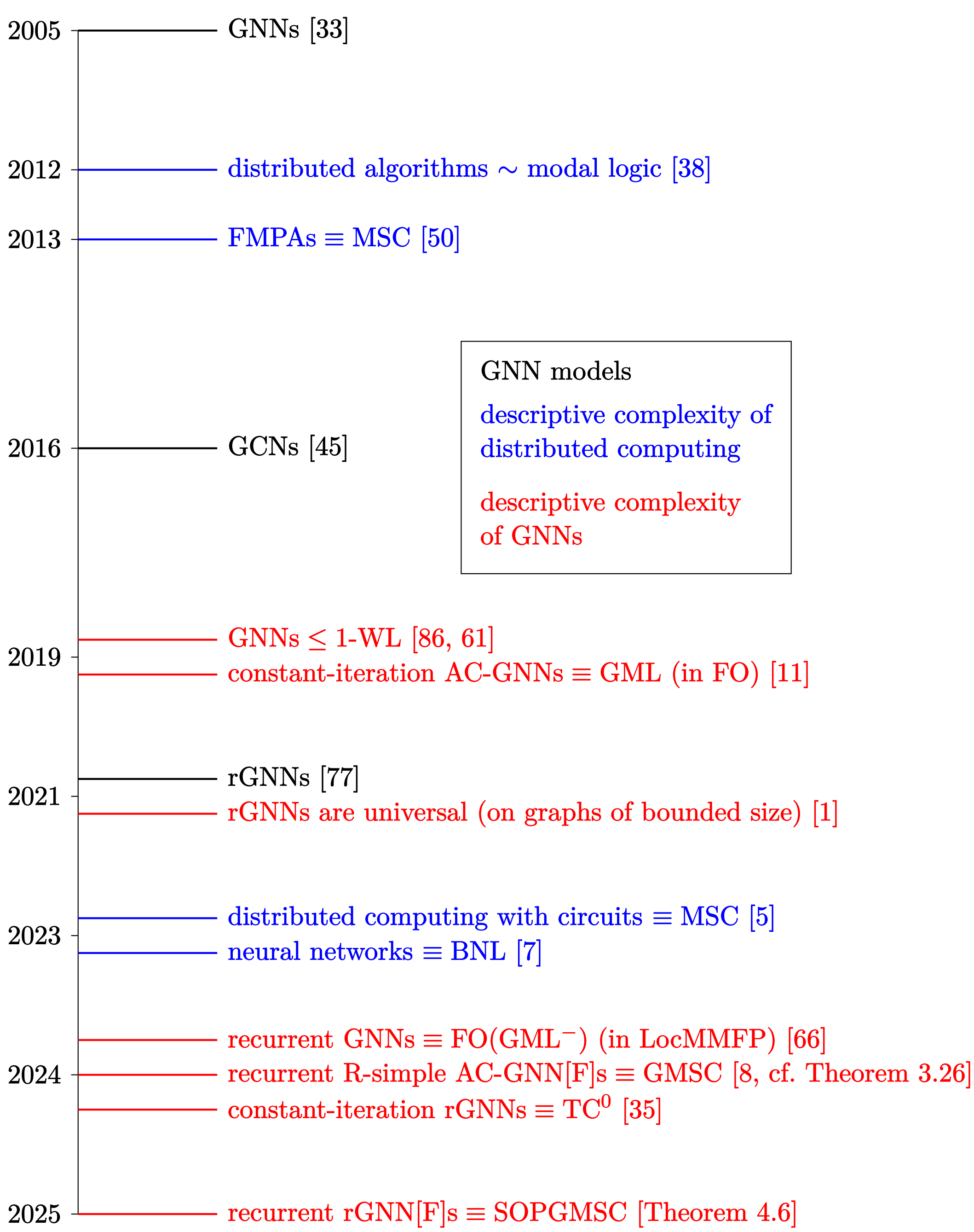



**Figure 2.1.** Timeline of key developments in logical classifications for distributed computing models.

# 3 Logically Characterizing Graph Neural Networks

In this chapter, we present a logical characterization of recurrent graph neural networks that calculate with floating-point numbers. Unless denoted otherwise, all results and definitions in this chapter are taken from Ahvonen et al. [8].

## 3.1 Preliminaries

**Definition 3.1.** By $\mathbb{N}$, $\mathbb{Z}_+$ and $\mathbb{R}$ we denote, respectively, the non-negative integers, the positive integers and the real numbers. We denote the power set of a set $X$ by $\mathcal{P}(X)$ and the cardinality of a set $X$ by $|X|$. Vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ are written in bold to distinguish them from scalar numbers $x, y \in \mathbb{R}$.

**Definition 3.2.** Let $X$ be a set. A *multiset over* $X$ is a function $M : X \to \mathbb{N}$. A $k$*-multiset over* $X$ is a function $M_k : X \to \{0, \dots, k\}$, where $k \in \mathbb{N}$. We denote the set of all multisets over $X$ by $\mathcal{M}(X)$ and the set of all $k$-multisets over $X$ by $\mathcal{M}_k(X)$. The *size* of a multiset $M$ is defined as $|M| := \sum_{x \in X} M(x)$.

Intuitively, a multiset is simply a set that can store multiple copies of identical elements, and a $k$-multiset is one that can count up to $k$ distinct copies. We often denote multisets by double curly brackets; for example, the multiset $\{\{a, a, b, c, c, c\}\}$ is formally the function $\{(a, 2), (b, 1), (c, 3)\}$ and its size is 6.

**Definition 3.3.** We denote the (countably infinite) set of *node label symbols* with $\mathrm{LAB} := \{p_i \mid i \in \mathbb{N}\}$, and an arbitrary finite subset of it with $\Pi \subseteq \mathrm{LAB}$. A $\Pi$*-labeled directed graph* is a structure $G = (V, E, \lambda)$, where

1. $V$ is a finite set of *nodes*,
2. $E \subseteq V \times V$ is a set of *edges* that connect the nodes, also referred to as the *accessibility relation*, and
3. $\lambda : V \to \mathcal{P}(\Pi)$ is a *node labeling function.*

We often denote node label symbols by $p, q, r$, and so on. Note that each node label is a *set* of these symbols; a node label can thus contain a single symbol, multiple different symbols, or no symbol at all. We also allow the graph to contain self-loops, i.e., edges of the form $(v, v) \in E$.

The set of *out-neighbours* of a node $v \in V$ is $N(v) := \{u \in V \mid (v, u) \in E\}$. A *pointed graph* is a pair $(G, v)$, where $G = (V, E, \lambda)$ is a $\Pi$-labeled graph and $v \in V$.

In this thesis, by "graph" we always refer to $\Pi$-labeled directed graphs, where $\Pi$ is often clear from context. In the literature, the terminology corresponding to the notions given in the previous definition can vary. We use terms common in the GNN world, but in modal logic, node label symbols are called *proposition symbols* and $\Pi$-labeled directed graphs are called *Kripke models over* $\Pi$. We use these terms interchangably throughout the thesis.

**Definition 3.4.** Let $G = (V, E, \lambda)$ and $H = (V', E', \lambda')$ be $\Pi$-labeled graphs. We call a bijection $f : V \to V'$ a *(graph) isomorphism between* $G$ *and* $H$ if

1. $(v, u) \in E \iff (f(v), f(u)) \in E'$ and
2. $\lambda(v) = \lambda'(f(v))$ for all $v \in V$.

If there exists an isomorphism between $G$ and $H$, we call $G$ and $H$ *isomorphic* and write $G \cong H$. Similarly, if $f(v) = v'$ for some isomorphism $f$ between $G$ and $H$, we call the pointed graphs $(G, v)$ and $(H, v')$ *isomorphic* and write $(G, v) \cong (H, v')$.

Let $\mathbb{G}$ be a class of graphs. We call $\mathbb{G}$ *isomorphism-closed* if for all $G \in \mathbb{G}$, if $G$ and some graph $H$ are isomorphic, then $H \in \mathbb{G}$. A function $g$ that has $\mathbb{G}$ as its domain is *isomorphism-invariant* if $g(G) = g(H)$ for all isomorphic graphs $G, H \in \mathbb{G}$. These definitions easily extend to pointed graphs. Finally, a *graph property over* $\Pi$ is an isomorphism-closed class of $\Pi$-labeled graphs and a *node property over* $\Pi$ is an isomorphism-closed class of pointed $\Pi$-labeled graphs.

## 3.2 Graph Neural Networks

**Definition 3.5.** Let $d \in \mathbb{Z}_+$ and $\Pi \subseteq \mathrm{LAB}$. A *(real number) graph neural network* GNN[$\mathbb{R}$] *for* $(\Pi, d)$ is a structure $\mathcal{G} = (\mathbb{R}^d, \pi, \delta, F)$, where

1. $\pi : \mathcal{P}(\Pi) \to \mathbb{R}^d$ is an *initialization function*,
2. $\delta : \mathbb{R}^d \times \mathcal{M}(\mathbb{R}^d) \to \mathbb{R}^d$ is a *transition function*, consisting of an *aggregation function* $\mathrm{AGG} : \mathcal{M}(\mathbb{R}^d) \to \mathbb{R}^d$ and a *combination function* $\mathrm{COM} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ such that $\delta(\mathbf{x}, Y) := \mathrm{COM}(\mathbf{x}, \mathrm{AGG}(Y))$, and
3. $F \subseteq \mathbb{R}^d$ is a set of *accepting feature vectors*.

The GNN $\mathcal{G}$ computes on a $\Pi$-labeled graph $G = (V, E, \lambda)$ by first, in round $t = 0$, assigning all nodes $v \in V$ an initial *feature vector* $\mathbf{x}(v)^0 := \pi(\lambda(v))$ of length $d$ based on its label, and in each subsequent round $t \geq 1$, updating the feature vector of $v$ by the equation

$$\begin{aligned}\mathbf{x}(v)^t &:= \delta(\mathbf{x}(v)^{t-1}, \{\{\mathbf{x}(u)^{t-1} \mid u \in N(v)\}\}) \\ &= \mathrm{COM}(\mathbf{x}(v)^{t-1}, \mathrm{AGG}(\{\{\mathbf{x}(u)^{t-1} \mid u \in N(v)\}\})).\end{aligned}$$

GNNs that use a two-phase transition function like this are also called *aggregate-combine* GNNs or AC-GNNs, as in [11].[6] GNNs with unrestricted

[6]Note that our definition assumes that we use the same aggregation and combination functions in each round, i.e., that the layers of the underlying neural network are identical. Such GNNs are variously called *homogeneous* [11] or *uniform* [66]. This simplification does not affect expressive power; the proof of this in [8, Appendix B.7] applies to the recurrent case as well.

aggregation and combination functions would be extremely powerful, being able to compute, for instance, that the number of out-neighbours of a node belongs to an undecidable subset of the natural numbers [8, Remark 2.2]. We thus define a restricted version of GNNs with functions used in actual implementations as follows.

**Definition 3.6.** GNNs that update the feature vector of $v$ by the equation

$$\mathbf{x}(v)^t := \mathrm{ReLU}^*(\mathbf{x}(v)^{t-1} \cdot C + \sum_{u \in N(v)} \mathbf{x}(u)^{t-1} \cdot A + \mathbf{b}),$$

where

1. $\mathrm{ReLU}^* : \mathbb{R}^d \to \mathbb{R}^d$ is the *truncated rectified linear unit*, applied point-wise to each element of the input vector as $\mathrm{ReLU}^*(x) := \min(\max(0, x), 1)$,
2. $C, A \in \mathbb{R}^{d \times d}$ are $d \times d$ matrices, and
3. $\mathbf{b} \in \mathbb{R}^d$ is a vector of length $d$,

are called *R-simple aggregate-combine* GNNs. This technical definition is illustrated in Figure 3.1.

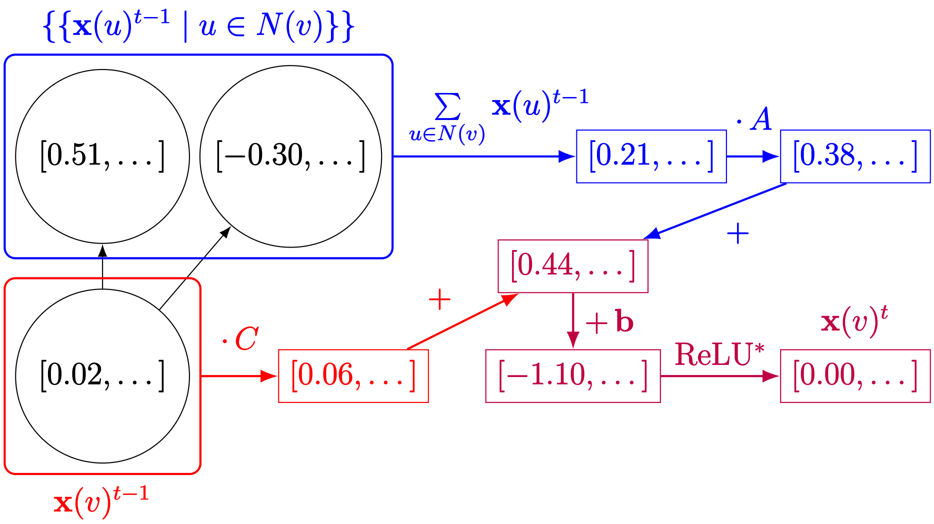



**Figure 3.1.** An R-simple aggregate-combine GNN computing a feature vector for $v$ in the round $t$. The transition function consists of summing the feature vectors of the neighbours and multiplying the result with the matrix $A$, multiplying the previous feature vector of $v$ with the matrix $C$, then summing both resulting vectors together with the bias vector, and finally applying the activation function $\mathrm{ReLU}^*$.

The matrices $C, A$ and the *bias vector* $\mathbf{b}$ (together also called *weights*) are learned during the training of the GNN. In this thesis, our focus is on the theoretical expressive power of GNNs and we thus do not consider the question of whether or not the appropriate weights can be learned by common training methods like *stochastic gradient descent*. As part of the main result of this chapter, however, we show that when using floating points, restricting our attention to R-simple aggregate-combine GNNs has no effect on the expressive power of the model; any graph property expressed by an arbitrary GNN can also be expressed by an R-simple aggregate-combine GNN.

The task of (binary) node classification can now be formalized as a GNN taking a pointed graph $(G, v)$ as input, updating the node feature vectors of the graph for some amount of time, and either accepting or rejecting the input based on the feature vector of $v$. Similarly, graph classification can be formalized as running the GNN on each node $v \in V$ separately and accepting the graph if and only if each pointed graph $(G, v)$ is accepted.

But what do we precisely mean by "accept"? We have left the definition open so far, as there is an important distinction to be made here. The leading class of GNNs in practical use are ones that operate for some fixed number of rounds $L$, i.e., pass the graph through an $L$-layer neural network, with, quite naturally, the final feature vectors determining whether or not the input is accepted. Another option is to let the GNN run "forever", halting and accepting or rejecting an input only if a certain condition is met. This is the difference between *constant-iteration* GNNs and *recurrent* GNNs. Resembling the case of finite-state automata extended from finite to infinite inputs splitting into different models by their acceptance condition, there are numerous different ways to define acceptance for recurrent GNNs, and there is no standard choice that subsumes all of them. In this thesis, we follow the lead of [8] and simply accept $(G, v)$ if $v$ obtains an accepting feature vector in some round of computation.

**Definition 3.7.** Let $\mathcal{G} = (\mathbb{R}^d, \pi, \delta, F)$ be a GNN. By itself, we call $\mathcal{G}$ a *recurrent* GNN and say that it *accepts* a pointed graph $(G, v)$, denoted by $G, v \models \mathcal{G}$, if $\mathbf{x}(v)^t \in F$ for some $t \in \mathbb{N}$. In contrast, we call the pair $(\mathcal{G}, L)$, where $L \in \mathbb{N}$, a *constant-iteration* GNN and say that $(\mathcal{G}, L)$ accepts $(G, v)$, denoted by $G, v \models (\mathcal{G}, L)$, if $\mathbf{x}(v)^L \in F$.

In [9, Appendix D], an extended version of [8], it was shown that characterizations similar to the one in this thesis can be obtained for a comprehensive class of other natural acceptance conditions. These include *Büchi semantics*, where an accepting feature vector must be visited an infinite number of times, the *fixed-point semantics* of Pflueger et al. [66], where the feature vectors of the accepted nodes ultimately circulate within a set of accepting feature vectors, and a variant of it called *convergence-based fixed-point semantics* where the entire graph must stabilize before the GNN halts and accepts precisely those nodes that have stabilized into an accepting feature vector.

### 3.2.1 Floating Points

In theoretical computer science, we often treat the systems we study as if they compute with real numbers of infinite precision. In real-life implementations, of course, we must use finite-accuracy approximations of real numbers, the most common choice of which is *floating-point arithmetic*. It turns out that in the case of GNNs, this approximation has an effect on the theoretical expressive power of the model. Namely, real-number GNNs are equivalent to *graded modal logic with infinite disjunctions* ($\omega$-GML) [8, Theorem 3.4], while floating-point GNNs are equivalent to *graded modal substitution calculus* (GMSC) [8, Theorem 3.2]. Interestingly, in a more specific context of restriction to properties definable in MSO, these models are equivalent [8, Theorem 4.3].

To keep the presentation at a reasonable length—and since we are mostly interested in what GNNs can do in the real world—we choose to focus on GNNs that compute with floating-point numbers. We begin by formally defining floating-point arithmetic.

**Definition 3.8.** Let $p \in \mathbb{Z}_+$, $n \in \mathbb{N}$ and $\beta \in \mathbb{Z}_+ \setminus \{1\}$. By $D_S$, we denote the set of rational numbers accurately representable in the form $0.d_1 \dots d_p \times \beta^e$ or $-0.d_1 \dots d_p \times \beta^e$, where $0 \leq d_i \leq \beta - 1$ for all $1 \leq i \leq p$, and $e \in \{-n, \dots, n\}$. We call numbers $f \in D_S$ *floating-point numbers*. A *floating-point system* is a tuple $S = ((p, n, \beta), +_S, \cdot_S)$, where $+_S$ and $\cdot_S$ are operations $D_S \times D_S \to D_S$ that first calculate the real number sum or product of the two operands and then round the result to the nearest floating-point number in $D_S$, breaking ties by rounding to the number with an even least significant digit. We often use the notation $f \in S$ to mean that $f$ is *representable in* $S$, that is, that $f \in D_S$.

Before giving a formal definition of floating-point GNNs, we investigate an interesting arithmetic property of floating-point systems that differentiates them from the real numbers.

**Example 3.9.** Let $p = 2$, $n = 3$ and $\beta = 10$. The floating-point system $S$ now contains all numbers of the form $0.d_1 d_2 \times 10^e$, where $d_1, d_2 \in \{0, \dots, 9\}$ and $e \in \{-3, \dots, 3\}$, or in other words, numbers in the range $[-999, 999]$ that can be represented with two significant decimal digits in normalized scientific notation. For example, 0.0012, 9.7, and $-340$ are all representable in $S$, but 12.5 is not, and thus $12 +_S 0.5 = 12$. Such rounding implies that the sum of two floating-point numbers is in fact a non-associative operation. For example,

$$1 +_S (0.01 +_S (-1)) = 1 +_S (-0.99) = 0.01,$$

but

$$(1 +_S 0.01) +_S (-1) = 1 +_S (-1) = 0,$$

since 1.01 is not representable in $S$.

In real-life implementations of GNNs, summation in the aggregation function is often done using an implicit linear ordering of the nodes of the graph. Since the sum of floats is non-associative, however, this breaks the desired isomorphism-invariance of GNNs [8]; different orderings of the nodes would result in different values when summing and thus different end results. A natural fix is to then assume that whenever we sum (multisets of) floats in GNNs, we do it in increasing order of *the floats themselves*, not with any ordering of nodes. This choice is also widely used in practice [8]. Such an assumption leads to a bound in the multiplicities of the elements of the sum. In other words, floating-point GNNs are limited by how many out-neighbours each node can count.

**Definition 3.10.** Let $M$ be a multiset over $X$ and $k \in \mathbb{N}$. The *$k$-projection* of $M$ is a $k$-multiset $M_{|k}$ over $X$, where

$$M_{|k}(x) := \min(M(x), k)$$

for all $x \in X$. Additionally, for a multiset $N = \{\{f_1, \dots, f_\ell\}\}$ of floats in a floating-point system $S$, we let $\mathrm{SUM}_S(N)$ denote the output of the sum $f_1 +_S \cdots +_S f_\ell$, where the floats are summed in increasing order.

**Theorem 3.11.** *For all floating-point systems $S$, there exists a $k \in \mathbb{N}$ such that for all multisets $M$ of floats in $S$, we have* $\mathrm{SUM}_S(M) = \mathrm{SUM}_S(M_{|k})$.

*Proof.* We will prove that $k := |S|$ suffices, since summing that many times guarantees that we "run out of floats" and get stuck. Consider the effect of an arbitrary number $f \in M$ with the multiplicity $n$ on the sum, carried out in the increasing order of the floats.

Suppose first that $n > |S|$. If $f$ is negative, adding it to the current value of the sum cannot increase its value. If it decreases it, then doing so $|S|$ times suffices to reach the smallest negative number representable in $S$, after which additional summing has no effect, and if it does not decrease it, then the current value of the sum stays the same no matter how many times we repeat this. If $f = 0$, then clearly it has no effect on the current value of the sum. Finally, if $f$ is positive, it cannot decrease the current value of the sum. Hence, by similar arguments as in the negative case, the value stays the same.

If $n \leq |S|$, then the $k$-projection has no effect on the multiplicity of $f$. We thus conclude that the effect of $f$ is the same in both the original and the $k$-projected case, and since $f$ and $M$ were arbitrary, we have the desired result.[7] $\square$

We may now finally present the formal definition of a floating-point graph neural network.

**Definition 3.12.** Let $S$ be a floating-point system. A *floating-point graph neural network* GNN[F] *over $S$* is a GNN $\mathcal{G} = (S^d, \pi, \delta, F)$, where the elements of the feature vectors are floating-point numbers in $S$ and which is *bounded* in the sense that its aggregation function can be written as a function AGG : $\mathcal{M}_k(S^d) \to S^d$ for some *bound* $k \in \mathbb{Z}_+$.

## 3.3 GML and GMSC

We now turn to the main logical definitions we will use in the rest of the thesis. We begin by defining *modal logic*, which is the base logic we will shortly extend.

[7] A smaller upper bound of $k = \beta^{p+1} + \beta^p + \beta^{p-1}$ can also be found [8, Appendix A.2], but this is not relevant for our purposes.

**Definition 3.13.** Let $\Pi \subseteq \text{LAB}$. The set of $\Pi$-formulas of *modal logic* (ML) is generated by the grammar

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi,$$

where $p \in \Pi$. To shorten proofs that use induction over the formula structure, the other logical symbols are defined as abbreviations in the usual manner:

1. $\bot := \neg\top$.[8]
2. $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$.
3. $\varphi \rightarrow \psi := \neg(\varphi \wedge \neg\psi)$.
4. $\varphi \leftrightarrow \psi := \neg(\varphi \wedge \neg\psi) \wedge \neg(\neg\varphi \wedge \psi)$.
5. $\Box\varphi := \neg\Diamond\neg\varphi$.

Let $G = (V, E, \lambda)$ be a $\Pi$-labeled graph and $v \in V$. The satisfaction relation $\models$ for ML-formulas in the pointed graph $(G, v)$ is defined as follows:

1. $G, v \models \top$.
2. $G, v \models p \iff p \in \lambda(v)$.
3. $G, v \models \neg\varphi \iff G, v \not\models \varphi$.
4. $G, v \models \varphi \wedge \psi \iff G, v \models \varphi$ and $G, v \models \psi$.
5. $G, v \models \Diamond\varphi \iff$ there exists a $u \in N(v)$ such that $G, u \models \varphi$.

*Graded modal logic* is an extension of modal logic which allows *counting diamonds* $\Diamond_{\geq k}$ that express that a formula is true in at least $k$ out-neighbours.

**Definition 3.14.** Let $\Pi \subseteq \text{LAB}$. The set of $\Pi$-formulas of *graded modal logic* (GML) is generated by the grammar

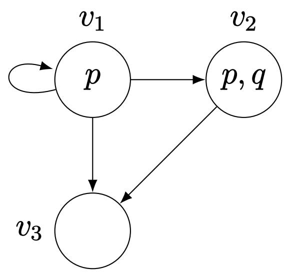
$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond_{\geq k}\varphi,$$

where $p \in \Pi$ and $k \geq 1$. We may also refer to $\Diamond_{\geq 1}$ as the ordinary modal diamond $\Diamond$. Let $G = (V, E, \lambda)$ be a $\Pi$-labeled graph and $v \in V$. The satisfaction relation $\models$ for the formula $\Diamond_{\geq k}\varphi$ in the pointed graph $(G, v)$ is defined as follows:

$$G, v \models \Diamond_{\geq k}\varphi \iff |\{u \in N(v) \mid G, u \models \varphi\}| \geq k.$$

**Example 3.15.** Let $\Pi = \{p, q\}$ and consider the following $\Pi$-labeled graph $G = (V, E, \lambda)$, where $V = \{v_1, v_2, v_3\}$, $\lambda(v_1) = \{p\}$, $\lambda(v_2) = \{p, q\}$ and $\lambda(v_3) = \emptyset$.

[8]The symbols $\bot$ and $\top$ are pronounced "bot" and "top" respectively.




It is clear that $G, v_1 \models p$, since $p$ is true in $v_1$. Furthermore, $G, v_1 \models \Diamond_{\geq 2} p$ but $G, v_1 \not\models \Diamond_{\geq 3} p$, since $v_1$ has precisely two neighbours in which $p$ is true.

We now define *graded modal substitution calculus* (GMSC), the logic that characterizes recurrent floating-point graph neural networks. This is a counting extension of *modal substitution calculus* (MSC), which is a recursive logic originally used in [50] to characterize a class of distributed message passing automata.

**Definition 3.16.** Let VAR $:= \{X_i \mid i \in \mathbb{N}\}$ be a (countably infinite) set of *schema variables*, $\mathcal{T} = \{X_1, \ldots, X_n\} \subseteq$ VAR be a finite set of *head predicates* and $\Pi \subseteq$ LAB. The set of $(\Pi, \mathcal{T})$*-schemata of GMSC* is generated by the grammar

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond_{\geq k} \varphi \mid X_i,$$

where $p \in \Pi$, $k \geq 1$ and $1 \leq i \leq n$. A $(\Pi, \mathcal{T})$*-program of GMSC* is a tuple $\Lambda = (T, I, \mathcal{A})$, where

1. $T$ is an $n$-sized set of *terminal clauses*, which are of the form $X_i(0) :- \varphi_i$,
2. $I$ is an $n$-sized set of *iteration clauses*, which are of the form $X_i :- \psi_i$, and
3. $\mathcal{A} \subseteq \mathcal{T}$ is a set of *appointed predicates*.

Here, $\varphi_i$ are $\Pi$-formulas of GML and $\psi_i$ are $(\Pi, \mathcal{T})$-schemata of GMSC for all $1 \leq i \leq n$, and both $\varphi_i$ and $\psi_i$ are called the *bodies* of their respective clauses. Usually the set of head predicates $\mathcal{T}$ is clear from context, and we speak simply of $\Pi$-programs of GMSC. The clauses of a GMSC-program can be depicted as follows:

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

Let $1 \leq i, j \leq n$. The *iteration formula* $X_i^t$ of the head predicate $X_i$ in *round* $t \in \mathbb{N}$ is a GML-formula defined as follows:

1. $X_i^0$ is $\varphi_i$.
2. For all $t \geq 1$, $X_i^t$ is obtained by replacing each head predicate $X_j$ that appears in the schema $\psi_i$ by the iteration formula $X_j^{t-1}$.

Let $\Lambda$ be a $\Pi$-program of GMSC. We say that $\Lambda$ *accepts* a pointed graph $(G, v)$, denoted by $G, v \models \Lambda$, if $G, v \models X^t$ for some appointed predicate $X \in \mathcal{A}$ and some $t \in \mathbb{N}$.

**Example 3.17.** [50, Proposition 6] [8, Example 2.5] The $\Pi$-program $\Lambda$, containing a single head predicate $X$ with the clauses

$$X(0) :- \Box\bot \qquad\qquad X :- \Diamond X \wedge \Box X,$$

produces the following iteration formulas:

$$\begin{aligned}
X^0 &= \Box\bot \\
X^1 &= \Diamond\Box\bot \wedge \Box\Box\bot \\
X^2 &= \Diamond(\Diamond\Box\bot \wedge \Box\Box\bot) \wedge \Box(\Diamond\Box\bot \wedge \Box\Box\bot) \\
&\vdots
\end{aligned}$$

Intuitively, the body $\Box\bot$ of the terminal clause says that the node has no out-neighbours, $X^1$ says that the node does have out-neighbours, but all of which themselves have no out-neighbours, and so on. Let $X$ be an appointed predicate and $G$ be the graph from Example 3.15. Now, since $G, v_2 \models X^1$, $\Lambda$ accepts $(G, v_2)$.

It can be shown that this program accepts precisely those pointed graphs that have the *centre-point property*, that is, those pointed graphs $(G, v)$ for which there exists an $n \in \mathbb{N}$ such that every directed walk starting from $v$ leads to a node with no out-neighbours in exactly $n$ steps.[9] The centre-point property is thus an example of a node property expressible in MSC.

**Definition 3.18.** Let $\varphi$ and $\psi$ be ML-formulas (or GNNs, programs of GMSC, or any other formalism evaluated over pointed graphs). We say that $\varphi$ *expresses* a node property $\mathcal{P}$ over $\Pi$ if for each pointed $\Pi$-labeled graph $(G, v)$ we have $(G, v) \in \mathcal{P} \iff G, v \models \varphi$.

If $G, v \models \varphi \iff G, v \models \psi$ for all pointed graphs $(G, v)$, we write $\varphi \equiv \psi$ and say that $\varphi$ and $\psi$ are *equivalent.* If for every formula $\varphi$ of a logic (or class of GNNs or class of GMSC-programs) $L$ there exists a formula $\psi$ of a logic $L'$ such that $\varphi \equiv \psi$, then we write $L \leq L'$ and say that $L'$ is *at least as expressive as L.* If $L \leq L'$ and $L' \leq L$, then we write $L \equiv L'$ and say that $L$ and $L'$ are *equivalent in expressive power*, or simply *equivalent.* If $L \leq L'$ and $L \not\equiv L'$, then we write $L < L'$ and say that $L'$ is *strictly more expressive than L.*

**Example 3.19.** Rather trivially, we have

$$\text{ML} < \text{GML} < \text{GMSC},$$

and in the next section, we will show that

$$\text{GNN[F]s} \equiv \text{R-simple aggregate-combine GNN[F]s} \equiv \text{GMSC}.$$

[9] A *directed walk starting from* $v_0$ is a sequence of nodes $v_0, \ldots, v_n$ such that $(v_i, v_{i+1}) \in E$ for each $0 \leq i \leq n-1$. Note that not every node in the walk is necessarily distinct.

## 3.4 GNN[F]s and GMSC Are Equivalent

In this section, we present the first main result of [8]: GNN[F]s, R-simple aggregate-combine GNN[F]s and GMSC are all equivalent in expressive power.

### 3.4.1 Simulating GMSC with GNN[F]s

This direction of the equivalence proof comes from [8, Appendix B.3]. We first introduce some auxiliary definitions related to schemata.

**Definition 3.20.** The *formula depth of a* $(\Pi, \mathcal{T})$*-schema of GMSC* $\varphi$ is the maximum number of nested operators $\neg$, $\wedge$ and $\Diamond_{\geq k}$ in $\varphi$, and is denoted $\mathrm{fdepth}(\varphi)$. Formally, we define:

1. $\mathrm{fdepth}(\top) := 0$.

2. $\mathrm{fdepth}(p) := 0$ for all $p \in \Pi$.

3. $\mathrm{fdepth}(X) := 0$ for all $X \in \mathcal{T}$.

4. $\mathrm{fdepth}(\neg\varphi) := \mathrm{fdepth}(\varphi) + 1$ for all $(\Pi, \mathcal{T})$-schemata $\varphi$.

5. $\mathrm{fdepth}(\varphi \wedge \psi) := \max(\mathrm{fdepth}(\varphi), \mathrm{fdepth}(\psi)) + 1$ for all $(\Pi, \mathcal{T})$-schemata $\varphi$ and $\psi$.

6. $\mathrm{fdepth}(\Diamond_{\geq k}\varphi) := \mathrm{fdepth}(\varphi) + 1$ for all $(\Pi, \mathcal{T})$-schemata $\varphi$ and $k \geq 1$.

The *formula depth of a GMSC-program* is the maximum formula depth of the bodies of its clauses.

**Definition 3.21.** The *set of subschemata of a* $(\Pi, \mathcal{T})$*-schema of GMSC* is defined recursively as follows:

1. $\mathrm{sub}(\top) := \{\top\}$.

2. $\mathrm{sub}(p) := \{p\}$ for all $p \in \Pi$.

3. $\mathrm{sub}(X) := \{X\}$ for all $X \in \mathcal{T}$.

4. $\mathrm{sub}(\neg\varphi) := \mathrm{sub}(\varphi) \cup \{\neg\varphi\}$ for all $(\Pi, \mathcal{T})$-schemata $\varphi$.

5. $\mathrm{sub}(\varphi \wedge \psi) := \mathrm{sub}(\varphi) \cup \mathrm{sub}(\psi) \cup \{\varphi \wedge \psi\}$ for all $(\Pi, \mathcal{T})$-schemata $\varphi$ and $\psi$.

6. $\mathrm{sub}(\Diamond_{\geq k}\varphi) := \mathrm{sub}(\varphi) \cup \{\Diamond_{\geq k}\varphi\}$ for all $(\Pi, \mathcal{T})$-schemata $\varphi$ and $k \geq 1$.

The *set of subschemata of a GMSC-program* is the union of all sets of subschemata of schemas present in the program.

Next, the following technical lemma shows that GMSC-programs can be turned into a form more easily simulated with GNN[F]s. The key idea is to move all computation into the iteration clauses, then "balance" them such that each clause has the same formula depth and each conjunction splits the formula depth evenly. This makes it possible to synchronize the evaluation of each iteration formula when processing them one formula-depth layer at a time.

**Lemma 3.22.** For each $\Pi$-program $\Lambda$ of GMSC with formula depth $D$, we can construct an equivalent $\Pi$-program $\Gamma$ of GMSC which has the following properties:

1. Each terminal clause is of the form $X(0) :- \bot$.
2. Each iteration clause has the same formula depth $\max(3, D+2)$.
3. If $\varphi \wedge \psi$ is a subschema of $\Lambda$ such that neither $\varphi$ nor $\psi$ is $\top$, then $\varphi$ and $\psi$ have the same formula depth.

*Proof.* We begin constructing our program $\Gamma$ by first defining a new head predicate $I$ with the clauses $I(0) :- \bot$ and $I :- \top$. Then, for each head predicate $X_\Lambda$ of $\Lambda$, we add a head predicate $X_\Gamma$ with the rules

$$X_\Gamma(0) :- \bot \qquad\qquad X_\Gamma :- (\neg I \wedge \varphi) \vee (I \wedge \psi),$$

where $\varphi$ and $\psi$ are the bodies of the terminal and iteration clauses, respectively, of $X_\Lambda$. The first requirement of the lemma is now satisfied. We satisfy the second requirement by increasing the formula depth of each iteration clause to $\max(3, D+2)$, which can be done purely syntactically by adding an even number of dummy negations, along with a dummy conjunction with $\top$ if an odd increase is required.

The third requirement is satisfied with a similar syntactic modification. Let $\varphi \wedge \psi$ be a subschema of some schema in $\Gamma$ such that neither $\varphi$ nor $\psi$ is $\top$. Without loss of generality, we may assume that $\mathrm{fdepth}(\varphi) > \mathrm{fdepth}(\psi)$. Now, denoting $n := \mathrm{fdepth}(\varphi) - \mathrm{fdepth}(\psi)$ and $n$ repeated negations with $(\neg)^n$, we replace, one-by-one, each instance of $\varphi \wedge \psi$ in $\Gamma$ with $(\neg)^n \varphi \wedge \psi$ if $n$ is even and with $(\neg)^{n-1}(\varphi \wedge \top) \wedge \psi$ if $n$ is odd.

$\Gamma$ is now ready, and it is easy to see that if $X_\Lambda$ is a head predicate of $\Lambda$ that appears in $\Gamma$ as $X_\Gamma$, then we have $X_\Lambda^n \equiv X_\Gamma^{n+1}$ for all $n \in \mathbb{N}$. Since $X_\Gamma^0 = \bot$ for each head predicate $X_\Gamma$ of $\Gamma$, the programs $\Lambda$ and $\Gamma$ are therefore equivalent. □

With the preliminaries dealt with, we are now ready to show how we can simulate GMSC with R-simple aggregate-combine GNN[F]s.

**Lemma 3.23.** For each $\Pi$-program of GMSC over $\Pi$, we can construct an equivalent R-simple aggregate-combine GNN[F] for $\Pi$.

*Proof.* Let $\Lambda$ be a $\Pi$-program of GMSC. By Lemma 3.22, we obtain an equivalent balanced $\Pi$-program $\Gamma$, which we proceed to simulate.

We begin by describing the basic specifications of our GNN. $\mathcal{G}_\Gamma$ is a GNN[F] for $(\Pi, N+D+1)$, where $N$ is the number of head predicates and distinct subschemata of the bodies of the iteration clauses of $\Gamma$ and $D$ is the (unique) formula depth of the iteration clauses of $\Gamma$. Intuitively, the first $N$ components of the feature vectors keep track of the truth values of each head predicate and each subschema of the iteration clauses, and the $D+1$ components that follow keep track of the formula depth being evaluated by functioning as a "one-hot clock" where the position of a single 1 among 0s signals the depth of the formula we are evaluating. The goal is thus to simulate a single round of $\Gamma$ with $D+1$ rounds of $\mathcal{G}_\Gamma$.

We will later define the initialization and transition functions such that all feature vectors that can actually be obtained are Boolean, i.e., vectors $\mathbf{v} \in \{0,1\}^{N+D+1}$. Despite this, for the floating-point system of $\mathcal{G}_\Gamma$, we choose a system $S$ that can express all integers from 0 to $K_{\max}$, where $K_{\max}$ is the maximum number $k$ that appears in a diamond $\Diamond_{\geq k}$ in the bodies of the clauses of $\Gamma$ (also called the *modal width* of $\Gamma$). This is because the intermediate computations when simulating diamonds require summing values that can reach up to $K_{\max}$.

We now define the initialization function $\pi$. Assume that all the subschemata and head predicates of $\Gamma$ are enumerated from 1 to $N$ such that if $\varphi_k$ is a subschema of $\varphi_l$, then $k \leq l$. For each $P \subseteq \Pi$, we define the initial feature vector $\pi(P) \in \{0,1\}^{N+D+1}$ such that it records the truth values of the logical atoms and sets up the $D+1$ "one-hot clock" as follows:

1. The component of $\pi(P)$ corresponding to $\top$ is 1.
2. The component of $\pi(P)$ corresponding to $p$ is 1 if and only if $p \in P$.
3. The very last $(N+D+1)$th component of $\pi(P)$ is 1.
4. The other components of $\pi(P)$ are 0.

The aggregation function AGG is simply the sum in increasing order of floats, but the combination function COM is of course more complex. For each $\mathbf{x}, \mathbf{y} \in S^{N+D+1}$, we set

$$\mathrm{COM}(\mathbf{x}, \mathbf{y}) = \mathrm{ReLU}^*(\mathbf{x} \cdot C + \mathbf{y} \cdot A + \mathbf{b}),$$

where $C$, $A$ and $\mathbf{b}$ are defined in the style of [11] as follows. The intuitive idea is to use the matrix $C$ to collect information about the basic logical symbols, the matrix $A$ to collect information about the local neighbourhood and the bias vector $\mathbf{b}$ to adjust the desired output value in the cases where formula depth increases.

First, a few notations. If $k, \ell \leq N+D+1$, we denote the element of $C$ at the $k$th row and $\ell$th column as $C_{k,\ell}$, and respectively for $A$. Similarly, we denote the $\ell$th value of $\mathbf{b}$ as $\mathbf{b}_\ell$. We now define for all $\ell \leq N$:

1. If $\varphi_\ell = \top$ or $\varphi_\ell \in \Pi$, then $C_{\ell,\ell} = 1$.
2. If $\varphi_\ell$ is a head predicate with the iteration clause $X :- \varphi_k$, then $C_{k,\ell} = 1$.
3. If $\varphi_\ell = \varphi_j \wedge \varphi_k$ for some $j, k \leq \ell$, then $C_{j,\ell} = C_{k,\ell} = 1$ and $\mathbf{b}_\ell = 1$.
4. If $\varphi_\ell = \neg\varphi_k$ for some $k \leq \ell$, then $C_{k,\ell} = -1$ and $\mathbf{b}_\ell = 1$.
5. If $\varphi_\ell = \Diamond_{\geq K}\varphi_k$ for some $k \leq \ell$ and $K \leq K_{\max}$, then $A_{k,\ell} = 1$ and $\mathbf{b}_\ell = -K+1$.

We have thus defined the top-left $N \times N$ corner of $C$. For the bottom-right $(D+1) \times (D+1)$ corner of $C$, we define it to be the identity matrix, except we move the rightmost column to be the leftmost column. Every other element of $C$, $A$ and $\mathbf{b}$ is 0.

Finally, we define the set of accepting feature vectors $F$ such that $\mathbf{v} \in F$ if and only if $\mathbf{v}_{N+D+1} = 1$ and $\mathbf{v}_\ell = 1$ for some appointed predicate $\varphi_\ell$. Informally, the first condition states that a full round of $\Gamma$ has been simulated and the second states that an appointed predicate has produced an iteration formula that is true.

The definition of $\mathcal{G}_\Gamma$ is now complete, and we proceed to show that it is equivalent to $\Gamma$. Let $\mathbf{v}(v)_\ell^n$ denote the $\ell$th element of the feature vector of the node $v$ in round $n$. Let $(G, v)$ be a pointed $\Pi$-labeled graph, $\varphi_\ell$ be a schema of formula depth $d$, and $n \in \mathbb{N}$. Our strategy is to first show by induction on $n$ that

$$\mathbf{v}(v)_\ell^{n(D+1)+d} = 1 \text{ if and only if } G, v \models \varphi_\ell^n,$$

noting that since the feature vectors are indeed Boolean, this is equivalent with the parallel claim that $\mathbf{v}(v)_\ell^{n(D+1)+d} = 0$ if and only if $G, v \not\models \varphi_\ell^n$. Then, we will show that for any appointed predicate $X$ of $\Gamma$, we have

$$G, v \models X^n \iff \mathbf{v}(v)^{n(D+1)} \in F.$$

We begin with the induction. For the base case, let $n = 0$. We prove this by another induction, this time over the formula depth $d$ of $\varphi_\ell$. If $d = 0$, there are three possible options for $\varphi_\ell$:

1. Suppose that $\varphi_\ell = \top$. Now, we always have $\mathbf{v}(v)_\ell^0 = 1$ and $G, v \models \top^0$.

2. Suppose that $\varphi_\ell = p$ for some $p \in \Pi$. By the definition of the initialization function $\pi$, we have $\mathbf{v}(v)_\ell^0 = 1 \iff p \in \lambda(v) \iff G, v \models p^0$, where $\lambda$ is the node labeling function of $G$.

3. Suppose that $\varphi_\ell = X$ for some head predicate $X$ of $\Gamma$. By the definition of the initialization function $\pi$, we have $\mathbf{v}(v)_\ell^0 = 0$. Since $\Gamma$ was created using Lemma 3.22, $X$ has the terminal clause $X(0) \coloneq \bot$, and thus $G, v \not\models X^0$.

Now, suppose the claim holds for all schemas of formula depth $d$. If $\varphi_\ell$ has formula depth $d+1$, we again have three possible options for $\varphi_\ell$:

1. Suppose that $\varphi_\ell = \varphi_j \wedge \varphi_k$ for some schemas $\varphi_j$ and $\varphi_k$. By Lemma 3.22, we may assume that $\varphi_j$ and $\varphi_k$ both have the same formula depth $d$. By the definition of $C$ and $\mathbf{b}$, we have $C_{j,\ell} = C_{k,\ell} = 1$ and $\mathbf{b}_\ell = 1$. Note that now $C_{m,\ell} = 0$ for all $m \neq j, k$ and $A_{m,\ell} = 0$ for all $m$. Now, since

   $$\mathbf{v}(v)_\ell^{d+1} = \mathrm{ReLU}^*(\mathbf{v}(v)_j^d + \mathbf{v}(v)_k^d - 1),$$

   we have

   $$\begin{aligned} \mathbf{v}(v)_\ell^{d+1} = 1 &\iff \mathbf{v}(v)_j^d = \mathbf{v}(v)_k^d = 1 \\ &\iff G, v \models \varphi_j^0 \text{ and } G, v \models \varphi_k^0 \\ &\iff G, v \models (\varphi_j \wedge \varphi_k)^0, \end{aligned}$$

   where the second equivalence follows from the induction hypothesis.

2. Suppose that $\varphi_\ell = \neg\varphi_k$ for some schema $\varphi_k$. By the definition of $C$ and $\mathbf{b}$, we have $C_{k,\ell} = -1$ and $\mathbf{b}_\ell = 1$. Again, note that $C_{m,\ell} = 0$ for all $m \neq k$ and $A_{m,\ell} = 0$ for all $m$. Now, since

$$\mathbf{v}(v)^{d+1}_\ell = \mathrm{ReLU}^*(-\mathbf{v}(v)^d_k + 1),$$

we have

$$\begin{aligned}\mathbf{v}(v)^{d+1}_\ell = 1 &\iff \mathbf{v}(v)^d_k = 0\\ &\iff G, v \not\models \varphi^0_k\\ &\iff G, v \models (\neg\varphi_k)^0,\end{aligned}$$

where the second equivalence follows from the induction hypothesis.

3. Suppose that $\varphi_\ell = \Diamond_{\geq K}\varphi_k$ for some schema $\varphi_k$. By the definition of $A$ and $\mathbf{b}$, we have $A_{k,\ell} = 1$ and $\mathbf{b}_\ell = -K + 1$. We also have $A_{m,\ell} = 0$ for all $m \neq k$ and $C_{m,\ell} = 0$ for all $m$. Now,

$$\mathbf{v}(v)^{d+1}_\ell = \mathrm{ReLU}^*(\mathrm{SUM}_S(\{\{\mathbf{v}(u)^d_k \mid u \in N(v)\}\} - K + 1)),$$

where $\mathrm{SUM}_S : \mathcal{M}(S) \to S$ is the sum in increasing order of floats. This means that

$$\begin{aligned}\mathbf{v}(v)^{d+1}_\ell = 1 &\iff |\{u \in N(v) \mid \mathbf{v}(u)^d_k = 1\}| \geq K\\ &\iff |\{u \in N(v) \mid G, v \models \varphi^0_k\}| \geq K\\ &\iff G, v \models (\Diamond_{\geq K}\varphi_k)^0,\end{aligned}$$

where the second equivalence follows from the induction hypothesis.

The base case of the original induction is thus proved. Assume now that the claim holds for $n$ for all formula depths. The cases where $\varphi_\ell$ is a conjunction, negation or a modal diamond are proved as in the base case, so we only consider the following two cases:

1. Suppose that $\varphi_\ell = \top$ or $\varphi_\ell = p$ for some $p \in \Pi$. By the definition of $C$ and $\mathbf{b}$, we have $C_{\ell,\ell} = 1$ and $\mathbf{b}_\ell = 0$. Again, note that $C_{m,\ell} = 0$ for all $m \neq k$ and $A_{m,\ell} = 0$ for all $m$. Now, since

$$\mathbf{v}(v)^{(n+1)(D+1)}_\ell = \mathrm{ReLU}^*(\mathbf{v}(v)^{n(D+1)+D}_\ell),$$

we have $\mathbf{v}(v)^{(n+1)(D+1)} = \mathbf{v}(v)^{n(D+1)+D}_\ell$. By induction, it is also easy to show that $\mathbf{v}(v)^{(n+1)(D+1)}_\ell = \mathbf{v}(v)^{n(D+1)}_\ell$. Hence, we have

$$\begin{aligned}\mathbf{v}(v)^{(n+1)(D+1)} = 1 &\iff \mathbf{v}(v)^{n(D+1)}_\ell = 1\\ &\iff G, v \models \varphi^n_\ell\\ &\iff G, v \models \varphi^{n+1}_\ell,\end{aligned}$$

where the second equivalence follows from the induction hypothesis and the third from the fact that $\varphi^{n+1}_\ell = \varphi^n_\ell$.

2. Suppose that $\varphi_\ell = X$ for some head predicate $X$ of $\Gamma$. By Lemma 3.22, the iteration clause $\varphi_k$ of $X$ has formula depth $D$. By the definition of $C$ and $\mathbf{b}$, we have $C_{k,\ell} = 1$ and $\mathbf{b}_\ell = 0$. Again, note that $C_{m,\ell} = 0$ for all $m \neq k$ and $A_{m,\ell} = 0$ for all $m$. Now, since

$$\mathbf{v}(v)_\ell^{(n+1)(D+1)} = \mathrm{ReLU}^*(\mathbf{v}(v)_k^{n(D+1)+D}),$$

we have

$$\begin{aligned} \mathbf{v}(v)^{(n+1)(D+1)} = 1 &\iff \mathbf{v}(v)_\ell^{n(D+1)+D} = 1 \\ &\iff G, v \models \varphi_k^n \\ &\iff G, v \models X^{n+1}, \end{aligned}$$

where the second equivalence follows from the induction hypothesis.

The induction proof is complete. We now show that the timing works as intended, simulating one round of $\Gamma$ in exactly $D+1$ rounds of of $\mathcal{G}_\Gamma$. Let $n \in \mathbb{N}$ be arbitrary and $\ell \geq 1$ be such that $N+1 \leq \ell < N+D+1$. Now, since $C_{\ell,\ell+1} = 1$, $C_{\ell',\ell+1} = 0$ for all $\ell' \neq \ell$ and $\mathbf{b}_{\ell+1} = 0$, we have

$$\begin{aligned} \mathbf{v}(v)_{\ell+1}^{n+1} &= \mathrm{ReLU}^*(\mathbf{v}(v)_\ell^n) \\ &= \mathbf{v}(v)_\ell^n, \end{aligned}$$

and similarly, since $C_{N+D+1,N+1} = 1$, $C_{\ell',N+1} = 0$ for all $\ell' \neq N+D+1$ and $\mathbf{b}_{N+1} = 0$, we have

$$\begin{aligned} \mathbf{v}(v)_{N+1}^{n+1} &= \mathrm{ReLU}^*(\mathbf{v}(v)_{N+D+1}^n) \\ &= \mathbf{v}(v)_{N+D+1}^n. \end{aligned}$$

Recall that the initialization function $\pi$ assigns $\mathbf{v}_{N+D+1}^0 = 1$ and $\mathbf{v}_\ell^0 = 0$ for all $N+1 \leq \ell < N+D+1$. The equations above thus mean, in plain words, that the 1 is passed "upwards" round by round, first looping from the $(N+D+1)$th element to the $(N+1)$th element, then returning to its original position after $D+1$ rounds in total. Formally, for all $n \in \mathbb{N}$ and $1 \leq \ell, \ell' \leq D+1$, we have

$$\mathbf{v}(v)_{N+\ell}^{n(D+1)+\ell'} = 1 \text{ if and only if } \ell' = \ell$$

and choosing $\ell = D+1$ gives us the special case

$$\mathbf{v}(v)_{N+D+1}^{n(D+1)+\ell'} = 1 \text{ if and only if } \ell' = D+1$$

For our final conclusion, let $n \in \mathbb{N}$ and $\varphi_\ell$ be an appointed predicate $X$ of $\Gamma$. Since $X$ has a formula depth of 0, by the induction we performed earlier, we have $G, v \models X^n$ if and only if $\mathbf{v}(v)_\ell^{n(D+1)} = 1$. In addition, since $\mathbf{v}(v)_{N+D+1}^{n(D+1)} = 1$ always holds, we have $\mathbf{v}(v)^{n(D+1)} \in F$. Thus $\Gamma$ accepts $(G, v)$ if and only if $v$ obtains an accepting feature vector in $\mathcal{G}_\Gamma$, meaning $\Gamma$ and $\mathcal{G}_\Gamma$ are equivalent. Since $\Gamma$ is equivalent to $\Lambda$ by Lemma 3.22, we are finished. □

**Example 3.24.** Suppose we want to simulate with a GNN the following MSC-program $\Lambda$ over $\Pi$, which expresses the *reachability of the proposition* $p \in \Pi$:

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

where $X$ is an appointed predicate. We first balance $\Lambda$ according to Lemma 3.22, creating the following program $\Gamma$:

$$\begin{array}{ll} I(0) :- \bot & \qquad I :- \top \\ X(0) :- \bot & \qquad X :- (\neg I \wedge p) \vee (I \wedge \Diamond X). \end{array}$$

Now, proceeding as in Lemma 3.23, we have $N = 10$ and $D = 3$, and thus our GNN $\mathcal{G}_\Gamma$ is a GNN[F] for $(\Pi, 14)$. The floating-point system $S$ can be very simple, as it merely has to express the integers 0 and $K_{\max} = 1$. We may choose the enumeration of the subschemata and head predicates of $\Gamma$ to be, say, as follows:

1. $\bot$,
2. $\top$,
3. $I$,
4. $\neg I$,
5. $p$,
6. $X$,
7. $\Diamond X$,
8. $\neg I \wedge p$,
9. $I \wedge \Diamond X$,
10. $(\neg I \wedge p) \vee (I \wedge \Diamond X)$.

The initial feature vector of each node with label $P \subseteq \Pi$ is thus

$$\pi(P) = [\overbrace{0,1,0,0,x,0,0,0,0,0}^{N},\overbrace{0,0,0,1}^{D+1}] \in \{0,1\}^{14},$$

where $x = 1$ if and only if $p \in P$. The combination matrix $C \in \{0,1\}^{14\times 14}$ is

$$
\begin{array}{ccc}
 & \overbrace{\hphantom{0\quad 0\quad 0\quad \cdots}}^{N}\hphantom{\quad 0\quad 0\quad 0\quad 0} & \\
\begin{matrix} N\left\{\vphantom{\begin{matrix}0\\0\\0\\\vdots\end{matrix}}\right. \\ \vphantom{\begin{matrix}0\\0\\0\\0\end{matrix}} \end{matrix} &
\begin{bmatrix}
0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & \cdots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & \cdots & 1 & 0 & 0 & 0
\end{bmatrix} &
\begin{matrix} \vphantom{\begin{matrix}0\\0\\0\\\vdots\end{matrix}} \\ \left.\vphantom{\begin{matrix}0\\0\\0\\0\end{matrix}}\right\} D+1 \end{matrix} \\
 & \hphantom{0\quad 0\quad 0\quad \cdots\quad}\underbrace{\hphantom{0\quad 0\quad 0\quad 0}}_{D+1} &
\end{array}
$$

The aggregation matrix $A \in \{0,1\}^{14\times 14}$ contains a single 1 in the cell $A_{6,7}$, with all other values being 0, and the bias vector is

$$\mathbf{b} = [0,0,0,1,0,0,0,1,1,1,0,0,0,0],$$

where the 1s correspond to the 4th, 8th, 9th and 10th subschemata respectively; the 7th subschemata $\Diamond X$ is not noted since it only contains an ordinary modal diamond $\Diamond$.

The set of accepting feature vectors $F$ now contains those vectors $\mathbf{v} \in S^{14} = \{0,1\}^{14}$ that satisfy $\mathbf{v}_{14} = \mathbf{v}_6 = 1$. By straightforward matrix multiplication, it is easy—if somewhat laborious—to see that a node $v$ eventually obtains such a feature vector if and only if the proposition symbol $p$ can be reached from it in a finite number of steps.

### 3.4.2 Simulating GNN[F]s with GMSC

We now prove the converse of Lemma 3.23 by constructing an equivalent GMSC-program for a given GNN[F]. This proof is originally from [50] and was adapted for GML and multisets in [8, Appendix B.2], where bounded *finite counting message-passing automata* (FCMPAs) were converted into equivalent GMSC-programs. Here, we do the conversion from GNN[F]s instead of FCMPAs, but this is a very minor change as their definitions are functionally identical.

**Lemma 3.25.** For each GNN[F] over $\Pi$, we can construct an equivalent GMSC-program over $\Pi$.

*Proof.* Let $\Pi \subseteq \mathrm{LAB}$ and $\mathcal{G} = (S^d, \pi, \delta, F)$ be a GNN[F] over $\Pi$. For each feature vector $\mathbf{x} \in S^d$, we create a head predicate $X_{\mathbf{x}}$. Note that this is possible since $S^d$ is finite. The terminal clause of this predicate is

$$X_{\mathbf{x}}(0) :- \bigvee_{P \subseteq \Pi,\ \pi(P)=\mathbf{x}} \left( \bigwedge_{p \in P} p \wedge \bigwedge_{p \in \Pi \setminus P} \neg p \right).$$

For the iteration clause of $X_{\mathbf{x}}$, we first define some auxiliary formulas. Recall that by Definition 3.12, the aggregation function of $\mathcal{G}$ is bounded, i.e., is of the form $\mathcal{M}_k(S^d) \to S^d$ for some bound $k \in \mathbb{Z}_+$. If $\mathbf{x}, \mathbf{x}' \in S^d$, we now define $M_k(\mathbf{x}, \mathbf{x}') := \{A \in \mathcal{M}_k(S^d) \mid \delta(\mathbf{x}, A) = \mathbf{x}'\}$. Again, note that $M_k(\mathbf{x}, \mathbf{x}')$ is always finite. For each $A \in M_k(\mathbf{x}, \mathbf{x}')$, we then define the schema

$$\varphi_A := \bigwedge_{\mathbf{x} \in S^d,\ A(\mathbf{x})=n,\ 0<n<k} \Diamond_{=n} X_{\mathbf{x}} \wedge \bigwedge_{\mathbf{x} \in S^d,\ A(\mathbf{x})=k} \Diamond_{\geq k} X_{\mathbf{x}},$$

where $\Diamond_{=n} X_{\mathbf{x}}$ is shorthand for $\Diamond_{\geq n} X_{\mathbf{x}} \wedge \neg \Diamond_{\geq n+1} X_{\mathbf{x}}$. Now, the iteration clause of $X_{\mathbf{x}}$ is

$$X_{\mathbf{x}} :- \bigwedge_{\mathbf{x}' \in S^d} \left( X_{\mathbf{x}'} \to \bigvee_{A \in M_k(\mathbf{x}', \mathbf{x})} \varphi_A \right).$$

Finally, we define the appointed predicates to be the set of head predicates $\{X_{\mathbf{x}} \mid \mathbf{x} \in F\}$. To show that $\mathcal{G}$ and the $\Pi$-program defined in such a way are equivalent, we prove by induction on $n \in \mathbb{N}$ that for every pointed $\Pi$-labeled graph $(G, v)$, we have $G, v \models X_{\mathbf{x}}^n$ if and only if the feature vector of $v$ in round $n$ is $\mathbf{x}$.

Let $n = 0$. Directly from the definition of the terminal clause and the initialization function $\pi$, we have

$$
\begin{aligned}
G, v \models X^0_{\mathbf{x}} \iff\; & G, v \models \bigvee_{P \subseteq \Pi,\ \pi(P) = \mathbf{x}} \left( \bigwedge_{p \in P} p \wedge \bigwedge_{p \in \Pi \setminus P} \neg p \right) \\
\iff\; & G, v \models \bigwedge_{p \in P} p \wedge \bigwedge_{p \in \Pi \setminus P} \neg p \text{ and } \pi(P) = \mathbf{x} \text{ for some } P \subseteq \Pi \\
\iff\; & \lambda(v) = P \text{ and } \pi(P) = \mathbf{x} \text{ for some } P \subseteq \Pi \\
\iff\; & \text{the feature vector of } v \text{ in round } 0 \text{ is } \mathbf{x},
\end{aligned}
$$

where $\lambda$ is the node labeling function of $G$ and the second equivalence follows from the fact that the empty disjunction is false.

For the induction case, suppose that the statement holds for some $n \in \mathbb{N}$. First, assume that $G, v \models X^{n+1}_{\mathbf{x}'}$ for some $\mathbf{x}' \in S^d$. This means that

$$
G, v \models \bigwedge_{\mathbf{x} \in S^d} \left( X^n_{\mathbf{x}} \to \bigvee_{A \in M_k(\mathbf{x}, \mathbf{x}')} \varphi^n_A \right),
$$

where $\varphi^n_A$ is $\varphi_A$ with each head predicate $X_{\mathbf{x}}$ replaced by the corresponding iteration formula $X^n_{\mathbf{x}}$. Let $\mathbf{r} \in S^d$ be the feature vector of $v$ in round $n$. Now, by the induction hypothesis, we have $G, v \models X^n_{\mathbf{r}}$, which implies that $G, v \models \bigvee_{A \in M_k(\mathbf{r}, \mathbf{x}')} \varphi_A$, so $G, v \models \varphi^n_A$ for some $A \in \mathcal{M}_k(\mathbf{r}, \mathbf{x}')$. By the definition of $\varphi_A$, each formula $X^n_{\mathbf{x}}$ is satisfied by exactly $A(\mathbf{r}) \leq k$ neighbours of $v$. Again, by the induction hypothesis, this implies that the multiset of feature vectors of the neighbours of $v$ in round $n$ is $A$, and since $A \in \mathcal{M}(\mathbf{r}, \mathbf{x}')$, the feature vector of $v$ in round $n + 1$ is $\mathbf{x}'$.

For the converse direction, assume that the feature vector of $v$ in round $n + 1$ is $\mathbf{x}'$. Again, let $\mathbf{r} \in S^d$ be the feature vector of $v$ in round $n$, and let $A$ be the multiset of feature vectors of the neighbours of $v$ in round $n$. Thus $\mathbf{x}'$ is the output of the transition function $\delta$ with the inputs $\mathbf{r}$ and $A$. Thus we know that the iteration clause of $X_{\mathbf{x}'}$ contains, as a conjunct, the formula

$$
X_{\mathbf{r}} \to \bigvee_{A \in M_k(\mathbf{r}, \mathbf{x}')} \varphi_A.
$$

Now, by the induction hypothesis, we have $G, v \models X^n_{\mathbf{q}}$ precisely when $\mathbf{q} = \mathbf{r}$, and for each $u \in N(v)$ we have $G, u \models X^n_{\mathbf{a}}$ for the corresponding feature vector $\mathbf{a} \in A$ of $u$ in round $n$. Thus $G, v \models X^{n+1}_{\mathbf{x}'}$. □

**Theorem 3.26.** GNN[F]*s, R-simple aggregate-combine* GNN[F]*s and* GMSC *are all equivalent in expressive power.*

*Proof.* The result follows directly from Lemmas 3.23 and 3.25. □

### 3.4.3 Global Readout

An obvious expressive limitation of GNNs is their locality. Since information is passed purely through edges, nodes in separate components of the input graph can never communicate. This is illustrated in Figure 3.2.

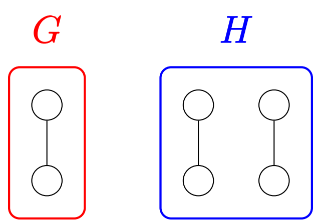



**Figure 3.2.** The graphs $G$ and $H$ are trivially non-isomorphic but nevertheless indistinguishable to a GNN in the sense that the feature vectors of each node are identical in both graphs.

This is an easy problem to fix, as we can simply augment GNNs with *global readout*. In such GNNs, each node also aggregates the feature vectors of all nodes in the graph, and combines them with its own feature vector and those of its neighbours when determining what feature vector to transition into. GNNs with global readout were introduced in [11] and were denoted *aggregate-combine-readout GNNs* or ACR-GNNs.[10] Ahvonen et al. showed in [9], an extended version of [8], that GNN[F]s with global readout correspond to GMSC extended with the *counting global modality*. In this subsection, we present an abridged version of the proof.

**Definition 3.27.** A *GNN[F] with global readout* (GNN[F]+G) is a tuple $\mathcal{G}_G = (S^d, \pi, \delta, F)$, where $S^d$, $\pi$ and $F$ are as in a GNN[F]. The transition function $\delta$ is of the form

$$\delta : S^d \times \mathcal{M}(S^d) \times \mathcal{M}(S^d) \to S^d,$$

and consists of the usual aggregation function AGG, a combination function COM, extended with an additional input, and a *readout function* READ that aggregates the feature vectors of all nodes in the graph. In round $t \geq 1$, each node $v$ computes its feature vector as follows:

$$\begin{aligned}\mathbf{x}(v)^t := &\, \mathrm{COM}(\mathbf{x}(v)^{t-1}, \mathrm{AGG}(\{\{\mathbf{x}(u)^{t-1} \mid u \in N(v)\}\}),\\ &\, \mathrm{READ}(\{\{\mathbf{x}(w)^{t-1} \mid w \in V\}\})).\end{aligned}$$

GNN[F]s that update the feature vector of $v$ by the equation

$$\mathbf{x}(v)^t := \mathrm{ReLU}^*(\mathbf{x}(v)^{t-1} \cdot C + \sum_{u \in N(v)} \mathbf{x}(u)^{t-1} \cdot A + \sum_{w \in V} \mathbf{x}(w)^{t-1} \cdot R + \mathbf{b}),$$

where

1. $\mathrm{ReLU}^* : S^d \to S^d$ is the truncated rectified linear unit,
2. $C, A, R \in S^{d \times d}$ are $d \times d$ matrices, and
3. $\mathbf{b} \in S^d$ is a vector of length $d$,

[10] Global readout via adding a virtual "master" node that is connected to each node in the graph was experimented with earlier in [31], and [12] studied the computation of *global attributes* in a more general *graph network* framework. Barceló et al. were the first to formalize this notion as part of the GNN transition function.

are called *R-simple aggregate-combine-readout* GNN[F]s.

**Definition 3.28.** Let $\Pi \subseteq \text{LAB}$. The set of $\Pi$-formulas of *graded modal logic with the counting global modality* ($\text{GML} + \text{CG}$) is generated by the grammar:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \Diamond_{\geq k}\varphi \mid \langle G\rangle_{\geq k}\varphi,$$

where $p \in \Pi$ and $k \geq 1$. Let $G = (V, E, \lambda)$ be a $\Pi$-labeled graph and $v \in V$. The satisfaction relation for the formula $\langle G\rangle_{\geq k}$ in the pointed graph $(G, v)$ is defined as follows:

$$G, v \models \langle G\rangle_{\geq k}\varphi \iff |\{w \in V \mid G, w \models \varphi\}| \geq k.$$

We define a $\Pi$-*program of GMSC with the counting global modality* ($\text{GMSC} + \text{CG}$) to be a $\Pi$-program of GMSC that allows free use of the counting global modality $\langle G\rangle_{\geq k}$.

Proving that $\text{GNN[F]} + \text{G} \equiv \text{GMSC} + \text{CG}$ holds is achieved by simple modifications of the proofs of Lemmas 3.23 and 3.25, noting that any use of global readout can be simulated with the use of the counting global modality $\langle G\rangle_{\geq k}$ and vice-versa.

**Lemma 3.29.** For each $\Pi$-program of GMSC with the counting global modality, we can construct an equivalent R-simple aggregate-combine-readout GNN[F] for $\Pi$.

*Proof.* Let $\Lambda$ be a $\Pi$-program of GMSC. It is clear that by treating the global diamond the exact same as the ordinary modal diamond, Lemma 3.22 applies also to $\Pi$-programs of GMSC with the counting global modality. Let $\Gamma$ be the $\Pi$-program obtained in such a way. We define the R-simple ACR-GNN[F] in the same way as in the proof of Lemma 3.23, except for the combination function.

For each $\mathbf{x}, \mathbf{y}, \mathbf{z} \in S^{N+D+1}$, we set

$$\text{COM}(\mathbf{x}, \mathbf{y}, \mathbf{z}) := \text{ReLU}^*(\mathbf{x} \cdot C + \mathbf{y} \cdot A + \mathbf{z} \cdot R + \mathbf{b}),$$

where $C$, $A$ and $\mathbf{b}$ are defined as before, with the additional case where if $\varphi_\ell = \langle G\rangle_{\geq K}$ for some $k \leq \ell$ and $K \leq K_{\max}$, then $R_{k,\ell} = 1$ and $\mathbf{b}_\ell = -K + 1$. All other elements of $R$ are 0.

We now prove that $\mathbf{v}(v)_\ell^{n(D+1)+d} = 1$ if and only if $G, v \models \varphi_\ell^n$. The only additional case we need to examine is the one where $n = 0$, the claim holds for the formula $\varphi_k$ of depth $d$, and $\varphi_\ell = \langle G\rangle_{\geq K}\varphi_k$. By the definition of $R$ and $\mathbf{b}$, we have $R_{k,\ell} = 1$ and $\mathbf{b}_\ell = -K + 1$. We also have $R_{m,\ell} = 0$ for all $m \neq k$ and $A_{m,\ell} = C_{m,\ell} = 0$ for all $m$. Now,

$$\mathbf{v}(v)_\ell^{d+1} = \text{ReLU}^*(\text{SUM}_S(\{\{\mathbf{v}(w)_k^d \mid w \in V\}\} - K + 1)),$$

where $\text{SUM}_S : \mathcal{M}(S) \to S$ is the sum in increasing order of floats. This means that

$$\begin{aligned}\mathbf{v}(v)_\ell^{d+1} = 1 &\iff |\{w \in V \mid \mathbf{v}(w)_k^d = 1\}| \geq K \\ &\iff |\{w \in V \mid G, w \models \varphi_k^0\}| \geq K \\ &\iff G, v \models (\langle G\rangle_{\geq K}\varphi_k)^0,\end{aligned}$$

where the second equivalence follows from the induction hypothesis. $\square$

**Lemma 3.30.** For each ACR-GNN[F] over $\Pi$, we can construct an equivalent $\Pi$-program of GMSC with the counting global modality.

*Proof.* Let $\Pi \subseteq \text{LAB}$ and $\mathcal{G} = (S^d, \pi, \delta, F)$ be an ACR-GNN[F] over $\Pi$. As in the proof of Lemma 3.25, we create a head predicate $X_{\mathbf{x}}$ for each feature vector $\mathbf{x} \in S^d$. The terminal clause of $X_{\mathbf{x}}$ is defined as before.

If $\mathbf{x}, \mathbf{x}' \in S^d$, we define $M_k'(\mathbf{x}, \mathbf{x}') := \{(A, R) \in \mathcal{M}_k(S^d) \times \mathcal{M}_k(S^d) \mid \delta(\mathbf{x}, A, R) = \mathbf{x}'\}$. Now, for each $(A, R) \in M_k'(\mathbf{x}, \mathbf{x}')$ we define $\varphi_A$ as before and

$$\psi_R := \bigwedge_{\mathbf{x} \in S^d,\ R(\mathbf{x}) = n,\ 0 < n < k} \langle G \rangle_{=n} X_{\mathbf{x}} \wedge \bigwedge_{\mathbf{x} \in S^d,\ R(\mathbf{x}) = k} \langle G \rangle_{\geq k} X_{\mathbf{x}}$$

where $\Diamond_{=n} X_{\mathbf{x}}$ is shorthand for $\Diamond_{\geq n} X_{\mathbf{x}} \wedge \neg \Diamond_{\geq n+1} X_{\mathbf{x}}$. Now, the iteration clause of $X_{\mathbf{x}}$ is

$$X_{\mathbf{x}} :- \bigwedge_{\mathbf{x}' \in S^d} \left( X_{\mathbf{x}'} \to \bigvee_{(A,R) \in M_k'(\mathbf{x}', \mathbf{x})} (\varphi_A \wedge \psi_R) \right).$$

The induction proof is essentially the same as in Lemma 3.25, so we simply remark that it is easy to show that for every pointed graph $(G, v)$ over $\Pi$, we have $G, v \models X_{\mathbf{x}}^n$ if and only if the feature vector of $v$ in round $n$ is $\mathbf{x}$. $\square$

**Theorem 3.31.** GNN[F]*s with global readout, R-simple aggregate-combine-readout* GNN[F]*s and* GMSC *with the counting global modality are all equivalent in expressive power.*

*Proof.* The result follows directly from Lemmas 3.29 and 3.30. $\square$

Global readout might seem like a trivial addition but it in fact remarkably increases the expressive power of GNNs. For example, *graph transformers* (GTs) have recently been shown to be equivalent to GML with the counting global modality [4].

# 4 Logically Characterizing Random Node Initialization

Since Cybenko's inaugural 1989 work on arbitrary-width feed-forward neural networks [18], various *universal approximation theorems* have been established in machine learning. The aim of these theorems is to prove the *universality* of a given model: the fact that, on a given class of input data, *any* function (that satisfies some conditions, such as continuity or isomorphism-invariance) can be approximated with arbitrary accuracy by the model. Though universality results have been shown for neural networks of eiher bounded width or bounded depth [10] as well as for other architectures, such as *convolutional neural networks* (CNNs) [91], GNNs are known *not* to be universal, since they are bounded in discriminatory power by the well-known 1-WL-test [61, 86].

In this chapter, we discuss ways of raising the expressive power of GNNs past this limit, focusing specifically on *random node initialization* (RNI). We give a new characterization of recurrent GNNs with RNI in the style of [8] by extending GMSC with propositional quantification. We then end the thesis with an investigation of some questions inspired by this novel recursive logic.

## 4.1 Weisfeiler–Leman and rGNNs

A fundamental limitation of ordinary GNNs is the *one-dimensional Weisfeiler–Leman graph isomorphism test.*[11] The 1-WL-test is a simple but powerful heuristic for determining whether two graphs are isomorphic or not, and works by recursively re-colouring nodes based on the colours of their neighbours.

**Definition 4.1.** Let $G = (V, E)$ be a graph. We call a function $\chi^G : V \to \mathbb{N}$ a *colouring* of $G$, interpreting each natural number as a distinct colour. Fixing an arbitrary injective function $\mathrm{HASH} : \mathbb{N} \times \mathcal{M}(\mathbb{N}) \to \mathbb{N}$, we define that the *1-WL-algorithm* generates a sequence of colourings $(\chi_i^G)_{i\in\mathbb{N}}$ of a graph $G = (V, E)$ as follows:

1. In round 0, assign the same colour $\chi_0^G(v) = 0$ to all nodes $v \in V$.[12]

2. In round $i + 1$, assign

$$\chi_{i+1}^G(v) = \mathrm{HASH}(\chi_i^G(v), \{\{\chi_i^G(u) \mid u \in N(v)\}\}),$$

to all nodes $v \in V$.

[11]The algorithm is named after Boris Weisfeiler and Andrey Leman, who described a two-dimensional version of it in a 1968 paper [84]. However, 1-WL, also known as the *colour refinement algorithm* or *naive vertex classification*, goes back to at least 1965 [59].

[12]The algorithm can easily be generalized to node-labeled graphs by assigning the first colour to correspond with the label. Formally, if $\lambda$ is a labeling function that assigns the labels $l_1, \ldots, l_n$ to nodes of $G$, then for all $v \in V$, we set $\chi_0^G(v) = i$ if and only if $\lambda(v) = l_i$.

3. Repeat step 2 until the number of colours used does not change between $\chi_i^G$ and $\chi_{i+1}^G$, i.e., the cardinalities of the images of $\chi_i^G$ and $\chi_{i+1}^G$ are equal. We then call $\chi_i^G$ the *stable colouring* of $G$ and denote it simply $\chi^G$.

If two graphs have non-equivalent stable colourings, i.e, the node partitions induced by the colourings are different, we say that the 1-WL-algorithm *distinguishes* the graphs.

If this algorithm distinguishes two graphs, they are indeed non-isomorphic (cf. Figure 4.1). However, despite the original hopes of Weisfeiler and Leman, it does not constitute a polynomial-time solution to the graph isomorphism problem; though 1-WL distinguishes many non-isomorphic graph pairs, some of them have equivalent stable colourings (cf. Figure 4.2).

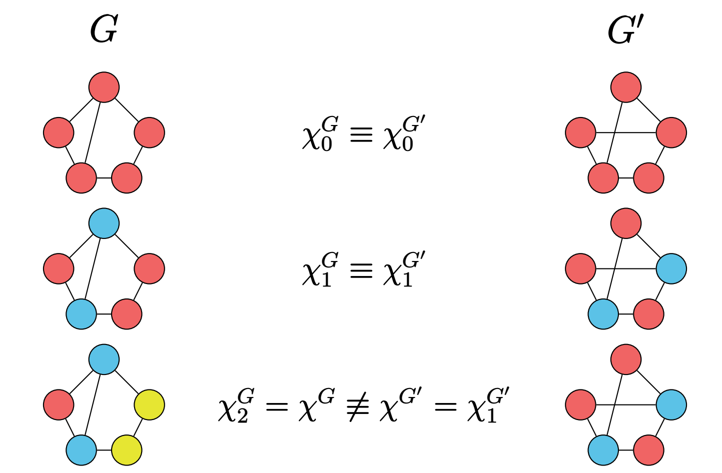



**Figure 4.1.** The 1-WL-algorithm distinguishes these non-isomorphic graphs after two iterations of colour refinement.[13]

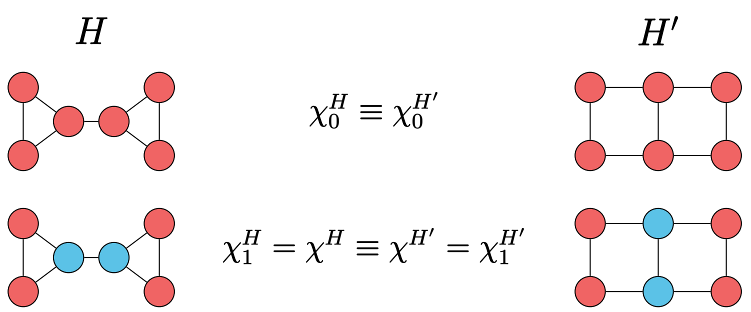



**Figure 4.2.** The 1-WL-algorithm does not distinguish these non-isomorphic graphs. Figure adapted from [20].

[13]To be precise, the node colours in Figures 4.1 and 4.2 are not the actual outputs of the HASH function—otherwise it would not be an injection. Rather, the colours depict the node partitions induced by the colourings.

Recall Definition 3.5 and note how similar this algorithm is to how GNNs refine node feature vectors. Indeed, as shown by Morris et al. [61] and Xu et al. [86], 1-WL and GNNs are constrained by the same upper bound in being able to distinguish non-isomorphic graphs. Thus, if any GNN gives two nodes two different feature vectors in some round, then the 1-WL-test will colour those nodes differently in that round. By ensuring the transition function of the GNN is injective, we can also obtain a converse: for each $t \in \mathbb{N}$, there exists a GNN that has the exact same power to distinguish graphs as the 1-WL-test up to the round $t$.

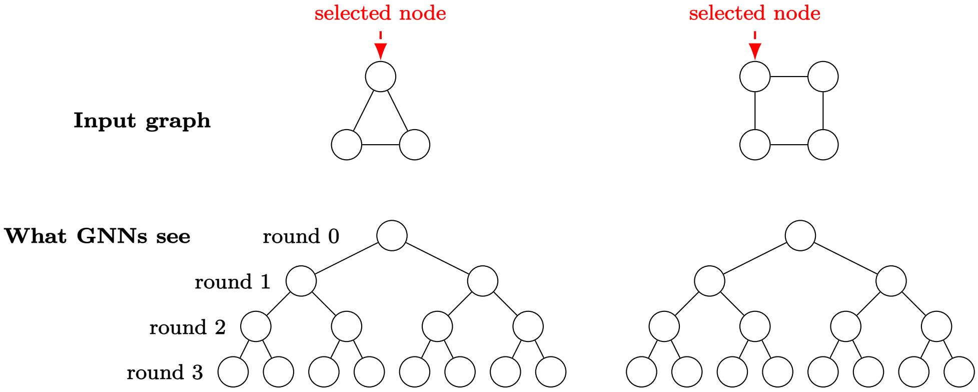
There are ways to augment GNNs such that their discriminatory power rises above the 1-WL-limit, but they often come with the cost of a prohibitive increase in computational complexity. A natural example is *$k$-dimensional GNNs* [61], which are inspired by higher-dimensional versions of the WL-test that colour $k$-tuples of nodes. Such GNNs pass messages between these $k$-tuples and thus have the expressive power of the $k$-WL-test, but their memory consumption scales very poorly as the amount of nodes in the graph grows. In addition, simply adding more dimensions is not enough to guarantee universality; for any $k \geq 1$, there exists a pair of non-isomorphic graphs that are indistinguishable by the $k$-WL-test [17].

However, this curse has been broken by a somewhat recently developed technique: *random node initialization* (RNI). Here, we train and run GNNs that, before the first round of computation, concatenate a randomly sampled number to the initial feature vector of each node, which they then learn to use to separate nodes and thus distinguish larger-scale structures in the graph. This method was first introduced in 2021 by Sato et al. [77], who called such models *rGNNs*.[14] The authors showed that this technique improved approximation ratios for a class of combinatorial problems on graphs and gave GNNs the ability to distinguish some 1-WL-indistinguishable graph pairs.

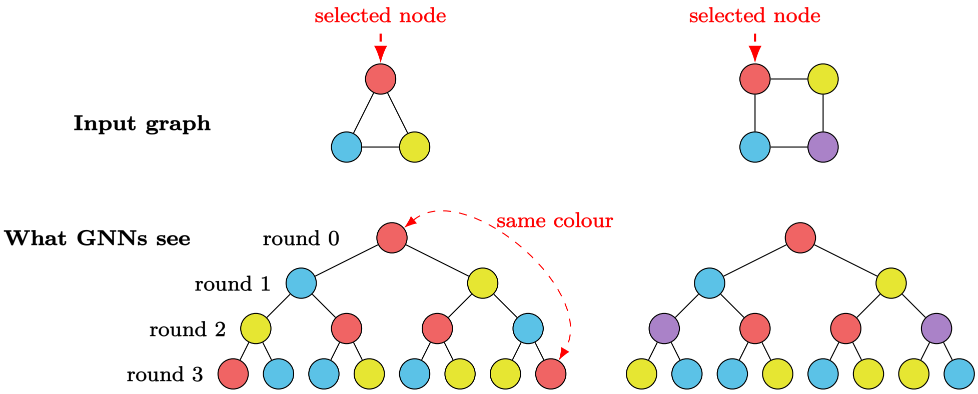
Why is random initialization so powerful? The key point is that since the randomly guessed floats are typically sampled from a uniform distribution over a large set of numbers—typically from all numbers in the interval $[0, 1]$ expressible in the chosen floating-point system—they are, with high probability, *unique* to each node. RNI thus equates to giving us an *individualized colouring* of the graph, where each node has its own unique colour. This allows us to break symmetries and distinguish nodes that, without colouring, are indistinguishable (cf. Figure 4.3).

Soon after the introduction of RNI, its effect on expressive power was studied from a point-of-view of descriptive complexity by Abboud et al. [1]. Utilizing the work of Barceló et al. [11], namely the connection between constant-iteration ACR-GNNs and $\mathrm{FOC}_2$, they showed that constant-iteration rGNNs not only surpass the 1-WL-test but indeed achieve universality—albeit on a

[14] It should be noted that the closely related idea of augmenting GNNs with random colourization schemes to obtain universality was first introduced by Dasoulas et al. [19] in 2020, with a method they call *colored local iterative procedure* (CLIP). The authors also emphasized the importance of *separability* for universality in neural network architectures, which is essentially why RNI works.




(a) Without random initialization.




(b) With random initialization.

**Figure 4.3.** Ordinary GNNs (or the 1-WL-test) cannot distinguish whether a node belongs to a 3-cycle or a 4-cycle, but rGNNs can, by learning to detect if the initial node colour appears in the colours of nodes three edges away. Figure adapted from [77]; see also [55, Appendix D].

very limited class of functions. Specifically, the result is *non-uniform*: [1] proved only that any isomorphism-invariant function $f : \mathbb{G}_n \to \mathbb{R}$ defined on graphs of a specific size $n$ can be approximated by an rGNN, while nothing was said about functions that process graphs of arbitrary size. To make matters worse, the approximation requires the rGNNs to have a size exponential in $n$ in the worst case.

Later, research led by Grohe linked rGNNs to *circuit complexity theory*—classifying problems by the minimum size and depth of Boolean circuits required to solve them—by showing that polynomial-weight constant-iteration rGNNs with global readout are equivalent to the circuit complexity class $\mathrm{TC}^0$, which consists of bounded-depth Boolean circuits with threshold gates [35]. Constant-iteration rGNNs were also studied in [2], where it was shown that they can approximate any graph property that satisfies a *zero-one law*, i.e., that is obtained with a probability approaching 0 or 1 as the size of the graph grows, but are also constrained by a similar limit; for example, the graph property of having an even number of nodes does not satisfy a zero-one law and thus cannot be expressed by an rGNN. Very recently, the first work studying *recurrent* rGNNs with finite-precision parameters showed that, with global readout, they are able to compute any isomorphism-invariant computable function on graphs with only a polynomial time and space overhead [74]. Finally, noting that random node initialization basically means giving each node a *unique identifier*, a related research direction is the study of distributed computing with IDs. For example, a characterization of the recurrent circuit case with MSC was given in [6], which could be combined with the characterization result in [7] to cover a recurrent GNN model with IDs with the local computation based on neural networks.

Of course, the universality results described above merely guarantee the *existence* of an rGNN that approximates a given function, not its practical implementability. However, both Sato et al. [77] and Abboud et al. [1] perform experiments and achieve real-life performance gains when implementing RNI. This technique thus shows promise as a way to boost the power of GNNs in both theory and practice.

## 4.2 rGNN[F]s and SOPGMSC Are Equivalent

In this section, we investigate recurrent floating-point GNNs with random node initialization by extending the work of Ahvonen et al. [8]. Our goal is to show that *second-order propositional* GMSC characterizes such architectures. To the author's knowledge, this is the first characterization of the uniform expressive power of this class of GNNs.

For technical reasons, our definition of RNI is slightly different: instead of appending random floating-point numbers to the initial feature vectors, we instead add random proposition symbols to nodes. Since we are studying floating-point GNNs, this inherent finiteness is not a problem; one could, of course, simulate guessing floating-point numbers by guessing multiple proposition symbols. For example, guessing 64 proposition symbols would be equivalent to assigning each node a random floating-point number from $2^{64}$ different options.[15] The key point is that each node is individualized with high probability.

**Definition 4.2.** Let $\Pi \subseteq \mathrm{LAB}$. A *floating-point graph neural network* rGNN[F] *with random initialization over* $\Pi$ is a triple $(\mathcal{G}, \mathcal{R}, \chi)$, where $\mathcal{G} = (S^d, \pi, \delta, F)$ is a GNN[F] over $\Pi \cup \mathcal{R}$, $\mathcal{R} \subseteq \mathrm{LAB}$ is finite and $\chi$ is a probability distribution over $\mathcal{P}(\mathcal{R})$ *with full support*, meaning that each member of $\mathcal{P}(\mathcal{R})$ has a non-zero probability of being obtained. It computes on a $\Pi$-labeled graph $G = (V, E, \lambda)$ as follows:

[15] In general, the amount of distinct numbers representable in a floating-point system is $2 \cdot \beta^p \cdot (2n+1)$. It is thus sufficient to guess $\lceil \log_2(2 \cdot \beta^p \cdot (2n+1)) \rceil$ proposition symbols to represent any such number.

1. First, the label of each node $v$ is changed from $\lambda(v)$ to $\lambda(v) \cup R_v$, where $R_v \sim \chi$ is sampled independently for each node. Denote this $(\Pi \cup \mathcal{R})$-labeled graph as $G' = (V, E, \lambda')$.

2. Then, $\mathcal{G}$ processes the graph $G'$ as normal.

We say that an rGNN[F] $(\mathcal{G}, \mathcal{R}, \chi)$ *accepts* a pointed $\Pi$-labeled graph $(G, v)$ if the random features $(R_u)_{u \in V}$ can be sampled such that $\mathcal{G}$ accepts $(G', v)$.

As usual in logic, this non-determinism can be expressed in terms of *quantification* over different models; in this case, quantifying over graphs with different valuations for a specific proposition symbol.

**Definition 4.3.** Let $G = (V, E, \lambda)$ be a $\Pi$-labeled graph and $\mathcal{R} \subseteq \mathrm{LAB}$ be finite. A *randomized graph* is a set of $(\Pi \cup \mathcal{R})$-labeled graphs

$$\begin{aligned} G[\mathcal{R}] = \{ &(V, E, \lambda') \mid \\ &\text{for all } v \in V \text{ there is a } R_v \subseteq \mathcal{R} \text{ such that } \lambda'(v) = \lambda(v) \cup R_v \}. \end{aligned}$$

A *program of second-order propositional GMSC* (SOPGMSC) is a pair $(\Lambda, \mathcal{R})$, where $\Lambda = (\mathcal{T}, I, \mathcal{A})$ is a $(\Pi \cup \mathcal{R}, \mathcal{T})$-program of GMSC. We say that $(\Lambda, \mathcal{R})$ *accepts* a pointed $\Pi$-labeled graph $(G, v)$ if $\Lambda$ accepts $(G_r, v)$ for some $G_r \in G[\mathcal{R}]$.

This definition means placing a finite amount of existential propositional quantifiers outside the entire GMSC-program. If $\mathcal{R} = \{p_1, \ldots, p_m\}$, the program $(\Lambda, \mathcal{R})$ can be depicted as follows:

$$\exists p_1 \ldots \exists p_m \begin{pmatrix} X_1(0) :- \varphi_1 & X_1 :- \psi_1 \\ \vdots & \vdots \\ X_n(0) :- \varphi_n & X_n :- \psi_n \end{pmatrix}.$$

**Lemma 4.4.** For each SOPGMSC-program over $\Pi$, we can construct an equivalent rGNN[F] for $\Pi$.

*Proof.* Let $(\Lambda, \mathcal{R})$ be a SOPGMSC-program over $\Pi$. We first turn $\Lambda$ into a GNN[F] $\mathcal{G}_\Lambda$ over $\Pi \cup \mathcal{R}$ by Lemma 3.23. Then, we turn $\mathcal{G}_\Lambda$ into an rGNN[F] $(\mathcal{G}_\Lambda, \mathcal{R}, \chi)$ over $\Pi$, where $\chi$ is a uniform distribution over $\mathcal{P}(\mathcal{R})$.

Let $(G, v)$ be a pointed $\Pi$-labeled graph. Then

$$\begin{aligned} G, v \models (\Lambda, \mathcal{R}) &\iff G_r, v \models \Lambda \text{ for some } G_r \in G[\mathcal{R}] \\ &\iff G_r, v \models \mathcal{G}_\Lambda \text{ for some } G_r \in G[\mathcal{R}] \\ &\iff G, v \models (\mathcal{G}_\Lambda, \mathcal{R}, \chi), \end{aligned}$$

where the second equivalence follows from Lemma 3.23 and the third from the fact that the uniform distribution has full support. $\square$

**Lemma 4.5.** For each rGNN[F] $\mathcal{G}_\chi$ over $\Pi$, we can construct an equivalent SOPGMSC-program over $\Pi$.

*Proof.* Let $(\mathcal{G}, \mathcal{R}, \chi)$ be an rGNN[F] over $\Pi$, where $\mathcal{G} = (S^d, \pi, \delta, f)$. We first turn $\mathcal{G}$ into an equivalent $(\Pi \cup \mathcal{R})$-program of GMSC $\Lambda_{\mathcal{G}}$ by Lemma 3.25. Then, our desired program is the SOPGMSC-program $(\Lambda_{\mathcal{G}}, \mathcal{R})$.

Let $(G, v)$ be a pointed $\Pi$-labeled graph. Then

$$\begin{aligned} G, v \models (\mathcal{G}, \mathcal{R}, \chi) &\iff G_r, v \models \mathcal{G} \text{ for some } G_r \in G[\mathcal{R}] \\ &\iff G_r, v \models \Lambda \text{ for some } G_r \in G[\mathcal{R}] \\ &\iff G, v \models (\Lambda_{\mathcal{G}}, \mathcal{R}), \end{aligned}$$

where the second equivalence follows from Lemma 3.25 and the third from the definition of SOPGMSC-acceptance. □

**Theorem 4.6.** rGNN[F]*s and* SOPGMSC *are equivalent in expressive power.*

*Proof.* The result follows directly from Lemmas 4.4 and 4.5. □

This characterization allows us to immediately extend previous results by Ahvonen et al. [8] to rGNNs. For example, one can add global readout to rGNNs in the exact same way as in Section 3.4.3, resulting in the following corollary.

**Corollary 4.7.** rGNN[F]*s with global readout and* SOPGMSC *with the counting global modality are equivalent in expressive power.*

Extending deterministic distributed algorithms with randomness has previously been studied in the framework of *distributed decision*, which investigates decision problems where the goal is to determine whether or not a system, represented as a graph, satisfies some global property. First, each node must communicate with the other nodes to compute a local answer to the problem. These answers are then aggregated to determine a global answer. Unrestricted randomization leads to these models being able to determine any Turing-decidable graph property in a single round by simply using $\mathcal{O}(n^2)$ bits, where $n$ is the amount of nodes in the graph, to encode the adjacency matrix of the graph into each node [24]. Given that, in the words of Reiter [72, p. 5], distributed decision "aims to develop a counterpart of computational complexity theory for distributed computing", the *amount* of randomization needed is of key interest. For example, adding an amount of random bits that is logarithmic to the amount of nodes in the graph leads to algorithms being able to decide, in constant time, any graph property expressible in *existential monadic second-order logic* (EMSO) [32].

Given that GNNs are a special case of distributed algorithms, one could expect to find similar results regarding the amount of randomized information needed to express certain graph properties. Empirical support for this claim can be found in Abboud et al. [1]. In the paper, the authors tested GNN models with a varying amount of randomization and found that a model that gave randomized node labels to just 12.5% of the nodes of the graph in fact performed better than a model with full randomization. This was explained as the models being able to "propagate" the randomness to the non-randomized nodes, while avoiding unnecessary noise. Another paper investigating this phenomenon is [28], where the convergence speed and robustness with respect to hyperparameter tuning of rGNNs were improved with the "surgical incision" of only a few bits of random information to specific nodes, while retaining the expressive power of the model. We leave further theoretical investigation of this phenomenon for future work.[16]

## 4.3 Further Investigations of SOPGMSC

In this section, we investigate some possible generalizations of SOPGMSC. For simplicity's sake and to focus on the pure interplay between propositional quantification and the substitution calculus, we set ordinary modal logic as our base logic. SOPMSC, meaning ordinary MSC extended with propositional quantification, resembles *second-order propositional modal logic* (SOPML), which is modal logic extended with existential and universal quantification over proposition symbols, i.e., unary relations over the domain [25]. More specifically, SOPMSC is a recursive extension of *existential* SOPML (ESOPML), where quantification is restricted to existential quantifiers placed at the beginning of formulas.

**Definition 4.8.** Let $\Pi \subseteq \mathrm{LAB}$. The set of $\Pi$-formulas of *second-order propositional modal logic* (SOPML) is generated by the grammar

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \Diamond\varphi \mid \exists q\, \varphi \mid \forall q\, \varphi,$$

where $p \in \Pi$ and $q \in \mathrm{LAB}$.

Let $G = (V, E, \lambda)$ be a $\Pi$-labeled graph, $v \in V$ and $p \in \mathrm{LAB}$. By $G[p]$, we denote the set of all possible interpretations of $G$ extended with $p$, that is, the set of $(\Pi \cup \{p\})$-labeled graphs of the form $(V, E, \lambda')$, where $\lambda' : V \to \mathcal{P}(\Pi \cup \{p\})$ such that for each $u \in V$, the label of $u$ is separately chosen to be $\lambda'(u) = \lambda(u)$ or $\lambda'(u) = \lambda(u) \cup \{p\}$. The satisfaction relation $\models$ for the formulas $\exists p\, \varphi$ and $\forall p\, \varphi$ in the pointed graph $(G, v)$ is then defined as follows:

1. $G, v \models \exists p\, \varphi \iff G', v \models \varphi$ for some $G' \in G[p]$.
2. $G, v \models \forall p\, \varphi \iff G', v \models \varphi$ for all $G' \in G[p]$.

We call SOPML restricted to formulas of the form $\exists p_1 \ldots \exists p_n \psi$, where $\psi$ is quantifier-free, *existential second-order propositional modal logic* (ESOPML).

**Example 4.9.** The ESOPML-formula

$$\exists p(p \wedge \Diamond\neg p)$$

[16]Note that the results described in the previous two paragraphs use different frameworks, such as studying only undirected graphs and making other assumptions, so they may not be directly applicable to the definitions used in this thesis; the interested reader is referred to the papers for details.

is true in precisely those nodes that have an out-neighbour which is not the node itself. The similar formula

$$\exists p(\Diamond p \wedge \Diamond\neg p)$$

expresses that a node has at least two distinct out-neighbours.

The semantics for SOPMSC described in the previous section are not the only possible choice. Instead of evaluating the original iteration formulas over a set of graphs with different node labeling functions, we could prepend the existential quantifiers to the start of each iteration formula, thus producing formulas of ESOPML, which are then evaluated on the original graph. For example, the program

$$\exists p\left( X(0) :- p \wedge \Diamond\neg p \qquad\qquad X :- X \wedge \Diamond X \right)$$

could be interpreted to produce the ESOPML-formulas

$$\begin{aligned} X^0 &= \exists p(p \wedge \Diamond\neg p) \\ X^1 &= \exists p(p \wedge \Diamond\neg p \wedge \Diamond(p \wedge \Diamond\neg p)) \\ &\vdots \end{aligned}$$

As it turns out, there is no difference between these two ways of interpreting SOPMSC-programs. We demonstrate this by defining the different semantics as different formulas of infinitary SOPML and showing that they are equivalent. Note that acceptance in the ordinary semantics as defined below exactly corresponds to the notion of acceptance in Definition 3.16.

**Definition 4.10.** We say that a SOPMSC-program $(\Lambda, \{p_1, \ldots, p_n\})$ accepts a pointed $\Pi$-labeled graph $(G, v)$ *in the ordinary semantics* if

$$G, v \models \exists p_1 \ldots \exists p_n \bigvee_{k\in\mathbb{N},\, X\in\mathcal{A}} X^k$$

and that it accepts $(G, v)$ *in the alternative semantics* if

$$G, v \models \bigvee_{k\in\mathbb{N},\, X\in\mathcal{A}} \exists p_1 \ldots \exists p_n X^k,$$

where $\mathcal{A}$ is the set of appointed predicates of $\Lambda$.

**Theorem 4.11.** *The ordinary semantics and the alternative semantics for* SOPMSC *are equivalent.*

*Proof.* By a straightforward unpacking of the definitions, we have

$$\begin{aligned} & G, v \models \exists p_1 \ldots \exists p_n \bigvee_{k\in\mathbb{N},\, X\in\mathcal{A}} X^k \\ \iff\ & G_r, v \models \bigvee_{k\in\mathbb{N},\, X\in\mathcal{A}} X^k \text{ for some } G_r \in G[\{p_1, \ldots, p_n\}] \\ \iff\ & G_r, v \models X^k \text{ for some } G_r \in G[\{p_1, \ldots, p_n\}],\ k \in \mathbb{N} \text{ and } X \in \mathcal{A} \\ \iff\ & G, v \models \exists p_1 \ldots \exists p_n X^k \text{ for some } k \in \mathbb{N} \text{ and } X \in \mathcal{A} \\ \iff\ & G, v \models \bigvee_{k\in\mathbb{N},\, X\in\mathcal{A}} \exists p_1 \ldots \exists p_n X^k. \end{aligned}$$

□

So far, we have placed the existential quantifiers outside the MSC-program. What if we put the quantifiers *inside* the program, that is, allow the MSC-program to use formulas of ESOPML as clause bodies?

**Definition 4.12.** We call SOPMSC as we have used it so far SOPMSC *with outer quantification*, and we call MSC-programs that allow the use of ESOPML-formulas only in the bodies of their terminal clauses SOPMSC *with inner quantification.*

This is an interesting variation, as it places all non-determinism inside the terminal clause. In other words, SOPMSC with inner quantification is not really a randomized logic; each quantifier block is evaluated completely locally, and it is perhaps better to think of it as simply MSC with "boosted" local expressivity.

What is the expressive power of this logic, and how does it relate to SOPMSC with outer quantification? This has turned out to be a quite difficult question to give a precise answer to. We begin with a relatively trivial lower bound.

**Theorem 4.13.** SOPMSC *with inner quantification is at least as expressive as* ESOPML *closed under negation, conjunction and the modal diamond.*

*Proof.* We first prove that for any formula $\varphi$ of ESOPML closed under negation, conjunction and the modal diamond, we can construct a SOPMSC-program with inner quantification with a head predicate that produces $\varphi$ as an iteration formula in some round. We proceed by induction on the formula structure. The case where $\varphi$ is a formula of ESOPML is simple, since the head predicate

$$X(0) :- \varphi \qquad\qquad X :- \bot$$

produces the iteration formula $X^0 = \varphi$. Suppose then that $\varphi$ is of the form $*_1 \cdots *_k\, \psi$ for some $k \geq 1$, where $*_i \in \{\neg, \Diamond\}$ for all $1 \leq i \leq k$ and $\psi$ is a formula of ESOPML. Then the head predicate

$$X(0) :- \psi \qquad\qquad X :- *_1 \cdots *_k\, X$$

produces the iteration formula $X^1 = *_1 \cdots *_k\, \psi$.

Assume now that the claim holds for $\varphi_1$ and $\varphi_2$. Let $X_1$ and $X_2$ be the head predicates that produce the iteration formulas $X_1^n = \varphi_1$ and $X_2^m = \varphi_2$ in rounds $n, m \in \mathbb{N}$. Suppose $n = m$. Then the head predicate

$$X(0) :- \bot \qquad\qquad X :- X_1 \wedge X_2$$

produces the iteration formula $X^{n+1} = \varphi_1 \wedge \varphi_2$. Suppose then, without loss of generality, that $n < m$. We now create a chain of fresh predicates $Y_1, \ldots, Y_{m-n}$ with the rules

$$Y_1(0) :- \bot \qquad\qquad Y_1 :- X_1$$

and

$$Y_i(0) :- \bot \qquad\qquad Y_i :- Y_{i-1}$$

for all $2 \leq i \leq m - n$. Now, the head predicate

$$X(0) :- \bot \qquad\qquad X :- Y_{m-n}$$

produces the iteration formula $X^{m+1} = \varphi_1 \wedge \varphi_2$. The induction is thus finished.

It is now easy to see that with the addition of a clock predicate, we can construct a head predicate that produces $\varphi$ in some round and a formula equivalent with $\bot$ in all other rounds. Setting this as our sole appointed predicate gives us a SOPMSC-program with inner quantification equivalent with $\varphi$. $\square$

Interestingly, since $\neg\exists p\, \varphi \equiv \forall p\, \neg\varphi$ for all SOPML-formulas $\varphi$, this result implies that SOPMSC with inner quantification contains *universal* SOPML, where quantification is restricted to universal quantifiers placed at the beginning of formulas.

Comparing the expressive power of inner and outer quantification directly is difficult since they are both very strong, easily being able to express NP-complete properties. However, utilizing the previous observation, it can at least be shown that the former is not a subset of the latter.

**Theorem 4.14.** *There is a node property expressible in* SOPMSC *with inner quantification that is not expressible in* SOPMSC *with outer quantification.*

*Proof.* Consider the node property of having at most one neighbour. First, it is easy to see that the formula $\forall p(\Box p \vee \Box\neg p)$, which is a formula of universal SOPML and can thus be produced by SOPMSC with inner quantification, expresses this property.

Assume then, for the sake of contradiction, that an ESOPML-formula $\varphi$ expresses this property, meaning it is true in $(G, v)$, where $v$ has one neighbour $u$. Then there is some valuation of the proposition symbols quantified in $\varphi$ for $u$ such that $G, v \models \varphi$. But in the graph $G'$ where $v$ also has another neighbour $u'$ that has that exact same valuation, $\varphi$ cannot distinguish between $u$ and $u'$, meaning $G', v \models \varphi$. Thus this property cannot be expressed by any ESOPML-formula. Therefore it is easy to show that no SOPMSC-program with outer quantification can express it either. $\square$

What about the converse? Is outer quantification a strict subset of inner quantification or are they orthogonal in expressive power? We believe the latter is the more likely option, but to conserve time and space, we leave a detailed proof for future work and put forward this claim as a conjecture for now.

**Conjecture 4.15.** *There is a node property expressible in* SOPMSC *with outer quantification that is not expressible in* SOPMSC *with inner quantification.*

The property in question would be some property—such as 3-colorability—that must hold, up to a depth of $n$, where $n$ varies in the input class of graphs and must be deduced in each case from the graph structure. The idea is to choose the property such that the fact that outer quantification can fix a valuation for the proposition symbols *globally* for the whole graph while inner quantification is forced to *locally* guess them separately in each node proves to be the differentiating factor. The point of $n$ depending on the graph is to ensure that inner quantification cannot "cheat" by simply using a single (finitary) formula of existential or universal SOPML to check the property up to some fixed depth. This idea is illustrated in Figure 4.4.

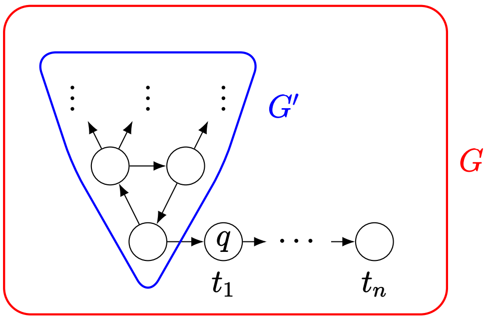



**Figure 4.4.** We conjecture that, given the entire graph $G$ as input, up to depth $n$, the subgraph $G'$ having a certain property not immediately deducible from the length of the "tail", consisting of $n$ nodes originating from the root and distinguishable with a helper proposition symbol $q$, is expressible with outer quantification but not with inner quantification.

A key direction for future research would involve studying the expressive power of random node initialization by establishing similar results for SOPGMSC, or in other words, investigating the interplay of propositional quantification and graded modal logic. Another, more hypothetical investigation would involve a variant of inner quantification where the quantifiers could be placed inside the iteration clauses, thus allowing infinite randomization and blowing up the expressive power of the logic even further.

# Bibliography

[1] Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The Surprising Power of Graph Neural Networks with Random Node Initialization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021*, pages 2112–2118, 2021.

[2] Sam Adam-Day, Theodor-Mihai Iliant, and İsmail İlkan Ceylan. Zero-one laws of graph neural networks. *Advances in Neural Information Processing Systems*, 36:70733–70756, 2023.

[3] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques & Tools*. Pearson Education, 2007.

[4] Veeti Ahvonen, Maurice Funk, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Expressive Power of Graph Transformers via Logic. *arXiv preprint arXiv:2508.01067*, 2025.

[5] Veeti Ahvonen, Damian Heiman, Lauri Hella, and Antti Kuusisto. Descriptive Complexity for Distributed Computing with Circuits. In *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, volume 272 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:15, 2023. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.MFCS.2023.9.

[6] Veeti Ahvonen, Damian Heiman, Lauri Hella, and Antti Kuusisto. Descriptive complexity for distributed computing with circuits. *Journal of Logic and Computation*, page exae087, 2025.

[7] Veeti Ahvonen, Damian Heiman, and Antti Kuusisto. Descriptive Complexity for Neural Networks via Boolean Networks. In *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:22, 2024. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CSL.2024.9.

[8] Veeti Ahvonen, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Logical characterizations of recurrent graph neural networks with reals and floats. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL: https://openreview.net/forum?id=atDcnWqG5n.

[9] Veeti Ahvonen, Damian Heiman, Antti Kuusisto, and Carsten Lutz. Logical Characterizations of Recurrent Graph Neural Networks with Reals and Floats. *arXiv preprint arXiv:2405.14606*, 2025.

[10] Midhun T. Augustine. A survey on universal approximation theorems. *arXiv preprint arXiv:2407.12895*, 2024.

[11] Pablo Barceló, Egor V. Kostylev, Mikael Monët, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*, 2020.

[12] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[13] Maya Bechler-Speicher, Ben Finkelshtein, Fabrizio Frasca, Luis Müller, Jan Tönshoff, Antoine Siraudin, Viktor Zaverkin, Michael M. Bronstein, Mathias Niepert, Bryan Perozzi, Mikhail Galkin, and Christopher Morris. Position: Graph learning will lose relevance due to poor benchmarks. *arXiv preprint arXiv:2502.14546*, 2025.

[14] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[15] Jeroen Bollen, Jan Van den Bussche, Stijn Vansummeren, and Jonni Virtema. Halting Recurrent GNNs and the Graded $\mu$-Calculus. *arXiv preprint arXiv:2505.11050*, 2025.

[16] Julius Büchi. Weak second-order arithmetic and finite automata. *MLQ Math. Log. Q.*, 6(1-6):66–92, 1960.

[17] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[18] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[19] George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. Coloring graph neural networks for node disambiguation. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, 2021.

[20] Mohammed Haroon Dupty and Wee Sun Lee. Graph representation learning with individualization and refinement. *arXiv preprint arXiv:2203.09141*, 2022.

[21] Calvin Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Am. Math. Soc.*, 98(1):21–51, 1961.

[22] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A Fair Comparison of Graph Neural Networks for Graph Classification. In *International Conference on Learning Representations*, 2020. URL: https://openreview.net/forum?id=HygDF6NFPB.

[23] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation*, 7:43–73, 1974.

[24] Laurent Feuilloley and Pierre Fraigniaud. Survey of distributed decision. *Bulletin of EATCS*, 1(119), 2016.

[25] Kit Fine. Propositional quantifiers in modal logic. *Theoria*, 36(3), 1970.

[26] Bettina Finzel, Anna Saranti, Alessa Angerschmid, David Tafler, Bastian Pfeifer, and Andreas Holzinger. Generating explanations for conceptual validation of graph neural networks: An investigation of symbolic predicates learned on relevance-ranked sub-graphs. *KI-Künstliche Intelligenz*, 36(3):271–285, 2022.

[27] Michael J. Fischer and Michael Merritt. Appraising two decades of distributed computing theory research. *Distributed Computing*, 16(2):239–247, 2003.

[28] Billy Joe Franks, Markus Anders, Marius Kloft, and Pascal Schweitzer. Trainability for Universal GNNs Through Surgical Randomness. *CoRR*, 2021.

[29] Paolo Frasconi, Marco Gori, and Alessandro Sperduti. A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks*, 9(5):768–786, 1998.

[30] Daniel M. Gerstein and Erin N. Leidy. *Emerging Technology and Risk Analysis: Artificial Intelligence and Critical Infrastructure.* RAND Corporation, Santa Monica, CA, 2024.

[31] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[32] Mika Göös and Jukka Suomela. Locally Checkable Proofs in Distributed Computing. *Theory of Computing*, 12(19):1–33, 2016. URL: https://theoryofcomputing.org/articles/v012a019.

[33] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE international joint conference on neural networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[34] Bernardo Cuenca Grau and Przemysław A. Wałęga. The Correspondence Between Bounded Graph Neural Networks and Fragments of First-Order Logic. *arXiv preprint arXiv:2505.08021*, 2025.

[35] Martin Grohe. The descriptive complexity of graph neural networks. *TheoretiCS*, 3:1–93, 2024.

[36] Stan P. Hauke and Przemysław Andrzej Wałęga. Aggregate-Combine-Readout GNNs Are More Expressive Than Logic C2. *arXiv preprint arXiv:2508.06091*, 2025.

[37] Yang-Hui He. Machine-learning mathematical structures. *International Journal of Data Science in the Mathematical Sciences*, 1(01):23–47, 2023.

[38] Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. In *Proceedings of the 2012 ACM Symposium on Principles of distributed computing*, pages 185–194, 2012.

[39] Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 2001.

[40] Moritz Herrmann, F. Julian D. Lange, Katharina Eggensperger, Giuseppe Casalicchio, Marcel Wever, Matthias Feurer, David Rügamer, Eyke Hüllermeier, Anne-Laure Boulesteix, and Bernd Bischl. Position: Why We Must Rethink Empirical Research in Machine Learning. In *Forty-first International Conference on Machine Learning*, 2024. URL: https://openreview.net/forum?id=DprrMz24tk.

[41] Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 1998.

[42] Reijo Jaakkola, Tomi Janhunen, Antti Kuusisto, Magdalena Ortiz, Matias Selin, and Mantas Šimkus. Graph Learning via Logic-Based Weisfeiler-Leman Variants and Tabularization. *arXiv preprint arXiv:2508.10651*, 2025.

[43] Fei Jiang, Yong Jiang, Hui Zhi, Yi Dong, Hao Li, Sufeng Ma, Yilong Wang, Qiang Dong, Haipeng Shen, and Yongjun Wang. Artificial intelligence in healthcare: past, present and future. *Stroke and vascular neurology*, 2(4), 2017.

[44] Tae Wan Kim and Bryan R. Routledge. Why a right to an explanation of algorithmic decision-making should exist: A trust-based approach. *Business Ethics Quarterly*, 32(1):75–102, 2022.

[45] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*, 2017. URL: https://openreview.net/forum?id=SJU4ayYgl.

[46] Nils M. Kriege, Matthias Fey, Denis Fisseler, Petra Mutzel, and Frank Weichert. Recognizing Cuneiform Signs Using Graph Based Methods. In Luís Torgo, Stan Matwin, Gary Weiss, Nuno Moniz, and Paula Branco, editors, *Proceedings of The International Workshop on Cost-Sensitive Learning*, volume 88 of *Proceedings of Machine Learning Research*, pages 31–44. PMLR, 05 May 2018. URL: https://proceedings.mlr.press/v88/kriege18a.html.

[47] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5:1–42, 2020.

[48] Andrej Kristofik. Bias in AI (Supported) Decision Making: Old Problems, New Technologies. In *International Journal of Court Administration*, volume 16, page 1. HeinOnline, 2025.

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[50] Antti Kuusisto. Modal logic and distributed message passing automata. In *Computer Science Logic 2013 (CSL 2013)*, pages 452–468. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2013.

[51] Antti Kuusisto. Infinite Networks, Halting and Local Algorithms. In *GandALF*, pages 147–160, 2014. URL: https://doi.org/10.4204/EPTCS.161.14.

[52] Antti Kuusisto and Fabian Reiter. Emptiness problems for distributed automata. *Information and Computation*, 272:104503, 2020.

[53] Jintang Li. ICLR 2023 OpenReview Data, 2022. Accessed on April 19th, 2025. URL: https://github.com/EdisonLeeeee/ICLR2023-OpenReviewData.

[54] Helena Löfström, Karl Hammar, and Ulf Johansson. A meta survey of quality evaluation criteria in explanation methods. In *International Conference on Advanced Information Systems Engineering*, pages 55–63. Springer, 2022.

[55] Andreas Loukas. What graph neural networks cannot learn: Depth vs width. In *8th International Conference on Learning Representations, ICLR 2020*, 2020.

[56] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. Are we really making much progress? Revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM*

*SIGKDD conference on knowledge discovery & data mining*, pages 1150–1160, 2021.

[57] Joao Marques-Silva. Logic-based explainability in machine learning. In *Reasoning Web. Causality, Explanations and Declarative Knowledge: 18th International Summer School 2022, Berlin, Germany, September 27–30, 2022, Tutorial Lectures*, pages 24–104. Springer, 2023.

[58] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[59] Harry L. Morgan. The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of chemical documentation*, 5(2):107–113, 1965.

[60] Christopher Morris, Fabrizio Frasca, Nadav Dym, Haggai Maron, İsmail İlkan Ceylan, Ron Levie, Derek Lim, Michael M. Bronstein, Martin Grohe, and Stefanie Jegelka. Position: Future Directions in the Theory of Graph Machine Learning. In *Forty-first International Conference on Machine Learning, ICML*, 2024. URL: https://openreview.net/forum?id=wBr5ozDEKp.

[61] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

[62] Shayan Mousavi. NeurIPS 2024 AI Research Trends, 2024. Accessed on April 19th, 2025. URL: https://shmouses.github.io/NeurIPS2024AITrends/.

[63] Nina Narodytska, Aditya Shrotri, Kuldeep S. Meel, Alexey Ignatiev, and Joao Marques-Silva. Assessing heuristic machine learning explanations with model counting. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 267–278. Springer, 2019.

[64] Pierre Nunn, Marco Sälzer, François Schwarzentruber, and Nicolas Troquard. A logic for reasoning about aggregate-combine graph neural networks. *arXiv preprint arXiv:2405.00205*, 2024.

[65] Sebastian Palacio, Adriano Lucieri, Mohsin Munir, Sheraz Ahmed, Jörn Hees, and Andreas Dengel. XAI Handbook: Towards a Unified Framework for Explainable AI. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3766–3775, 2021.

[66] Maximilian Pflueger, David Tena Cucala, and Egor V. Kostylev. Recurrent graph neural networks and their connections to bisimulation and logic. In

*Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 14608–14616, 2024.

[67] Alexander Pluska, Pascal Welke, Thomas Gärtner, and Sagar Malhotra. Logical distillation of graph neural networks. In *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning*, pages 920–930, 2024.

[68] Simon J.D. Prince. *Understanding Deep Learning.* The MIT Press, 2023. URL: http://udlbook.com.

[69] Tim Räz and Claus Beisbart. The Importance of Understanding Deep Learning. *Erkenntnis*, 89(5), 2024.

[70] Fabian Reiter. Distributed graph automata. In *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 192–201. IEEE, 2015.

[71] Fabian Reiter. Asynchronous Distributed Automata: A Characterization of the Modal Mu-Fragment. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 100:1–100:14, 2017. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2017.100.

[72] Fabian Reiter. *Distributed automata and logic.* PhD thesis, Université Sorbonne Paris Cité, 2017. URL: https://theses.hal.science/tel-01953461.

[73] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[74] Eran Rosenbluth and Martin Grohe. Repetition Makes Perfect: Recurrent Sum-GNNs Match Message Passing Limit. *arXiv preprint arXiv:2505.00291*, 2025.

[75] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings. In *International Conference on Learning Representations*, 2020. URL: https://openreview.net/forum?id=BkxSmlBFvr.

[76] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems. *Advances in Neural Information Processing Systems*, 32, 2019.

[77] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random Features Strengthen Graph Neural Networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining, SDM 2021*, pages 333–341. SIAM, 2021. URL: https://doi.org/10.1137/1.9781611976700.38.

[78] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Toward verified artificial intelligence. *Communications of the ACM*, 65(7):46–55, 2022.

[79] Claude E. Shannon. A symbolic analysis of relay and switching circuits. *Electrical Engineering*, 57(12):713–723, 1938.

[80] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE transactions on neural networks*, 8(3):714–735, 1997.

[81] Jan Tönshoff, Martin Ritzert, Eran Rosenbluth, and Martin Grohe. Where Did the Gap Go? Reassessing the Long-Range Graph Benchmark. *arXiv preprint arXiv:2309.00367*, 2023.

[82] Boris Trakhtenbrot. Finite automata and the logic of single-place predicates. In *Doklady Akademii Nauk*, volume 140, pages 326–329. Russian Academy of Sciences, 1961.

[83] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

[84] Boris Weisfeiler and Andrey Leman. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series*, 2(9):12–16, 1968.

[85] Bradford Wiggins and Cody Christopherson. The replication crisis in psychology: An overview for theoretical and philosophical psychology. *Journal of Theoretical and Philosophical Psychology*, 39(4):202, 2019.

[86] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*, 2019. URL: https://openreview.net/forum?id=ryGs6iA5Km.

[87] Pinar Yanardag and S. V. N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015.

[88] Douglas Yeung, Inez Khan, Nidhi Kalra, and Osonde Osoba. Identifying Systemic Bias in the Acquisition of Machine Learning Decision Aids for Law Enforcement Applications. Technical report, RAND Corporation, 2021. URL: http://www.jstor.org/stable/resrep29576.

[89] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017. URL: https://openreview.net/forum?id=Sy8gdB9xx.

[90] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[91] Ding-Xuan Zhou. Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2):787–794, 2020.