Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
KR meets Machine Learning and Explanation

Recurrent Graph Neural Networks and Arithmetic Circuits

Timon Barlag[1] , **Vivian Holzapfel**[1] , **Laura Strieker**[1]
Jonni Virtema[2] , **Heribert Vollmer**[1]
[1]Institut für Theoretische Informatik, Leibniz Universität Hannover, Germany
[2]School of Computing Science, University of Glasgow, UK
{barlag, holzapfel, strieker, vollmer}@thi.uni-hannover.de,
jonni.virtema@glasgow.ac.uk

Abstract

We characterise the computational power of recurrent graph neural networks (GNNs) in terms of arithmetic circuits over the real numbers. Our networks are not restricted to aggregate-combine GNNs or other particular types. Generalising similar notions from the literature, we introduce the model of recurrent arithmetic circuits, which can be seen as arithmetic analogues of sequential or logical circuits. These circuits utilise so-called *memory gates* which are used to store data between iterations of the recurrent circuit. While (recurrent) GNNs work on labelled graphs, we construct arithmetic circuits that obtain encoded labelled graphs as real valued tuples and then compute the same function. For the other direction we construct recurrent GNNs which are able to simulate the computations of recurrent circuits. These GNNs are given the circuit-input as initial feature vectors and then, after the GNN-computation, have the circuit-output among the feature vectors of its nodes. In this way we establish an exact correspondence between the expressivity of recurrent GNNs and recurrent arithmetic circuits operating over real numbers. Our results both deepen our understanding of the capabilities of trained neural networks and open new approaches to study recurrent neural networks using the lens of circuit complexity theory.

1 Introduction

Graph neural networks (GNNs) have become a widely-used and popular machine learning model on graph inputs. Graph nodes are labelled with numerical feature values, which are updated in a sequence of steps (so called layers) depending on the feature values of the neighbours. Graph neural networks have been studied intensively from a theoretical perspective, in particular regarding their expressive (or: computational) power. In this context, the training process, i. e. the question how a GNN can be obtained from its training data is ignored, but its intrinsic power as an optimally trained network is considered. Going back to a seminal paper by Barceló et al. (2020) the expressive power of GNNs was closely related to unary formulas of a certain counting variant of first-order predicate logic that classify nodes in a given graph. Grohe (2024) extended these results and related GNNs to the Boolean circuit class TC^0 of families of polynomial size and constant depth threshold circuits.

A further step in understanding the capabilities of GNNs, that is very important for the present paper, was done by Barlag et al. (2024a). First, since GNNs intrinsically compute functions over graphs labelled with feature vectors of real numbers, they were not compared to Boolean logic (like Barceló) or Boolean circuits (like Grohe), but to circuits computing over real numbers. Second, this paper considered a general form of GNNs where the message passing from one layer to the next was not restricted to so-called AC-GNNs. These are GNNs where the computation in each node of a layer of the network consists first of an aggregation step (collecting information from the adjacent nodes of the graph—very often simply summation), followed by a combination step (combining the aggregated information with the current information of the node—mostly a linear function followed by a unary non-linear activation function such as sigmoid or ReLU). Barlag et al. retained the message-passing architecture of GNNs, but allowed any computation that can be performed by constant-depth arithmetic circuits to be used; their model was called circuit-GNN (C-GNN). In this way, a close analogy between GNNs (with a constant number of layers) and arithmetic circuits of polynomial size and constant depth was obtained.

In 2024, Pflueger, Cucala, and Kostylev (2024) introduced recurrent GNNs which are allowed to perform more than a fixed number of message-passing iterations. They considered several variants for termination, one by performing iterations until a form of stabilisation was achieved, and another in which the number of iterations can depend on the graph size. They then proved that both variants are more powerful than GNNs with constant number of layers, and related them to a fragment of monadic monotone fixpoint logic. Further halting conditions were studied (Ahvonen et al. 2024; Bollen et al. 2025) and related to modal and predicate logics.

Our contributions. Our goal is to exactly characterise the computational power of recurrent GNNs as a model of computation over real numbers or labelled graphs, by relating it to other well-studied computational models in theoretical computer science. In the above mentioned works, recurrent GNNs have been compared with Boolean circuits, and their expressive power has been compared with certain logical languages. Since GNNs compute over real numbers, significant effort in these results is placed into issues of coding or approximating real numbers, and no exact correspondence between a Boolean model and GNNs has been proven (with the exception of some special cases for Boolean node classifiers). We

aim at relating recurrent GNNs to well-studied computation models operating over the reals. More concretely, we follow the above mentioned approach by Barlag et al. (2024a) and study the power of recurrent GNNs by comparing them to arithmetic circuits.

Interestingly a somewhat surprising extension of usual circuits turns out to be important: In the area of digital systems, one distinguishes between combinational circuits, i. e., regular circuits with gates and connecting wires, but without any form of memory, and sequential circuits (sometimes called logic circuits) with memory gates. These consist of combinational circuits plus memory states and feedback edges that allow memory states to be used in the next iteration of the circuit computation (Lewin and Protheroe 1992). Sequential circuits are used to implement finite automata or Moore and Mealy machines (Wagner 2003). Here, we use the well-studied model of arithmetic circuits (Vollmer 1999) (as previously also done by Barlag et al. in the context of non-recurrent GNNs) but now extend it with memory gates and backward edges, and consider the computation of the circuit in iterations. We define halting conditions that for their part can be computed by arithmetic circuits depending on values of certain so-called *halting gates*. We call these circuits *recurrent arithmetic circuits*. We define *recurrent graph neural networks* to iterate a periodic sequence of GNN layers. Halting of the computation is determined by a halting function, depending on all feature vectors of the current layer.

Our main result is a comparison of the computational power of recurrent GNNs and recurrent arithmetic circuits. We do not restrict our attention to AC-GNNs, but as in Barlag et al. we allow the message passing to be computed by arithmetic circuits, which then can be recurrent themselves. Hence we distinguish inner recurrence (where the circuits computing the messages passed are recursive), outer recurrence (the recursion by iterating GNN layers), and their combination.

We show that any recurrent C-GNN can be simulated by a recurrent arithmetic circuit. The circuit obtains an encoding of the input graph of the C-GNN as an input and computes an encoding of the output graph of the GNN. Conversely, we show that any recurrent arithmetic circuit can be simulated by a recurrent C-GNN. The GNN obtains (an encoding of) the circuit's input as feature values and computes the output of the circuit among the feature values in its terminating layer. Both results hold for inner and outer recurrence, but as we will see, certain restrictions are necessary for syntactic reasons.

By studying GNNs in comparison with another computation model over the reals, we obtain a close correspondence between recurrent GNNs and arithmetic circuits. In order to understand the capabilities of GNNs as Boolean classifiers, arithmetic circuits can be related with a Boolean computation model in a subsequent step. Mixing these two steps obscures statements about the computational power of the networks by mixing up two different issues. By separating the computational aspect of recurrent GNNs from the Boolean coding aspect, we do not only obtain an upper bound but an equivalence between recurrent GNNs and recurrent arithmetic circuits over the reals. In this way we shed more light on the computational power of recurrent GNNs by showing very explicitly what elementary operations they can perform. In particular, any known limitations of recurrent arithmetic circuits known or to be proven in the future transfer immediately to recurrent GNNs.

Comparison to previous work. Recurrent GNNs were defined by Pflueger, Cucala, and Kostylev (2024) considering two halting conditions, as explained above. Subsequent papers (Bollen et al. 2025; Ahvonen et al. 2024; Rosenbluth and Grohe 2025) have considered further halting conditions and related the obtained recurrent GNN model to various logics. For some cases of Boolean node classifiers, an equivalence has been shown.

Contrary to the cited papers, we do not consider Boolean classifiers, but see both GNNs and circuits as devices that compute functions (from labelled graphs to labelled graphs, or from tuples of reals to tuples of reals), and show that both are equivalent (modulo an appropriate input encoding). We do not restrict ourselves to the AC-GNN architecture but allow a more general message passing in the form of C-GNNs. We do not restrict ourselves to a particular type of halting condition, but allow arbitrary halting functions computed by arithmetic circuits, subsuming all cases studied so far in the literature.

Organisation. We start by defining our models of recurrent arithmetic circuits in Section 2 and recurrent C-GNNs in Section 3. Subsequently, we give a short overview of our results in Section 4.1, after which we delve into the relation between the models in Section 4, where we show how and under which restrictions recurrent arithmetic circuits can simulate recurrent C-GNNs and vice versa. The full proofs can be found in (Barlag et al. 2026).

2 Recurrent Arithmetic Circuits

Throughout this paper, the graphs we consider have an ordered set of vertices and are undirected unless otherwise specified. We use overlined letters to denote tuples, and write $|\overline{x}|$ to denote the length of $\overline{x}$ (i. e. the number of elements of $\overline{x}$) and x_j to denote the jth element $\overline{x}$. For a $k \in \mathbb{N}$, we write $\mathbb{R}^k$ for the set of all k-tuples from $\mathbb{R}$ and $\mathbb{R}^*$ for the set of all n-tuples from $\mathbb{R}$, for $n \in \mathbb{N}$. The notation $\{\!\{ \}\!\}$ is used for multisets and tuple notation for ordered multisets. Hence $\mathbb{R}^*$ is also interpreted as the set of all multisets over $\mathbb{R}$. We write $\chi[f_B] \to \{0, 1\}$ for the characteristic function of a Boolean expression f_B.

We start with defining the base model of arithmetic circuits and then add recurrence. For simplicity, we define our model of computation over the reals $\mathbb{R}$, but all our results would also hold for $\mathbb{R}^k$ with $k \in \mathbb{N}$. Equivalence of $\mathbb{R}$- and $\mathbb{R}^k$-circuits was shown in (Barlag et al. 2024a, Remark 2.4).

Definition 1. Let $n, m \in \mathbb{N}$. An *arithmetic circuit* with n inputs and m outputs is a simple directed acyclic graph of labelled nodes, also called *gates*, such that

- there are exactly n input gates, which each have indegree 0,
- there are exactly m output gates, which have indegree 1 and outdegree 0,
- there are constant gates, which have indegree 0 and are labelled with a value $c \in \mathbb{R}$,

- there are arithmetic gates, which are associated with addition or multiplication.

Both the input and the output gates are ordered.

The *depth* of C (written *depth*(C)) is the length of the longest path from an input gate to an output gate in C and the *size* of C (written *size*(C)) is the number of gates in C. For a gate g in C, we will write *depth*(g) to denote the length of the longest path from an input gate to g in C.

Definition 2. An arithmetic circuit C with n inputs and m outputs computes the function $f_C\colon \mathbb{R}^n \to \mathbb{R}^m$ as follows: First, the input to the circuit is placed in the input gates. Then, recursively, each arithmetic gate whose predecessor gates all have a value, computes its own value by applying the function it is labelled with to the values of its predecessors. Finally, each output gate takes the value of its predecessor, once its predecessor has one. The output of f_C is then the tuple of values in the m output gates of C after the computation. The function computed up to an individual gate g of a circuit C is defined as $f_g^C\colon \mathbb{R}^n \to \mathbb{R}$, where $f_g^C(\overline{x})$ is the value that g takes in the computation of C on input $\overline{x}$.

Adding recurrence extends the definition of an arithmetic circuit by a halting function and the notion of memory gates. It allows for a circuit to be iteratively executed multiple times until a halting condition is met. In order to save information between the iterations of computation, we first extend our model of arithmetic circuits with so called *auxiliary memory gates* and call the resulting model an *extended arithmetic circuit*.

Definition 3. Let $\ell, m, n \in \mathbb{N}$. An *extended arithmetic circuit* with n inputs and m outputs is an arithmetic circuit with ℓ additional gates labelled *auxiliary memory gates* which are ordered and behave in the same way as input gates. The term *memory gates* is used to refer to the set of input and auxiliary memory gates.

An extended arithmetic circuit C extends the computation of an arithmetic circuit by dependence on the auxiliary memory gates and computes the function $f_C\colon \mathbb{R}^n \times \mathbb{R}^\ell \to \mathbb{R}^m$. The functions computed up to individual gates are extended in the same way such that for each gate g in C, we define $f_g^C\colon \mathbb{R}^n \times \mathbb{R}^\ell \to \mathbb{R}$, where $f_g^C(\overline{x}, \overline{a})$ is the value that g takes in the computation of C on input $\overline{x}$ and with auxiliary memory gate values $\overline{a}$.

The size of extended arithmetic circuits is defined as in Definition 1, whereas the depth is the maximum distance between any two gates.

Definition 4. An *extended arithmetic circuit family* $\mathcal{C}$ is a sequence $(C_n)_{n\in\mathbb{N}}$ of extended circuits, where for all $n \in \mathbb{N}$ the circuit C_n has exactly n input gates. Its *depth* and *size* are functions mapping natural number n to *depth*(C_n) and *size*(C_n), respectively.

An extended arithmetic circuit family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ computes the function $f_{\mathcal{C}}\colon \mathbb{R}^* \times \mathbb{R}^* \to \mathbb{R}^*$ defined as $f_{\mathcal{C}}(\overline{x}, \overline{a}) \coloneqq f_{C_{|\overline{x}|}}(\overline{x}, \overline{a})$. If $C_{|\overline{x}|}$ has more than $|\overline{a}|$ auxiliary memory gates, the input to $f_{C_{|\overline{x}|}}$ is padded with zeroes, and if it has fewer, the overflow is cut off. The number of auxiliary memory gates need not correlate with the input size and can change for every circuit in the family.

Definition 5. For $s, d\colon \mathbb{N} \to \mathbb{N}$, $\text{FSIZE-DEPTH}_{\mathbb{R}}(s, d)$ is the class of all functions $\mathbb{R}^* \times \mathbb{R}^* \to \mathbb{R}^*$ that are computable by extended arithmetic circuit families of size $\mathcal{O}(s)$ and depth $\mathcal{O}(d)$. Let $\mathcal{A}$ be a set of functions $f\colon \mathbb{R} \to \mathbb{R}$. Then $\text{FSIZE-DEPTH}_{\mathbb{R}}(s, d)[\mathcal{A}]$ is the class of functions computable by extended arithmetic circuit families with the same constraints, but with additional gate types g_f, with indegree and outdegree 1 that compute f, for each $f \in \mathcal{A}$.

We are interested in functions that are "simple" in the sense that they can be computed by circuit families with reasonable size and depth.

Definition 6. For $i \in \mathbb{N}$, the class $\text{FAC}^i_{\mathbb{R}}$ denotes the set of functions $f\colon \mathbb{R}^* \times \mathbb{R}^* \to \mathbb{R}^*$ computable by extended arithmetic circuit families of size $\mathcal{O}(n^{\mathcal{O}(1)})$ and depth $\mathcal{O}((\log n)^i)$.

For a set $\mathcal{A}$, the class $\text{FAC}^i_{\mathbb{R}}[\mathcal{A}]$ is defined as in Definition 5. We call such classes, i. e., classes of the form $\text{FAC}^i_{\mathbb{R}}[\mathcal{A}]$, *circuit function classes*. Since those are the only classes we are interested in, whenever we write $\mathfrak{F}$ in the remainder of the paper, we mean that $\mathfrak{F}$ is a circuit function class.

Furthermore, for any $\mathfrak{F} = \text{FAC}^i_{\mathbb{R}}[\mathcal{A}]$ we will use the term $\mathfrak{F}$*-circuit family* to denote those circuit families adhering to the size and depth requirements of $\mathfrak{F}$.

Remark 1. Circuit families and circuit function classes are defined analogously for (non-extended) arithmetic circuits as described in Definition 1.

The second component of a recurrent arithmetic circuit is the halting condition. We define the halting condition as a function based on a natural number which will represent the number of iterations of the circuit, and a tuple of reals which will represent values of some predefined gates of the circuit. Note that this functionality could also be simulated by a function without the additional parameter for the number of iterations by using a sub circuit to count this instead, but we keep it to be consistent with the model of recurrent GNNs we introduce later on. This equivalence is shown in Lemma 2 in (Barlag et al. 2026).

Definition 7. A *halting function* is a function of the form $f_{\text{halt}}\colon \mathbb{N}_{>0} \times \mathbb{R}^* \to \{0, 1\}$.

Note that the halting function maps to discrete values. In what follows, we will also consider halting functions that can be computed by circuits, i. e. they are of some circuit function class. Since plain arithmetic circuits can only compute continuous functions, we have to make use of the additional gate types to achieve the non-continuous nature of the halting function. For this we often use the sign function as an additional gate type:

$$\text{sign}(x) \coloneqq \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise.} \end{cases}$$

Given a circuit function class $\mathfrak{F}$ we write $\mathfrak{F}_s$ as a shorthand for $\mathfrak{F}[\text{sign}]$.

Definition 8. Let $\ell, m, n \in \mathbb{N}$. A *recurrent arithmetic circuit* with n inputs, ℓ auxiliary memory gates and m outputs is a tuple $(C, \overline{a}, E_{\text{rec}}, f_{\text{halt}}, V_{\text{halt}})$, where C is an extended arithmetic circuit with n inputs, m outputs and ℓ auxiliary memory

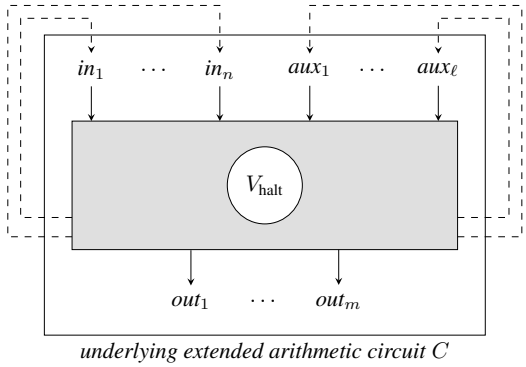


Figure 1: Structure of a recurrent arithmetic circuit: in_i are the input gates, aux_i are the auxiliary memory gates, out_i are the output gates and V_{halt} the halting gates. Dashed edges are the recurrent edges of the set E_{rec}. The halting function is not depicted.

gates with a tuple of initial values $\overline{a} \in \mathbb{R}^\ell$. We call C the *underlying extended arithmetic circuit*. E_{rec} is a set of edges consisting of exactly one incoming edge for each memory gate in C. As before, the memory gates consist of the input and auxiliary memory gates. Every memory gate has fan-in one, the output gates have fan-out zero.

The halting of the recurrent circuit is determined by a halting function $f_{\text{halt}} \colon \mathbb{N}_{>0} \times \mathbb{R}^{|V_{\text{halt}}|} \to \{0,1\}$, where V_{halt} are the *halting gates*, a subset of the gates of C.

The structure of a recurrent circuit is depicted in Figure 1 and a complete example is given in Example 1.

A recurrent arithmetic circuit computes a function as follows: In addition to the input gates, the auxiliary memory gates get assigned their initial constant. The underlying extended arithmetic circuit is then executed with the input values and the initial memory gate values as in Definition 3 until all gates have a value. If the halting function evaluates to 1 the output of the function is the tuple of values in the output gates of the recurrent circuit. Otherwise the memory gates take the value of their respective predecessors, all other non-constant gates are reset to have no value and the computation of the underlying circuit is executed again.

Definition 9. Let $\ell, m, n, p \in \mathbb{N}$. Let *rec-C* $= (C, \overline{a}, E_{\text{rec}}, f_{\text{halt}}, V_{\text{halt}})$ be a recurrent arithmetic circuit with n inputs, ℓ auxiliary memory gates and m outputs, an ordered set of p halting gates $V_{\text{halt}} = \{v_1, \ldots, v_p\}$ and a halting function $f_{\text{halt}} \colon \mathbb{N}_{>0} \times \mathbb{R}^p \to \{0,1\}$. The halting function operates on the current iteration number and the ordered values of the gates V_{halt}. For all gates g in C let $f_g \colon \mathbb{R}^n \times \mathbb{R}^\ell \to \mathbb{R}$ be the function that computes the value that g takes in the computation of C. The recurrent arithmetic circuit *rec-C* computes the function $f_{\textit{rec-C}} \colon \mathbb{R}^n \to \mathbb{R}^m$ defined as $f_{\textit{rec-C}}(\overline{x}) \coloneqq f^1_{\text{mem}}(\overline{x}, \overline{a})$, where $\overline{a}$ are the initial constants in the auxiliary memory gates and $\overline{x}$ is the input. The function $f^i_{\text{mem}} \colon \mathbb{R}^n \times \mathbb{R}^\ell \to \mathbb{R}^m$ is recursively defined as

$$f^i_{\text{mem}}(\overline{x}, \overline{a}) \coloneqq \begin{cases} f^{i+1}_{\text{mem}}\left(\overline{y}, \overline{b}\right), & f_{\text{halt}}(i, \overline{v}) = 0 \\ \overline{o}, & \text{else} \end{cases},$$

where $\overline{y} \in \mathbb{R}^n$ with $y_j = f^C_{\text{in}_j}(\overline{x}, \overline{a})$ for each $j \leq n$, $\overline{b} \in \mathbb{R}^\ell$ with $b_j = f^C_{\text{aux}_j}(\overline{x}, \overline{a})$ for each $j \leq \ell$, $\overline{v} \in \mathbb{R}^p$ with $v_j = f^C_{v_j}(\overline{x}, \overline{a})$ for each $j \leq p$, and $\overline{o} \in \mathbb{R}^m$ with $o_j = f^C_{\text{out}_j}(\overline{x}, \overline{a})$ for each $j \leq m$. Here, in_j (aux_j, out_j, out_j, resp.) refers to the predecessor of the jth input gate (auxiliary memory gate, halting gate, output gate, resp.) and i is the number of the current iteration. The tuples $\overline{y}, \overline{b}, \overline{o}, \overline{v}$ are the new values of these gates after one iteration of the underlying circuit C. The function f^i_{mem} maps memory gate values to memory gate values until the halting condition is reached.

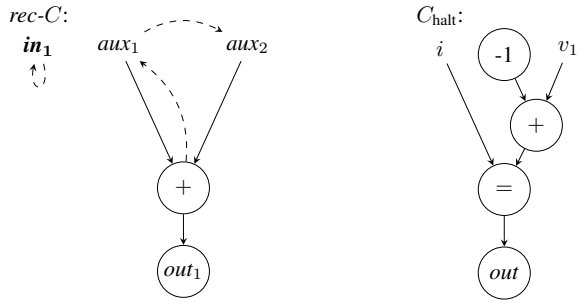


Figure 2: Illustration of recurrent circuit *rec-C* with halting circuit C_{halt} computing the x_1-th Fibonacci number for $1 < x_1 \in \mathbb{N}$. The dashed arrows mark the recurrent edges and the bold gate in_1 is the single halting gate, i. e. the gate whose value forms the input for C_{halt} along with the iteration number i.

Example 1. For $x_1 \in \mathbb{N}$, the recurrent circuit *rec-C* $= (C, \overline{a}, E_{\text{rec}}, f_{\text{halt}}, V_{\text{halt}})$ as illustrated in Figure 2 computes the x_1-th Fibonacci number when given $x_1 > 1$ as the input.

The recurrent circuit *rec-C* has one input gate in_1 which receives the input x_1, two auxiliary memory gates aux_1 and aux_2 and one output gate out_1. The set V_{halt} consists only of in_1.The halting function f_{halt} of *rec-C* is the function computed by the circuit C_{halt} on the right.

The circuit computes the x_1-th Fibonacci number as follows. Initially, the input x_1 to *rec-C* is the index of the desired Fibonacci number and the auxiliary memory gates are set up with the initial values $\overline{a} = (a_1, a_2)$ where $a_1 = 1$ and $a_2 = 0$, respectively, which are the first (and 0th) Fibonacci numbers. The addition gate now computes the sum of the values of the gates aux_1 and aux_2. Before the next iteration of the circuit begins, the halting condition is checked. In this case, C_{halt} examines whether the number of iteration rounds (the value of input gate i) and the input to *rec-C*, x_1, are equal. But since the first Fibonacci number was stored in aux_1 and not calculated by the recursive scheme, we need to subtract 1 from x_1. The function of the equality gate (=) can be computed using a (non-recurrent) circuit of depth 7 facilitating the following formula: $x_1 = x_2 \coloneqq ((\text{sign}(x_1 - x_2) + \text{sign}(x_2 - x_1)) \times -1) + 1$. If C_{halt} outputs 1, *rec-C* outputs the value given to the output gate. Otherwise, the values of the auxiliary memory gates are updated: The value of aux_1 gets updated with the sum of the previous values of aux_1 and aux_2, while the value of aux_2 gets updated with the previous value of aux_1.

As the Fibonacci sequence cannot be expressed as a polynomial there is no non-recurrent arithmetic circuit family that computes it.

Next, we define the notion of recurrent circuit families similar to Definition 4.

Definition 10. A *recurrent arithmetic circuit family rec-$\mathcal{C}$* is a sequence $(\textit{rec-}C_n)_{n\in\mathbb{N}}$ of recurrent circuits, where each circuit *rec-*C_n has exactly n input gates. The underlying circuits of *rec-*$\mathcal{C}$ define an *extended arithmetic circuit family* $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$, also called the *underlying circuit family*.

The *depth* and *size* of *rec-*$\mathcal{C}$ are defined by the *depth* and *size* of the underlying circuit family $\mathcal{C}$.

A recurrent arithmetic circuit family *rec-*$\mathcal{C} = (\textit{rec-}C_n)_{n\in\mathbb{N}}$ computes the function $f_{\textit{rec-}\mathcal{C}} \colon \mathbb{R}^* \to \mathbb{R}^*$ defined as $f_{\textit{rec-}\mathcal{C}}(\overline{x}) \coloneqq f_{\textit{rec-}C_{|\overline{x}|}}(\overline{x})$. The set of halting functions of a recurrent circuit family *rec-*$\mathcal{C} = (\textit{rec-}C_n)_{n\in\mathbb{N}}$ is defined as $\mathcal{F} = \{f^n_{\text{halt}} \mid f^n_{\text{halt}} \text{ in } \textit{rec-}C_n,\ n \in \mathbb{N}\}$, where $(f^n_{\text{halt}})_{n\in\mathbb{N}}$ is the sequence of halting functions of *rec-*$\mathcal{C}$.

Remark 2. Without loss of generality, we stipulate that if two recurrent circuits *rec-*C and *rec-*C' in a recurrent circuit family have the same number of halting gates, then they use the same halting functions, i.e., $|V_{\text{halt}}| = |V'_{\text{halt}}|$ implies $f_{\text{halt}} = f'_{\text{halt}}$. This is not a restriction as we can always increase the arity of a function by adding additional inputs that do not affect the output.

The halting functions for each circuit are determined by the number of halting gates of the circuit. Note that a recurrent circuit family might only use a subset of the functions of a circuit function family as its halting functions, as not all arities are necessarily utilised, and it might use the same halting function for circuits of different input size. When expressing that the halting complexity of a recurrent circuit family is $\mathfrak{F}_s$, we mean that the respective set $\mathcal{F}$ is a subset of an $\mathfrak{F}_s$ function family, rather than an element of $\mathfrak{F}_s$.

Definition 11. We say that a function $\mathbb{R}^* \to \mathbb{R}^*$ is in the *recurrent circuit function class* $\text{rec}[\mathfrak{F}_{\text{halt}}]\text{-}\mathfrak{F}_C$ if it can be computed by a recurrent arithmetic circuit family where the function which its underlying arithmetic circuit family computes is in the circuit function class $\mathfrak{F}_C$ and its set of halting functions is a subset of a circuit family in the circuit function class $\mathfrak{F}_{\text{halt}}$.

If not specified otherwise, we consider recurrent circuit function classes of the form $\text{rec}[\mathfrak{F}_s]\text{-}\mathfrak{F}$, i. e., recurrent circuit families where the underlying circuit family is in a circuit function class $\mathfrak{F}$ and the set of halting functions is a subset of a $\mathfrak{F}_s$-family, the same class extended by sign. As every circuit family is also a recurrent circuit family that uses a constant halting function we have that $\mathfrak{F} \subseteq \text{rec}[\mathfrak{F}_s]\text{-}\mathfrak{F}$.

Note that our recurrent circuits are able to simulate halting by reaching a fixed point by adding an additional auxiliary memory gate for each output gate that saves the value of the last iteration of the circuit. The procedure is similar to that used in our Example 1 to save the previously computed Fibonacci numbers. Similarly it is able to simulate halting by some (input dependent) initial counter value reaching zero via adding an auxiliary memory gate to the underlying circuit that is decreased by one in each iteration. The halting function is used to check whether the value of that gate reached zero.

For our later results we need to define the composition of two function families from a circuit function class $\text{rec}[\mathfrak{F}_s]\text{-}\mathfrak{F}$. Note that the circuit family computing the composition of those function families needs to have access to the sign gate.

Theorem 1. *Let $(f_n)_{n\in\mathbb{N}}$, $(f'_n)_{n\in\mathbb{N}}$ be two function families in $\text{rec}[\mathfrak{F}_s]\text{-}\mathfrak{F}$. Then $(f'_n)_{n\in\mathbb{N}} \circ (f_n)_{n\in\mathbb{N}} = (f'_{n'} \circ f_n)_{n\in\mathbb{N}}$, where $n' \in \mathbb{N}$ is the output dimension of f_n, is a function family in $\text{rec}[\mathfrak{F}_s]\text{-}\mathfrak{F}_s$.*

Proof Sketch. The circuit computing $f'_{n'} \circ f_n$ is constructed by using the circuits computing f_n and $f'_{n'}$, as well as the circuit computing the halting function of f_n. They are connected in such a way, that the first circuit is iterated until the first halting condition is met. Only then the computations of the second circuit are started with meaningful values. To be able to check whether halting function of the first circuit evaluated to 1 the computation of the halting function needs to be directly incorporated into the circuit computing the composition. As the halting function is from a circuit function class that allows for sign gates the composed function is also in a such a circuit function class. Our restriction to only $\text{FAC}^i_{\mathbb{R}}$ classes to ensure the halting function is implementable in the same class. □

3 Network Model

3.1 Recurrent Graph Neural Networks

In order to construct a circuit based model of recurrent graph neural networks (GNNs) in the next section, we fix the notion of a recurrent GNN first. Generally, recurrent GNNs like their non-recurrent counterpart can classify either individual nodes or whole graphs. For the purpose of this paper, we will only introduce node classification, since graph classification works analogously. We focus on aggregate combine GNNs that aggregate the information of every neighbour of a node and then combine this information with the information of the node itself. Typically GNNs are defined with a fixed number of layers of computations. A recurrent GNN does not have this restriction. Instead the feature values of a given labelled input graph are updated multiple times according to the defined computations of the GNN until a predefined halting condition is met.

Definition 12. Let $G = (V, E)$ be a graph with an ordered set of vertices V and a set of undirected edges E on V. Let $g_V \colon V \to \mathbb{R}$ be a function which labels the vertices with so called *feature values*. We then call $\mathfrak{G} = (V, E, g_V)$ a *labelled graph* and denote by $\mathfrak{Graph}$ the class of all labelled graphs. We denote the multiset of all labels of a set of nodes $V' \subseteq V$ by $\text{val}(V') \coloneqq \{\!\{g_V(v) \mid v \in V'\}\!\}$. For a node $v \in V$, the *neighbourhood of* v is defined as $N_{\mathfrak{G}}(v) \coloneqq \{w \in V \mid \{v, w\} \in E\}$.

As we did in the case of recurrent arithmetic circuits, we define recurrent GNNs with feature values in $\mathbb{R}$, but we remark that all our results also hold in the more general setting of circuits and GNNs over $\mathbb{R}^k$ for some $k \in \mathbb{N}$. A recurrent GNN is defined by the following types of functions.

An *aggregation function* is a permutation-invariant function $\text{AGG} : \mathbb{R} \times \cdots \times \mathbb{R} \to \mathbb{R}$, a *combination function* is a function $\text{COM} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and a (Boolean) *classification function* is a function $\text{CLS} : \mathbb{R} \to \{0, 1\}$ that classifies a real value as either true or false. A *halting function* is a function

HLT: $\mathbb{N}_{>0} \times \mathbb{R}^* \to \{0,1\}$ that gets the current layer number and the unordered multiset of feature values as an input. Finally, an *activation function* is a function $\sigma\colon \mathbb{R} \to \mathbb{R}$.

Definition 13. Let $d \in \mathbb{N}$. A *recurrent aggregate combine graph neural network* (rec-AC-GNN) of period d is a tuple $\mathcal{D} = (\{\text{AGG}^{(i)}\}_{i=1}^{d}, \{\text{COM}^{(i)}\}_{i=1}^{d}, \{\sigma^{(i)}\}_{i=1}^{d}, \text{HLT}, \text{CLS})$, where $\{\text{AGG}^{(i)}\}_{i=1}^{d}$ and $\{\text{COM}^{(i)}\}_{i=1}^{d}$ are respectively sequences of aggregation and combination functions, $\{\sigma^{(i)}\}_{i=1}^{d}$ is a sequence of activation functions, HLT is a halting function and CLS is a classification function.

Given a labelled graph $\mathfrak{G} = (V, E, g_V)$, the initial feature values $x_v^{(0)} = g_V(v)$ are set for every $v \in V$ and for every layer $1 \leq i \leq d$ the rec-AC-GNN model computes $x_v^{(i)}$ for every $v \in V$ as follows:

$$x_v^{(i)} = \sigma^{(i)}\left(\text{COM}^{(i)}\left(x_v^{(i-1)}, y\right)\right),$$
$$\text{where } y = \text{AGG}^{(i)}\left(\{\!\{x_u^{(i-1)} \mid u \in N_{\mathfrak{G}}(v)\}\!\}\right).$$

After every layer i the halting function $\text{HLT}\left(i, \left(x_v^{(i)} \mid v \in V\right)\right)$ is computed on the current layer number and the tuple of resulting feature values of that layer. If HLT evaluates to 1 in layer i, the computation of the recurrent GNN stops and the classification function CLS: $\mathbb{R} \to \{0,1\}$ is applied to the feature values $x_v^{(i)}$. Otherwise the computation continues with the next layer $i+1$ or once layer d is reached, periodically starts again with layer 1.

Non-recurrent GNNs can be recovered as the special case, where the value of the halting function is determined by the inputted layer number.

In this paper we focus on the real-valued computation part of GNNs and discard the classification function. We consider the feature values $x_v^{(\ell)}$ after the computation of layer ℓ that satisfy our halting condition as our output. While we could also integrate CLS into our model, for our concerns this is not needed.

3.2 Recurrent Circuit Graph Neural Networks

Our goal is to analyze the expressive power of recurrent graph neural networks, without restricting to aggregate-combine GNNs or any other particular type. Following (Barlag et al. 2024a), we will study *circuit GNNs* and their recurrent version.

A *basis* of a recurrent C-GNN is a set of functions of a specific recurrent circuit function class along with a set of activation functions and a halting function which is also in a specific circuit function class via a circuit family which computes it. The respective networks will be based on these functions. As GNNs are permutation invariant we also assume that our halting functions for the recurrent C-GNNs have this property. We call such a function *tail-symmetric* and write $t\text{-}f_{\text{halt}}\colon \mathbb{N}_{>0} \times \mathbb{R}^* \to \{0,1\}$ to emphasize it. The first parameter of the halting function is the number of the current layer. We can generally define tail-symmetry of a function as follows.

Definition 14. A function $f\colon \mathbb{R}^n \to \{0,1\}$ is *tail-symmetric*, if $f(v_1 \ldots, v_n) = f(v_1, \pi(v_2, \ldots, v_n))$ for all permutations π.

The same requirement of tail-symmetry holds for the functions that are computed in each layer of a GNN. We write $t\mathfrak{F}$ to denote only the tail-symmetric functions of the circuit function class $\mathfrak{F}$ and call $t\mathfrak{F}$ a *tail-symmetric class*.

Definition 15. A *recurrent C-GNN-basis* is a set $\mathcal{S} \times \mathcal{A} \times \{t\text{-}f_{\text{halt}}\}$, where $\mathcal{S}$ is a subset of some tail-symmetric recurrent circuit function class $t\mathfrak{F}$, $\mathcal{A}$ is set of activation functions, and $t\text{-}f_{\text{halt}}$ is from some tail-symmetric circuit function class $\mathfrak{F}_{\text{halt}}$. We call bases of this kind as $(t\mathfrak{F}, t\mathfrak{F}_{\text{halt}})$-*bases*.

Recurrent C-GNNs of a particular basis essentially consist of a function assigning recurrent circuit families and activation functions from its basis to its different layers, and in addition, a halting function that governs how often those layers are iterated.

Definition 16. Let $B = \mathcal{S} \times \mathcal{A} \times \{t\text{-}f_{\text{halt}}\}$ be a $(t\mathfrak{F}, t\mathfrak{F}_{\text{halt}})$-basis. A *recurrent circuit graph neural network rec-*$\mathcal{N} = (\mathcal{N}, t\text{-}f_{\text{halt}})$ of basis B consists of a periodic function $\mathcal{N}\colon \mathbb{N} \to \mathcal{S} \times \mathcal{A}$ and a tail-symmetric halting function $t\text{-}f_{\text{halt}}\colon \mathbb{N}_{>0} \times \mathbb{R}^* \to \{0,1\}$. We say that *rec-*$\mathcal{N}$ is a rec$[t\mathfrak{F}_{\text{halt}}]$-$(t\mathfrak{F}, \mathcal{A})$-GNN.

Definition 17. Let *rec-*$\mathcal{N} = (\mathcal{N}, t\text{-}f_{\text{halt}})$ be a recurrent C-GNN. The partial function $f_{\textit{rec-}\mathcal{N}}\colon \mathfrak{Graph} \to \mathfrak{Graph}$ computed by *rec-*$\mathcal{N}$ is defined as follows: Let $\mathfrak{G} = (V, E, g_V)$ be a labelled graph. The labelled graph $\mathfrak{G}' = f_{\textit{rec-}\mathcal{N}}(\mathfrak{G})$ has the same structure as $\mathfrak{G}$, however, its nodes have different feature vectors. That is $\mathfrak{G}' = (V, E, h_V)$ and h_V is defined as follows:

$$h_V^{(0)}(w) \coloneqq g_V(w)$$
$$h_V^{(i)}(w) \coloneqq \sigma^{(i)}\left(f_{\mathcal{C}^{(i)}}\left(h_V^{(i-1)}(w), M\right)\right),$$
$$\text{with } M = \left\{\!\!\left\{h_V^{(i-1)}(u) \mid u \in N_{\mathfrak{G}}(w)\right\}\!\!\right\}$$

where $\mathcal{N}(i) = \left(\mathcal{C}^{(i)}, \sigma^{(i)}\right)$. Then $h_V = h_V^{(\ell)}$ is the smallest value such that $t\text{-}f_{\text{halt}}\left(\ell, \left\{\!\!\left\{h_V^{(\ell)}(v) \mid v \in V\right\}\!\!\right\}\right) = 1$ where $\ell \in \mathbb{N}$. Otherwise $f_{\textit{rec-}\mathcal{N}}$ is undefined.

The halting function of a recurrent C-GNN is applied to the feature vectors of all nodes contrary to recurrent arithmetic circuits where only the values of some of the gates were taken into consideration.

The recurrent C-GNN model that we have just defined computes functions from labelled graphs to labelled graphs (with the same graph structure) with recurrence in two places. To distinguish those two types of recurrence we call the recurrence of the complete network *outer recurrence* and that of the internal circuits *inner recurrence*.

We call rec$[t\mathfrak{F}_{\text{halt}}]$-$(t\mathfrak{F}, \mathcal{A})$-GNNs where $t\mathfrak{F}$ is a non-recurrent circuit function class a *circuit graph neural networks with outer recurrence*. On the other hand, we call (rec$[\mathfrak{F}_s]$-$t\mathfrak{F}, \mathcal{A})$-GNNs, where rec$[\mathfrak{F}_s]$-$t\mathfrak{F}$ is a recurrent circuit function class and the halting is determined by a fixed layer number, a *circuit graph neural network with inner recurrence*. Finally we call $(t\mathfrak{F}, \mathcal{A})$-GNNs, where $t\mathfrak{F}$ is a non-recurrent

C-GNN with Outer recurrence	C-GNN with Inner recurrence	Recurrent circuit without sign	Recurrent circuit with sign	
✓	✗	✓	✗	(Thm. 2)
✗	✓	✗	✓	(Thm. 3)
✓	✓	✗	✓	(Cor. 1)

Table 1: Simulating Recurrent C-GNNs with Recurrent Circuits. One row in the table represents one result.

circuit function class, *circuit graph neural networks*. This model is closest to that of a standard non-recurrent GNN and was already introduced in (Barlag et al. 2024a). By our definition of recurrent arithmetic circuits every function computable by a C-GNN without recurrence is also computable by any of the introduced recurrent C-GNNs. An overview of the different models is given in Figure 3.

An overview of the relations between the different C-GNN models is given in Figure 4.

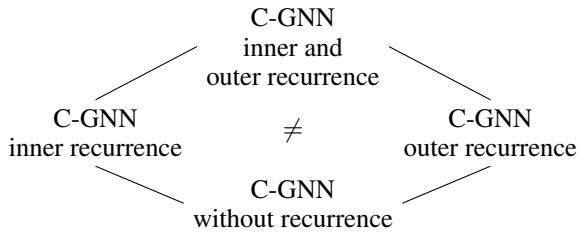


Figure 4: Assume the same circuit function classes for recurrence/ underlying circuit and the same set of activation functions, lines denote subclasses of computable functions

4 Relations Between Recurrent Circuits and Recurrent GNNs

Our aim is to relate our different models of recurrent C-GNNs to the model of recurrent arithmetic circuits we introduced before, i. e. to show which functions that can be computed by recurrent arithmetic circuits can also be computed using a recurrent C-GNN and vice versa. We generally assume that the the circuits and C-GNNs have the same resources, i.e. the same circuit function classes on both sides and access to the same additional functions or activation functions. This allows us to investigate the similarities and differences in expressive power between the two models.

4.1 Overview of Results

We show that there exist recurrent arithmetic circuits that are able to simulate the computations of our recurrent C-GNN model in the sense that when given an encoding of a labelled graph they compute the updated feature values and output the updated (encoded) labelled graph. We distinguish between recurrent arithmetic circuits where the underlying circuits have access to the sign function an can therefore compute non-continuous functions and those that do not. An overview of the results of that direction is given in Table 1.

We moreover show that functions computed by recurrent circuits with different restrictions on the functions or the circuits themselves can be simulated by our recurrent C-GNN models in the sense that there exists a recurrent C-GNN for which there exists an input (labelled graph) such that the output (labelled graph) has the output of the recurrent circuit among its feature values. Here we differentiate between C-GNNs by their modes of recurrence — inner or outer recurrence — as well as how they are allowed to use activation functions — either arbitrarily locally in their internal circuits or globally on the whole graph as in the usual sense of activation functions in GNNs. Also of interest is whether the internal circuits of the C-GNN have access to sign gates which allow them to compute non-continuous functions. The results are shown in Table 2. We can also show that recurrent circuits (even when restricted to symmetric functions) cannot be simulated by Inner Recurrent C-GNNs using activation functions in the usual GNN sense (Lemma 1).

We get the correspondences of the two models under reasonable reductions/encodings that are necessary as C-GNNs and arithmetic circuits operate on different types of input, namely labelled graphs and real valued input strings. For the most general models (for C-GNNs and recurrent circuits) we have a one-to-one correspondence. Every recurrent circuit can be simulated with an outer and inner recurrent C-GNN and every outer and inner recurrent C-GNN can be simulated with a recurrent circuit. This also shows that under logspace reductions outer recurrent C-GNNs have the same expressivity as C-GNNs with both inner and outer recurrence. For the other simulations some restrictions are assumed.

4.2 Simulating Recurrent GNNs With Recurrent Arithmetic Circuits

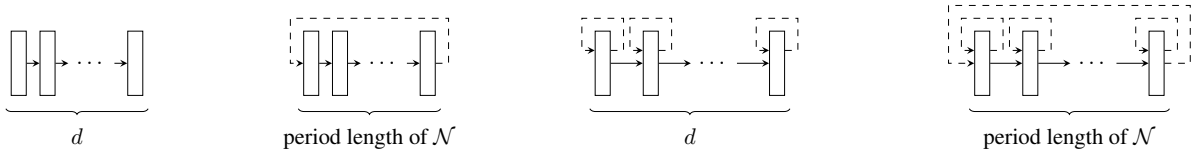
While recurrent C-GNNs work on labelled graphs, arithmetic circuits only get an ordered tuple of $\mathbb{R}$-values as an input. To be able to talk about arithmetic circuits computing the same functions as recurrent C-GNNs, we encode labelled graphs as real valued tuples that can then be used as an input to an arithmetic circuit. We write $\langle \mathfrak{G} \rangle$ for the encoding of a labelled graph $\mathfrak{G}$ as its adjacency matrix followed by the respective feature values.

As defined in Section 2, we assume that our recurrent arithmetic circuits only have access to the sign function inside their halting function. However by a result from (Barlag et al. 2024b, Lemma A.4) we are still able to check for equality for a set of fixed values in the underlying circuits in constant depth.

The following theorem states that any C-GNN with outer recurrence can be simulated by a recurrent circuit. Given an encoding of a labelled graph as an input the recurrent circuit computes the encoding of the labelled graph the C-GNN would output. Using the reverse procedure of the encoding this can then be decoded back to a labelled graph.

Theorem 2. *Let rec-$\mathcal{N}$ be a* $\mathrm{rec}[t\mathfrak{F}_s]$*-$(t\mathfrak{F}, \mathcal{A})$-GNN. Then there exists a function* $f_{\mathrm{circ}} \in \mathit{rec}[\mathfrak{F}_s]\text{-}\mathfrak{F}[\mathcal{A}]$ *s. t. for all labelled graphs* $\mathfrak{G}$ *we have* $f_{\mathrm{circ}}(\langle \mathfrak{G} \rangle) = \langle f_{\mathit{rec}\text{-}\mathcal{N}}(\mathfrak{G}) \rangle$.

Proof Sketch. The main idea here is to build a circuit $C_n^{\mathcal{N}}$ for every input length n where we have the d different circuit



(a) C-GNN without recurrence and a fixed depth $d \in \mathbb{N}$

(b) C-GNN with only outer recurrence

(c) C-GNN with only inner recurrence and a fixed depth $d \in \mathbb{N}$

(d) C-GNN with inner and outer recurrence

Figure 3: The different models of (recurrent) C-GNNs. One block stands for one layer of the different (recurrent) C-GNN models $\mathcal{N}$, dashed edges mark recurrence

Recurrent circuit	Recurrent C-GNN					
	Outer recurrence	Inner recurrence	sign in internal circuits	Activation functions local	Activation functions global	
no restrictions	✓	✗	✓	✓	✗	(Thm. 5)
Predecessor Form	✓	✗	✗	✗	✓	(Thm. 4)
Symmetric Functions	✗	✓	✗	✓	✗	(Thm. 6)

Table 2: Simulating Recurrent Circuits with Recurrent C-GNNs.
One row in the table represents one result.

families of rec-$\mathcal{N}$ in parallel for period length d of rec-$\mathcal{N}$. We use a modulus d counter to decide which result to feed back into the next iteration. □

Contrary to the previous result, in order to simulate the function computed by an inner recurrent C-GNN via a recurrent circuit, we need the underlying circuit of that recurrent circuit to have access to a sign gate., i. e. to compute a function from a class of the form $\mathfrak{F}_s[\mathcal{A}]$. This is due to the fact that the proof is based on the composition result of Theorem 1.

Theorem 3. *Let rec-$\mathcal{N}$ be a* $(\mathrm{rec}[t\mathfrak{F}_s]\text{-}t\mathfrak{F}, \mathcal{A})$*-GNN. Then there exists a function* $f_{\mathrm{circ}} \in \mathit{rec}[\mathfrak{F}_s]\text{-}\mathfrak{F}_s[\mathcal{A}]$ *s. t. for all labelled graphs* $\mathfrak{G}$ *we have* $f_{\mathrm{circ}}(\langle \mathfrak{G} \rangle) = \langle f_{\mathit{rec}\text{-}\mathcal{N}}(\mathfrak{G}) \rangle$.

Proof Sketch. The inner recurrent C-GNN has a fixed number of layers. We therefore compose the functions computed by the recurrent circuits of each layer while ensuring the correct connections between nodes are represented by the recurrent circuit that simulates the computations of the recurrent C-GNN. □

From Theorems 2 and 3 follows that a recurrent C-GNN with both inner and outer recurrence can be simulated by a recurrent circuit. The proof combines the techniques used in the proofs of the mentioned theorems.

Corollary 1. *Let rec-$\mathcal{N}$ be a* $\mathrm{rec}[t\mathfrak{F}_s]\text{-}(\mathrm{rec}[t\mathfrak{F}_s]\text{-}t\mathfrak{F}, \mathcal{A})$*-GNN. Then there exists a function* $f_{\mathrm{circ}} \in \mathit{rec}[\mathfrak{F}_s]\text{-}\mathfrak{F}_s[\mathcal{A}]$ *defined such that for all labelled graphs* $\mathfrak{G}$ *it holds that* $f_{\mathrm{circ}}(\langle \mathfrak{G} \rangle) = \langle f_{\mathit{rec}\text{-}\mathcal{N}}(\mathfrak{G}) \rangle$.

4.3 Simulating Recurrent Arithmetic Circuits With Recurrent C-GNNs

Simulation with Outer Recurrent C-GNNs

Our goal is to construct recurrent C-GNNs which are able to simulate the computations of recurrent circuits, i. e. they have the output of a given circuit among the feature vectors of the output labelled graph when given a labelled graph as input. Those input labelled graphs are constructed given the recurrent arithmetic circuit. We therefore need to fix the notion of a graph encoding. We bound them to be logspace Turing computabile to ensure that our encodings are not too powerful. As recurrent arithmetic circuits are usually defined via their tuple as described in Definition 8 and then given an input, we use a so called *symbolic logspace labelled graph encoding* $\mathfrak{G}(C)$ first which, given a recurrent circuit C as input, uses symbols for the input, auxiliary memory and constant gates instead of the values from $\mathbb{R}$ as node labels. This ensures that our encoding can be computed by a Turing machine which only works over a finite alphabet. Afterwards we use the corresponding *labelled graph encoding* $\mathfrak{G}(C, \overline{x})$ that, additionally given the input $\overline{x}$, replaces the symbols for inputs, auxiliary memory and constant values with their respective values and outputs a labelled graph with labels from $\mathbb{R}$.

To simplify the proofs of the theorems of this section, from now on we assume the recurrent circuits to be *balanced DAGs*, meaning that the following property holds for both the underlying circuits and the halting circuits.

Definition 18. A circuit C is a *balanced DAG* if for each gate g in C every path from an input to g has the same length.

It was shown that this assumption does not form a restriction in the non-recurrent setting (Barlag, Chudigiewitsch, and Gaube 2024) and the same argument can be applied here, since the property of being a balanced DAG only concerns the structure of the underlying extended circuit and has no bearing on the recurrent part of the circuit.

We show that recurrent arithmetic circuits can be simulated by C-GNNs with outer recurrence. Here the halting of the recurrent circuit needs to be simulated by the outer recurrent C-GNN, which only has a tail-symmetric halting function available that operates on all feature values. Therefore, we

have to move some of the computations of the halting functions to the internal circuits of the C-GNN. This means that they need to have access to the sign function, however their size and depth restrictions are the same as for the underlying circuit of the recurrent circuit which they are simulating.

The halting function of the C-GNN stays in the same circuit function class as that of the simulated circuit.

Theorem 4. *Let $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ be a rec$[\mathfrak{F}_s]$-$\mathfrak{F}[\mathcal{A}]$-circuit family. Then there exists a rec$[\mathfrak{tF}_s]$-$(\mathfrak{tF}_s[\mathcal{A}], id)$-GNN rec-$\mathcal{N} = (\mathcal{N}, t\text{-}f_{halt})$ and a symbolic logspace graph encoding $\mathfrak{G}(C_n)$, such that for all $n \in \mathbb{N}$ and $\overline{x} \in \mathbb{R}^n$ the feature values of $f_{\mathcal{N}}(\mathfrak{G}(C_n, \overline{x}))$ include $f_{C_n}(\overline{x})$.*

Proof Sketch. The underlying circuit and the halting circuit of the recurrent circuit C_n that is being simulated are encoded into one labelled graph ensuring the following criteria: the nodes representing halting gates are connected to the nodes representing input gates of the halting circuit and all nodes that represent circuit gates have a distinct degree by the addition of dummy nodes. In each layer of the recurrent C-GNN the operations of the circuit C_n are now performed stepwise via arithmetic circuit families in the nodes of the labelled graph. The degree of the nodes serves as an identifier which circuit of the internal circuit family of the C-GNN is executed, e. g. which operation is performed such that the feature value of each node is equal to the value of the gate that it takes during the execution of the circuit C_n. The computations of the halting function are simulated in the same way. By a special construction the halting of the C-GNN only has to check for a predefined value in a fixed number of feature values. Notably the halting function is even in $\mathrm{FAC}^0_{\mathbb{R}}[\mathrm{sign}]$. □

In the previous result the recurrent C-GNN that simulated the computation of functions from a circuit function class rec$[\mathfrak{F}_s]$-$t\mathfrak{F}[\mathcal{A}]$ realised the additional functions from the set $\mathcal{A}$ in the internal circuits, as those were from the circuit function class $t\mathfrak{F}_s[\mathcal{A}]$ (with the additional sign gate). Naturally in the GNN model additional functions like those from $\mathcal{A}$ are applied as activation functions only after the computations of a layer are completed. To be able to simulate functions by recurrent C-GNNs of this form, restrictions on the arithmetic circuit families computing them are necessary. The following type of circuit form was originally introduced in (Barlag et al. 2024a). We now use and extend this notion under the assumption of balanced DAGs to our model of recurrent circuits.

Definition 19. A recurrent circuit is in *predecessor form* if its underlying circuit C is a balanced DAG, for each depth $d \leq \mathit{depth}(C)$ all gates of C at depth d have the same gate type and the last function layer is at depth d, if all halting gates and predecessors of memory gates are at depth $> d$.

We write $p\mathfrak{F}$ to denote only the functions that are computable by $\mathfrak{F}$-circuit families in predecessor form and call classes of the form $p\mathfrak{F}$ *predecessor form classes*.

Functions computed by arithmetic circuit families of this form can be simulated by C-GNNs with outer recurrence that use the additional functions from the set $\mathcal{A}$ only as activation functions.

Theorem 5. *Let $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ be a rec$[\mathfrak{F}_s]$-$p\mathfrak{F}[\mathcal{A}]$-circuit family. Then there exists a rec$[\mathfrak{tF}_s]$-$(\mathfrak{tF}_s, \mathcal{A} \cup \{id\})$-GNN rec-$\mathcal{N}$ and a symbolic logspace graph encoding $\mathfrak{G}(C_n)$, such that for all $n \in \mathbb{N}$ and $\overline{x} \in \mathbb{R}^n$ the feature values of $f_{\mathcal{N}}(\mathfrak{G}(C_n, \overline{x}))$ include $f_{C_n}(\overline{x})$.*

Proof Sketch. The proof proceeds in the same way as that of Theorem 4. The predecessor form ensures that during the simulation of the recurrent circuit no results are overwritten by the global application of the activation function. □

Simulation with Inner Recurrent C-GNNs

To be able to simulate recurrent arithmetic circuits with inner recurrent C-GNNs, a different restriction on the functions computed by the circuits is needed.

Definition 20. A function $f\colon \mathbb{R}^n \to \{0,1\}$ is *symmetric*, if $f(v_1 \ldots, v_n) = f(\pi(v_1, \ldots, v_n))$ for all permutations π.

We write $s\mathfrak{F}$ to denote only the symmetric functions of the circuit function class $\mathfrak{F}$. We call classes of the form $s\mathfrak{F}$ *symmetric classes*.

Functions from recurrent function classes of this form can now be simulated by inner recurrent C-GNNs by essentially simulating the computations of the recurrent circuit computing them in parallel in each of the nodes of the input graph. The need for the symmetry arises because computations done in a node of a C-GNN have no order on their inputs, i. e. on the feature values of their neighbouring nodes. Notice that due to this method of simulation the underlying circuits of the internal circuits do not need to have access to the sign gate, as was required in the previous results.

Theorem 6. *Let $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ be a rec$[\mathfrak{F}_s]$-$s\mathfrak{F}[\mathcal{A}]$-circuit family. Then there exists a (rec$[\mathfrak{F}_s]$-$\mathfrak{tF}[\mathcal{A}], \{id\}$)-GNN rec-$\mathcal{N}$ and a symbolic logspace graph encoding $\mathfrak{G}(C_n)$, such that for all $n \in \mathbb{N}$ and $\overline{x} \in \mathbb{R}^n$ the feature values of $f_{\mathcal{N}}(\mathfrak{G}(C_n, \overline{x}))$ include $f_{C_n}(\overline{x})$.*

Proof Sketch. For every n, a complete bipartite labelled graph is constructed, such that it has one set of n nodes with the n input values as feature values and one set of m nodes with feature values 1 to m for the m outputs of the circuit C_n. The C-GNN consists of one layer where in all nodes the computations of C_n are executed. For the m nodes the inputs are the feature values of their n neighbouring nodes, i. e. the input values of C_n. Each of the m nodes then contains one of the outputs of C_n determined by the initial feature value of the node. □

As in Theorem 4 the model of C-GNN used here does not use any activation functions besides the identity function. Additional functions from the simulated circuit family are again realised directly in the internal circuits.

It can be shown that there exist functions from some recurrent circuit function class without the previously mentioned restrictions that cannot be simulated by a C-GNN with only inner recurrence that simulates the additional functions form $\mathcal{A}$ only as activation functions. This is based on the fact that an inner recurrent C-GNN has a fixed number of layers that is determined during construction of the network. The lemma also holds for any non-continuous activation function.

Lemma 1. *There exists a rec$[\mathfrak{F}_s]$-$s\mathfrak{F}[\exp]$-circuit family $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$, with $\exp$ the exponential function, where there is no symbolic logspace graph encoding $\mathfrak{G}(C_n)$ such that*

there exists a $(rec[\mathfrak{F}_s]\text{-}\mathfrak{tF}, \{\exp\})$*-GNN rec-*$\mathcal{N}$*, such that the feature values of* $f_{\mathcal{N}}(\mathfrak{G}(C_n, \overline{x})))$ *include* $f_{C_n}(\overline{x})$.

The circuit family in Lemma 1 can be defined to be in predecessor form. By Theorem 5 there exists an outer recurrent C-GNN that has the output of the circuit family among its final feature vectors.

Lemma 1 also holds for an inner recurrent C-GNN where the underlying circuits of the internal recurrent circuits have access to the sign function. It follows that there exist functions that an outer recurrent C-GNN can compute that an inner recurrent C-GNN cannot.

5 Conclusion

In this paper, we introduced a model of recurrent arithmetic circuits and a generalisation of recurrent graph neural networks based on arithmetic circuits. We defined different models of recurrence for GNNs and showed correspondence of those to arithmetic circuits. However, some restrictions needed to be imposed on the particular circuits used in our constructions. In particular, the circuits used in our recurrent C-GNNs needed to be tail-symmetric. We established that inner recurrent C-GNNs can simulate symmetric circuit functions. To simulate recurrent circuits by an outer recurrent C-GNN with activation functions (as in Theorem 5), we restricted the circuits to be in the more restrictive *predecessor form*.

An interesting avenue for further research is the question, whether those restrictions can be relaxed or if it can be proven that they are necessary.

One concept in circuit complexity that we did not touch on is the notion of *uniformity*. Without further restrictions, the individual circuits within one circuit family do not need to have anything in common. This means that in general circuit families do not need to be finitely describable, though that is a desirable property for computational models. A circuit family which can be finitely described is called *uniform*, since its circuits need to be of a uniform structure. It is worth noting that all our proofs are constructive, which means that our circuit families are all uniform in some sense, though the exact nature of that uniformity has yet to be investigated.

A further area of research is to directly study the connections between the different recurrent C-GNN models we introduced and to confirm our conjectures based on our results in relation to arithmetic circuits. An interesting factor here is that all models work on labelled graphs that cannot be altered when directly comparing computations. We conjecture the two models of inner and outer recurrence to be provably incomparable via bisimulation and memory capacity arguments. Likewise, the study of capabilities between different models is interesting under reasonably simple reductions.

Further investigations are required to obtain practical implications of our theoretical results. Once we fix the architecture of our recurrent C-GNNs (i. e., a circuit function class for the GNN nodes), we obtain a specific circuit class that characterises the computational power of the recurrent C-GNN model which can then be rigorously studied.

AI Declaration

The authors have not employed any Generative AI tools.

References

Ahvonen, V.; Heiman, D.; Kuusisto, A.; and Lutz, C. 2024. Logical characterizations of recurrent graph neural networks with reals and floats. In Globersons, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J. M.; and Zhang, C., eds., *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Barceló, P.; Kostylev, E. V.; Monet, M.; Pérez, J.; Reutter, J. L.; and Silva, J. P. 2020. The Logical Expressiveness of Graph Neural Networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Barlag, T.; Holzapfel, V.; Strieker, L.; Virtema, J.; and Vollmer, H. 2024a. Graph neural networks and arithmetic circuits. In Globersons, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J. M.; and Zhang, C., eds., *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Barlag, T.; Holzapfel, V.; Strieker, L.; Virtema, J.; and Vollmer, H. 2024b. Graph neural networks and arithmetic circuits. *CoRR* abs/2402.17805. Extended version with Appendix.

Barlag, T.; Holzapfel, V.; Strieker, L.; Virtema, J.; and Vollmer, H. 2026. Recurrent graph neural networks and arithmetic circuits. *CoRR* abs/2603.05140. Extended version with Appendix.

Barlag, T.; Chudigiewitsch, F.; and Gaube, S. A. 2024. Logical characterizations of algebraic circuit classes over integral domains. *Math. Struct. Comput. Sci.* 34(5):346–374.

Bollen, J.; Van den Bussche, J.; Vansummeren, S.; and Virtema, J. 2025. Halting Recurrent GNNs and the Graded mu-Calculus. In *Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning*, 175–184.

Grohe, M. 2024. The descriptive complexity of graph neural networks. *TheoretiCS* 3.

Kozen, D. 2006. *Theory of Computation*. Texts in Computer Science. Springer.

Lewin, D., and Protheroe, D. 1992. *Design of Logical Systems*. Springer-Science+Business-Media, B.V., 2 edition.

Pflueger, M.; Cucala, D. T.; and Kostylev, E. V. 2024. Recurrent graph neural networks and their connections to bisimulation and logic. In Wooldridge, M. J.; Dy, J. G.; and Natarajan, S., eds., *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, 14608–14616. AAAI Press.

Rosenbluth, E., and Grohe, M. 2025. Repetition makes perfect: Recurrent sum-gnns match message passing limit. *CoRR* abs/2505.00291.

Vollmer, H. 1999. *Introduction to Circuit Complexity - A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer.

Wagner, K. W. 2003. *Theoretische Informatik - eine kompakte Einführung (2. Aufl.)*. Springer.