An Algebraic Characterization of Local Weisfeiler–Leman

Yi Chen[1], Eugenia Ternovska[1]

[1]School of Computing Science, Simon Fraser University, Canada

Abstract

The Weisfeiler–Leman (WL) algorithm was originally introduced as a refinement procedure for analysing graph symmetries and as a tool for graph isomorphism testing. It distinguishes a large class of graphs up to isomorphism. For the k-dimensional WL algorithm, its indistinguishability power coincides with that of the finite-variable fragment of first-order logic with counting quantifiers, C_k.

We study indistinguishability from an algebraic perspective. We introduce a program algebra with counting capabilities and a local version of k-WL, and show that both coincide in expressive power with the guarded fragment of counting logic, GC_k.

Keywords

Graph isomorphism problem, Weisfeiler–Leman algorithm, Colour refinement, Approximate isomorphism testing, Algebraic characterization

1. Introduction

Graph neural networks (GNNs) are deep learning models for graph-structured data that learn relational representations via message passing. While highly effective, their reasoning mechanism remains largely implicit: aggregation over multisets of neighbours captures structural information through counting, but without an explicit symbolic interpretation [1, 2, 3, 4].

The Weisfeiler–Leman (WL) algorithm provides a canonical bridge between combinatorial structure and counting-based reasoning. Originally introduced for graph isomorphism testing [5], WL iteratively refines vertex colours by aggregating neighbourhood information, and its k-dimensional extension refines k-tuples of vertices. WL has been widely studied in graph theory, finite model theory, and machine learning [6, 1, 2, 3]. In finite model theory, WL is closely connected to finite-variable counting logics: 1-WL corresponds to the two-variable guarded counting logic, while for $k \geq 2$, k-WL corresponds to the k-variable counting logic with counting quantifiers [6, 4, 7].

Despite these correspondences, no analogous characterization is known for guarded counting logic (GC), the local counterpart of counting logic in which quantification is restricted by guards. This gap is particularly intriguing in light of recent work on localized variants of WL, which demonstrate that locality can be incorporated into colour refinement while retaining substantial expressive power [8, 9].

[0]This paper is a short version of a manuscript currently submitted to MFCS 2026.

LOGICNN 2026: Workshop on Logical Methods for Neural Network Analysis, July 25, 2026, Lisbon, Portugal

yca455@sfu.ca (Y. Chen); ter@sfu.ca (E. Ternovska)

0009-0003-1353-1775 (Y. Chen); 0000-0003-0751-4031 (E. Ternovska)

A possible obstacle is that the two formalisms propagate information differently. Guarded quantification extends an assignment by moving along a guard relation through an already assigned variable, whereas WL refinement updates colours directly from neighbourhood information. In this paper, we define an algebraic framework whose propagation behaviour mirrors the locality constraints of GC. Based on this fragment, we develop a refinement procedure that captures guarded propagation, moving toward a WL-style characterization of guarded counting logic.

Our goal is to develop a compositional, isomorphism-invariant semantic framework for sequential computation with counting. We propose an algebra of nondeterministic programs over relational structures, equipped with actions, tests, sequential composition, and antidomain (dynamic negation). Unlike counting logics, our language contains no quantifiers or explicit counting operators; instead, counting emerges operationally from program structure. The antidomain operation traces back to dynamic negation in modal and dynamic logic [10, 11].

Contributions. We introduce a new variant of the local k-WL algorithm that combines locality in the input graph with locality induced by the underlying algebra at each iteration. We also consider guarded variants of all three formalisms—our algebra, WL, and counting logic—and prove the following equivalence with respect to their ability to distinguish vertex tuples:

$$\mathsf{lwl}_k^r \equiv \mathrm{GRL}_k^r \equiv \mathrm{GC}_k^r.$$

To the best of our knowledge, this is the first exact characterization of guarded counting logic [3, 12] in terms of a local Weisfeiler–Leman procedure and an algebraic formalism.

2. Algebra and Logic

In this section, we define the syntax and semantics of our algebra, and subsequently introduce the logic built upon it. We begin with an informal overview.

Our algebra represents nondeterministic computations over relational structures. Programs manipulate unary "registers" via atomic actions, and are combined using sequential composition and tests; in addition, the algebra includes a dynamic negation operator inspired by antidomain operations [10, 11].

Crucially, the algebra features a novel construct that induces counting-like behaviour: it restricts successful executions to traces in which exactly one specified register changes. This mechanism enables threshold-style reasoning without explicit numeric quantifiers or arithmetic operations over domain elements.

The semantics is trace-based: each program denotes a set of successful executions (i.e., strings of relational structures). Double negation induces a logic that determines whether a program admits a successful execution from a state (antidomain applied twice gives domain).

A relational structure over a vocabulary τ consists of a universe together with an interpretation of each relation symbol in τ according to its arity. A (labelled) graph can be viewed as a relational structure whose vocabulary comprises a binary edge relation E and, optionally, a collection of unary predicates $P_1, \ldots, P_m$ representing vertex labels. Throughout this paper, we restrict our attention to finite, undirected, simple, vertex-labelled graphs.

Syntax We call our logic *register logic* (RL). RL is based on an algebra of terms. The vocabulary τ is fixed and partitioned into two disjoint parts: $\tau := \tau_{\text{EDB}} \uplus \tau_{\text{REG}}$. Here, τ_{EDB} denotes the vocabulary of the underlying input structure.[1] In this paper, $\tau_{\text{EDB}} := \{ E, =, P_1, P_2, \dots \}$, where E denotes the binary edge relation and each P_i is a unary vertex label predicate. Vocabulary $\tau_{\text{REG}} := \{R_1, \dots, R_n\}$ is a set of unary relational symbols representing registers. The key distinction between τ_{EDB} and τ_{REG} is that interpretations of τ_{EDB} relations (e.g., P_i) remain fixed during execution, whereas the values of registers R_i may change.

Terms RL terms are built from a collection of atomic actions and atomic tests, and are closed under a set of operators. The syntax of well-formed RL terms is given by

$$\text{(RL terms)} \qquad t ::= A \mid T \mid t_1; t_2 \mid \sim t \mid \mathbf{C}_{R_i}(t).$$

Intuitively, terms are programs. Atomic actions A represent elementary computation steps on registers, while atomic tests T check whether the current register values satisfy certain atomic predicates, without modifying them. The construct $t_1; t_2$ denotes sequential composition. The operator $\sim t$ tests whether the term t has no successful execution. The operator $\mathbf{C}_{R_i}(t)$ restricts executions of t to those in which only the contents of R_i may change, and these values are pairwise distinct throughout. We read $\mathbf{C}_{R_i}$ as "Count on R_i". This operator allows us to iterate over distinct values for R_i while keeping the other registers fixed.

We also consider a guarded variant of RL, called guarded register logic (GRL). The syntax of GRL differs only in its set of atomic actions:

$$\text{(GRL terms)} \qquad t ::= A_{\text{guarded}} \mid T \mid t_1; t_2 \mid \sim t \mid \mathbf{C}_{R_i}(t).$$

Atomic actions The atomic actions of RL and GRL are represented by singleton rule sets.[2] Their unique rules are of the following forms:

$$A ::= \{R_i(x) \leftharpoondown\},$$
$$A_{\text{guarded}} ::= \{ R_j(y) \leftharpoondown E(x,y), R_i(x) \} \mid \{ R_j(x) \leftharpoondown R_i(x) \},$$

where $i \neq j$ and $R_i, R_j \in \tau_{\text{REG}}$. The first variant assigns an arbitrary value to the register. There are two kinds of atomic actions in the second (guarded) variant: (1) propagation along edges and (2) copying the contents of registers. Each atomic action represents a nondeterministic modification of the register values, constrained by the associated rules. In the context of graphs, the possible updated values are guarded by the edge relation.[3]

Atomic tests Atomic tests T are defined as

$$T ::= \{ \leftharpoondown E(x,y), R_i(x), R_j(y) \} \mid \{ \leftharpoondown P_j(x), R_i(x) \} \mid \{ \leftharpoondown R_i(x), R_j(x) \}$$

[1] EDB stands for "Extensional Database", a standard term from the Datalog literature, referring to a relational structure given as input.

[2] For readers familiar with Datalog, we note that, unlike Datalog, our rules place only one nondeterministically chosen element into the relation in the head of the rule.

[3] Although $A_{\text{guarded}} \not\subseteq A$, the guarded atomic actions are merely syntactic sugar. Specifically, $\{ R_j(y) \leftharpoondown E(x,y), R_i(x) \}$ abbreviates $\{R_j(x) \leftharpoondown\}; \{\leftharpoondown E(x,y), R_i(x), R_j(y)\}$, while $\{ R_j(x) \leftharpoondown R_i(x) \}$ abbreviates $\{R_j(x) \leftharpoondown\}; \{\leftharpoondown R_i(x), R_j(x)\}$.

where $R_i, R_j \in \tau_{\mathrm{REG}}$. There are three kinds of atomic tests: (1) checking whether R_i and R_j contain adjacent vertices in the input graph, (2) checking whether R_i contains a vertex labeled P_j in the input graph, and (3) equality comparisons between register contents. Tests capture Boolean conditions over register values. They do not modify the values stored in the registers.

Semantics Given a τ_{EDB}-structure Γ, a *state* is a τ-structure that extends Γ by interpreting the symbols in τ_{REG}. Each register contains exactly one vertex, so the interpretation of each $R_i^{\mathfrak{A}}$ is a singleton set. We use $R_i^{\mathfrak{A}} = v$ as a shorthand for $R_i^{\mathfrak{A}} = \{v\}$. Given a tuple $\bar{v} = (v_1, \ldots, v_n)$, we write $\Gamma_{\bar{v}}$ for the extension of Γ satisfying $R_i^{\Gamma_{\bar{v}}} = v_i$ for all i.

In this paper, register values change one at a time. We write $\phi_i(\bar{v}, w)$ for the tuple obtained from $\bar{v}$ by replacing its i-th component with w. Accordingly, we write $\Phi_i(\mathfrak{A}, w)$ for the state obtained from $\mathfrak{A}$ by updating the value of register R_i to w, with $R_i^{\Phi_i(\mathfrak{A},w)} = w$. Thus, $\Phi_i(\mathfrak{A}, w) = \Gamma_{\phi_i(\bar{v},w)}$. Moreover, we write $N(v) := \{\, w \mid (v, w) \in E^{\Gamma} \,\}$ for the set of neighbours of v.

Let W be the set of states, and let W^+ denote the set of finite non-empty sequences over W. A *trace* is an element of W^+. For a trace π, we write $|\pi|$ for its length, $\pi(i)$ for its i-th state (for $1 \le i \le |\pi|$), and $\pi(i, j)$ for its subtrace from i to j. For traces π, π', we write $\pi \cdot \pi'$ for their concatenation. We write $\pi \sqsubseteq \pi'$ if π is a prefix of π'.

The semantics of a term t is then defined as a subset of W^+, the *traces* of t, denoted by $\Delta^{\Gamma}(t)$.

- $\Delta^{\Gamma}(\{R_i(x) \hookleftarrow\}) = \{\, \mathfrak{A} \cdot \Phi_i(\mathfrak{A}, w) \;\mid\; w \in V^{\Gamma} \,\}$,
- $\Delta^{\Gamma}(\{R_j(y) \hookleftarrow E(x, y), R_i(x)\}) = \{\, \mathfrak{A} \cdot \Phi_j(\mathfrak{A}, w) \;\mid\; w \in N(R_i^{\mathfrak{A}}) \,\}$,
- $\Delta^{\Gamma}(\{R_j(x) \hookleftarrow R_i(x)\}) = \{\, \mathfrak{A} \cdot \Phi_j(\mathfrak{A}, R_i^{\mathfrak{A}}) \,\}$,
- $\Delta^{\Gamma}(\{\hookleftarrow E(x, y), R_i(x), R_j(y)\}) = \{\, \mathfrak{A} \;\mid\; R_j^{\mathfrak{A}} \in N(R_i^{\mathfrak{A}}) \,\}$,
- $\Delta^{\Gamma}(\{\hookleftarrow R_i(x), R_j(x)\}) = \{\, \mathfrak{A} \;\mid\; R_i^{\mathfrak{A}} = R_j^{\mathfrak{A}} \,\}$,
- $\Delta^{\Gamma}(\{\hookleftarrow P_j(x), R_i(x)\}) = \{\, \mathfrak{A} \;\mid\; R_i^{\mathfrak{A}} \in P_j^{\Gamma} \,\}$,
- $\Delta^{\Gamma}(t_1; t_2) = \{\, \pi(1, |\pi| - 1) \cdot \pi' \;\mid\; \pi \in \Delta^{\Gamma}(t_1), \pi' \in \Delta^{\Gamma}(t_2) \text{ and } \pi(|\pi|) = \pi'(1) \,\}$,
- $\Delta^{\Gamma}(\sim t) = \{\, \mathfrak{A} \;\mid\; \forall\, \pi \in \Delta^{\Gamma}(t),\ \pi(1) \neq \mathfrak{A} \,\}$,
- $\Delta^{\Gamma}(\mathbf{C}_{R_i}(t)) = \{\, \pi \in \Delta^{\Gamma}(t) \;\mid\; \pi(j) = \Phi_i(\pi(1), w_j) \text{ for distinct } w_1, \ldots, w_{|\pi|} \,\}$.

For a state $\mathfrak{A} \in W$, we define the abbreviation $\Delta^{\Gamma}_{\mathfrak{A}}(t) := \{\, \pi \in \Delta^{\Gamma}(t) \mid \pi(1) = \mathfrak{A} \,\}$, which denotes the set of traces whose initial state is $\mathfrak{A}$. Since every state expands Γ, we may omit the superscript Γ when it is clear from the context and write $\Delta_{\mathfrak{A}}$ instead of $\Delta^{\Gamma}_{\mathfrak{A}}$.

We say that a term t *successfully executes* on a state $\mathfrak{A}$ if it admits at least one execution trace starting from $\mathfrak{A}$, that is, if $\Delta_{\mathfrak{A}}(t)$ is non-empty.

Logic We now obtain a logic by associating each term with a *test term* via the mapping $t \mapsto \sim\!\sim\, t$. For a given initial state $\mathfrak{A}$, the test $\sim\!\sim\, t$ succeeds precisely if there exists an execution trace of t starting from $\mathfrak{A}$. Formally, we define $\mathfrak{A} \models \sim\!\sim t \;:\Longleftrightarrow\; \Delta_{\mathfrak{A}}(t) \neq \emptyset$.
If t is already a test, we may omit the double negation $\sim\!\sim$.

Example 1.

Problem: Extend to 4-Cycle
Given: *A simple vertex-labeled graph* Γ *and an edge(in forms of 2 vertices* w_1, w_2*).*
Question: *Does the edge extend to a 4-cycle?*

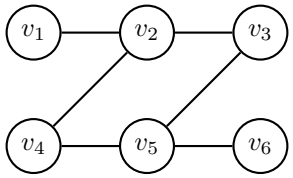

Figure 1: Example graph Γ_1.

We construct a term t such that the edge (w_1, w_2) extends to a 4-cycle if and only if $\Gamma_{(w_1,w_2,w)} \models t$ for every vertex w.

$t_{diagonal} = \mathbf{C}_{R_3}\big((\{\, R_3(y) \leftharpoondown E(x,y), R_1(x) \,\}; \{\, \leftharpoondown E(x,y), R_2(x), R_3(y) \,\})^2\big)$[4]

The term $t_{diagonal}$ checks whether the vertices stored in R_1 and R_2 have at least two common neighbours, and hence can serve as opposite vertices of a 4-cycle.

$t = \{\, \leftharpoondown E(x,y), R_1(x), R_2(y) \,\}; \{\, R_3(y) \leftharpoondown E(x,y), R_2(x) \,\}; \{\, R_2(x) \leftharpoondown R_3(x) \,\}; t_{diagonal}$

The term t propagates the vertex stored in R_2 to one of its neighbours and then checks whether the resulting vertices in R_1 and R_2 can serve as a diagonal of a 4-cycle.

For instance, the following trace on the graph Γ_1 shown in Figure 1 successfully executes t.

$$\begin{bmatrix} R_1: & v_2 & v_2 & v_2 & v_2 & v_2 \\ R_2: & v_4 & v_4 & v_5 & v_5 & v_5 \\ R_3: & w & v_5 & v_5 & v_3 & v_4 \end{bmatrix},$$

where each column represents a state and each row records the value of a register throughout the trace. Since w is arbitrary, this trace witnesses that the edge (v_2, v_4) extends to a 4-cycle.

3. Depth of a Term

In this section, we introduce the notion of depth of a term, which corresponds to the refinement level in the colour refinement algorithm and to the quantifier rank in counting logic.

In the colour refinement algorithm, the number of refinement iterations determines how far the colouring deviates from the initial assignment. Similarly, in counting logic, the quantifier rank controls the number of nested variable substitutions. To capture this intuition for algebraic terms, we introduce a notion of depth that bounds how far register values can influence the evaluation of a term.

At first glance, the length of a trace—i.e., the number of atomic actions executed—appears to be a natural measure of depth, since each atomic action modifies a register. However, dynamic negation does not fully align with this intuition: a term consisting of a single test always produces a trace of length 1, even though satisfying the test may require the hypothetical execution of multiple atomic actions. To account for this discrepancy, we introduce two notions of depth: *actual depth*, which records the maximum length of executed traces, and *depth*, which captures the maximum length including hypothetical executions.

Definition 1.

[4]For a term t, the notation t^n denotes the n-fold composition of t with itself.

- *Atomic actions have depth* 1 *and actual depth* 1.
- *Atomic tests have depth* 0 *and actual depth* 0.

Suppose terms $t_0, \ldots, t_k$ *have depth* $x_0, \ldots, x_k$ *and actual depth* $y_0, \ldots, y_k$*, respectively. Then:*

- *The sequential composition* $t_1; t_2$ *has depth* $\max\{x_1,\ y_1 + x_2\}$ *and actual depth* $y_1 + y_2$.
- *The negation* $\sim t_1$ *has depth* x_1 *and actual depth* 0.
- *If* $t := \mathbf{C}_{R_i}(t_0; m_1; \ldots; m_k; t_k)$*, where each* m_j *is an atomic action and each* t_j *is a test, then* t *has depth* $\max\{x_0,\ x_1 + 1,\ \ldots,\ x_k + 1\}$ *and actual depth* 1.

Intuitively, an execution can be viewed as a labeled computation tree. Nodes correspond to states, which may be actual states appearing in the execution trace or hypothetical states introduced during the evaluation of tests. Edges are labeled by atomic actions. Branching arises from the counting operator $\mathbf{C}$, which aggregates multiple possible continuations. In this view, the depth of a term corresponds to the height of the resulting computation tree, while the *actual depth* is defined as the maximal depth of nodes that correspond to actual (trace-reachable) states.

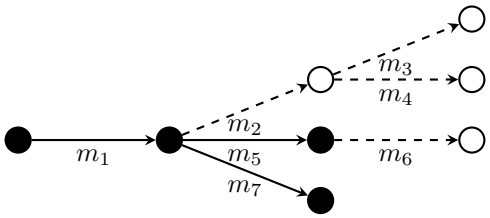

Figure 2: Tree(t), where thick edges indicate connections between the actual nodes, and dotted edges indicate other edges.

Example 2. *Let* $m_1, \ldots, m_7$ *be atomic actions. The syntactic tree of the term*

$$m_1; \mathbf{C}_{R_1}\Big(\sim \big(m_2; \sim m_3; m_4\big); m_5; \sim m_6; m_7\Big)$$

is illustrated in Figure 2. The term has depth 3 and actual depth 2.

Definition 2. *We define the fragment* GRL_k^d *is the set of guarded terms that use at most* k *registers and have depth at most* d.

4. Main Results

In this section, we introduce a local version of the Weisfeiler–Leman algorithm and compare its distinguishing power with our algebra and guarded counting logic. We begin by giving the formal definitions.

The *counting logic* (C) [3, 12] is a variant of first-order logic extended with counting quantifiers of the form $\exists^{\geq k}$. Its guarded fragment (GC) restricts quantification to be guarded by the edge relation, i.e., formulas of the form $\exists^{\geq k} x\,(E(y,x) \wedge \varphi)$, where all free occurrences of x in φ are within the scope of the guard $E(y,x)$. We write GC_k^r for the fragment using at most k variables and quantifier rank at most r.

Definition 3. *We say that* GC_k^r *distinguishes vertex tuples* $\bar{u}, \bar{v}$ *of* Γ, Γ' *if there exists a formula* $\varphi \in \mathrm{GC}_k^r$ *such that* $\Gamma \models \varphi(\bar{u}) \iff \Gamma' \not\models \varphi(\bar{v})$.

The WL algorithm was originally introduced as a heuristic for graph isomorphism testing [5, 6]. The k-dimensional WL algorithm (k-WL) is a fundamental tool in this area: if two graphs yield different stable colourings, then they are not isomorphic. Although k-WL is not complete for graph isomorphism, it provides a powerful polynomial-time approximation that distinguishes many non-isomorphic graphs. We introduce a local variant of it that captures the aggregation behaviour of the guarded fragment of our algebra.

For a graph $\Gamma = (V^\Gamma, E^\Gamma, P_1^\Gamma, \ldots, P_m^\Gamma)$, we write V for V^Γ for simplicity. The k-WL algorithm generates a sequence of colouring functions $\{\mathsf{wl}_k^r\}_{r\in\mathbb{N}}$, where r denotes the refinement level. They are defined as follows:

- $\mathsf{lwl}_k^0(\bar{v}) = \mathsf{atp}(\bar{v})$, where $\mathsf{atp} : V^k \to \{0,1\}^{2\binom{k}{2}+km}$ is a vector that, for $1 \le i < j \le k$, has two entries indicating whether $v_i = v_j$ and whether $E(v_i, v_j)$, and for $1 \le i \le k$, has m entries indicating whether $P_\ell(v_i)$. We call this the atomic type of $\bar{v}$.
- For $r \ge 1$, $\mathsf{lwl}_k^r(\bar{v}) := (\mathsf{lwl}_k^{r-1}(\bar{v}), M_{11}^{r-1}(\bar{v}), M_{12}^{r-1}(\bar{v}), \ldots, M_{kk}^{r-1}(\bar{v}))$ where

$$M_{ij}^{r-1}(\bar{v}) := \begin{cases} \big(\mathsf{lwl}_k^{r-1}(\phi_i(\bar{v}, v_j)), \{\!\{\, \mathsf{lwl}_k^{r-1}(\phi_i(\bar{v}, w)) \mid w \in N(v_j) \,\}\!\}\big) & \text{if } i \neq j \\ \emptyset & \text{if } i = j \end{cases}$$

  where $\{\!\{ \cdot \}\!\}$ denotes a *multiset*, i.e., elements may occur with multiplicity.
  Each $M_{ij}^{r-1}(\bar{v})$ collects the colours of all possible states obtained by applying atomic actions to $\Gamma_{\bar{v}}$. Here, $\Phi_i(\Gamma_{\bar{v}}, v_j)$ is the unique state resulting from the atomic action $\{ R_i(x) \leftharpoonup R_j(x) \}$, while $\{\!\{ \Phi_i(\Gamma_{\bar{v}}, w) \mid w \in N(v_j) \}\!\}$ is the multiset of all possible states resulting from $\{ R_i(y) \leftharpoonup E(x,y), R_j(x) \}$.

Definition 4. *We say that* lwl_k^r *distinguishes vertex tuples* $\bar{u}, \bar{v}$ *of* Γ, Γ' *if* $\mathsf{lwl}_k^r(\bar{u}) \neq \mathsf{lwl}_k^r(\bar{v})$.

Definition 5. *We say that* GRL_k^r *distinguishes vertex tuples* $\bar{u}, \bar{v}$ *of* Γ, Γ' *if there exists a term* $t \in \mathrm{GRL}_k^r$ *such that* $\Gamma_{\bar{u}} \models {\sim\sim}\, t \iff \Gamma'_{\bar{v}} \not\models {\sim\sim}\, t$.

We write $A_1 \sqsupseteq A_2$ if A_1 distinguishes all vertex tuples that A_2 distinguishes, that is, if A_1 has greater expressive power than A_2. Moreover, we define $A_1 \equiv A_2$ if both directions hold.[5]

Theorem 1 ($\mathrm{GRL}_k^r \sqsupseteq \mathrm{GC}_k^r$)**.** *Given an underlying graph* Γ *and a formula* φ *from* GC_k^d*. There exists a guarded test term* $t \in \mathrm{GRL}_k^d$ *such that, for any tuple* $\bar{v}$*, we have*

$$\Gamma \models \varphi(\bar{v}) \quad \iff \quad \Gamma_{\bar{v}} \models t.$$

Proof. We proceed by structural induction on the formula φ to prove the theorem. Assume the statement holds for the subformulas φ_1 and φ_2 with test terms t_1, t_2 respectively.

- **Case 1.** $\varphi = \exists^{\ge n} x_i \big(E(x_j, x_i) \wedge \varphi_1\big)$. Therefore $qr(\varphi) = qr(\varphi_1) + 1$. We define

$$t = {\sim\sim}\, \mathbf{C}_{R_i}\Big(\big(\{R_i(x) \leftharpoonup E(x,y), R_j(y)\}; t_1\big)^n\Big).$$

[5] In [9], the authors use $A_1 \sqsubseteq A_2$ instead.

Here, the superscript n denotes the n-fold composition of the enclosed term. Then,

$\Gamma \models (\exists^{\geq n} x_i\, \varphi_1)(\bar{v})$

$\iff$ there exist n distinct $w \in N(v_j)$ such that $\Gamma \models \varphi_1(\phi_i(\bar{v}, w))$,

$\iff$ there exist n distinct $w \in N(v_j)$ such that $\Phi_i(\Gamma_{\bar{v}}, w) \models t_1$ by induction,

$\iff$ there exist distinct $w^1, \ldots, w^n \in N(v_j)$ such that $\Gamma_{\bar{v}} \cdot \Phi_i(\Gamma_{\bar{v}}, w^1) \cdot \ldots \cdot \Phi_i(\Gamma_{\bar{v}}, w^n)$ is a trace of $\big(\{R_i(x) \leftarrow E(x,y), R_j(y)\}; t_1\big)^n$,

$\iff \Gamma_{\bar{v}} \models {\sim}{\sim}\, \mathbf{C}_{R_i}\Big(\big(\{R_i(x) \leftarrow E(x,y), R_j(y)\}; t_1\big)^n\Big)$.

By Definition 1, the depth of t is the depth of $\{R_i(x) \leftarrow E(x,y), R_j(y)\}; t_1$, and thus equals the depth of t_1 plus 1, which is $qr(\varphi)$.

- **Case 2.** $\varphi = \varphi_1 \wedge \varphi_2$. In this case, we define $t = t_1; t_2$.
- **Case 3.** $\varphi = \neg\varphi_1$. In this case, we define $t = {\sim}\, t_1$. □

Theorem 2 ($\mathrm{GC}_k^r \sqsupseteq \mathsf{lwl}_k^r$)**.** *Given a vertex tuple $\bar{w} \in V^k$ and an integer r, there exists a formula $\varphi \in \mathrm{GC}_k^r$ that characterizes the colour $\mathsf{lwl}_k^r(\bar{w})$, that is, for all $\bar{v}$,*

$$\Gamma \models \varphi(\bar{v}) \iff \mathsf{lwl}_k^r(\bar{v}) = \mathsf{lwl}_k^r(\bar{w}).$$

Proof. Suppose the statement holds for $r-1$. We show how to characterize the multiset

$$\{\!\{\, \mathsf{lwl}_k^{r-1}(\phi_i(\bar{w}, u)) \mid u \in N(w_j) \,\}\!\}, \text{ where } i \neq j.$$

We group the vertices of $N(w_j)$ by colour; suppose there are n groups with representatives $u^1, \ldots, u^n$, and each group has size $k_1, \ldots, k_n$, respectively. For each u^l, by the induction hypothesis, there exists a formula φ_{i,u^l} characterizing the colour $\mathsf{lwl}_k^{r-1}(\phi_i(\bar{w}, u^l))$.

Consider the formula $\varphi_{ij} := (\exists^{\geq k_1} x_i\, E(x_i, x_j) \wedge \varphi_{i,u^1}) \wedge \cdots \wedge (\exists^{\geq k_n} x_i\, E(x_i, x_j) \wedge \varphi_{i,u^n})$.

$\Gamma \models \varphi_{ij}(\bar{v})$

$\iff$ there exist k_l distinct $u \in N(v_j)$ such that $\Gamma \models \varphi_{i,u^l}(\phi_i(\bar{v}, u))$ for $1 \leq l \leq n$

$\iff$ there exist k_l distinct $u \in N(v_j)$ such that $\mathsf{lwl}_k^{r-1}(\phi_i(\bar{v}, u)) = \mathsf{lwl}_k^{r-1}(\phi_i(\bar{w}, u^l))$ for $1 \leq l \leq n$

$\iff \{\!\{\, \mathsf{lwl}_k^{r-1}(\phi_i(\bar{v}, u)) \mid u \in N(v_j) \,\}\!\} \supseteq \{\!\{\, \mathsf{lwl}_k^{r-1}(\phi_i(\bar{w}, u)) \mid u \in N(w_j) \,\}\!\}$.

Moreover, $\Gamma \models \neg\exists^{\geq |N(w_j)|+1} x_i\, E(x_i, x_j)(\bar{v}) \iff |N(v_j)| \leq |N(w_j)|$. Thus, their exists a formula that characterizes the multiset $\{\!\{\, \mathsf{lwl}_k^{r-1}(\phi_i(\bar{w}, u)) \mid u \in N(w_j) \,\}\!\}$. The remaining details are straightforward and are omitted. □

Lemma 1. *Suppose $\mathsf{lwl}_k^r(\bar{u}) = \mathsf{lwl}_k^r(\bar{v})$. Consider colour $C = \mathsf{lwl}_k^{r-1}(\phi_i(\bar{u}, w))$ for some $w \in N(u_j)$. Then,*

$$|\{\, w \in V \mid \mathsf{lwl}_k^{r-1}(\phi_i(\bar{u}, w)) = C \,\}| = |\{\, w \in V' \mid \mathsf{lwl}_k^{r-1}(\phi_i(\bar{v}, w)) = C \,\}|.$$

The lemma states that tuples of the same colour have the same number of neighbouring tuples of every colour. This follows directly from the fact that $\{\, w \in V \mid \mathsf{lwl}_k^{r-1}(\phi_i(\bar{u}, w)) = C \,\} = \{\, w \in N(u_j) \mid \mathsf{lwl}_k^{r-1}(\phi_i(\bar{u}, w)) = C \,\}$, since the colour C contains the atomic-type information specifying whether w is adjacent to u_j. Combined with $\{\!\{\, \mathsf{lwl}_k^{r-1}(\phi_i(\bar{u}, w)) \mid w \in N(u_j) \,\}\!\} = \{\!\{\, \mathsf{lwl}_k^{r-1}(\phi_i(\bar{v}, w)) \mid w \in N(v_j) \,\}\!\}$, the lemma follows immediately.

Lemma 2. *For any term of the form $\mathbf{C}_{R_i}(t)$, there exist atomic actions $m_1, \ldots, m_k$ and tests $t_0, \ldots, t_k$ such that $\Delta(\mathbf{C}_{R_i}(t)) = \Delta(\mathbf{C}_{R_i}(t_0; m_1; \ldots; m_k; t_k))$*

The lemma states that nested applications of the $\mathbf{C}$ operator can be eliminated. Intuitively, the $\mathbf{C}$ operator restricts changes to a single register, rendering traces satisfying multiple nested $\mathbf{C}$ operators trivial. We omit the proof.

Theorem 3 ($\mathsf{lwl}_k^r \sqsupseteq \mathrm{GRL}_k^r$)**.** *Suppose $\bar{u} \in V^k$ and $\bar{v} \in V'^k$ are two tuples such that $\mathsf{lwl}_k^r(\bar{u}) = \mathsf{lwl}_k^r(\bar{v})$. Let t be a guarded term of depth $d \leq r$ using k registers. Then, for every trace $\pi \in \Delta_{\Gamma_{\bar{u}}}(t)$, there exists a trace $\pi' \in \Delta_{\Gamma'_{\bar{v}}}(t)$ such that*

$$\mathsf{lwl}_k^{r-a}(\pi(|\pi|)) = \mathsf{lwl}_k^{r-a}(\pi'(|\pi'|)).$$

where a is the actual depth of t.

Proof. We proceed by structural induction and omit routine details.

- **Case 1.** t is an atomic test. We simply take $\pi' = \mathfrak{B}$.
- **Case 2.** $t := \{R_i(x) \leftharpoonup E(x, y), R_j(x)\}$ is an atomic action. Then $\pi = \mathfrak{A} \cdot \Phi_i(\mathfrak{A}, u')$ for some u'. Since $\{\!\{ \mathsf{lwl}_k^{r-1}(\phi_i(\bar{u}, w)) \mid w \in N(u_j) \}\!\} = \{\!\{ \mathsf{lwl}_k^{r-1}(\phi_i(\bar{v}, w)) \mid w \in N(v_j) \}\!\}$, we may take $\pi' = \mathfrak{B} \cdot \Phi_i(\mathfrak{B}, v')$, where $\phi_i(\bar{v}, v')$ has the same colour as $\phi_i(\bar{u}, u')$.
- **Case 3.** $t := t_1; t_2$. Then $\pi = \pi_1 \cdot \pi_2$. By the induction hypothesis, there exist traces π'_1 and π'_2 corresponding to π_1 and π_2, respectively. Hence, we may take $\pi' = \pi'_1 \cdot \pi'_2$.
- **Case 4.** $t :=\!\sim t'$. We simply take $\pi' = \mathfrak{B}$.
- **Case 5.** $t := \mathbf{C}_{R_i}(t')$. By induction, there exists a trace $\pi' \in \Delta_{\mathfrak{B}}(t')$. If π' contains no repeated states, then $\pi' \in \Delta_{\mathfrak{B}}(\mathbf{C}_{R_i}(t'))$. Otherwise, we construct a trace π'' by replacing repeated states with states of the same colour, i.e., $\mathsf{lwl}_k^{r-1}(\pi''(i)) = \mathsf{lwl}_k^{r-1}(\pi'(i))$. By Lemma 1, there are sufficiently many such states to eliminate repetitions. By Lemma 2, t' does not contain $\mathbf{C}$ operator and $\pi'' \in \Delta(t')$ follows by induction on the other cases. □

The main theorem follows as a corollary.

Theorem 4.

$$\mathsf{lwl}_k^r \equiv \mathrm{GRL}_k^r \equiv \mathrm{GC}_k^r$$

Conclusion. We introduced a compositional algebraic framework for reasoning with counting over relational structures, and established its exact correspondence with guarded counting logic and a local variant of the Weisfeiler–Leman algorithm. This provides a unified perspective on counting, locality, and compositionality. Several directions for future work remain, including extensions to richer logics and algorithmic applications.

Declaration on Generative AI

The authors used ChatGPT-4 for language editing only. After using these tools, the authors reviewed and edited the content and take full responsibility for the publication's content.

References

[1] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, in: International Conference on Learning Representations (ICLR), 2019. URL: https://openreview.net/forum?id=ryGs6iA5Km.

[2] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, M. Grohe, Weisfeiler and leman go neural: Higher-order graph neural networks, in: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), volume 33(01), 2019, pp. 4602–4609. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4664.

[3] P. Barceló, E. V. Kostylev, M. Monet, J. Pérez, J. L. Reutter, J. P. Silva, The logical expressiveness of graph neural networks, in: 8th International Conference on Learning Representations (ICLR 2020), Addis Ababa, Ethiopia, 2020. URL: https://openreview.net/forum?id=H1gYWsCcKX.

[4] M. Grohe, The logic of graph neural networks, in: Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), IEEE, 2021, pp. 1–17. URL: https://arxiv.org/abs/2104.14624. doi:10.1109/LICS52264.2021.9470677.

[5] B. Weisfeiler, A. Leman, The reduction of a graph to canonical form and the algebra which appears therein, Nauchno-Technicheskaya Informatsia, Series 2 (1968) 12–16. URL: https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf, english translation by G. Ryabov is available.

[6] J. Cai, M. Fürer, N. Immerman, An optimal lower bound on the number of variables for graph identification, Combinatorica 12 (1992) 389–410.

[7] N. Immerman, E. Lander, Describing graphs: A first-order approach to graph canonization, in: Complexity Theory Retrospective, Springer, Berlin, 1990, pp. 59–81.

[8] A. Kumar, S. Das, S. Roy, B. Maity, A. Dasgupta, Improving expressivity of graph neural networks using localization, arXiv preprint arXiv:2305.19659 (2023).

[9] C. Morris, G. Rattan, P. Mutzel, Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings, in: Advances in Neural Information Processing Systems 33 (NeurIPS 2020), 2020, pp. 21824–21840. URL: https://papers.nips.cc/paper/2020/hash/f81dee42585b3814de199b2e88757f5c-Abstract.html.

[10] M. Hollenberg, A. Visser, Dynamic negation, the one and only, J. Log. Lang. Inf. 8 (1999) 137–141.

[11] B. McLean, Complete representation by partial functions for composition, intersection and anti-domain, J. Log. Comput. 27 (2017) 1143–1156.

[12] M. de Rijke, A note on graded modal logic, Studia Logica 64 (2000) 271–283. doi:10.1023/A:1005245900406.