EXPRESSIVE POWER OF HOMOMORHPISM QUERY ALGORITHMS

MSc Thesis *(Afstudeerscriptie)*

written by

Arnar Ágúst Kristjánsson
(born August 1, 2001 in Reykjavík, Iceland)

under the supervision of **Dr. Balder ten Cate** and **Prof. Dr. Phokion G. Kolaitis**, and submitted to the Examinations Board in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
June 27, 2025	Dr. Maria Aloni (chair)
	Dr. Benno van den Berg
	Dr. Balder ten Cate (supervisor)
	Dr. Ronald de Haan
	Prof. Dr. Phokion G. Kolaitis (supervisor)

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Abstract

A query algorithm based on homomorphism counts is a procedure for deciding membership in a class of finite relational structures using only homomorphism count queries. A left query algorithm can ask for the number of homomorphisms from any structure *to* the input structure, while a right query algorithm can ask for the number of homomorphisms *from* the input structure to any other structure.

We systematically compare the expressive power of various types of left and right query algorithms, including non-adaptive query algorithms, adaptive query algorithms that can ask a bounded number of queries, and adaptive query algorithms that can ask an unbounded number of queries. Among other results, we show that there exist classes that cannot be decided by adaptive left query algorithms with a bounded number of queries. We also provide a complete comparison of the expressive power of adaptive and non-adaptive left query algorithms.

We further consider query algorithms in which the homomorphism counting is done over the Boolean semiring $\mathbb{B}$, so that only the existence of a homomorphism is recorded, not the precise number of them. We characterize the expressive power of adaptive unbounded query algorithms over $\mathbb{B}$ and use this characterization to derive simpler descriptions in the special cases of homomorphic equivalence classes, CSPs, and Datalog-definable classes.

Finally, we investigate the question of whether counting over $\mathbb{N}$, rather than $\mathbb{B}$, increases the expressive power of these query algorithms. We answer this question affirmatively for all adaptive query algorithms considered.

Acknowledgements

First and foremost, I would like to thank my supervisors, Balder and Phokion. Balder, for taking me under his wing throughout this project, for introducing me to the topic, listening to my ideas, offering thoughtful feedback, and guiding me through the final steps of my master's and beyond. Phokion, for generously sharing his deep expertise and providing insightful and constructive comments. I am also grateful to Víctor Dalmau and Susanna de Rezende for their suggestions on how to improve the arguments presented in this thesis. Lastly, I want to thank my fiancée, Marín, whose presence and support made every day of this journey more meaningful.

Contents

Introduction

The subject of this thesis falls within the field of *finite model theory*, which studies the interaction between finite models and logical languages. The models most commonly studied in finite model theory, and the ones we focus on in this thesis, are *relational structures*, which are sets equipped with relations. A simple example of a relational structure is a directed graph. However, the concept is very general: relational structures can capture data of almost any kind. This is exemplified by the fact that the most commonly used database model, the relational model, essentially treats a database as a relational structure.

Homomorphisms are functions between relational structures that preserve all of the relations. Homomorphisms are central to finite model theory as highlighted by one of the field's most celebrated results: the *finite homomorphism preservation theorem*, proved by Rossman [27]. It states that a *first-order logic formula* is preserved under homomorphisms on finite structures if and only if the formula is equivalent to a *positive-existential* formula. Homomorphism counts—that is, the number of homomorphisms between two relational structures—have also proven to be deeply connected to the expressive power of logical languages and have yielded many new insights within finite model theory.

A classic result by Lovász [24] states that a finite relational structure A is determined up to isomorphism by the numbers $\hom(B, A)$ of homomorphisms from B to A for every finite relational structure B with the same signature. This list of numbers is called the *left homomorphism profile* of A and is written as $\hom(\mathcal{C}, A)$, where $\mathcal{C}$ is the class of all finite relational structures with the same signature as A. Similarly, one can define the *right homomorphism profile* as the list of numbers of homomorphisms from A to each structure. The right profile also determines the structure up to isomorphism, as shown by Chaudhuri and Vardi [12].

Recently, there has been considerable interest in exploring which structures can be distinguished when the class $\mathcal{C}$ is restricted. As an example, in 2010, Dvořák [15] proved that if $\mathcal{C}$ is the class of graphs of *treewidth* at most k, the left profile distinguishes precisely the same pairs of non-isomorphic graphs as the *k-dimensional Weisfeiler-Leman algorithm*, a well-known polynomial time graph isomorphism test. Cai, Fürer, and Immerman [7] had already proved that the $k + 1$ variable fragment of first-order logic extended with *counting quantifiers* can distinguish precisely the same graphs as the k-dimensional Weisfeiler-Leman algorithm. Also, in 2020, Grohe [20] proved that first-order logic with counting of *quantifier rank* at most k can distinguish exactly the same graphs as the left homomorphism profile restricted to graphs of *treedepth* k. Mančinska and Roberson [25] showed that the graphs that are *quantum isomorphic* are exactly the graphs that have the same left homomorphism profile restricted to planar graphs. In his 2023 Master of Logic thesis, Comer [14] showed that the equivalence of *labeled transition systems* with respect to some modal languages can also be captured with restrictions to the homomorphism profile. In a 2024 paper, Seppelt [28] provides further insight into the logical equivalence relations for graphs that can be characterized by restricted left homomorphism profiles, using techniques from structural graph theory. These results highlight the importance of homomorphism profiles in finite model theory by describing a close connection between homomorphism profiles and logical languages.

Taking a slightly different approach, in 2022, Chen et al. [13] ask what properties of graphs can be decided based on *finitely many* homomorphism count queries. They did this by introducing the notion of a homomorphism-count based *query algorithm.* A *non-adaptive left k-query algorithm* (also referred to as a left k-query algorithm) consists of a finite tuple $\mathcal{F} = (F_1, \ldots, F_k)$ of structures and a subset $X \subseteq \mathbb{N}^k$. The algorithm is said to decide the class of structures

$$\mathcal{A} \coloneqq \{A : (\hom(F_i, A))_{i=1}^k \in X\}.$$

Non-adaptive *right* k-query algorithms are defined similarly, but using $\hom(A, F_i)$ instead of $\hom(F_i, A)$. These notions allow for the study of what classes of structures can be defined using a finite restriction of the left and right homomorphism profiles, respectively. Chen et al. also define an *adaptive* version of left/right k-query algorithms. These are, roughly speaking, algorithms that can ask k homomorphism count queries where the choice of the $(i+1)$-th query may depend on the answers to the first i queries. Note that similar notions had already been explored by Bielecki and Van den Bussche [4]. Chen et al. [13] studied the expressive power of non-adaptive and adaptive query algorithms for the specific case of *simple graphs* (i.e., undirected graphs without self-loops). Among other things, they show that every first-order sentence ϕ that is a Boolean combination of universal first-order sentences can be decided by a k-left query algorithm for some k. On the other hand, they show that there are first-order sentences ϕ that cannot be decided by a non-adaptive left k-query algorithm for any k. When it comes to adaptive query algorithms, they show that every class of simple graphs can be decided by an adaptive left 3-query algorithm, but that there exists a class of simple graphs that is not decided by an adaptive right k-query algorithm for any k, when using only simple graphs for the queries.

In a 2024 paper, ten Cate et al. [8] extend the above analysis by exploring query algorithms for arbitrary relational structures (not only simple graphs) and by also considering query algorithms over the Boolean semiring $\mathbb{B}$, which means that instead of being able to query for the number of homomorphisms, the algorithm can only query for the existence of a homomorphism. The query algorithms as defined by Chen et al. [13] can be viewed as query algorithms over the semiring $\mathbb{N}$ of natural numbers. It is shown in [8] that, for classes of structures that are closed under homomorphic equivalence, non-adaptive left k-query algorithms (over $\mathbb{N}$) are no more powerful than non-adaptive left k-query algorithms over $\mathbb{B}$. In other words, "counting does not help" for such classes. Moreover, for such classes, non-adaptive left query algorithms are equivalent in expressive power to first-order logic.

Besides being interesting from a purely mathematical standpoint, the study of finite homomorphism profiles also has applications in other areas, most notably in database theory. Relational databases can be viewed as relational structures. In this setting, conjunctive queries—which are among the most fundamental types of database queries—correspond to homomorphism queries. When counting is done over the natural numbers, homomorphism queries correspond to conjunctive queries under *bag semantics*; when counting is done over the Boolean semiring $\mathbb{B}$, they correspond to conjunctive queries under *set semantics.* Thus, questions concerning the expressive power of homomorphism query algorithms can reveal insights about the expressive power of database query languages.

Finite homomorphism profiles have also recently attracted attention in the domain of artificial intelligence. Morris et al. [26] and Xu et al. [31] independently showed that *graph neural networks* (GNNs) can distinguish exactly the same structures as the 1-dimensional Weisfeiler-Leman algorithm, which in turn distinguishes the same structures as the homomorphism profile restricted to the class of trees. Barceló et al. [3] demonstrated that augmenting GNNs with the output of a finite number of homomorphism queries can significantly increase their expressive power and efficacy, with minimal computational overhead. Consequently, homomorphism profiles and their finite restrictions have become important tools in the analysis and design of GNNs.

Summary of contributions. This thesis studies the expressive power of various versions of homomorphism-count based query algorithms. It also considers the previously unexplored notion of adaptive *unbounded* query algorithms, i.e., adaptive query algorithms for which there is no uniform bound on the number of queries they can ask on a given input, but which nevertheless terminate on every input.

The main contributions are along two lines:

1. The symmetry of directed cycles and cycle-like structures is exploited to give a simple formula for the number of homomorphisms from a structure to a disjoint union of cycle-like structures of the same size. This is used to systematically investigate and compare the expressive power of adaptive and non-adaptive query algorithms over $\mathbb{N}$ for relational structures. For instance, in Theorem 3.15, it is shown that for every signature containing a non-unary relation, there exists a class of structures of that signature that is not decided by an adaptive left k-query algorithm over $\mathbb{N}$ for any k. A concrete example of such a class is the class of directed graphs whose number of connected components is an even power of 2. This result is in sharp contrast to the result of Chen et al., which states that every class of simple graphs can be decided by an adaptive left 3-query algorithm over $\mathbb{N}$. In Theorem 3.16, it is also shown that for each $k > 1$ there exists a class that is decided by a non-adaptive left k-query algorithm over $\mathbb{N}$ but not by any adaptive left $(k-1)$-query algorithm over $\mathbb{N}$. However, in Theorem 3.17 we also note that there exists a class that is decided by an adaptive left 2-query algorithm but not by any non-adaptive left query algorithm. We thus establish all inclusion and non-inclusion relations between the expressive powers of adaptive and non-adaptive left k-query algorithms over $\mathbb{N}$ (note: it follows from the proof of Lovász's theorem that every class is decided by an adaptive left unbounded query algorithm). The results are summarized in Figure 1. Adaptive left $f(n)$-query algorithms are also explored. These are query algorithms that can use $f(n)$ queries for inputs of size n, where f is a function on the natural numbers. The question of how fast f must grow for adaptive left $f(n)$-query algorithms to be able to decide any class is addressed. In Theorem 3.22, a lower bound is established for the asymptotic growth of such a function. An upper bound for f, derived from Lovász's proof, is also provided.

   Interestingly, the situation is quite different for *right* query algorithms over $\mathbb{N}$. We show that it follows from results in [30] that every class of structures can be decided with only two adaptive right queries. This is again in contrast to a prior result of Chen et al. [13], namely that there is no k such that every class of simple graphs can be decided by an adaptive right k-query algorithm (when the queries themselves are also required to be simple graphs).

$$\begin{array}{ccccccccccc}
\mathfrak{A}_1 & \subsetneq & \mathfrak{A}_2 & \subsetneq & \cdots & \mathfrak{A}_k & \cdots & \subsetneq & \bigcup_k \mathfrak{A}_k & \subsetneq & \mathfrak{A}_{unb} = \text{All classes of structures} \\
\parallel & & \rotatebox{90}{$\subsetneq$} & & & \rotatebox{90}{$\subsetneq$} & & & \rotatebox{90}{$\subsetneq$} & & \\
\mathfrak{N}_1 & \subsetneq & \mathfrak{N}_2 & \subsetneq & \cdots & \mathfrak{N}_k & \cdots & \subsetneq & \bigcup_k \mathfrak{N}_k & &
\end{array}$$

Figure 1: Summary of our results regarding the relative expressive power of different types of left query algorithms over $\mathbb{N}$. Here, $\mathfrak{N}_k$, $\mathfrak{A}_k$, and $\mathfrak{A}_{unb}$ denote the collections of classes of structures that admit a non-adaptive left k-query algorithm, an adaptive left k-query algorithm, and an adaptive left unbounded query algorithm, respectively. No inclusions hold other than those depicted. In particular, $\mathfrak{A}_2 \not\subseteq \bigcup_k \mathfrak{N}_k$ and $\mathfrak{N}_{k+1} \not\subseteq \mathfrak{A}_k$.

2. We investigate the expressive power of adaptive query algorithms over $\mathbb{B}$. We observe that adaptive *bounded* query algorithms over $\mathbb{B}$ have the same expressive power as *non-adaptive* query algorithms over $\mathbb{B}$, whose expressive power was characterized by ten Cate et al. [8]. However, we show that adaptive unbounded query algorithms over $\mathbb{B}$ are strictly more expressive. The classes decided by such algorithms exhibit an intricate pattern, which we show can be captured by a topological characterization. In the case of homomorphism-closed classes, we further show that the characterization can be simplified significantly, yielding an elegant condition for establishing whether such a class is decided by an adaptive unbounded query algorithm over $\mathbb{B}$ or not.

   We also apply the topological characterization to examine the expressive power in the special cases of classes that are homomorphism equivalence types, CSPs, or Datalog-definable. Specifically, we show that unboundedness does not increase the expressive power of adaptive left query algorithms over $\mathbb{B}$ for deciding CSPs and homomorphic-equivalence types, thereby yielding, based on existing results, an effective criterion for determining whether such classes are decided by an adaptive left unbounded query algorithm over $\mathbb{B}$. Additionally, we characterize the Datalog definable classes that are decided by such an algorithm in terms of their upper envelopes. We explore the problem of whether there exists an adaptive left unbounded query algorithm over $\mathbb{B}$ that is equivalent to a given Datalog program. Some light is shed on the decidability of this problem, but it remains open. Finally, we show that analogous results hold for adaptive *right* unbounded query algorithms over $\mathbb{B}$. Similar results are also obtained for adaptive *right* unbounded query algorithms over $\mathbb{B}$.

Finally, the two lines of research are brought together to explore the question of which query algorithms benefit from counting, that is, in which cases does counting over $\mathbb{N}$ instead of $\mathbb{B}$ increase the expressive power.

Some of the material from this thesis will appear in the proceedings of MFCS 2025.

Outline. Chapter 1 reviews basic facts and definitions. Chapter 2 introduces the main concepts studied in this thesis, namely adaptive and non-adaptive left/right query algorithms. Chapter 3 contains the main results regarding the expressive power of query algorithms over $\mathbb{N}$, while Chapter 4 contains the main results regarding the expressive power of query algorithms over $\mathbb{B}$. Finally, in Chapter 5, the results from the previous two chapters are used to determine for which query algorithms counting helps.

Chapter 1

Preliminaries

1.1 Relational structures and homomorphism counts

A *signature* $\tau = \{R_1, \dots, R_n\}$ is a set of relational symbols where each symbol R_i has an associated arity r_i. A *relational structure* $\mathfrak{A}$ of signature τ consists of a non-empty set A called the *domain* of $\mathfrak{A}$ and a relation $R_i^{\mathfrak{A}} \subseteq A^{r_i}$ for each symbol $R_i \in \tau$. If $\mathbf{a} \in R_i^{\mathfrak{A}}$ then $(R_i, \mathbf{a})$ is said to be a *fact* of $\mathfrak{A}$. We use $\mathsf{FIN}(\tau)$ to denote the class of all finite relational structures with signature τ. We will only study finite relational structures, we will thus write "structure" and mean "finite relational structure" throughout this text. If $R \subseteq A^n$ is a relation on a set A and $(a_1, \dots, a_n) \in A^n$ we often write $R(a_1, \dots, a_n)$ or $Ra_1 \dots a_n$ instead of $(a_1, \dots, a_n) \in R$.

Let $\mathfrak{A} = (A, (R_i^{\mathfrak{A}})_{i \in \{1,\dots,n\}})$ and $\mathfrak{B} = (B, (R_i^{\mathfrak{B}})_{i \in \{1,\dots,n\}})$ be structures of signature τ. We say that $\varphi : A \to B$ is a *homomorphism* from $\mathfrak{A}$ to $\mathfrak{B}$ if it preserves all the relations, i.e. if

$$(a_1, \dots, a_{r_i}) \in R_i^{\mathfrak{A}} \implies (\varphi(a_1), \dots, \varphi(a_{r_i})) \in R_i^{\mathfrak{B}}$$

for each $R_i \in \tau$. We then write $\varphi : \mathfrak{A} \to \mathfrak{B}$. If there exists a homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$ we write $\mathfrak{A} \to \mathfrak{B}$, otherwise, $\mathfrak{A} \nrightarrow \mathfrak{B}$. If $\mathfrak{A} \to \mathfrak{B}$ and $\mathfrak{B} \to \mathfrak{A}$, we then write $\mathfrak{A} \leftrightarrow \mathfrak{B}$ and say that $\mathfrak{A}$ and $\mathfrak{B}$ are homomorphically equivalent. A bijective homomorphism that also reflects all relations, i.e.

$$(\varphi(a_1), \dots, \varphi(a_{r_i})) \in R_i^{\mathfrak{B}} \implies (a_1, \dots, a_{r_i}) \in R_i^{\mathfrak{A}}$$

holds for each $R_i \in \tau$, is called an isomorphism. If there exists an isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$ we write $\mathfrak{A} \cong \mathfrak{B}$ and say that $\mathfrak{A}$ and $\mathfrak{B}$ are isomorphic. The relation $\cong$ is an equivalence relation. In general, we treat isomorphic structures as the same structure, and when we speak of a class of structures, we will always assume that it is closed under isomorphisms.

For structures A and B, we let $\hom(A, B)$ denote the number of homomorphisms from A to B. We also use $\hom_{\mathbb{N}}(A, B)$ to denote this same number. We then define

$$\hom_{\mathbb{B}}(A, B) := \begin{cases} 0 & \text{if } \hom(A, B) = 0 \\ 1 & \text{if } \hom(A, B) > 0 \end{cases}$$

where $\mathbb{B}$ denotes the Boolean semiring. For a class $\mathcal{C}$ of structures, the *left homomorphism profile of A restricted to $\mathcal{C}$* is the tuple $(\hom(C, A))_{C \in \mathcal{C}}$. Similarly, we can define the right homomorphism profile and the homomorphism profile over $\mathbb{B}$.

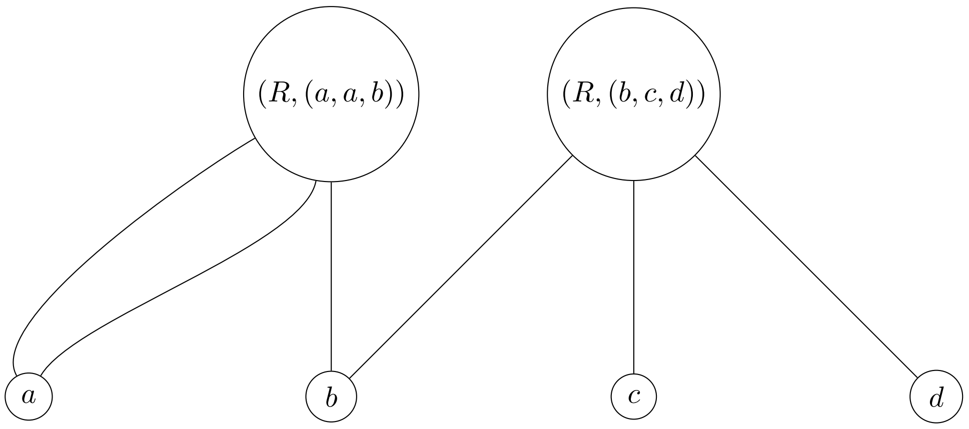

Figure 1.1: The incidence multigraph of the structure with domain $\{a,b,c,d\}$, one ternary relation R, and two facts $(R,(a,a,b))$ and $(R,(b,c,d))$. This structure is not acyclic, but it is connected and has 1 component.

1.2 Properties of structures.

The size of a structure A is the size of its domain. It is denoted $|A|$. The incidence multigraph $\mathrm{inc}(\mathfrak{A})$ of a structure $\mathfrak{A} = (A, (R^{\mathfrak{A}})_{R\in\tau})$ is the bipartite multigraph whose parts are the sets A and $\mathrm{facts}(\mathfrak{A}) = \{(R,\mathbf{t}) : R \in \tau \text{ and } \mathbf{t} \in R^{\mathfrak{A}}\}$, and there is an edge between a and $(R,\mathbf{t})$ for each entry in $\mathbf{t}$ that is a (see Figure 1.1 for an example). We say that $\mathfrak{A}$ is connected if $\mathrm{inc}(\mathfrak{A})$ is connected, and we say $\mathfrak{A}$ is acyclic if $\mathrm{inc}\,\mathfrak{A}$ is acyclic. In particular, if an element appears multiple times in a fact, then the structure is *not* acyclic. This notion of acyclicity is also known as *Berge-acyclicity.* The number $c(A)$ of components of A is the maximal n such that A can be written as the disjoint union of n structures.

For given structures A and B we let $A \oplus B$ denote their disjoint union and for a natural number m and a structure H we write

$$m \cdot H := \underbrace{H \oplus \ldots \oplus H}_{m\text{-times}}.$$

For a given signature $\tau = \{R_1,\ldots,R_n\}$, we let $\mathbf{1}_\tau$ denote the complete singleton structure of that signature, namely

$$\mathbf{1}_\tau := (\{0\}, (\{0\}^{r_i})_{R_i\in\tau}).$$

If the signature is clear from the context, we usually drop the subscript.

The *direct product* of two structures $\mathfrak{A} = (A, (R_i^{\mathfrak{A}})_{i\in\{1,\ldots,n\}})$ and $\mathfrak{B} = (B, (R_i^{\mathfrak{B}})_{i\in\{1,\ldots,n\}})$ is defined with

$$\mathfrak{A} \otimes \mathfrak{B} := (A \times B, R_i^{\mathfrak{A}\otimes\mathfrak{B}})$$

where

$$R_i^{\mathfrak{A}\otimes\mathfrak{B}} := \{((a_1,b_1),\ldots,(a_{r_i},b_{r_i})) : (a_1,\ldots,a_{r_i}) \in R_i^{\mathfrak{A}} \text{ and } (b_1,\ldots,b_{r_i}) \in R_i^{\mathfrak{B}}\}.$$

The following observations are easy to prove:

Proposition 1.1.

(i) For every structure $\mathcal{H}$ and natural number r we have $(r\cdot\mathbf{1})\otimes\mathcal{H} \cong r\cdot\mathcal{H}$.

(ii) *For all structures* $\mathcal{H}, \mathfrak{A}, \mathfrak{B}$ *we have* $\hom(\mathcal{H}, \mathfrak{A} \otimes \mathfrak{B}) = \hom(\mathcal{H}, \mathfrak{A}) \cdot \hom(\mathcal{H}, \mathfrak{B})$.

(iii) *For all structures* $\mathcal{H}, \mathfrak{A}, \mathfrak{B}$ *we have* $\hom(\mathfrak{A} \oplus \mathfrak{B}, \mathcal{H}) = \hom(\mathfrak{A}, \mathcal{H}) \cdot \hom(\mathfrak{B}, \mathcal{H})$.

(iv) *For all structures* $\mathcal{H}, \mathfrak{A}, \mathfrak{B}$ *with* $\mathcal{H}$ *connected we have* $\hom(\mathcal{H}, \mathfrak{A} \oplus \mathfrak{B}) = \hom(\mathcal{H}, \mathfrak{A}) + \hom(\mathcal{H}, \mathfrak{B})$.

Proof sketch:

(i) Let H be the domain of $\mathcal{H}$. Now, $r \cdot \mathbf{1}$ is isomorphic to r disjoint copies of $\mathbf{1}$ on the domain $\{1, \dots, r\}$. Thus, $(r \cdot \mathbf{1}) \otimes \mathcal{H}$ is isomorphic to the structure with domain $\{1, \dots, r\} \times H$ where for each $i \in \{1, \dots, r\}$ the substructure on $\{(i, h) : h \in H\}$ (i.e. the structure containing all the facts that are entirely contained in this set) is isomorphic to H and there are no facts containing $(i, h), (i', h')$ with $i \neq i'$. This structure is clearly isomorphic to $r \cdot \mathcal{H}$, which consists of r disjoint copies of $\mathcal{H}$.

(ii) Let H, A, and B be the domains of $\mathcal{H}$, $\mathfrak{A}$, and $\mathfrak{B}$ respectively. Each function $f : H \to A \otimes B$ can be written $f = (f_1, f_2)$ where $f_1 : H \to A$ and $f_2 : H \to B$. It is easy to see that f is a homomorphism if and only if both f_1 and f_2 are homomorphisms. The result follows.

(iii) This follows from the observation that each homomorphism $f : \mathfrak{A} \oplus \mathfrak{B} \to \mathcal{H}$ can be split up to homomorphisms $f_1 : \mathfrak{A} \to \mathcal{H}$ and $f_2 : \mathfrak{B} \to \mathcal{H}$, and vice versa.

(iv) The result follows from the fact that the image of a connected structure under a homomorphism is connected. Thus, the image $f[\mathcal{H}]$ of $f : \mathcal{H} \to \mathfrak{A} \oplus \mathfrak{B}$ is contained in either $\mathfrak{A}$ or $\mathfrak{B}$, showing that either $f : \mathcal{H} \to \mathfrak{A}$ or $f : \mathcal{H} \to \mathfrak{B}$. Likewise, every homomorphism from $\mathcal{H}$ to $\mathfrak{A}$ or $\mathfrak{B}$ is a homomorphism to $\mathfrak{A} \oplus \mathfrak{B}$.

□

As the categorically minded reader might have noticed, $\otimes$ is indeed a categorical product, $\mathbf{1}$ is a terminal object, and $\oplus$ is a categorical coproduct—in the category of relational structures with some specific signature.

1.3 Digraphs, walks, and homomorphisms to cycles.

A *digraph* (or a *directed graph*) is a structure with exactly one binary relation. So it is a pair (V, R) where $R \subseteq V \times V$. We call R the set of *edges* of the digraph. A *simple graph* is an undirected graph without loops, that is, a directed graph (V, R) where all $a, b \in V$ satisfy $(a, a) \notin R$ and $R(a, b)$ implies $R(b, a)$. We let C_n denote the directed cycle of length n:

$$\mathsf{C}_n \coloneqq (\{0, \dots, n-1\}, \{(0,1), \dots, (n-2, n-1), (n-1, 0)\})$$

and P_n denote the directed path of length n:

$$\mathsf{P}_n \coloneqq (\{0, \dots, n\}, \{(0,1), \dots, (n-1, n)\}).$$

A digraph D is said to be *bipartite* if $D \xrightarrow{f} \mathsf{C}_2$. Then $\{f^{-1}(0), f^{-1}(1)\}$ is a bipartition of D.

Proposition 1.2. *For every natural number* n *we have* $\mathsf{C}_n \otimes \mathsf{C}_n \cong n \cdot \mathsf{C}_n$.

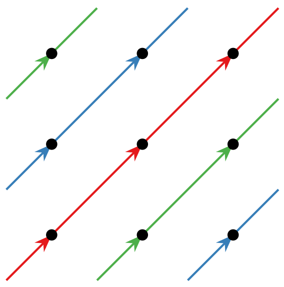

Figure 1.2: An illustration of $\mathsf{C}_3 \otimes \mathsf{C}_3$. The domain $\{0,1,2\}^2$ forms a grid. The resulting three disjoint directed cycles of length 3 are drawn in distinct colors. The edges that are cut short on the right and top sides continue on the opposite left and bottom sides in the same color.

Proof sketch: $\mathsf{C}_n \otimes \mathsf{C}_n$ has domain $\{0,\dots,n-1\}^2$. We have that $(i,j)R(i',j')$ if and only if $i' = i+1$ and $j' = j+1 \mod n$. Thus each diagonal line $\{(k+i,i) : i \in \{0,\dots,n-1\}\}$ forms a directed cycle of length n (see an example in figure 1.2). These cycles are disjoint, so this structure is isomorphic to n disjoint copies of C_n.

□

For our discussion of adaptive left query algorithms, walks in directed graphs play a large role. We will thus review the definition of an oriented walk and some related concepts.

Definition 1.3. *Let $A = (V,R)$ be a digraph.*

(i) *An* oriented walk *in A is a sequence $(a_0, r_0, a_1, r_1, \dots, a_{n-1}, r_{n-1}, a_n)$ where for each i we have $a_i \in V$, $r_i \in \{-,+\}$, and if $r_i = +$ then $(a_i, a_{i+1}) \in R$ while if $r_i = -$ then $(a_{i+1}, a_i) \in R$.*

(ii) *The* net length *of an oriented walk $W = (a_0, r_0, a_1, r_1, \dots, a_{n-1}, r_{n-1}, a_n)$ is defined as:*

$$l(W) := |\{i : i \in \{0,\dots,n-1\} \text{ and } r_i = +\}| - |\{i : i \in \{0,\dots,n-1\} \text{ and } r_i = -\}|.$$

(iii) *The oriented walk $(a_0, r_0, a_1, r_1, \dots, a_{n-1}, r_{n-1}, a_n)$ is said to be* closed *if $a_0 = a_n$.*

(iv) *A closed oriented walk $(a_0, r_0, a_1, r_1, \dots, a_{n-1}, r_{n-1}, a_0)$ is called an oriented* cycle *if $n \geq 1$ (meaning that it traverses at least one edge) and $a_i \neq a_j$ for all $i \neq j$.*

(v) *Let $W = (a_0, r_0, a_1, r_1, \dots, a_{n-1}, r_{n-1}, a_n)$ be an oriented walk. The inverse of W is the oriented walk*

$$-W := (a_n, r'_{n-1}, a_{n-1}, \dots, r'_1, a_1, r'_0, a_0)$$

where r'_i is the opposite of r_i, so $r'_i := \begin{cases} + & \text{if } r_i = - \\ - & \text{if } r_i = + \end{cases}$.

(vi) *For oriented walks $W = (a_0, r_0, \dots, a_n)$ and $W' = (b_0, s_0, \dots, b_n)$ with $a_n = b_0$ we can define the composition of W and W' as*

$$W \circ W' = (a_0, r_0, \dots, a_n = b_0, s_0, \dots, b_n).$$

(vii) A directed walk *is an oriented walk* $(a_0, r_0, \dots, a_n)$ *such that* $r_i = +$ *for each* i. Closed directed walks *and* directed cycles *are defined analogously.*

An important thing to note is that oriented walks and their net lengths are preserved under homomorphisms. So if $(a_0, r_0, a_1, r_1, \dots, a_{n-1}, r_{n-1}, a_n)$ is an oriented walk in A and $\varphi : A \to B$ is a homomorphism, then

$$(\varphi(a_0), r_0, \varphi(a_1), r_1, \dots, \varphi(a_{n-1}), r_{n-1}, \varphi(a_n))$$

is an oriented walk in B with the same net length.

For our proof of the existence of a class that is not decided by an adaptive left unbounded query algorithm, we take advantage of the fact that the existence of a homomorphism to a directed cycle can be described in simple terms.

Lemma 1.4 (Corollary 1.17 in [22]). *Let A be a digraph and n be a positive natural number. We have $A \to \mathsf{C}_n$ if and only if n divides the net length of every closed oriented walk in A.*

1.4 Conjunctive queries.

A conjunctive query (CQ) is a first-order logic (FO) sentence of the form $q \coloneqq \exists \mathbf{x}(\bigwedge_i \alpha_i)$ such that each α_i is of the form $R(y_1, \dots, y_n)$ where y_i can be among the variables in $\mathbf{x}$. If every y_i is among the variables in $\mathbf{x}$, then q is *Boolean*. It is well known that every Boolean CQ q has an associated "canonical structure" A_q such that for every structure B we have $B \vDash q$ if and only if $A_q \to B$. Similarly, vice versa, for every (finite) structure A, there is a Boolean CQ q_A such that for every structure B we have $A \to B$ if and only if $B \vDash q_A$.

Example 1.5. *Look at the conjuncive query*

$$q \coloneqq \exists x_1, x_2, x_3(Rx_1x_2 \wedge Rx_2x_3 \wedge Rx_3x_1).$$

A digraph B satisfies q if and only if B has a closed directed walk of length 3. It is thus clear that C_3 is the canonical structure of q.

Example 1.6. *The directed path of length n, P_n, has the canonical query*

$$\exists x_0, \dots, x_n \bigwedge_{i=0}^{n-1} Rx_ix_{i+1}.$$

A CQ is said to be Berge-acyclic if its canonical structure is acyclic. A (Boolean) *union of conjunctive queries (UCQ)* is a finite disjunction of (Boolean) CQs. Every Boolean UCQ is a positive existential FO sentence, and, conversely, every positive existential FO sentence is equivalent to a UCQ.

1.5 Datalog

Datalog is a logic-based database query language. Datalog programs operate on relational databases, which can be looked at as relational structures.

A Datalog program π is a finite set of rules of the form

$$\alpha \quad \leftarrow \quad \alpha_1, \dots, \alpha_l$$

where $\alpha, \alpha_1, \ldots, \alpha_l$ are atomic first-order formulas. Here, α is the head of the rule and $\alpha_1, \ldots, \alpha_l$ is the body of the rule. Informally, the rule states that if the conditions in the body hold, then the condition in the head also holds. The relation symbols occurring in the head of some rule of the program are called *intentional* (IDBs), while all other relation symbols are called *extensional* (EDBs) (note that IDBs can occur in the body). An example of a Datalog program is the following:

$$\begin{array}{llll} \pi_1: & Xxy & \leftarrow & x = y \\ & Xxy & \leftarrow & Xxz, Rzy \end{array}$$

Here X is an IDB and R is an EDB.

To run a Datalog program π on a relational structure A, every EDB in π must be interpreted as some relation in A. The IDBs are then generated iteratively until a fixpoint is found. Let us explain this in more detail. Let $X_1, \ldots, X_m$ be the IDBs of a program π and let $\mathcal{A} = (A, (R_i^A)_{i=0}^m)$ be an input structure that interprets all the EDBs of π. To simplify the presentation we assume that for each X_i there is a tuple $\mathbf{x}_{X_i}$ of distinct variables such that each rule containing X_i in its head is of the form

$$X_i \mathbf{x}_{X_i} \leftarrow \alpha_1, \ldots, \alpha_l.$$

To each X_i we associate a formula

$$\varphi_{X_i}(\mathbf{x}_{X_i}) := \bigvee \{\exists \mathbf{y}(\alpha_1 \wedge \ldots \wedge \alpha_l) : \quad X_i \mathbf{x}_{X_i} \leftarrow \alpha_1, \ldots, \alpha_l \quad \text{is a rule in } \pi\}$$

where $\mathbf{y}$ are all the variables in $\alpha_1 \wedge \ldots \wedge \alpha_l$ that do not appear in $\mathbf{x}_{\mathbf{X_i}}$. Note that φ_{X_i} has $\mathbf{x}_{\mathbf{X_i}}$ as its free first-order variables, but the IDBs can also appear in it as free second-order variables. We should therefore write the formula as $\varphi_{X_i}(\mathbf{x}_{\mathbf{X_i}}, X_1, \ldots, X_m)$. We can now define how to iteratively generate the IDBs simultaneously on a given input. We define $X_i^0(A) := \varnothing$ and

$$X_i^{k+1}(A) := \{\mathbf{a} \in A^{n_{X_i}} : A \vDash \varphi_{X_i}(\mathbf{a}, X_1^k(A), \ldots, X_m^k(A))\}$$

where n_{X_i} is the arity of X_i. We then define $X_i^\infty(A) := \bigcup_{k \geq 0} X_i^k(A)$. It is easy to see that

$$X_i^0(A) \subseteq X_i^1(A) \subseteq X_i^2(A) \subseteq \ldots \subseteq X_i^\infty(A)$$

since all of the formulas φ_{X_i} contain no negations. Since A is finite, it is also clear that there exists a k_A such that $X_i^{k_A}(A) = X_i^\infty(A)$. Moreover, it can be shown that for $k_A := |A|^{\sum_i n_{X_i}}$ we always have $X_i^{k_A}(A) = X_i^\infty(A)$. We say that the program π is *bounded* if this k_A can be chosen uniformly, i.e. if there exists a k such that for all structures B and all i we have $X_i^k(B) = X_i^\infty(B)$.

Example 1.7. *For the program π_1 described above and a digraph A, we see that $X^{k+1}(A)$ is the set of vertex pairs (a, b) in A such that there is a directed walk from a to b of length at most k. The program therefore computes the reachability relation, also known as the reflexive-transitive closure of R.*

Usually, one IDB is marked as the *goal predicate.* If X is the goal predicate of π, then $X^\infty(A)$ is the output of π on input A. We say that π is *Boolean* if the goal predicate is 0-ary, then π either accepts or rejects the input structure. Each Boolean Datalog program π defines a class $\mathcal{C}_\pi$ of the structures it accepts.

Example 1.8. *Define*

$$\begin{array}{rrcl} \pi_2: & Px & \leftarrow & Rxy, Ryx \\ & Px & \leftarrow & Py, Ryx \\ & \mathrm{Ans}() & \leftarrow & Px, Rxy, Ryz, Rzx \end{array}$$

This is a Boolean Datalog program with Ans() *as its goal predicate. It defines the class of digraphs that have a directed walk from a closed directed walk of length 2 to a closed directed walk of length 3.*

It can be proved that for every Datalog program π, the class $\mathcal{C}_\pi$ is homomorphism-closed, meaning that if $A \in \mathcal{C}_\pi$ and $A \to B$, then $B \in \mathcal{C}_\pi$. Thus, $\mathcal{C}_\pi$ is also closed under homomorphic equivalence.

If X is the goal predicate of π, then for each k there exists a CQs q_i such that for every structure A and every k we have

$$X^k(A) = \{\mathbf{a} : A \vDash \bigvee_{i=1}^{k} q_k(\mathbf{a})\}.$$

If X is 0-ary, then q_i are Boolean and $B \vDash q_i$ if and only if π accepts B in the i-th step of the iteration. The formulas q_i are called the *unfoldings* of π.

Some common fragments of Datalog are *monadic* Datalog and *linear* Datalog. A Datalog program is monadic if it only uses unary and 0-ary IDBs, and it is linear if every rule body has at most one occurrence of an IDB. Thus, the programs π_1 and π_2 from above are both linear, but only π_2 is monadic.

1.6 Constraint satisfaction problems

Many algorithmic problems can be formulated as *Constraint Satisfaction Problems* (CSPs). Using the language of relational structures, these problems can be formulated in simple terms. Each relational structure A produces a constraint satisfaction problem denoted by CSP(A). This is the problem of determining whether an input structure B has a homomorphism to A. We write $\mathrm{CSP}(A) = \{B : B \to A\}$. It is easy to see that this class of structures is always closed under homomorphic equivalence.

Examples of problems that have this form are the non-reachability and k-colorability problems in graphs, scheduling problems, and the number puzzle *Sudoku*. In recent years, considerable efforts have been devoted to classifying the complexity of CSPs, culminating with the proof of a dichotomy theorem [6][32] stating that CSPs are either are NP-complete or solvable in polynomial time.

1.7 Notation for asymptotic growth

In computer science, when estimating the resources required for some computation, one is often not interested in the precise amount needed but rather in how the amount grows when the input size increases. We use notation that allows us to sweep those unnecessary details under the rug.

For the rest of this section, we assume f, g are functions on the positive reals or naturals. The simplest asymptotic notation is the tilde notation. We say that $f \sim g$ if

$$\lim_{x \to +\infty} \frac{f(x)}{g(x)} = 1.$$

Similarly, if

$$\lim_{x\to+\infty} \frac{f(x)}{g(x)} \leq 1$$

then $f \lesssim g$. $f \gtrsim g$ is defined analogously.

The big-$\mathcal{O}$ notation is a bit rougher. We say that f is $\mathcal{O}(g)$ if there exist constants $c_1, c_2 > 0$ such that for all $x \geq c_1$ we have

$$f(x) \leq c_2 g(x).$$

Informally, this tells us that f does not grow faster than g. Additionally, we say that f is $\Omega(g)$ if g is $\mathcal{O}(f)$.

Example 1.9. *The function $2n^2 + 300$ is clearly $\Omega(n^2)$. It is also $\mathcal{O}(n^2)$, since we can pick $c_2 = 3$ and $c_1 = \sqrt{300}$ to make the inequality*

$$2n^2 + 300 \leq c_2 n^2$$

true for all $n \geq c_1$. However, we do not have $2n^2 + 300 \sim n^2$, since

$$\lim_{n\to+\infty} \frac{2n^2 + 300}{n^2} = 2.$$

Conversely, it is easy to see that if $f \sim g$ then f is $\Omega(g)$ and $\mathcal{O}(g)$.

The little-o notation will also be useful in later chapters. We say that f is $o(g)$ if

$$\lim_{x\to+\infty} \frac{f(x)}{g(x)} = 0.$$

Informally, this means that f grows strictly slower than g. In the preceding definition, we also allow f to take non-positive values, but g must always be positive.

Big-$\mathcal{O}$ and little-o notation are frequently used in mathematical expressions. For example we can write

$$f(n) = n^2 + \mathcal{O}(n)$$

meaning that $f(n) = n^2 + h(n)$ for some function h that is $\mathcal{O}(n)$. Thus we can write

$$n^2 + 3n = n^2 + \mathcal{O}(n)$$

and similarly

$$n^2 + 3n = n^2(1 + o(1)).$$

There is an interesting relation between the Ω notation and the negation of the little-o notation. Both of them provide lower bounds on the growth of a function, but they are not quite equivalent.

Example 1.10. *If f is $\Omega(g)$, then there are $c_1, c_2 > 0$ such that*

$$f(x) \geq c_2 g(x)$$

for all $x \geq c_1$. This implies that the limit of $\frac{f(x)}{g(x)}$ is greater than c_2 if it exists, so f is not $o(g)$.

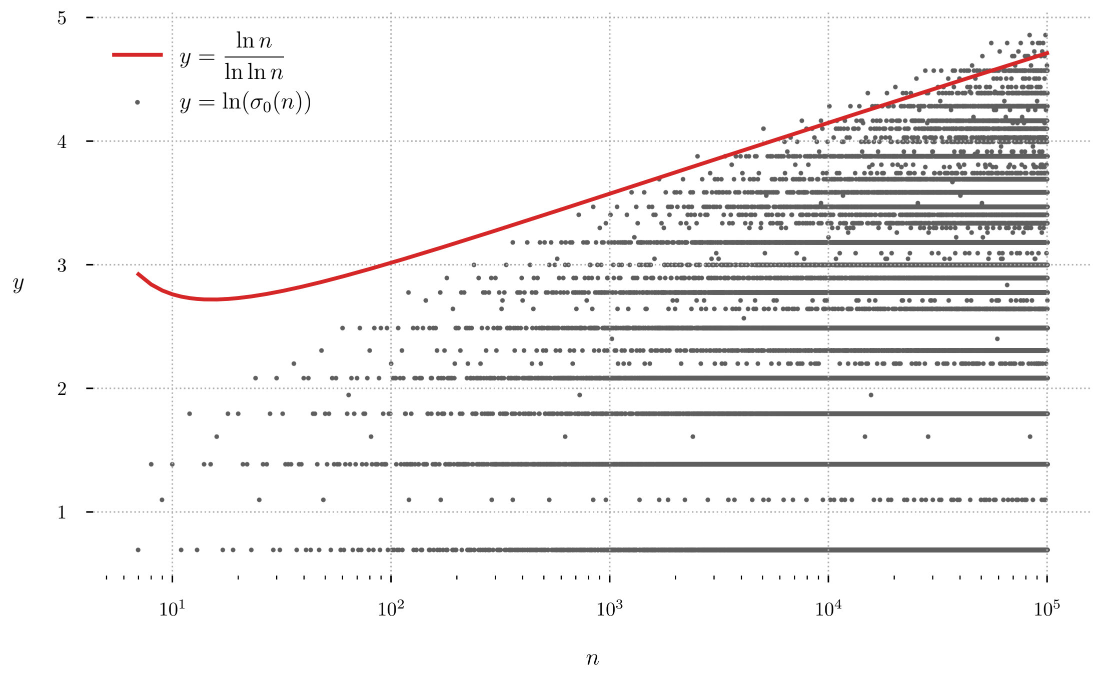

Figure 1.3: A plot showing the values of the functions $\frac{\ln n}{\ln \ln n}$ (red line) and $\ln(\sigma_0(n))$ (gray scattered points) up to $n = 10^5$.

The converse does not hold in general. For example, look at the divisor counting function on the positive natural numbers, $\sigma_0(n) := |\{d \in \mathbb{N} : d \mid n\}|$. It can be shown that $\log \sigma_0(n)$ is not $o(\frac{\log n}{\log \log n})$ (see Theorem 3.23). However, $\log \sigma_0(n)$ is also not $\Omega(\frac{\log n}{\log \log n})$: There are infinitely many primes, so $\sigma_0(n) = 2$ for infinitely many n. Since $\frac{\log n}{\log \log n} \to +\infty$ we also have that $c_2 \frac{\log n}{\log \log n} \to +\infty$ for all $c_2 > 0$. Thus there always exists a large enough prime n such that

$$\log \sigma_0(n) = \log 2 < c_2 \frac{\log n}{\log \log n}.$$

See Figure 1.3 for a visual representation of these functions.

1.8 Ordered sets, dualities and frontiers

A set X equipped with a binary relation R is a *preorder* if R is reflexive and transitive, i.e. if for all a we have $R(a, a)$ and for all a, b, c we have that if $R(a, b)$ and $R(b, c)$ then $R(a, c)$. It is easy to see that the homomorphism relation $\to$ is a preorder on $\mathsf{FIN}(\tau)$. A *partial order* (X, R) is a preorder that is also anti-symmetric, so $R(a, b)$ and $R(b, a)$ imply $a = b$. A preorder (X, R) can be converted into a partial order by identifying the elements $x, y \in X$ such that $R(x, y)$ and $R(y, x)$. More formally, we define an equivalence relation $\sim$ by letting $x \sim y$ if and only if $R(x, y)$ and $R(y, x)$. We then let $(X/\sim) := \{[x]_\sim : x \in X\}$ be the set of equivalence classes of this relation. We can then form the partial order $((X/\sim), R')$ where $R'([x]_\sim, [y]_\sim)$ if and only if $R(x, y)$.

An *upwards frontier* for an element a in a partial order $(X, \leq)$ is a finite set $\{a_1, \ldots, a_n\}$ such that $a_i > a$ for each i and for each $b > a$ there is an i such that $b \geq a_i$. Similarly, *downwards frontiers* are defined by flipping the inequalities. We also define frontiers for elements of preorders by working in the corresponding partial order.

Let $(X, \to)$ be a preorder. A *duality* in this preorder is a pair $(\mathcal{F}, \mathcal{D})$ of finite subsets of X such that for $x \in X$ we have that there exists $a \in \mathcal{F}$ such that $a \to x$ if and only if for all $b \in \mathcal{D}$ we have $x \nrightarrow b$. Then we call $\mathcal{F}$ an *obstruction set* for $\mathcal{D}$. If $\mathcal{D} = \{d\}$ is a singleton, we say that $\mathcal{F}$ is an obstruction set for d.

Example 1.11. *Look at the powerset partial order on $n > 2$ elements, i.e. $(\mathcal{P}(S), \subseteq)$ for $S = \{1, \ldots, n\}$. The set $\{1, \ldots, n-2\}$ has the downwards frontier*

$$\big\{\{1, \ldots, n-2\} \setminus \{i\} : i \in \{1, \ldots, n-2\}\big\}$$

and the upwards frontier

$$\big\{\{1, \ldots, n-2, n-1\}, \{1, \ldots, n-2, n\}\big\}.$$

For $A \in \mathcal{P}(S)$ we have that $A \subseteq \{1, \ldots, n-2\}$ if and only if $\{n-1\} \nsubseteq A$ and $\{n\} \nsubseteq A$. Thus we have that $\mathcal{F} := \{\{n-1\}, \{n\}\}$ is an obstruction set for $\{1, \ldots, n-2\}$ and

$$(\mathcal{F}, \{\{1, \ldots, n-2\}\})$$

is a duality.

1.9 Topology

A *topological space* is a pair $\mathcal{X} = (X, \mathfrak{T})$ where X is a set and $\mathfrak{T} \subseteq \mathcal{P}(X)$ is a *topology* on X. A topology on X must satisfy the following conditions:

- Closed under arbitrary union: If $\mathcal{U} \subseteq \mathfrak{T}$ then $\bigcup_{U \in \mathcal{U}} U \in \mathfrak{T}$.
- Closed under finite intersection: If $U_1, \ldots, U_n \in \mathfrak{T}$ then $\bigcap_{i=1}^{n} U_i \in \mathfrak{T}$.
- Contains the trivial subsets: $\varnothing \in \mathfrak{T}$ and $X \in \mathfrak{T}$

The elements of $\mathfrak{T}$ are called the *open sets* in $\mathcal{X}$. A set A is said to be *closed* if its complement is open, i.e., if $A^c \in \mathfrak{T}$. A set can be both open and closed, such a set is said to be *clopen*.

Given a set X and a collection $\mathcal{S}$ of subsets of X, one can form a topological space by taking the closure of $\mathcal{S}$ under the conditions above. So we define $\mathcal{S}' := \{S_1 \cap \ldots \cap S_n : S_1, \ldots, S_n \in \mathcal{S}\}$ and let

$$\mathfrak{T} := \{\bigcup_{S \in \mathcal{U}} S : \mathcal{U} \subseteq \mathcal{S}'\} \cup \{\varnothing, X\}.$$

It can be shown that $(X, \mathfrak{T})$ is then a topological space. The set $\mathcal{S}$ is called a *subbasis* for the topology $\mathfrak{T}$ and we say that $\mathcal{S}$ *generates* $\mathfrak{T}$. A collection $\mathcal{B}$ of subsets of X that is closed under finite intersection, contains X and satisfies

$$\mathfrak{T} := \{\bigcup_{S \in \mathcal{U}} S : \mathcal{U} \subseteq \mathcal{B}\} \cup \{\varnothing, X\}$$

is called a *basis* for $\mathfrak{T}$. The set $\mathcal{S}' \cup \{X\}$ is therefore a basis for the topology $\mathfrak{T}$ defined above.

The following proposition gives a neat characterization of open sets.

Proposition 1.12. *Let $\mathfrak{T}$ be a topology on X with a basis $\mathcal{B}$. A set $A \subseteq X$ is open if and only if for every $x \in A$ there exists $U \in \mathcal{B}$ such that $x \in U \subseteq A$.*

Chapter 2

Definitions of Query Algorithms

We begin this chapter by reviewing the definition of non-adaptive query algorithms. This definition is motivated by the question: *What properties can be defined using finitely many homomorphism count queries?* More precisely, for a given property (represented by a class $\mathcal{C}$ of structures), do there exist queries such that the pattern of their outputs distinguishes structures that satisfy the property? Query algorithms are a formalism used to tackle this question. They consist of a finite set of queries and a set X that specifies which output patterns lead to acceptance.

Definition 2.1. *Let $\mathcal{C}$ be a class of relational structures of some signature τ, let $\mathcal{K} \in \{\mathbb{N}, \mathbb{B}\}$, and let k be a positive integer.*

- *A* non-adaptive left k-query algorithm *over $\mathcal{K}$ for $\mathcal{C}$ is a pair $(\mathcal{F}, X)$, where $\mathcal{F} = (F_1, \ldots, F_k)$ is a tuple of relational structures of signature τ, and X is a set of k-tuples over $\mathcal{K}$, such that for all structures D of signature τ, we have $D \in \mathcal{C}$ if and only if*

$$(\hom_{\mathcal{K}}(F_1, D), \hom_{\mathcal{K}}(F_2, D), \ldots, \hom_{\mathcal{K}}(F_k, D)) \in X$$

- *A* non-adaptive right k-query algorithm *over $\mathcal{K}$ for $\mathcal{C}$ is a pair $(\mathcal{F}, X)$, where $\mathcal{F} = (F_1, \ldots, F_k)$ is a tuple of relational structures of signature τ, and X is a set of k-tuples over K, such that for all structures D of signature τ, we have that $D \in \mathcal{C}$ if and only if*

$$(\hom_{\mathcal{K}}(D, F_1), \hom_{\mathcal{K}}(D, F_2), \ldots, \hom_{\mathcal{K}}(D, F_k)) \in X.$$

Example 2.2. *Let $\mathcal{F} := \{F\}$ where $F = (\{0\}, \varnothing)$ is the singleton digraph with no edges, and let $X = \{3\}$. The non-adaptive left 1-query algorithm $(\mathcal{F}, X)$ over $\mathbb{N}$ decides the class $\mathcal{C}$ of digraphs that have exactly 3 vertices. It is not difficult to see that $\mathcal{C}$ does not admit a non-adaptive left query algorithm over $\mathbb{B}$. Indeed, this follows from the fact that the class is not closed under homomorphic equivalence. A query algorithm over $\mathbb{B}$ cannot distinguish between homomorphically equivalent structures, since if $A \leftrightarrow B$ then $D \to A$ if and only if $D \to B$ for every D.*

The example above shows that, in general, query algorithms over $\mathbb{N}$ have more expressive power than query algorithms over $\mathbb{B}$. However, for classes closed under homomorphic equivalence, the same does not hold, as shown by ten Cate et al. in the following theorem.

Theorem 2.3 (Corollary 5.6 in [8]). *Let $\mathcal{C}$ be a class of structures that is closed under homomorphic equivalence. Then the following are equivalent:*

- *$\mathcal{C}$ admits a non-adaptive left k-query algorithm over $\mathbb{N}$ for some k.*
- *$\mathcal{C}$ admits a non-adaptive left k-query algorithm over $\mathbb{B}$ for some k.*
- *$\mathcal{C}$ is FO-definable.*

We now turn to the definition of an *adaptive unbounded query algorithm.* These function more as traditional algorithms, as they adapt their next action based on previously obtained information. As in the non-adaptive case, these algorithms can only gain information about the input structure through homomorphism count queries. However, they can use the entire history of outputs from prior queries to decide whether to accept, reject, or ask another query.

Adaptive query algorithms were defined by Chen et al. [13]. The definition there is limited in the sense that it only considers query algorithms whose number of queries is bounded by some constant.

Before presenting the definition, we need to introduce some notation. For a set Σ, we let $\Sigma^{<\omega}$ denote the set of finite strings with alphabet Σ.

- For $\sigma, \sigma' \in \Sigma^{<\omega}$, we write $\sigma' \sqsubseteq \sigma$ if σ' is an initial segment of σ.
- We let $\sigma \bullet \sigma'$ denote the concatenation of the strings σ, σ' (we also use this notation if σ' is an element of Σ).
- If $\sigma_0 \sqsubseteq \sigma_1 \sqsubseteq \sigma_2 \sqsubseteq \ldots$ is a sequence, then we let $\bigsqcup_{n\geq 0} \sigma_n$ denote the (possibly infinite) string σ with length $m := \sup_{n\geq 0} |\sigma_n|$ such that for $i \leq m$, the i-th letter of σ is the i-th letter of σ_n for every n such that $|\sigma_n| \geq i$.

A set $\mathcal{T} \subseteq \Sigma^{<\omega}$ that is closed under initial segments (so if $\sigma \in \mathcal{T}$ and $\sigma' \sqsubseteq \sigma$ then $\sigma' \in \mathcal{T}$) is called a *subtree of* $\Sigma^{<\omega}$. An element σ of such a subtree $\mathcal{T}$ is called a *leaf* if for every $\sigma' \in \mathcal{T}$ such that $\sigma \sqsubseteq \sigma'$ we have $\sigma' = \sigma$.

We are now ready to define adaptive left unbounded query algorithms.

Definition 2.4. *Let $\mathcal{K} \in \{\mathbb{N}, \mathbb{B}\}$.*

(i) An adaptive left unbounded query algorithm over $\mathcal{K}$ *for structures with signature τ is a function*

$$G : \mathcal{T} \to \mathsf{FIN}(\tau) \cup \{\mathit{YES}, \mathit{NO}\}$$

where $\mathcal{T}$ is a subtree of $\mathcal{K}^{<\omega}$ and $G(\sigma) \in \{\mathit{YES}, \mathit{NO}\}$ if and only if σ is a leaf in $\mathcal{T}$.

(ii) For $A \in \mathsf{FIN}(\tau)$, the computation path *of an adaptive left unbounded query algorithm G on A is the string defined by $\sigma(A, G) := \bigsqcup_{n\geq 0} \sigma_n$ where*

$$\sigma_0 = \varepsilon$$
$$\sigma_{n+1} = \begin{cases} \sigma_n & \text{if } G(\sigma_n) \in \{\mathit{YES}, \mathit{NO}\} \\ \sigma_n \bullet \hom_{\mathcal{K}}(G(\sigma_n), A) & \text{if } G(\sigma_n) \in \mathsf{FIN}(\tau) \end{cases}.$$

The algorithm G halts *on input A if $\sigma(A, G)$ is finite.*

(iii) If G halts on all $A \in \mathsf{FIN}(\tau)$, then G is said to be total.

(iv) If G is total, we say that it decides the class

$$\mathcal{C} := \{A \in \mathsf{FIN}(\tau) : G(\sigma(A,G)) = \mathsf{YES}\}.$$

(v) We say that G is an adaptive left k-query algorithm if for every A, $\sigma(A,G)$ has length at most k. We say G is bounded *if it is an adaptive left k-query algorithm for some k.*

The notion of an adaptive right unbounded/bounded query algorithm G is defined analogously by appending $\hom_{\mathcal{K}}(A, G(\sigma_n))$ instead of $\hom_{\mathcal{K}}(G(\sigma_n), A)$ in the computation path.

We will only consider total adaptive query algorithms, so when we speak of adaptive query algorithms, we always assume it is total.

Example 2.5. *Let G be an adaptive left query algorithm over $\mathbb{N}$ for digraphs defined with the following:*

$$G(\sigma) := \begin{cases} (\{0\}, \varnothing) & \text{if } \sigma = \varepsilon \\ \mathsf{P}_n & \text{if } \sigma = \text{`}n\text{'} \\ \mathsf{YES} & \text{if } \sigma = \text{`}a_0a_1\text{' and } a_1 \neq 0 \\ \mathsf{NO} & \text{if } \sigma = \text{`}a_0a_1\text{' and } a_1 = 0 \end{cases}.$$

The corresponding tree $\mathcal{T} \subseteq \mathbb{N}^{<\omega}$ would here be the set of all strings of length at most 2.

We can describe a run of the algorithm on a target structure A as follows: First, the algorithm queries for the number of homomorphisms from the singleton digraph with no edges to A. Let n denote the output of that query. It then queries for $\hom_{\mathbb{N}}(\mathsf{P}_n, A)$. *If the result is positive, it accepts; otherwise, it rejects.*

To see what class this algorithm decides, we first note that the number of homomorphisms from the singleton digraph with no edges to A is precisely the number of vertices of A. We also have that there exists a homomorphism from P_n to A if and only if A has a directed walk of length n. Since n is the size of A, we see that such a walk has to visit the same vertex twice. We thus get that such a walk exists if and only if A has a directed cycle. The algorithm, therefore, decides the class of digraphs that contain a directed cycle.

Example 2.6. *In this example, we show that every class $\mathcal{C}$ of structures is decided by an adaptive left unbounded query algorithm over $\mathbb{N}$.*

The algorithm first queries for the size of the structure. Then it queries the number of homomorphisms from every structure that is not larger than it. By the proof of Lovász's theorem (see, for example, Section III of [2] for a proof of this), the algorithm has now distinguished the structure (up to isomorphism) and can thus classify it correctly.

To formally define the algorithm, let τ be the signature of the structures in $\mathcal{C}$. We fix an enumeration $A_1, A_2, A_3, \ldots$ of all structures of signature τ such that for each $n \in \mathbb{N}$ there exists a number s_n such that $A_1, \ldots, A_{s_n}$ is an enumeration of all of the structures of size at most n. Then the algorithm can be defined with:

$$G(\sigma) := \begin{cases} (\{0\}, \varnothing) & \text{if } \sigma = \varepsilon \\ A_{k+1} & \text{if } \sigma = \text{`}na_1a_2\ldots a_k\text{' and } k < s_n \\ \mathit{YES} & \text{if } \sigma = \text{`}na_1a_2\ldots a_{s_n}\text{' and the unique } A \text{ s.t. } \hom_{\mathbb{N}}(A_i, A) = a_i \\ & \quad \text{for each } 1 \leq i \leq s_n \text{ satisfies } A \in \mathcal{C} \\ \mathit{NO} & \text{if } \sigma = \text{`}na_1a_2\ldots a_{s_n}\text{' and the unique } A \text{ s.t. } \hom_{\mathbb{N}}(A_i, A) = a_i \\ & \quad \text{for each } 1 \leq i \leq s_n \text{ satisfies } A \notin \mathcal{C} \end{cases}$$

Chapter 3

Query Algorithms Over $\mathbb{N}$

The notion of a query algorithm admits many variations. They can be aptive or non-adaptive, left or right, unbounded or bounded; and they may operate over $\mathbb{N}$ or $\mathbb{B}$. In this chapter, we explore the expressive power of these different types of query algorithms over $\mathbb{N}$.

3.1 Unboundedness helps for adaptive left query algorithms

The starting point for our work in this section is the following result by Chen et al:

Theorem 3.1 (Theorem 8.2 in [13])**.** *Every class of simple graphs can be decided by an adaptive left 3-query algorithm over* $\mathbb{N}$.

The proof of this result relies, among other things, on the fact that for a simple graph A, exactly one of the following three possibilities holds:

- A has no edges.
- A is homomorphically equivalent to the graph consisting of two vertices and a single edge connecting them. This is the case when A is bipartite, but it has an edge.
- There is a homomorphism from the undirected cycle of length m to A, where m is the smallest odd number greater than $|A|$.

If one wants to emulate the proof for the case of arbitrary relational structures, one would then need to establish a similar tripartition in that case. However, already if one considers digraphs, this tripartition breaks down, since structures such as the single directed edge fall into none of the above categories. It turns out that extending the theorem to the general case of arbitrary relational structures is not possible. Not only is it impossible to decide every class with 3 queries, we even prove that there is a class of directed graphs that is not decided by an adaptive k-query algorithm over $\mathbb{N}$, for any k. We do this by utilizing the symmetry of directed cycles to show that it is hard to distinguish them from each other using left homomorphism queries.

We begin by defining a parameter very related to the condition in Lemma 1.4.

Definition 3.2. *For a finite digraph A we define*

$$\gamma(A) := \gcd(l_1, \dots, l_k)$$

where $l_1, \dots, l_k$ is a listing of the net lengths of all cycles of positive net length in A.

Here, gcd denotes the greatest common divisor of a collection of natural numbers. Note that we use the convention $\gcd(\varnothing) = 0$. For integers n, m we write $n \mid m$ if n divides m, and $n \nmid m$ otherwise. The next proposition relates the parameter $\gamma(A)$ to the condition in Lemma 1.4.

Proposition 3.3. *Let A be a digraph and n be a positive integer. We have $A \to \mathsf{C}_n$ if and only if $n \mid \gamma(A)$.*

Proof. The left-to-right direction follows immediately from Lemma 1.4, since every cycle of positive net length is a closed oriented walk.

Let us now prove the other direction. We assume $n \mid \gamma(A)$. By Lemma 1.4 it suffices to show $\gamma(A) \mid l(W)$ for every closed oriented walk $W = (a_0, r_0, a_1, r_1, \ldots, a_{n-1}, r_{n-1}, a_n)$ in A. To do that, we use strong induction over n, the length of the walk (not the net length). We have two cases:

- Assume W is an oriented cycle. If $l(W) = 0$, then trivially $\gamma(A) \mid l(W)$. If $l(W) > 0$, then by definition $\gamma(A) \mid l(W)$. Lastly, if $l(W) < 0$ then $l(-W) > 0$ so $-W$ has positive net length. Then $\gamma(A) \mid l(-W)$ but $l(-W) = -l(W)$ so $\gamma(A) \mid l(W)$.
- Assume W is not an oriented cycle. Then there exists a subwalk $C = (a_i, r_i, \ldots, a_{j+1})$ (with $i \leq j$) that is an oriented cycle. But then

$$W' = (a_0, r_0, \ldots, a_{i-1}, r_{i-1}, a_i = a_{j+1}, r_{j+1}, \ldots, a_{n-1}, r_n, a_n)$$

is a smaller closed oriented walk (it has length $n - (j - i + 1) < n$). So by the induction hypothesis, we get that $\gamma(A) \mid l(W')$. We also have $l(W) = l(W') + l(C)$ and by the argument from the previous case, $\gamma(A) \mid l(C)$. So we have $\gamma(A) \mid l(W)$.

Using induction, we conclude that $\gamma(A) \mid l(W)$ for every closed oriented walk W in A, which is what we wanted to show. $\square$

This result can quickly tell us what digraphs have homomorphisms to C_n.

Example 3.4.

(i) For a natural number m we have $\gamma(\mathsf{C}_m) = m$, so C_m has a homomorphism to C_n if and only if $n \mid m$.

(ii) Every T that is an orientation of a tree has no oriented cycle, so $\gamma(T) = 0$ and T has a homomorphism to every directed cycle.

(iii) The digraph A shown in figure 3.1 has one oriented cycle with net length 0. Thus $\gamma(A) = 0$, and it therefore also has a homomorphism to every directed cycle.

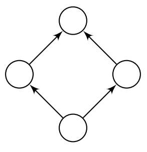

Figure 3.1: A digraph containing an oriented cycle that has net length 0.

(iv) The digraph $\mathsf{C}_r \oplus \mathsf{C}_m$ has oriented cycles of net lengths r and m, thus $\gamma(\mathsf{C}_r \oplus \mathsf{C}_m) = \gcd(r, m)$ and it has a homomorphism to C_n if and only if $n \mid \gcd(r, m)$.

We can now prove a lemma showing that the homomorphism count from a structure to a disjoint union of cycles can be described simply.

Lemma 3.5. *For every digraph A and all positive integers n, m we have*

$$\hom(A, m \cdot \mathsf{C}_n) := \begin{cases} 0 & \text{if } n \nmid \gamma(A) \\ (m \cdot n)^{c(A)} & \text{if } n \mid \gamma(A) \end{cases}.$$

Proof. First we note that $\hom(A, n \cdot \mathbf{1}) = n^{c(A)}$. This is because each connected component must map to a connected component, which are singletons in $n \cdot \mathbf{1}$. Thus, there are n possibilities for where to map each component of A. From Propositions 1.2 and 1.1 we get $\mathsf{C}_n \otimes \mathsf{C}_n \cong n \cdot \mathsf{C}_n \cong (n \cdot \mathbf{1}) \otimes \mathsf{C}_n$. From Proposition 1.1 we then get

$$\begin{aligned} \hom(A, \mathsf{C}_n) \cdot \hom(A, \mathsf{C}_n) &= \hom(A, \mathsf{C}_n \otimes \mathsf{C}_n) \\ &= \hom(A, (n \cdot \mathbf{1}) \otimes \mathsf{C}_n) \\ &= \hom(A, n \cdot \mathbf{1}) \cdot \hom(A, \mathsf{C}_n), \end{aligned}$$

so if $\hom(A, \mathsf{C}_n) \neq 0$ then $\hom(A, \mathsf{C}_n) = \hom(A, n \cdot \mathbf{1}) = n^{c(A)}$. Again using Proposition 1.1 we get

$$\begin{aligned} \hom(A, m \cdot \mathsf{C}_n) &= \hom(A, (m \cdot \mathbf{1}) \otimes \mathsf{C}_n) \\ &= \hom(A, m \cdot \mathbf{1}) \cdot \hom(A, \mathsf{C}_n) \\ &= \begin{cases} 0 & \text{if } \hom(A, \mathsf{C}_n) = 0 \\ m^{c(A)} \cdot n^{c(A)} & \text{if } \hom(A, \mathsf{C}_n) \neq 0 \end{cases} \\ &= \begin{cases} 0 & \text{if } n \nmid \gamma(A) \\ (m \cdot n)^{c(A)} & \text{if } n \mid \gamma(A) \end{cases} \end{aligned}$$

where the last step follows from Lemma 3.3 □

The above proof relies on a few lemmas, it therefore does not give a good intuitive explanation for why the result holds. An alternative explanation is as follows: In disjoint unions of directed cycles, each vertex has in-degree 1 and out-degree 1. For a homomorphism $A \to m \cdot \mathsf{C}_n$, the whole map is therefore decided by the value of the map in one vertex from each component of A. Since there are $m \cdot n$ ways to pick the value of a homomorphism on that initial vertex (if such a homomorphism exists), the lemma follows.

We say that classes $\mathcal{A}, \mathcal{B}$ of structures are *separated* by a query algorithm if it accepts all structures in $\mathcal{A}$ and rejects all structures in $\mathcal{B}$, or vice versa.

The importance of Lemma 3.5 is in the fact that it shows that for a given structure $m \cdot \mathsf{C}_n$ there are only two possible outcomes for $\hom(F, m \cdot \mathsf{C}_n)$. This limits what can be done using few queries, as is shown in the following theorem:

Theorem 3.6. *The class $\mathcal{D}_n := \left\{ 2^{n-m} \cdot \mathsf{C}_{2^m} : 0 \leq m \leq n \text{ and } m \text{ is even} \right\}$ can be separated from the class $\mathcal{D}'_n := \left\{ 2^{n-m} \cdot \mathsf{C}_{2^m} : 0 \leq m \leq n \text{ and } m \text{ is odd} \right\}$ by a non-adaptive left k-query algorithm over $\mathbb{N}$ if and only if $k \geq n$.*

Proof. By Lemma 3.5 we have that for every digraph F and every $2^{n-m} \cdot \mathsf{C}_{2^m} \in \mathcal{D}_n \cup \mathcal{D}'_n$:

$$\begin{aligned} \hom(F, 2^{n-m} \cdot C_{2^m}) &= \begin{cases} 0 & \text{if } 2^m \nmid \gamma(F) \\ (2^n)^{c(F)} & \text{if } 2^m \mid \gamma(F) \end{cases} \\ &= \begin{cases} 0 & \text{if } \nu_2(\gamma(F)) < m \\ (2^n)^{c(F)} & \text{if } \nu_2(\gamma(F)) \geq m \end{cases} \end{aligned}$$

where $\nu_2(k)$ denotes the largest positive integer r such that $2^r \mid k$ if $k \neq 0$, if $k = 0$ then $\nu_2(k) = +\infty$. It follows from this that to have $\hom(F, 2^{n-m} \cdot \mathsf{C}_{2^m}) \neq \hom(F, 2^{n-(m+1)} \cdot \mathsf{C}_{2^{m+1}})$ we must have $\nu_2(\gamma(F)) = m+1$. To distinguish between $2^{n-m} \cdot \mathsf{C}_{2^m}$ and $2^{n-(m+1)} \cdot \mathsf{C}_{2^{m+1}}$ for every $m \in \{0, \dots, n-1\}$ the algorithm then needs to query some F_m with $\nu_2(\gamma(F_m)) = m$ for each $m \in \{0, \dots, n-1\}$. Therefore, the algorithm needs at least n queries to separate the classes. It is clear that n non-adaptive queries suffice: the algorithm can query C_{2^m} for each $m \in \{0, \dots, n-1\}$. Every structure in $\mathcal{D}_n \cup \mathcal{D}'_n$ yields a unique outcome from the collection of these queries, and thus they can be classified correctly.

It is worth noting that homomorphism existence queries suffice here, and hence $\mathcal{D}_n$ and $\mathcal{D}'_n$ are already separated by a non-adaptive left n-query algorithm over $\mathbb{B}$. □

Corollary 3.7. *The class $\mathcal{D}_n$ can be separated from the class $\mathcal{D}'_n$ by an adaptive left k-query algorithm over $\mathbb{N}$ if and only if $k \geq \log(n+1)$.*

Proof. For every digraph $H \in \mathcal{D}_n \cup \mathcal{D}'_n$, there are only two possible outcomes for every query $\hom(F, H)$ (as explained above). Therefore, an adaptive left k-query algorithm over $\mathbb{N}$ that separates $\mathcal{D}_n$ from $\mathcal{D}'_n$ queries at most

$$2^0 + 2^1 + \ldots + 2^{k-1} = 2^k - 1$$

different structures in all of its possible computation paths on inputs from $\mathcal{D}_n \cup \mathcal{D}'_n$. It can thus be translated into a non-adaptive left (2^k-1)-query algorithm over $\mathbb{N}$ separating the two classes. By Theorem 3.6 it follows that such a non-adaptive left (2^k-1)-query algorithm over $\mathbb{N}$ exists if and only if $2^k - 1 \geq n$, so $k \geq \log(n+1)$. Thus, we have shown that the classes $\mathcal{D}_n$, $\mathcal{D}'_n$ can only be separated by an adaptive left k-query algorithm over $\mathbb{N}$ if $k \geq \log(n+1)$. It is also clear that $k \geq \log(n+1)$ suffices. The algorithm can simply use binary search to find the m of an input structure of the form $2^{n-m} \cdot C_{2^m}$, and use that to classify it correctly. □

Let $\mathfrak{N}_k$ denote the set of classes that are decided by a non-adaptive left k-query algorithm over $\mathbb{N}$, and $\mathfrak{A}_k$ denote the set of classes that are decided by an adaptive left k-query algorithm over $\mathbb{N}$. Theorem 3.6 and Corollary 3.7 show, among other things, that

$$\mathfrak{N}_1 \subsetneq \mathfrak{N}_2 \subsetneq \mathfrak{N}_3 \subsetneq \ldots \quad \text{and} \quad \mathfrak{A}_1 \subsetneq \mathfrak{A}_2 \subsetneq \mathfrak{A}_3 \subsetneq \ldots$$

Using Corollary 3.7, we can also prove an even stronger result:

Theorem 3.8. *Let $\mathcal{C}$ be the class of digraphs whose smallest directed cycle (if it exists) has length that is an even power of two. Then $\mathcal{C}$ is not decided by an adaptive left k-query algorithm over $\mathbb{N}$ for any k.*

Proof. Note that we have

$$\bigcup_{n>0} \mathcal{D}_n \subseteq \mathcal{C} \subseteq (\bigcup_{n>0} \mathcal{D}'_n)^c.$$

A given adaptive left k-query algorithm over $\mathbb{N}$ cannot separate $\mathcal{D}_{2^k}$ from $\mathcal{D}'_{2^k}$, by Corollary 3.7. To decide $\mathcal{C}$, it must, in particular, separate these classes. Therefore, the algorithm cannot decide $\mathcal{C}$. Thus, it is clear that $\mathcal{C}$ is not decided by an adaptive left k-query algorithm over $\mathbb{N}$ for any k. □

Using the same method, it can be shown that many other classes are not decided by an adaptive left k-query algorithm over $\mathbb{N}$ for any k. In fact, to show that a class is not decided by such an algorithm, it suffices to show that it separates $\mathcal{D}_n$ and $\mathcal{D}'_n$ for infinitely many n. Another example of such a class is the class of digraphs whose number of components is an even power of two.

Note 3.9. *As we saw in Example 2.6, every class is decided by an adaptive left unbounded query algorithm over $\mathbb{N}$. Theorem 3.8 therefore also shows that unboundedness increases the expressive power of adaptive left query algorithms over $\mathbb{N}$, i.e. $\bigcup_k \mathfrak{A}_k \subsetneq \mathfrak{A}_{unb}$ where $\mathfrak{A}_{unb}$ denotes the collection of classes decided by an adaptive left unbounded query algorithm over $\mathbb{N}$.*

3.2 Extending to n-ary relations

The preceding results show that there exist classes of digraphs that adaptive left bounded query algorithms over $\mathbb{N}$ do not decide. It follows immediately that for every signature containing a binary relation, there exists a class of structures of that signature that left bounded query algorithms over $\mathbb{N}$ cannot decide. What about signatures that contain no binary relation? We begin this section by proving that for signatures containing only unary relations, every class can be decided with an adaptive left bounded query algorithm over $\mathbb{N}$.

Theorem 3.10. *Every class of structures with signature $\tau = \{P_1, \dots, P_k\}$, where each P_i is a unary relation symbol, is decided by a non-adaptive left 2^k-query algorithm over $\mathbb{N}$.*

Proof. For each subset $S \subseteq \tau$ let $F_S \coloneqq (\{0\}, (P_i^{F_S})_{i=1}^k)$ where

$$P_i^{F_S} = \begin{cases} \{0\} & \text{if } P_i \in S \\ \varnothing & \text{if } P_i \notin S \end{cases}.$$

In words, F_S is the singleton structure satisfying the predicates in S and no others. Querying $\hom(F_S, A)$ for each S gives information about the number of elements in A satisfying each Boolean combination of the predicates in τ. This information determines A up to isomorphism and therefore suffices to classify A correctly. □

Theorem 3.8 shows that for every signature containing a binary relation, there exists a class of structures with that signature that is not decided by an adaptive left bounded query algorithm over $\mathbb{N}$. In the remainder of this section, we extend this result to every signature containing a non-unary relation. To prove this extended result, we first extend Lemma 3.5 to the n-ary setting by defining n-ary analogues of cycles.

Definition 3.11. *The* n-ary cycle *of length d is defined with*

$$\mathsf{C}_d^n \coloneqq (\{0, \dots, d-1\}, R)$$

where

$$R = \{(i \pmod d), \dots, i + (n-1) \pmod d)) : i \in \{0, \dots, d-1\}\}.$$

For a structure $\mathcal{A} = (A, R)$ with one n-ary relation, we define $\mathcal{A}^* \coloneqq (A, R^*)$ where R^* is the binary relation defined with: $R^*(a, b)$ holds if and only if there exists an i and elements $a_1, \dots, a_{i-1}, a_{i+2}, \dots, a_n$ such that

$$R(a_1, \dots, a_{i-1}, a, b, a_{i+2}, \dots, a_n).$$

Note that $(\mathsf{C}_d^n)^* \cong \mathsf{C}_d$. We can now prove:

Lemma 3.12. *Let $\mathcal{A} = (A, R)$ be a structure where R is an n-ary relation. A function $f : \mathcal{A} \to \mathsf{C}_d^n$ is a homomorphism if and only if $f : \mathcal{A}^* \to \mathsf{C}_d$ is a homomorphism.*

Proof. First, assume f is a homomorphism from $\mathcal{A}$ to C_d^n and let $R^*(a,b)$ be given. Then there exists i and elements a_j for $j \in \{1,\ldots,n\} \setminus \{i, i+1\}$ such that

$$R(a_1,\ldots,a_{i-1},a,b,a_{i+2},\ldots,a_n).$$

Then we get

$$R(f(a_1),\ldots,f(a_{i-1}),f(a),f(b),f(a_{i+2}),\ldots,f(a_n))$$

and therefore $R^*(f(a),f(b))$, so f preserves R^*.

For the other direction, assume f preserves R^* and let $R(a_1,\ldots,a_n)$ be given. For each $i \in \{1,\ldots,n-1\}$ we then have $R^*(a_i,a_{i+1})$, so $R^*(f(a_i),f(a_{i+1}))$ which means that $f(a_i)+1 = f(a_{i+1}) \pmod d$. Since this holds for every i we have

$$f(a_{j+1}) = f(a_1) + j \pmod d$$

for each $j \in \{0,\ldots,n-1\}$. So indeed $R(f(a_1),\ldots,f(a_n))$ and we have shown that f preserves R. □

The following now immediately follows from Lemmas 3.12 and 3.3.

Lemma 3.13. *Let $\mathcal{A} = (A,R)$ with R n-ary. Then $\mathcal{A} \to \mathsf{C}_d^n$ if and only if $d \mid \gamma(\mathcal{A}^*)$.*

We then obtain the analogue of Lemma 3.5 for structures with one n-ary relation.

Lemma 3.14. *For every $\mathcal{A} = (A,R)$ with R an n-ary relation and all positive integers d,m we have*

$$\hom(\mathcal{A}, m\cdot\mathsf{C}_d^n) = \begin{cases} 0 & \text{if } d \nmid \gamma(\mathcal{A}^*) \\ (m\cdot d)^{c(\mathcal{A})} & \text{if } d \mid \gamma(\mathcal{A}^*) \end{cases}.$$

Proof. In the same way as in Proposition 1.2, we have that $\mathsf{C}_d^n \otimes \mathsf{C}_d^n \cong d\cdot\mathsf{C}_d^n$ by observing that the diagonal lines in the product form cycles. Then, using Proposition 1.1 we get $\mathsf{C}_d^n \otimes \mathsf{C}_d^n \cong d\cdot\mathsf{C}_d^n \cong (d\cdot\mathbf{1})\otimes\mathsf{C}_d^n$. Using this and Lemma 3.13, the proof follows in the same way as the proof of Lemma 3.5. □

We can now use the same proofs as before to prove analogues of Theorem 3.6, Corollary 3.7, and Theorem 3.8, if $n \geq 2$.

The attentive reader might have noticed that the hypothesis $n \geq 2$ is not assumed in Lemma 3.14. Indeed, the lemma also holds in the case $n = 1$. However, the analogues of Theorem 3.6, Corollary 3.7, and Theorem 3.8 do not extend to this case. The reason is that if $n = 1$ we have $\mathsf{C}_d^n \cong d\cdot C_1^n$, whereas for all $n \geq 2$ and all $m > 1$ we have that $C_{m\cdot d}^n \not\cong m\cdot\mathsf{C}_d^n$. Therefore, we can only establish the analogues of these results for n-ary relations with $n \geq 2$. In particular, we obtain:

Theorem 3.15. *For every $n \geq 2$ and every signature τ containing an n-ary relation, there exists a class of structures with signature τ that is not decided by an adaptive left k-query algorithm over $\mathbb{N}$ for any k.*

3.3 Adaptive versus non-adaptive left k-query algorithms

We now turn to a comparison of the expressive power of adaptive and non-adaptive left query algorithms over $\mathbb{N}$. It follows from Theorem 3.6 and Corollary 3.7 that $\mathfrak{N}_{2^k} \nsubseteq \mathfrak{A}_k$. We also trivially have the relation $\mathfrak{N}_k \subseteq \mathfrak{A}_k$. This raises the question of where the bound lies, that is, for which l we have $\mathfrak{N}_l \subseteq \mathfrak{A}_k$ but $\mathfrak{N}_{l+1} \nsubseteq \mathfrak{A}_k$? This is answered by the following theorem:

Theorem 3.16. *For each k, there exists a class of structures that is decided by a non-adaptive left k-query algorithm over $\mathbb{N}$ but not by an adaptive left $(k-1)$-query algorithm over $\mathbb{N}$.*

Proof. Let $p_1, \dots, p_{2k}$ be distinct primes and write $P := \prod_{i=1}^{2k} p_i$ and $q_i := P/p_i$. Define a non-adaptive left k-query algorithm $((F_1, \dots, F_k), X)$ with

$$F_i := p_i \cdot \mathsf{C}_{q_i} \text{ and } X := \{(P^{p_1} \cdot \delta_{1,j}, \dots, P^{p_k} \cdot \delta_{k,j}) : j \in \{1, \dots, k\}\}$$

where $\delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$ is the Kronecker delta. Let $\mathcal{C}_{2k}$ be the class decided by the above query algorithm. It follows from Lemma 3.5 that

$$\hom(F_i, p_j \cdot \mathsf{C}_{q_j}) = P^{p_i} \cdot \delta_{i,j},$$

so for $j \in \{1, \dots, 2k\}$ we have $p_j \cdot \mathsf{C}_{q_j} \in \mathcal{C}_{2k}$ if and only if $j \leq k$. We will show that $\mathcal{C}_{2k}$ cannot be decided by an adaptive left $(k-1)$-query algorithm over $\mathbb{N}$.

Let G be a given adaptive left $(k-1)$-query algorithm over $\mathbb{N}$. Using an adversarial argument, we find two structures that are classified in the same way by G but differently by $\mathcal{C}_{2k}$. We do this by inductively defining sets $\mathcal{A}_n$ such that G has the same computation path on all elements of $\mathcal{A}_n$ up to stage n. We define $\mathcal{A}_0 := \{p_j \cdot \mathsf{C}_{q_j} : j \in \{1, \dots, 2k\}\}$. Now let $\mathcal{A}_n \subseteq \mathcal{A}_0$ be given such that G has the same computation path σ on all elements of $\mathcal{A}_n$ up to stage n. Write $F := G(\sigma)$. We have two cases:

- If $\hom(F, A)$ is the same for all elements A of $\mathcal{A}_n$ we can simply define $\mathcal{A}_{n+1} := \mathcal{A}_n$ and the computation path is the same up to stage $n+1$.
- Assume otherwise. Note that for every element $p_j \cdot \mathsf{C}_{q_j}$ of $\mathcal{A}_n$ we have by Lemma 3.5 that

  $$\hom(F, p_j \cdot \mathsf{C}_{q_j}) = \begin{cases} 0 & \text{if } q_j \nmid \gamma(F) \\ P^{c(F)} & \text{if } q_j \mid \gamma(F). \end{cases}$$

  So there are only two possible outcomes. Moreover, if $q_i, q_j \mid \gamma(F)$ for $i \neq j$, then $P \mid \gamma(F)$ and thus $q_r \mid \gamma(F)$ for all $r \in \{1, \dots, 2k\}$ and we are in the former case. Thus, there is an i such that $q_j \mid \gamma(F)$ if and only if $j = i$. We can thus define $\mathcal{A}_{n+1} := \mathcal{A}_n \setminus \{p_i \cdot \mathsf{C}_{q_i}\}$. It is clear that $\hom(F, A) = 0$ for all elements A of $\mathcal{A}_{n+1}$, so they share the same computation path up to stage $n+1$.

The above construction gives us a set $\mathcal{A}_{k-1}$ such that G has the same computation path up to stage $k-1$ on all elements of it. Clearly we also have that $|\mathcal{A}_{k-1}| \geq |\mathcal{A}_0| - (k-1) = k+1$. This means that there must exist $p_i \cdot \mathsf{C}_{q_i}, p_j \cdot \mathsf{C}_{q_j} \in \mathcal{A}_{k-1}$ such that $i \leq k$ and $j \geq k+1$. Thus, we have found elements that G classifies in the same way but $\mathcal{C}_{2k}$ classifies differently. We have thus shown that G does not decide $\mathcal{C}_{2k}$. □

The above theorem describes a class of structures where adaptiveness does not help at all when trying to decide the class, and it shows that $\mathfrak{N}_{k+1} \nsubseteq \mathfrak{A}_k$ for each k. This is interesting as in many cases, adaptiveness greatly reduces the number of queries needed.

In this regard, we now prove that there is an adaptive left 2-query algorithm over $\mathbb{N}$ that decides a class that no non-adaptive left query algorithm over $\mathbb{N}$ decides.

Theorem 3.17. *The class of all digraphs that contain a directed cycle is decided by an adaptive left 2-query algorithm over $\mathbb{N}$, but not by a non-adaptive left k-query algorithm over $\mathbb{N}$, for any k.*

Proof. Note that directed cycles are preserved by homomorphisms. It then follows that the class is closed under homomorphic equivalence. It is also well known that this class is not first-order definable; this is a standard application of the Ehrenfeucht-Fraïsse method, which can be found in textbooks on finite model theory [16]. It then follows from Theorem 2.3 that the class is not decided by a non-adaptive left k-query algorithm over $\mathbb{N}$, for any k.

In example 2.5 we saw an adaptive left 2-query algorithm over $\mathbb{N}$ that decides the class. This completes the proof. □

The preceding results now show that the inclusions that hold between $\mathfrak{N}_k$ and $\mathfrak{A}_k$ are precisely the ones that are shown in Figure 1.

3.4 Adaptive left $f(n)$-query algorithms over $\mathbb{N}$

At this point, we have seen that adaptive left k-query algorithms are unable to decide some classes of structures, but adaptive left unbounded query algorithms can decide them all. A natural restriction on adaptive unbounded query algorithms is to bound the number of queries they can use on a given input based on the size of that input. This leads to the following definition.

Definition 3.18. *Let $\mathcal{K} \in \{\mathbb{N}, \mathbb{B}\}$, $f : \mathbb{N} \to \mathbb{N}$ be a function, and G be an adaptive left unbounded query algorithm over $\mathcal{K}$. We say that G is an* adaptive left $f(n)$-query algorithm over $\mathcal{K}$ *if for every structure A with $|A| = n$ we have that $\sigma(A, G)$ has length at most $f(n)$.*

Recall that $\sigma(A, G)$ is the computation path of G on input A. So G is an $f(n)$-query algorithm if and only if G uses at most $f(n)$ queries on inputs of size n.

Example 3.19. *If $f : \mathbb{N} \to \mathbb{N}, n \mapsto k$ for some constant k, then G is an $f(n)$-query algorithm if and only if G is a k-query algorithm.*

Example 3.20. *In Example 2.6, we described a template for creating an adaptive left unbounded query algorithm for deciding any given class $\mathcal{C}$ (of structures of a given signature τ). For an input structure of size n, the algorithm uses $s_n + 1$ queries where s_n is the number of structures of size at most n up to isomorphism. The number of structures with the same domain of size k is*

$$2^{\sum_i k^{r_i}}$$

where r_i are the arities of the relation symbols in τ. This shows that we can upper bound s_n with

$$\sum_{k=1}^{n} 2^{\sum_i k^{r_i}} \leq 2 \cdot 2^{\sum_i n^{r_i}}.$$

This tells us that there exists $f \in \mathcal{O}(2^{\sum_i n^{r_i}})$ such that every class of structures with signature τ is decided by some adaptive left $f(n)$-query algorithm over $\mathbb{N}$.

The algorithm in the above example is a bit rough, it probably uses more queries than it needs to. But how many are needed? More precisely:

Question 3.21. *What is the smallest f such that every class of structures of a given signature is decided by an adaptive left $f(n)$-query algorithm over $\mathbb{N}$?*

A similar problem, studied by Comer in his master's thesis [14], concerns how many queries are needed to characterize a structure A; this can equivalently be phrased as the question of finding the smallest non-adaptive query algorithm that decides the class $\{A\}$. It should be pointed out that Comer only allows small queries, i.e., the structures used in the queries cannot be larger than the target structure.

Turning back to Question 3.21, Example 3.20 provides the upper bound $O(2^{\sum_i n^{r_i}})$ for this f. We can use earlier results in this chapter to get a lower bound. In particular, it follows from Corollary 3.7 and the proof of Theorem 3.8 that the class of digraphs whose number of components is an even power of two cannot be decided by an adaptive left $f(n)$-query algorithm over $\mathbb{N}$ if f is $o(\log\log n)$. The reason for this is that for some input structures of size 2^n, $\log(n+1)$ queries are needed to determine whether the structure falls into the class or not.

The proof of Theorem 3.16 gives a stronger lower bound:

Theorem 3.22. *There exists a class of structures that is not decided by an adaptive left $f(n)$-query algorithm for any f that is $o(\frac{\log n}{\log\log n})$.*

Proof. In the proof of Theorem 3.16 it is shown that there exists a class $\mathcal{C}_{2k}$ of structures of size $P_{2k} = \prod_{i=1}^{2k} p_i$ that is not decided by any adaptive left $(k-1)$-query algorithm over $\mathbb{N}$. Here, $p_1, p_2, \dots$ is an enumeration of all primes in increasing order. Define $\mathcal{C} \coloneqq \bigcup_{k\geq 1} \mathcal{C}_{2k}$ and let f be such that there is an adaptive left $f(n)$-query algorithm that decides $\mathcal{C}$. We show that f is not $o(\frac{\log n}{\log\log n})$.

This algorithm deciding $\mathcal{C}$ can be turned into an algorithm deciding $\mathcal{C}_{2k}$ by adding a single query asking for the size of the structure. Since $\mathcal{C}_{2k}$ cannot be decided in fewer than k queries it follows that $f(P_{2k}) + 1 \geq k$. It is a well-known consequence of the Prime Number Theorem that $P_n = e^{n\log(n)(1+o(1))}$. Write $g(n) = \frac{\log n}{\log\log n}$. We then get that

$$\begin{aligned} g(P_n) = \frac{\log P_n}{\log\log P_n} &= \frac{\log(e)n\log(n)(1+o(1))}{\log(\log(e)n\log(n)(1+o(1)))} \\ &\leq \frac{\log(e)n\log(n)(1+o(1))}{\log n} = \log(e)n(1+o(1)) \sim \log(e)n. \end{aligned}$$

We have now shown that $f(P_{2k}) \geq k-1 \gtrsim \frac{1}{2\log(e)} g(P_{2k})$ so f is not $o(g)$. $\square$

Both of the preceding lower bounds were proved by building on Lemma 3.5, in the sense that they show that classes consisting of disjoint unions of directed cycles of the same size are hard to distinguish from each other. This poses the question of whether we have reached the best lower bound obtainable using this technique. This is indeed the case. To show this, it suffices to prove that there exists an $\mathcal{O}(\frac{\log n}{\log\log n})$ function f such that for every two disjoint classes consisting of disjoint unions of equal-sized cycles, there exists an adaptive left $f(n)$-query algorithm that separates them.

In our proof of this, we need a classical result from number theory, proved by Carl Severin Wigert (see [21, § 18.1] for a proof). We let $\sigma_0(n)$ denote the number of divisors of a natural number n.

Theorem 3.23 (Wigert).

$$\limsup_{n\to\infty} \frac{\ln\sigma_0(n)}{\ln n/\ln\ln n} = \ln 2.$$

In particular, this theorem tells us that $\log\sigma_0(n)$ is $\mathcal{O}(\frac{\log n}{\log\log n})$.

Theorem 3.24. *There exists a function f that is $\mathcal{O}(\frac{\log n}{\log\log n})$ such that for all disjoint classes $\mathcal{D}_1$ and $\mathcal{D}_2$ of structures of the form $m\cdot\mathsf{C}_d$, there exists an adaptive left $f(n)$-query algorithm separating $\mathcal{D}_1$ and $\mathcal{D}_2$.*

Proof. As in Example 2.6, we construct an algorithm that has a distinct computation path for every element. Thus, only the final deciding step needs to be changed depending on the class to be decided. The difference is that now we only have to consider input structures that are disjoint unions of equal-sized cycles.

A structure $m\cdot\mathsf{C}_d$ is identified from the other unions of equal-sized cycles by the two numbers d and $m\cdot d$. The latter number is the size of the structure, which can be obtained in a single query, as shown in Example 2.2. Write $n = m\cdot d$ and prime factorize it: $n = p_1^{e_1}\cdots p_r^{e_r}$. Then $d = p_1^{e_1'}\cdots p_r^{e_r'}$ with $0 \le e_i' \le e_i$ for each $1 \le i \le r$, so to find d it suffices to find $e_1',\dots,e_r'$. Let $q_i \coloneqq \prod_{j\in\{1,\dots,r\}\setminus\{i\}} p_j^{e_j}$. Now note that Lemma 3.5 tells us that

$$\hom(\mathsf{C}_{q_i\cdot p_i^k}, m\cdot\mathsf{C}_d) = \begin{cases} 0 & \text{if } d\nmid q_i\cdot p_i^k \\ n & \text{if } d\mid q_i\cdot p_i^k \end{cases} = \begin{cases} 0 & \text{if } e_i' > k \\ n & \text{if } e_i' \le k \end{cases}.$$

This shows that we can, for all k and i, obtain whether $e_i' \le k$ holds using one query. We can thus use binary search to obtain e_i' in $\lceil\log(e_i+1)\rceil$ queries. Thus d is obtainable in

$$\sum_{i=1}^{r}\lceil\log(e_i+1)\rceil \le \sum_{i=1}^{r} 2\log(e_i+1) = 2\log\left(\prod_{i=1}^{r}(e_i+1)\right) = 2\log(\sigma_0(n))$$

queries. This shows that by setting the algorithm up in this way, every structure of the form $m\cdot\mathsf{C}_d$ with $m\cdot d = n$ has a different computation path after $2\log(\sigma_0(n))+1$ queries. This allows for the construction of an algorithm that uses only $2\log(\sigma_0(n))+1$ queries on inputs of size n and separates any two disjoint classes of cycles of the form $m\cdot\mathsf{C}_d$. Since $2\log(\sigma_0(n))+1$ is $\mathcal{O}(\frac{\log n}{\log\log n})$ by Theorem 3.23, we are done. □

What we have essentially shown in this theorem is that if given natural numbers d, n such that $d \mid n$, then d can be identified in $\mathcal{O}(\frac{\log n}{\log\log n})$ queries of the form: "does d divide a?" This, in turn, means that the question of whether d belongs to some set $S \subseteq \{s\in\mathbb{N} : s\mid n\}$ can also be determined in $\mathcal{O}(\frac{\log n}{\log\log n})$ such queries. Conversely, Theorem 3.22 shows that determining whether a natural number d dividing n belongs to some set $S \subseteq \{s\in\mathbb{N} : s\mid n\}$ cannot always be done in $o(\frac{\log n}{\log\log n})$ queries of the form "does d divide a?". Thus, we have shown that $\mathcal{O}(\frac{\log n}{\log\log n})$ is an asymptotically tight upper bound for this problem, and we have shown that the lower bound given in Theorem 3.22 is asymptotically the best lower bound for f in Question 3.21 that can be obtained using disjoint unions of equal-sized cycles.

We are still far from settling Question 3.21 as there remains a wide gap between our lower bound of $\frac{\log n}{\log\log n}$ and the upper bound of $2^{\sum_i n^{r_i}}$. A promising method for improving the lower bound is to consider disjoint unions of cycles of varying lengths. Although this approach sacrifices the clarity provided by Lemma 3.5, Proposition 3.3 can still be used to impose some structure on the outputs of the homomorphism queries. To simplify the analysis, one might restrict the attention to structures of the form $\sum_{d\mid n} a_d\cdot\mathsf{C}_d$ such that $\sum_{d\mid n} a_d\cdot d = n$. Since then, if F is connected we have

$$\hom(F, \sum_{d\mid n} a_d\cdot\mathsf{C}_d) = \sum_{d\mid\gamma(F)} a_d\cdot d.$$

Thus, the output of a homomorphism query has a predictable structure, albeit one of considerable complexity.

This line of inquiry has not yet been fully explored, but it appears to be a promising direction for future work.

3.5 Adaptive right query algorithms over $\mathbb{N}$

In their paper, Chen et al. [13] also prove a result about adaptive *right* query algorithms on simple graphs:

Proposition 3.25 (Proposition 9.3 in [13]). *There exists a class of simple graphs that is not decided by an adaptive right k-query algorithm over $\mathbb{N}$ for any k, using only simple graphs in its queries.*

In the general case, we again get a different result. It turns out that using looped structures in the queries changes the picture completely. This result was essentially proved by Wu [30], but it is not stated in this form in his work.

Theorem 3.26 (From [30]). *Every class of structures can be decided by an adaptive right 2-query algorithm over $\mathbb{N}$.*

This theorem follows almost immediately from the following result, which is a variation of a result by Wu (Theorem 5.38 in [30]).

Lemma 3.27. *For every $n > 0$ and every signature τ, there exists a structure $F(n,\tau)$ of signature τ such that for every two structures H, H' with signature τ and of size n we have*

$$H \cong H' \text{ if and only if } \hom(H, F(n,\tau)) = \hom(H', F(n,\tau)).$$

Proof idea. The proof is the same as Wu's proof; we therefore only sketch out the idea here.

Let $A_1, \ldots, A_{s_n}$ be an enumeration of all of the structures with signature τ and size at most n. We know, from the proof of the Chaudhuri-Vardi Theorem (which is analogous to the proof of Lovász's Theorem), that the sequence $\hom(B, A_1), \ldots, \hom(B, A_{s_n})$ determines the structure B up to isomorphism. The idea here is to ask a single homomorphism query containing all of this information. To do that, the structure

$$F(n,\tau) := \bigoplus_{j=1}^{s_n} \bigoplus_{D^{e_j}} A_j$$

is formed, where $D, e_1, \ldots, e_{s_n}$ are numbers determined based on n. Then

$$\hom(B, F(n,\tau)) = \prod_{i=1}^{r} \hom(B_i, F(n,\tau)) = \prod_{i=1}^{r} \sum_{j=1}^{s_n} \hom(B_i, A_j) \cdot D^{e_j}$$

where $B = B_1 \oplus \ldots \oplus B_r$ and each B_i is connected. With some technical manipulation, it can be shown that D and $e_1, \ldots, e_{s_n}$ can be chosen such that the re_j-th digit in the D-ary representation of $\hom(B, F(n,\tau))$ is exactly $\hom(B, A_j) = \hom(B_1, A_j) \cdots \hom(B_r, A_j)$. This shows that the number $\hom(B, F(n,\tau))$ contains all the information needed to deduce the isomorphism class of B. $\square$

Proof of Theorem 3.26. Let τ be a signature. Let $\mathfrak{2}_\tau$ be the complete structure with domain of size 2 and signature τ. Concretely, $\mathfrak{2}_\tau := (\{0,1\}, (R_i^{\mathfrak{2}})_{R_i \in \tau})$ where

$$R_i^{\mathfrak{2}} = \{0,1\}^{r_i}$$

where r_i is the arity of R_i.

Then, for every structure H with signature τ we have

$$\hom(H, \mathfrak{2}_\tau) = 2^{|H|}.$$

Our algorithm thus runs as follows: Given input H, query $\hom(H, \mathfrak{2}_\tau)$ to obtain $2^{|H|}$. Then query $\hom(H, F(|H|, \tau))$.

By Lemma 3.27, this determines the structure H up to isomorphism. Thus, every input A has a unique computation path $\sigma(A)$ up to this point. We can thus set

$$G(\sigma(A)) := \begin{cases} \mathsf{YES} & \text{if } A \in \mathcal{C} \\ \mathsf{NO} & \text{if } A \notin \mathcal{C} \end{cases}$$

where $\mathcal{C}$ is the class we are trying to decide. □

Theorem 3.26 contrasts our result for left query algorithms, which stated that there exist classes that cannot be decided using a bounded number of queries. One cause of this difference is that, for right query algorithms, we have more constructions that give useful information about the homomorphism counts. In particular, the proof of Lemma 3.27 used that if H is connected then

$$\hom(H, A \oplus B) = \hom(H, A) + \hom(H, B).$$

No construction on the left side allows for the addition of the homomorphism profiles.

Chapter 4

Query Algorithms Over $\mathbb{B}$

We now move on to studying adaptive query algorithms over $\mathbb{B}$. We focus on left query algorithms, and we will comment on the case of right query algorithms afterwards.

4.1 Unboundedness helps for adaptive left query algorithms

Every class that can be decided by an adaptive k-query algorithm over $\mathbb{B}$ can also be decided by a non-adaptive $(2^k - 1)$-query algorithm over $\mathbb{B}$. This non-adaptive algorithm queries all the structures that the adaptive algorithm can possibly query in the first k steps of its run. Since there are at most 2 branches from each node in the computation tree, this is at most $1 + 2 + 2^2 + \ldots + 2^{k-1} = (2^k - 1)$ queries. This means that *adaptive* and *non-adaptive* left bounded query algorithms over $\mathbb{B}$ have the same expressive power.

The question we answer below is whether every class decided by an adaptive unbounded query algorithm over $\mathbb{B}$ can be decided by an adaptive bounded query algorithm (and thus a non-adaptive one)? The answer is no.

Theorem 4.1. *The class $\mathcal{C} = \{A : A \text{ is a digraph that contains a directed cycle}\}$ can be decided by an adaptive left unbounded query algorithm over $\mathbb{B}$, but not by an adaptive left k-query algorithm over $\mathbb{B}$, for any k.*

Proof. In theorem 3.17 we showed that $\mathcal{C}$ is not decided by a non-adaptive left k-query algorithm over $\mathbb{N}$, it is thus not decided by a non-adaptive left k-query algorithm over $\mathbb{B}$.

We now provide an adaptive left unbounded query algorithm over $\mathbb{B}$ that decides $\mathcal{C}$. The algorithm runs as follows:

Let A be the given input. For $n = 1, 2, \ldots$ query $\hom_{\mathbb{B}}(\mathsf{P}_n, A)$ and $\hom_{\mathbb{B}}(\mathsf{C}_n, A)$. If at some point $\hom_{\mathbb{B}}(\mathsf{P}_n, A) = 0$ then halt and output NO. If at some point $\hom_{\mathbb{B}}(\mathsf{C}_n, A) = 1$ then halt and output YES.

To see that this algorithm decides $\mathcal{C}$ we note that the following equivalence holds:

$$\begin{aligned} A \text{ has no directed cycle} &\iff \text{the lengths of directed walks in } A \text{ are bounded} \\ & \text{ by a constant } k \\ &\iff \hom_{\mathbb{B}}(\mathsf{P}_n, A) = 0 \text{ for some } n. \end{aligned}$$

Also, directed cycles are preserved under homomorphism, so A has a directed cycle if and only if $\hom_{\mathbb{B}}(\mathsf{C}_n, A) = 1$ for some n. Thus, it is clear that the algorithm halts on all inputs and produces the correct answer. □

4.2 Characterization of the expressive power of adaptive left unbounded query algorithms over $\mathbb{B}$

We can use the language of topology to characterize the expressive power of adaptive unbounded query algorithms exactly. The topology we define is generated by all “basic observations”, where a basic observation is a class defined by the answer to a single homomorphism query.

More formally, for a structure $A \in \mathsf{FIN}(\tau)$ we define

$$\uparrow A \coloneqq \{B : A \to B\},$$

which, phrased in terms of *conjunctive queries* (CQs), is simply the set of structures that satisfy the canonical CQ q_A of A. We then let

$$\mathcal{S} \coloneqq \{\uparrow A : A \in \mathsf{FIN}(\tau)\} \cup \{\mathsf{FIN}(\tau) \setminus \uparrow A : A \in \mathsf{FIN}(\tau)\}$$

be the subbasis for our topology $\mathfrak{T}$. In other words, phrased again in terms of conjunctive queries, the open sets of the topology are all classes of structures that can be defined by a CQ or the negation of a CQ, as well as all classes generated from these by closing under finite intersection and arbitrary union.

We have the following characterization of classes that are decided by adaptive left unbounded query algorithms over $\mathbb{B}$.

Theorem 4.2. *A class $\mathcal{C} \subseteq \mathsf{FIN}(\tau)$ is decided by an adaptive left unbounded query algorithm over $\mathbb{B}$ if and only if $\mathcal{C}$ is a clopen set in the topological space* $(\mathsf{FIN}(\tau), \mathfrak{T})$.

Proof.

$\Rightarrow$: Let G be an adaptive left unbounded query algorithm over $\mathbb{B}$ that decides $\mathcal{C}$. We want to show that $\mathcal{C}$ is clopen. Let $A \in \mathcal{C}$. Now $\sigma(A, G)$ is finite so there exist $A_0, \dots, A_n$ such that

$$\sigma(A, G) = (\hom_{\mathbb{B}}(A_0, A), \dots, \hom_{\mathbb{B}}(A_n, A)).$$

Let

$$U_A \coloneqq \bigcap_{A_i \to A} \uparrow A_i \cap \bigcap_{A_i \not\to A} \mathsf{FIN}(\tau) \setminus \uparrow A_i.$$

Then indeed U_A is a basis element such that $A \in U_A$. Now for $A' \in U_A$ we have

$$(\hom_{\mathbb{B}}(A_0, A'), \dots, \hom_{\mathbb{B}}(A_{n'}, A')) = (\hom_{\mathbb{B}}(A_0, A), \dots, \hom_{\mathbb{B}}(A_{n'}, A))$$

for every $n' \leq n$. The computation path of G on A' is thus the same as the computation path on A, i.e. $\sigma(A', G) = \sigma(A, G)$. It therefore follows that the algorithm accepts A' since it accepts A, and thus $U_A \subseteq \mathcal{C}$. It now follows from Proposition 1.12 that $\mathcal{C}$ is open. If $\mathcal{C}$ is decided by an adaptive left unbounded query algorithm over $\mathbb{B}$, then so is $\mathcal{C}^c$ (by switching YES and NO in the deciding step of the algorithm). Our argument therefore also shows that $\mathcal{C}^c$ is open, so $\mathcal{C}$ is clopen.

$\Leftarrow$: Let $\mathcal{C} \subseteq \mathsf{FIN}(\tau)$ be clopen. Let $A_0, A_1, A_2, \ldots$ be an enumeration of all structures with signature τ up to isomorphism. For all $\sigma = (\sigma_0, \ldots, \sigma_n) \in \mathbb{B}^{<\omega}$ we have a corresponding basis element

$$U_\sigma = \bigcap_{\substack{i \leq n \\ \sigma_i = 1}} \uparrow A_i \cap \bigcap_{\substack{i \leq n \\ \sigma_i = 0}} \mathsf{FIN}(\tau) \setminus \uparrow A_i.$$

Moreover, if A is a target structure defining σ, so

$$\sigma = (\hom_{\mathbb{B}}(A_0, A), \ldots, \hom_{\mathbb{B}}(A_n, A))$$

then we have that $A \in U_\sigma$. We now define the following algorithm:

$$G(\sigma) = \begin{cases} \mathsf{YES} & \text{if } U_\sigma \subseteq \mathcal{C} \\ \mathsf{NO} & \text{if } U_\sigma \subseteq \mathcal{C}^c \\ A_{|\sigma|} & \text{otherwise} \end{cases}.$$

If $\sigma = (\hom_{\mathbb{B}}(A_0, A), \ldots, \hom_{\mathbb{B}}(A_n, A))$ and $U_\sigma \subseteq \mathcal{C}$ then clearly $A \in \mathcal{C}$. Similarly, if $U_\sigma \subseteq \mathcal{C}^c$, then $A \notin \mathcal{C}$, so the algorithm always classifies correctly when it terminates. To see that it always terminates, we note that for every A we have either $A \in \mathcal{C}$ or $A \in \mathcal{C}^c$. Assume, without loss of generality, that $A \in \mathcal{C}$. Then, since $\mathcal{C}$ is open, there is a basis element

$$U = \bigcap_{i=0}^{k} \uparrow B_i \cap \bigcap_{i=0}^{m} \mathsf{FIN}(\tau) \setminus \uparrow D_i$$

such that $A \in U \subseteq \mathcal{C}$. Now, there exists an n such that $B_i, D_j \in \{A_0, \ldots, A_n\}$ for all $i = 0, \ldots, k$ and $j = 0, \ldots, m$ (here we do not distinguish between isomorphic structures). So, then indeed $A \in U_\sigma \subseteq U \subseteq \mathcal{C}$ for

$$\sigma = (\hom_{\mathbb{B}}(A_0, A), \ldots, \hom_{\mathbb{B}}(A_n, A)).$$

This shows that the algorithm terminates. The argument is the same for $A \in \mathcal{C}^c$. We have thus shown that $\mathcal{C}$ is decided by an adaptive left unbounded query algorithm over $\mathbb{B}$.

□

We say that a class $\mathcal{C}$ of structures is homomorphism-closed if whenever $A \in \mathcal{C}$ and $A \to B$ for some structure B, then $B \in \mathcal{C}$. We can use the topological characterization to get a simpler characterization for the existence of an adaptive left unbounded query algorithm over $\mathbb{B}$ in the case of homomorphism-closed classes.

Theorem 4.3. *Let $\mathcal{C}$ be a homomorphism-closed class, then the following hold:*

(i) $\mathcal{C} = \bigcup_{A \in \mathcal{C}} \uparrow A$, hence $\mathcal{C}$ is open.

(ii) Furthermore, $\mathcal{C}$ is closed if and only if $\mathcal{C} = \bigcap_{i \in I} \bigcup_{j=0}^{n_i} \uparrow A_{i,j}$ for some structures $A_{i,j}$.

Proof. Since $\mathcal{C}$ is homomorphism-closed it follows directly that $\mathcal{C} = \bigcup_{A \in \mathcal{C}} \uparrow A$. Since $\uparrow A$ is open for each A, we have that $\mathcal{C}$ is open. We now turn to proving (ii). The right-to-left direction is immediate since $\bigcup_{j=0}^{n} \uparrow A_{i,j}$ are closed sets. For the other direction, we assume $\mathcal{C}$ is closed and thus clopen. Since $\mathcal{C}$ is closed and the basis is clopen, we have that $\mathcal{C}$ is an intersection of basis elements:

$$\mathcal{C} = \bigcap_{i \in I} \left(\bigcup_{j=0}^{n_i} {\uparrow} B_{i,j} \cup \bigcup_{j=0}^{k_i} \mathsf{FIN}(\tau) \setminus {\uparrow} D_{i,j} \right). \tag{4.2.1}$$

In fact, we can assume that $B_{i,j}$ is connected for each (i, j):

Each $B_{i,j}$ can be written as $B_{i,j} = A_1 \oplus \ldots \oplus A_n$ where A_k is connected for each k. Then ${\uparrow} B_{i,j} = \bigcap_{k=1}^{n} {\uparrow} A_k$ since $\oplus$ is a least upper bound operation in the homomorphism lattice. Using the distributive law it is clear that each term $\bigcup_{j=0}^{n_i} {\uparrow} B_{i,j} \cup \bigcup_{j=0}^{k_i} \mathsf{FIN}(\tau) \setminus {\uparrow} D_{i,j}$ can be written in conjunctive normal form as:

$$\bigcap_{k=0}^{n} \left(\bigcup_{j=0}^{n_i'} {\uparrow} A_{i,k,j} \cup \bigcup_{j=0}^{k_i'} \mathsf{FIN}(\tau) \setminus {\uparrow} D'_{i,k,j} \right)$$

where each $A_{i,k,j}$ is connected. Thus, we can write

$$\mathcal{C} = \bigcap_{i \in I} \bigcap_{k=0}^{n} \left(\bigcup_{j=0}^{n_i'} {\uparrow} A_{i,k,j} \cup \bigcup_{j=0}^{k_i'} \mathsf{FIN}(\tau) \setminus {\uparrow} D'_{i,k,j} \right).$$

The outer intersections can be taken together to form an equation of the form (4.2.1) where each $B_{i,j}$ is connected.

We can now assume equation (4.2.1) holds and all $B_{i,j}$'s are connected. If

$$\mathcal{C} = \bigcap_{i \in I} \bigcup_{j=0}^{n_i} {\uparrow} B_{i,j}$$

then we are done. Otherwise there exists a structure $H \in \mathcal{C}$ such that $H \notin \bigcup_{j=0}^{n_i} {\uparrow} B_{i,j}$ for some i. Since $\mathcal{C}$ is homomorphism-closed we also have that $H' := H \oplus \bigoplus_{j=0}^{k_i} D_{i,j} \in \mathcal{C}$, but $H' \notin \bigcup_{j=0}^{k_i} \mathsf{FIN}(\tau) \setminus {\uparrow} D_{i,j}$. This means that we must have $H' \in \bigcup_{j=0}^{n_i} {\uparrow} B_{i,j}$, so for some j we have $B_{i,j} \to H'$. Then, since $B_{i,j}$ is connected and $B_{i,j} \not\to H$, we must have $B_{i,j} \to D_{i,j'}$ for some j'. But then ${\uparrow} D_{i,j'} \subseteq {\uparrow} B_{i,j}$ so

$${\uparrow} B_{i,j} \cup (\mathsf{FIN}(\tau) \setminus {\uparrow} D_{i,j'}) = \mathsf{FIN}(\tau)$$

and thus

$$\bigcup_{j=0}^{n_i} {\uparrow} B_{i,j} \cup \bigcup_{j=0}^{k_i} \mathsf{FIN}(\tau) \setminus {\uparrow} D_{i,j} = \mathsf{FIN}(\tau).$$

We can therefore define

$$I' := \{ i \in I : \bigcup_{j=0}^{n_i} {\uparrow} B_{i,j} \cup \bigcup_{j=0}^{k_i} \mathsf{FIN}(\tau) \setminus {\uparrow} D_{i,j} \neq \mathsf{FIN}(\tau) \}$$

and obtain

$$\mathcal{C} = \bigcap_{i \in I'} \bigcup_{j=0}^{n_i} {\uparrow} B_{i,j}$$

as desired. $\square$

The preceding two theorems imply that a homomorphism-closed class $\mathcal{C}$ is decided by an adaptive left unbounded query algorithm over $\mathbb{B}$ if and only if $\mathcal{C}$ is an intersection of UCQs.

We also have a similar, but slightly less elegant, characterization for classes that are not necessarily homomorphism-closed.

Theorem 4.4. *$\mathcal{C}$ is closed if and only if for every $A \notin \mathcal{C}$ there exists a finite set $X \subseteq \uparrow\{A\}$ such that $\mathcal{C} \cap \uparrow\{A\} \subseteq \uparrow X$ but $A \notin \uparrow X$.*

Proof.

- $\Rightarrow$: Assume $\mathcal{C}$ is closed and let $A \in \mathcal{C}^c$. Then there exists a basis element

$$U = \bigcap_{i=0}^{k} \uparrow\{B_i\} \cap \bigcap_{i=0}^{m} \mathsf{FIN}(\tau) \setminus \uparrow\{D_i\}$$

  such that $A \in U \subseteq \mathcal{C}^c$. Since $A \in U$ we must have that $B_i \to A$ and thus $\uparrow\{A\} \subseteq \uparrow\{B_i\}$ for every $i = 0, \dots, k$. Let

$$U' := \uparrow\{A\} \cap \bigcap_{i=0}^{m} \mathsf{FIN}(\tau) \setminus \uparrow\{D_i\}.$$

  Then $A \in U' \subseteq U \subseteq \mathcal{C}^c$. We let $X := \{D_0 \oplus A, \dots, D_m \oplus A\}$. Clearly $X \subseteq \uparrow\{A\}$. Also, since $A \in U$ we have $D_i \nrightarrow A$ for each i, so $A \notin \uparrow X$. Moreover, if $B \in \mathcal{C} \cap \uparrow\{A\}$ then $B \notin U'$ so $B \in \uparrow\{D_i\}$ for some i (since $B \in \uparrow\{A\}$). But then $A \oplus D_i \to B$, so $B \in \uparrow X$. We have therefore shown that $\mathcal{C} \cap \uparrow\{A\} \subseteq \uparrow X$.

- $\Leftarrow$: Assume that for every $A \in \mathcal{C}^c$ there is a finite set $X \subseteq \uparrow\{A\}$ such that $\mathcal{C} \cap \uparrow\{A\} \subseteq \uparrow X$ and $A \notin \uparrow X$. Let A and the corresponding X be given. We simply note that

$$U' = \uparrow\{A\} \cap \bigcap_{D \in X} \mathsf{FIN}(\tau) \setminus \uparrow\{D\}$$

  is a basis element such that $A \in U' \subseteq \mathcal{C}^c$, this shows that $\mathcal{C}$ is closed.

$\square$

Note 4.5. *The condition on X in the above corollary is related to the concept of upwards frontiers. For example, we get that for every A that does not have an upwards frontier and every $\mathcal{C}$ such that*

$$\uparrow\{A\} \setminus [A]_{\leftrightarrow} \subseteq \mathcal{C} \subseteq ([A]_{\leftrightarrow})^c$$

we have that $\mathcal{C}$ is not decided by an adaptive left unbounded query algorithm over $\mathbb{B}$. By looking at the complements, we also get that every $\mathcal{C}$ such that

$$[A]_{\leftrightarrow} \subseteq \mathcal{C} \subseteq (\uparrow\{A\} \setminus [A]_{\leftrightarrow})^c$$

is also not decided by an adaptive left unbounded query algorithm over $\mathbb{B}$.

4.3 Case studies

In the previous section, we have given a topological characterization of the classes that are decided by adaptive left unbounded query algorithms over $\mathbb{B}$. We will now apply this characterization to obtain simpler characterizations for classes of specific forms. Since every class that is not closed under homomorphic equivalence is not decided by an adaptive left unbounded query algorithm over $\mathbb{B}$, the question is only interesting for families of classes that are all closed under homomorphic equivalence. In this chapter, we will study three such families: homomorphic equivalence classes, Constraint Satisfaction Problems (CSPs), and Datalog-definable classes.

For homomorphic equivalence classes and CSPs, it turns out that unboundedness does not increase the expressive power. We can combine known results with our observations from Section 4.2 to get the following:

Theorem 4.6. *Let $\mathcal{C}$ be any class of the form $[A]_{\leftrightarrow}$ or $\mathrm{CSP}(A)$ for a structure A. Then the following are equivalent:*

(i) $\mathcal{C}$ is decided by an adaptive left unbounded query algorithm over $\mathbb{B}$.

(ii) $\mathcal{C}$ is decided by a non-adaptive left k-query algorithm over $\mathbb{B}$ for some k.

Proof. If $\mathcal{C}$ is decided by a non-adaptive left query algorithm over $\mathbb{B}$, then it is clearly also decided by an adaptive one. For the other direction, first assume $\mathcal{C} = [A]_{\leftrightarrow}$ is decided by an adaptive left unbounded query algorithm over $\mathbb{B}$. Then $[A]_{\leftrightarrow}$ is clopen, so it is a union of basis elements. Since it is a singleton modulo homomorphic equivalence, it must be a union of a single basis element. It is thus an intersection of k subbasis elements, which correspond to homomorphism queries. Thus, it is clear that $[A]_{\leftrightarrow}$ is decided by a non-adaptive left k-query algorithm.

Now, assume $\mathcal{C} = \mathrm{CSP}(A)$ is decided by an adaptive left unbounded query algorithm over $\mathbb{B}$. Note that such an algorithm can be turned into an adaptive left unbounded query algorithm over $\mathbb{B}$ for $[A]_{\leftrightarrow}$ by also querying for the existence of a homomorphism from A. If such a homomorphism exists and the target structure is in $\mathrm{CSP}(A)$, then the algorithm accepts. From the above, it therefore follows that $[A]_{\leftrightarrow}$ is also decided by a *non*-adaptive left query algorithm over $\mathbb{B}$. Proposition 4.4 in ten Cate et al. [8] states that $[A]_{\leftrightarrow}$ is decided by a non-adaptive left query algorithm over $\mathbb{B}$ if and only if $\mathrm{CSP}(A)$ is. Thus, we have that $\mathrm{CSP}(A)$ is also decided by a non-adaptive left query algorithm over $\mathbb{B}$. □

This theorem shows that unboundedness does not help when deciding CSPs or homomorphic equivalence classes, and it gives an effective characterization of the classes of this form that are decided by an adaptive left unbounded query algorithm over $\mathbb{B}$:

Corollary 4.7. $\mathrm{CSP}(A)$ *is decided by an adaptive left unbounded query algorithm over $\mathbb{B}$ if and only if it first-order definable. Moreover, there is an effective algorithm that decides for a given structure A whether this holds.*

The fact that $\mathrm{CSP}(A)$ is decided by a non-adaptive left query algorithm over $\mathbb{B}$ if and only if it is first-order definable follows from Theorem 2.3. The fact that there is an algorithm that decides whether $\mathrm{CSP}(A)$ is first-order definable is a well-known result by Larose, Loten, and Tardif [23].

Our results here can be summarized into the following sequence of equivalent conditions.

Corollary 4.8. *Let $A \in \mathsf{FIN}(\tau)$. The following are equivalent:*

(i) *A has an upwards frontier*

(ii) *$[A]_{\leftrightarrow}$ is decided by an adaptive left unbounded query algorithm.*

(iii) *$\mathrm{CSP}(A)$ is decided by an adaptive left unbounded query algorithm*

(iv) *$[A]_{\leftrightarrow}$ is decided by a non-adaptive left query algorithm.*

(v) *$\mathrm{CSP}(A)$ is decided by a non-adaptive left query algorithm*

(vi) *$\mathrm{CSP}(A)$ is first-order definable.*

(vii) *A has an obstruction set.*

Moreover, there is an effective algorithm that decides for a given structure A whether this holds.

Proof. The equivalence of (ii)-(v) and the last statement is established by the proof of Theorem 4.6 and Corollary 4.7. The fact that (ii) implies (i) follows from Note 4.5. It is also easy to show that if A has a frontier, then there is a non-adaptive left query algorithm for $[A]_{\leftrightarrow}$. This shows that (i) implies (iv). Finally, the equivalence of (vi) and (vii) is a well-known result by Atserias [1]. $\square$

This result, among other things, gives a characterization of the structures A that have an upwards frontier. In particular, it shows that A has an upwards frontier if and only if it has an obstruction set. By going through query algorithms and CSPs, this seems like a rather roundabout method for proving this fact. Indeed, in the following, we provide a direct proof and show how to construct a duality from an upwards frontier and vice versa.

Theorem 4.9. *A structure A has an upwards frontier if and only if it has an obstruction set.*

Proof.

$\Leftarrow$: Let $F_1, \ldots, F_n$ be an obstruction set for A, so $C \not\to A$ if and only if $F_i \to C$ for some i. Then $A \oplus F_1, \ldots, A \oplus F_n$ is a frontier for A:

Clearly $A \to A \oplus F_i$. Also, $F_i \not\to A$ so $A \oplus F_i \not\to A$ for each i. Now let C be such that $A \to C$ but $C \not\to A$. By the duality, we have that $F_i \to C$ for some i, but then $A \oplus F_i \to C$.

$\Rightarrow$: Let $A_1, \ldots, A_n$ be a frontier for A. Define the set

$$\mathcal{F} := \{F : F \text{ is a connected component of } A_i \text{ for some } i \text{ and } F \not\to A\ \}.$$

We show that $\mathcal{F}$ is an obstruction set for A, so we show that for a given B we have $B \not\to A$ if and only if $F \to B$ for some $F \in \mathcal{F}$.

Clearly, we have that if $F \to B$ for some $F \in \mathcal{F}$, then $B \not\to A$ (since otherwise $F \to A$). Now, assume $B \not\to A$. Then $A \to A \oplus B$ and $A \oplus B \not\to A$, so there exists an i such that $A_i \to A \oplus B$. Since $A_i \not\to A$, there must be a connected component C of A_i such that $C \not\to A$. Then $C \in \mathcal{F}$ and $C \to B$, so we have shown that there exists $F \in \mathcal{F}$ such that $F \to B$.

$\square$

After this short digression, let us now return to our case studies. Consider the database query language Datalog. Here, we only consider Boolean Datalog programs. As mentioned in the preliminaries, such a program π defines a class of structures $\mathcal{C}_\pi$ that is homomorphism-closed and thus closed under homomorphic equivalence.

The class $\mathcal{C}$ of digraphs that contain a directed cycle from Theorem 4.1 is Datalog-definable, using the program:

$$\begin{array}{lcl} X(x,y) & \leftarrow & R(x,y) \\ X(x,y) & \leftarrow & X(x,x_1), R(x_1,y) \\ \mathrm{Ans}() & \leftarrow & X(z,z) \end{array}$$

This shows that unboundedness helps even when we restrict ourselves to Datalog-definable classes. Contrastingly, Theorem 4.6 shows that for every Datalog program that defines the complement of a CSP, unboundedness does *not* help. Corollary 4.7 furthermore shows that a class defined by such Datalog programs is decided by an adaptive left unbounded query algorithm if and only if it is first-order definable. Note that the Datalog programs that define the complement of a CSP form a non-trivial fragment of Datalog. Indeed, every (Boolean) monadic Datalog program whose rule bodies are "tree-shaped" defines a complement of a CSP (and the CSP in question can be constructed effectively from the Datalog program), cf. [17, 9].

We can use these observations to prove a negative result on the expressive power of adaptive left query algorithms over $\mathbb{B}$.

Theorem 4.10. *There exists a class definable by a monadic linear Datalog program that is not decided by an adaptive left unbounded query algorithm over* $\mathbb{B}$.

Proof. Let $\tau = \{R, P, Q\}$ be a signature with R a binary relation while P and Q are unary. Define the structure $A := \{\{0,1\}, R^A, P^A, Q^A\}$ with $R^A := \{(0,0),(1,1)\}$, $P_A = \{0\}$ and $Q_A = \{1\}$. A structure B is an element of CSP(A) if and only if there is no oriented walk from an element in P^B to an element in Q^B. It is a standard application of the Ehrenfeucht-Fraïsse method to show that this is not first-order definable. It then follows from Corollary 4.7 that the complement of CSP(A) is not decided by an adaptive left unbounded query algorithm over $\mathbb{B}$. However, this class is defined by the following monadic linear Datalog program:

$$\begin{array}{lcl} X(x) & \leftarrow & P(x) \\ X(y) & \leftarrow & X(x), R(x,y) \\ X(y) & \leftarrow & X(x), R(y,x) \\ \mathrm{Ans}() & \leftarrow & X(y), Q(y) \end{array}$$

□

We now give a characterization of the Datalog-definable classes that are also decided by an adaptive left unbounded query algorithm in terms of their *upper envelopes*.

Chaudhuri and Kolaitis [10, 11], in their study of approximations of Datalog programs by non-recursive queries, introduced the notion of an upper envelope. For a Boolean Datalog program deciding a class $\mathcal{C}$, an upper envelope is a union of conjunctive queries (UCQ) q that defines a class $\mathcal{C}'$ such that $\mathcal{C} \subseteq \mathcal{C}'$. Such an upper envelope can be equivalently viewed as a class $\mathcal{C}'$ of the form $\bigcup_{j=0}^{n} \uparrow A_j$ that contains $\mathcal{C}$. Since Datalog-definable classes $\mathcal{C}$ are homomorphism-closed, Theorem 4.3 therefore gives us the following:

Corollary 4.11. *A Datalog-definable class* $\mathcal{C}$ *is decided by an adaptive left unbounded query algorithm over* $\mathbb{B}$ *if and only if it is the intersection of its upper envelopes.*

4.4 Is the existence of an adaptive left unbounded query algorithm over $\mathbb{B}$ decidable?

In Section 4.2 we provided a characterization of the classes decided by an adaptive left unbounded query algorithm over $\mathbb{B}$. The characterizing condition given is, however, far from being decidable. This is not surprising, as *a priori* a class of structures does not have a finite representation, thus preventing it from being the input to an algorithm. To even formulate the decidability problem, we must therefore restrict our attention to enumerable subsets of the collection of all classes. In the case studies, we have already encountered three such enumerations: homomorphic equivalence classes, CSPs, and Datalog-definable classes. We have already seen in Corollary 4.8 that the problem is decidable in both the homomorphic equivalence class case and the CSP case. The case of Datalog-definable classes is the focus of the remainder of this section.

We will not resolve this undecidability question in the case of Datalog programs in this chapter. We will, however, explain why a general and appropriate proof strategy—that is, using a *Rice-style theorem* for Datalog, proved by Gaifman et al. [19]—does not work in our setting. In doing so, we prove some insightful results about the expressive power of adaptive left unbounded query algorithms over $\mathbb{B}$. Before stating the Rice-style theorem, we need to introduce a few concepts. This discussion follows the paper by Gaifman et al. [19].

In the rest of this section, we assume all structures contain the relation symbols ZERO, SUCC, and MAX in their signature; ZERO and MAX are unary, while SUCC is binary. We say that a structure is *ordered* if these relations have their standard meaning, so there are unique elements where ZERO and MAX are true, and SUCC is the successor relation in a linear order of the whole structure, starting at the element marked with ZERO and ending at the element marked with MAX. We let $\mathcal{C}_{\text{ord}}$ denote the class of ordered structures.

A *property*, here, refers to a property of a Datalog program with respect to a class of structures. Formally, a property P is a subset of $\Pi \times \mathcal{P}(\mathcal{C}_{\text{all}})$, where Π is the set of all Datalog programs and $\mathcal{C}_{\text{all}}$ is the class of all structures. If $(\pi, \mathcal{D}) \in P$, we say that the Datalog program π has the property P with respect to the class $\mathcal{D}$ of structures. A property *contains boundedness* if every bounded Datalog program has the property with respect to $\mathcal{C}_{\text{all}}$. A property P is *semantic* if it is closed under program equivalence, i.e. if π defines the same class of structures as π' within the class $\mathcal{D}$ then $(\pi, \mathcal{D}) \in P$ if and only if $(\pi', \mathcal{D}) \in P$. A property is *stable* if whenever a program has the property with respect to the class of all structures, it also has it with respect to the class of all ordered structures. Finally, a property is *strongly non-trivial* if there exists a program that does not have the property with respect to the class of all ordered structures.

We can now state the Rice-style theorem:

Theorem 4.12 (Theorem 5.8 in [19]). *If P is a property that is semantic, stable, strongly non-trivial, and contains boundedness, then it is undecidable whether a program π has the property P with respect to the class of all structures.*

If we wanted to use this theorem to prove that the problem of determining whether a Datalog program admits an equivalent adaptive left unbounded query algorithm is undecidable, the straightforward way to define the property would be to let $(\pi, \mathcal{D}) \in P$ if and only if there exists an adaptive left unbounded query algorithm over $\mathbb{B}$ that decides the same class as π on $\mathcal{D}$, where the query algorithm can use any structure in its queries, not only the structures in $\mathcal{D}$. This almost works. It is not hard to see that this property

P is semantic, stable, and contains boundedness. It is also non-trivial, as shown in Theorem 4.10. However, it is not *strongly* non-trivial:

Theorem 4.13. *Every class of ordered structures that is closed under homomorphic equivalence can be decided by an adaptive left unbounded query algorithm over* $\mathbb{B}$.

Proof. We assume we are working with a signature τ containing ZERO, SUCC, and MAX. For each natural number n we let L_n be the linear order of size n using SUCC, so it has domain $\{0, \ldots, n-1\}$ and SUCC is the successor relation. All other relations, including ZERO and MAX, are empty in L_n. Given an ordered structure D as input, we have that $L_n \to D$ if and only if $n = |L_n| \leq |D|$.

Thus, we see that on input D, an adaptive left unbounded query algorithm over $\mathbb{B}$ can query $\hom_{\mathbb{B}}(L_n, D)$ for $n = 1, 2, \ldots$, until it finds the first L_n such that $L_n \nrightarrow D$. Then $|D| = n - 1$. Now, knowing the size of D, there are only finitely many possible homomorphic equivalence classes D can belong to, since there are only finitely many structures of this size. The algorithm then queries $\hom_{\mathbb{B}}(B, D)$ for one structure B from each of these finitely many homomorphic equivalence classes. Let $\mathcal{B}$ be the set of these structures B. The information obtained from these queries is enough to identify the homomorphic equivalence class of D, as shown by the following argument: Let D' is a structure with $|D'| = |D|$ and $\hom_{\mathbb{B}}(\mathcal{B}, D) = \hom_{\mathbb{B}}(\mathcal{B}, D')$. Then there exist $A, A' \in \mathcal{B}$ such that $A \leftrightarrow D$ and $A' \leftrightarrow D'$. We then get $\hom_{\mathbb{B}}(A', D) = \hom_{\mathbb{B}}(A', D') = 1$ and $\hom_{\mathbb{B}}(A, D') = \hom_{\mathbb{B}}(A, D) = 1$. This shows that

$$D \to A \to D' \to A' \to D,$$

so $D \leftrightarrow D'$, and thus D and D' belong to the same homomorphic equivalence class.

We have thus shown that the homomorphic equivalence class of an input ordered structure D can be determined by an adaptive left unbounded query algorithm over $\mathbb{B}$. This information suffices for the algorithm to be able to classify D with respect to any class closed under homomorphic equivalence □

Note that this proof shows that if given the size of the input structures, adaptive left unbounded query algorithms over $\mathbb{B}$ can decide any class closed under homomorphic equivalence. For ordered structures, the algorithm can obtain the size of the linear order, leading to every class closed under homomorphic equivalence to be decided by such an algorithm. Then, in particular, every Datalog-definable class of ordered structures is decided by such an algorithm. Therefore, this shows that the property P, defined previously, does not satisfy strong non-triviality, and we can thus not use it in Theorem 4.12.

This raises the question of whether we can redefine the property P such that it satisfies the conditions of Theorem 4.12 and can still be used to prove our desired undecidability result? Unfortunately, this is not possible. To see this, let P be a property that is semantic, stable, strongly non-trivial, and contains boundedness. To be able to use P to prove our undecidability result, we must also have that $(\pi, \mathcal{C}_{\text{all}}) \in P$ if and only if $\mathcal{C}_\pi$ is decided by an adaptive left unbounded query algorithm over $\mathbb{B}$. From the semantic condition, it follows that if

$$\mathcal{C}_\pi \cap \mathcal{C}_{\text{ord}} = \mathcal{C}_{\pi'} \cap \mathcal{C}_{\text{ord}}$$

then $(\pi, \mathcal{C}_{\text{ord}}) \in P$ if and only if $(\pi', \mathcal{C}_{\text{ord}}) \in P$. From the stability condition, we then get that if $\mathcal{C}_\pi$ is decided by an adaptive left unbounded query algorithm over $\mathbb{B}$ and $\mathcal{C}_\pi \cap \mathcal{C}_{\text{ord}} = \mathcal{C}_{\pi'} \cap \mathcal{C}_{\text{ord}}$, then $(\pi', \mathcal{C}_{\text{ord}}) \in P$. Thus, if P is strongly non-trivial, there exists a Datalog program π' such that for all π with $C_\pi \cap \mathcal{C}_{\text{ord}} = \mathcal{C}_{\pi'} \cap \mathcal{C}_{\text{ord}}$ we have that $\mathcal{C}_\pi$ is not decided by an adaptive left unbounded query algorihtm over $\mathbb{B}$. This is not true:

Theorem 4.14. *For every Datalog program π there exists a Datalog program π' such that π and π' define the same class of ordered structures and $\mathcal{C}_{\pi'}$ is decided by an adaptive left unbounded query algorithm over $\mathbb{B}$.*

Proof. The idea of the proof is to define π' such that it runs π, but only on the SUCC-linear orders (a SUCC-linear order is a linear order in the SUCC relation). We also let π' search for a directed cycle in the SUCC relation and accept if it finds it. Since ordered structures are SUCC-linear orders, it is clear that this π' defines the same class of ordered structures as π. Moreover, an adaptive left unbounded query algorithm over $\mathbb{B}$ can, using the method described in the proof of Theorem 4.1, simultaneously look for a SUCC-directed cycle and for the largest directed SUCC-linear order, and find either one. If it finds the size of the largest SUCC-linear order, it can run the Datalog program π' by using its unfoldings in its queries. Since it knows the size of the largest SUCC-component, it knows how long it needs to run the program to get the right answer.

Let us describe these constructions more formally. The program π' has the rules

$$\begin{aligned} \mathsf{SUCC}'(x,y) &\leftarrow \mathsf{SUCC}(x,y) \\ \mathsf{SUCC}'(x,y) &\leftarrow \mathsf{SUCC}'(x,z), \mathsf{SUCC}(z,y) \\ \text{Ans}() &\leftarrow \mathsf{SUCC}'(x,x) \end{aligned}$$

where Ans() is the goal predicate of π. SUCC' computes the transitive closure of SUCC, thus π' accepts if there is a SUCC-directed cycle. Let k be the maximum number of variables that appear in a single rule of π. For each $i \leq k$ we can define the predicate $L_i(x_1, \ldots, x_i)$, which holds if and only if $x_1, \ldots, x_i$ belong to the same SUCC-linear order, meaning that there exists a permutation $\sigma : \{1, \ldots, i\} \to \{1, \ldots, i\}$ such that $\mathsf{SUCC}'(x_{\sigma(1)}, x_{\sigma(2)}), \ldots, \mathsf{SUCC}'(x_{\sigma(i-1)}, x_{\sigma(i)})$ all hold. Note that this predicate can be computed directly from the transitive closure SUCC' of SUCC using a large disjunction. For each rule

$$\alpha \quad \leftarrow \quad \alpha_1, \ldots, \alpha_l$$

of π we add the rule

$$\alpha \quad \leftarrow \quad \alpha_1, \ldots, \alpha_l, L_r(x_1, \ldots, x_r)$$

to π', where $x_1, \ldots, x_r$ are all the variables that appear in $\alpha, \alpha_1, \ldots, \alpha_l$. This completes the description of π'. For ordered structures, $L_r(x_1, \ldots, x_r)$ always holds, so π' returns the same answer as π for those structures. Also note that for each IDB X of π' and each structure A that has no SUCC-directed cycle, we have that $X^{n+n^k}(A) = X^\infty(A)$, where n is the largest linear order in A. The first n steps are needed to compute SUCC', the remaining n^k steps suffice to run the rest of the program, because the program can be seen as running independently on each maximal linear order.

We now define the adaptive left unbounded query algorithm over $\mathbb{B}$ that decides the same class of structures as π'. As in Theorem 4.1, it queries for the existence of a homomorphism from the SUCC-directed cycle of length m and the SUCC-path of length m for $m = 1, 2, \ldots$ until it either finds a cycle or there is no homomorphism from the path. If it finds a cycle, it accepts, otherwise, it knows the size n of the largest SUCC-linear order. It can now query for the existence of a homomorphism from the canonical structure A_{q_j} of the j-th unfolding of π' for $j = 1, \ldots, n + n^k$. If any of these queries return 1, then the algorithm accepts, otherwise, it rejects. This algorithm decides the same class as π' since for all structures containing no directed cycles we have $X^{n+n^k}(A) = X^{\infty}(A)$. This concludes the proof. □

This is another positive theorem concerning the expressive power of adaptive left unbounded query algorithms over $\mathbb{B}$. Together with the preceding argument, it shows that we cannot use Theorem 4.12 to prove the undecidability of the existence of an adaptive left unbounded query algorithm over $\mathbb{B}$ deciding $\mathcal{C}_\pi$.

We now discontinue our efforts to clarify this question and leave it as an interesting open problem.

4.5 Adaptive right query algorithms over $\mathbb{B}$

For adaptive *right* query algorithms over $\mathbb{B}$ we have a result analogous to Theorem 4.1:

Theorem 4.15. *The class $\mathcal{C}$ of all digraphs that have an oriented cycle with non-zero net length can be decided by an adaptive right unbounded query algorithm over $\mathbb{B}$, and not by an adaptive right k-query algorithm over $\mathbb{B}$, for any k.*

Before we prove the theorem, we state and prove a few lemmas.

Lemma 4.16 (Proposition 1.13 in [22])**.** *A digraph H has no oriented cycle with non-zero net length if and only if $H \to P_{|H|-1}$.*

Lemma 4.17. *Let A be a finite directed graph. We have that A has no oriented cycle with non-zero net length if and only if there exists $n > 0$ such that $\hom(A, \mathsf{P}_n) > 0$.*

Proof.

- $\Rightarrow$: Since the net length of oriented walks is preserved under homomorphisms and the net length of the longest oriented walk in P_n is n, we see that if $A \to \mathsf{P}_n$ then A has no oriented walk with net length $> n$. Then indeed A has no oriented cycle with positive net length (and thus no oriented cycle with non-zero net length).

- $\Leftarrow$: If A has no oriented cycles with positive net length, then it follows from Lemma 4.16 that $A \to P_{|A|-1}$.

□

Lemma 4.18. *Let A be a finite directed graph. We have that A has an oriented cycle with non-zero net length if and only if there exists $n > 0$ such that $\hom(A, \mathsf{C}_n) = 0$.*

Proof.

- $\Rightarrow$: If $\hom(A, \mathsf{C}_n) = 0$, then by Lemma 3.3 we have $n \nmid \gamma(A)$. In particular, $\gamma(A) \neq 0$, so A has an oriented cycle with non-zero net length.

- $\Leftarrow$: If A has a oriented cycle of non-zero net length then $\gamma(A) \neq 0$. But then $\gamma(A)+1 \nmid \gamma(A)$ so by Lemma 3.3 we have $\hom(A, C_{\gamma(A)+1}) = 0$.

□

Surprisingly, essentially the same unbounded algorithm as we used to prove Theorem 4.1 works here.

Proof of Theorem 4.15. We first show that the class $\mathcal{C}$ of all digraphs that have an oriented cycle with non-zero net length can be decided by an adaptive right unbounded query algorithm over $\mathbb{B}$. The algorithm runs as follows: Let A be the given input. For $n = 1, 2, \dots$ query $\hom_{\mathbb{B}}(A, \mathsf{P}_n)$ and $\hom_{\mathbb{B}}(A, \mathsf{C}_n)$. If at some point $\hom_{\mathbb{B}}(A, \mathsf{P}_n) = 1$ then halt and output NO. If at some point $\hom_{\mathbb{B}}(A, \mathsf{C}_n) = 0$ then halt and output YES.

It follows from Lemmas 4.17 and 4.18 that on every input, this algorithm halts and produces the correct outcome.

We now show that the class $\mathcal{C}$ cannot be decided by a non-adaptive right k-query algorithm over $\mathbb{B}$ and thus not by an adaptive right bounded query algorithm over $\mathbb{B}$.

Let $(\mathcal{F}, X)$ be a non-adaptive right k-query algorithm. Let

$$m \coloneqq \max(\{|F| : F \in \mathcal{F}\}),$$

so $m \geq |F|$ for every $F \in \mathcal{F}$. Then, since the lengths of directed walks are preserved under homomorphism, it holds for all $F \in \mathcal{F}$ that we have $P_m \to F$ if and only if F has a directed cycle. Also, if F has a directed cycle, then that cycle has length $\leq m$, so $C_{m!} \to F$. It is also clear that if $C_{m!} \to F$ then F must have a directed cycle. Thus $C_{m!} \to F$ if and only if F has a directed cycle. We have thus shown that for $F \in \mathcal{F}$ we have

$$P_m \to F \quad \text{if and only if} \quad C_{m!} \to F.$$

So the class $\mathcal{C}'$ that the algorithm decides either contains both P_m and $C_{m!}$ or neither of them. In both cases $\mathcal{C}' \neq \mathcal{C}$, so the algorithm does not decide the class $\mathcal{C}$. □

The class from the above result is Datalog-definable:

Theorem 4.19. *The class of all digraphs that have an oriented cycle with non-zero net length is Datalog-definable.*

Proof. A Datalog program defining this class is the following:

$$\begin{array}{rcc}
X(a,b) & \leftarrow & a = b \\
X(a,b) & \leftarrow & X(a',b'), R(a',a), R(b',b) \\
X(a,b) & \leftarrow & X(a',b'), R(a,a'), R(b,b') \\
X(a,b) & \leftarrow & X(a,c), X(c,b) \\
\\
Y(a,b) & \leftarrow & R(a,b) \\
Y(a,b) & \leftarrow & Y(a,c), X(c,b) \\
Y(a,b) & \leftarrow & X(a,c), Y(c,b) \\
Y(a,b) & \leftarrow & Y(a,c), Y(c,b) \\
\\
\text{Ans}() & \leftarrow & Y(a,a)
\end{array}$$

To see that this program does indeed define the class of digraphs that have an oriented cycle with non-zero net length, we begin by showing that the IDB X defines the relation

$$X(a,b) \iff \text{there is an oriented walk from } a \text{ to } b \text{ with net length } 0. \tag{4.5.1}$$

The left-to-right direction is clear. To see the other direction, we use induction on the length of walks of net length 0. Let P be a walk of length $2n$ but net length 0. We have a few cases:

- If P is of the form $(a_0, -, a_1, r_1, \dots, a_{2n-1}, +, a_{2n})$ then $(a_1, r_1, \dots, a_{2n-1})$ is an oriented walk of length $2(n-1)$ but net length 0, so by the induction hypothesis $X(a_1, a_{2n-1})$. It then follows from the second rule in the program that $X(a_0, a_{2n})$.
- The case where P is of the form $(a_0, +, a_1, r_1, \dots, a_{2n-1}, -, a_{2n})$ is similar.
- Now assume $P = (a_0, +, a_1, r_1, \dots, a_{2n-1}, +, a_{2n})$. Since it has net length 0 there must exist subwalks Q, R and Z of net length 0 such that

$$P = (a_0, +, a_1) \circ Q \circ (a_i, -, a_{i+1}) \circ R \circ (a_j, -, a_{j+1}) \circ Z \circ (a_{2n-1}, +, a_{2n}).$$

(Namely, we pick i as the smallest number such that $(a_0, +, \dots, a_{i+1})$ is an oriented walk of net length 0 and we pick j as the smallest number such that $(a_0, +, \dots, a_{j+1})$ is an oriented walk of net length -1). By the induction hypothesis, $X(a_1, a_i)$, $X(a_{i+1}, a_j)$ and $X(a_{j+1}, a_{2n-1})$ hold. Using the rules, it is clear that we also have $X(a_0, a_{2n})$.

- The case where P is of the form $(a_0, -, a_1, r_1, \dots, a_{2n-1}, -, a_{2n})$ is similar.

We have now proved the bi-implication (4.5.1). It is then easy to see that $Y(a, b)$ holds if and only if there is an oriented walk of positive net length from a to b. Now we see that the program accepts if and only if there is a closed oriented walk of positive net length, which holds if and only if there is an oriented cycle of non-zero net length (this follows from our proof of Proposition 3.3). □

Theorems 4.15 and 4.19 show that, as in the case for left query algorithms, unboundedness sometimes helps for adaptive right query algorithms, even when we restrict ourselves to Datalog-definable classes.

We can also replicate the results from Section 4.2 for right query algorithms. We define $\mathfrak{T}'$ as the topology generated by the subbasis

$$\mathcal{S}' := \{\downarrow A : A \in \mathsf{FIN}(\tau)\} \cup \{\mathsf{FIN}(\tau) \setminus \downarrow A : A \in \mathsf{FIN}(\tau)\}.$$

where $\downarrow A$ denotes the set $\{B \in \mathsf{FIN}(\tau) : B \to A\}$. In other words, $\mathcal{S}'$ consists of all CSPs and their complements. Using the same argument as before, we can prove:

Theorem 4.20. *A class $\mathcal{C} \subseteq \mathsf{FIN}(\tau)$ is decided by an adaptive right unbounded query algorithm over $\mathbb{B}$ if and only if $\mathcal{C}$ is a clopen set in the topological space $(\mathsf{FIN}(\tau), \mathfrak{T}')$.*

Then we also get

Theorem 4.21. *Let $\mathcal{C}$ be any class of the form $[A]_{\leftrightarrow}$ or $\uparrow A$ for a structure A. Then the following are equivalent:*

(i) $\mathcal{C}$ is decided by an adaptive right unbounded query algorithm over $\mathbb{B}$.

(ii) $\mathcal{C}$ is decided by a non-adaptive right k-query algorithm over $\mathbb{B}$ for some k.

Proof. We prove this theorem by showing that the following are equivalent:

(1) $[A]_{\leftrightarrow}$ is decided by an adaptive right unbounded query algorithm over $\mathbb{B}$.

(2) $\uparrow A$ is decided by an adaptive right unbounded query algorithm over $\mathbb{B}$.

(3) $[A]_{\leftrightarrow}$ is decided by a non-adaptive right query k-algorithm over $\mathbb{B}$ for some k.

(4) $\uparrow A$ is decided by a non-adaptive right k-query algorithm over $\mathbb{B}$ for some k.

(5) A is homomorphically equivalent to an acyclic structure.

The equivalence of (3), (4), and (5) is Theorem 6.5 in ten Cate et al. [8]. It is easy to see that (3) implies (1) and that (4) implies (2). Now, if $[A]_{\leftrightarrow}$ is clopen, then it is a union of basis elements. Since it is a singleton modulo homomorphic equivalence, it must be a union of a single basis element. It is thus an intersection of k subbasis elements, which correspond to right homomorphism queries. Thus, it is clear that $[A]_{\leftrightarrow}$ is decided by a non-adaptive right k-query algorithm. This shows that (1) implies (3). Now, assume $\uparrow A$ is decided by an adaptive right unbounded query algorithm over $\mathbb{B}$. Note that such an algorithm can be turned into an adaptive right unbounded query algorithm over $\mathbb{B}$ for $[A]_{\leftrightarrow}$ by also querying for the existence of a homomorphism to A. If such a homomorphism exists and the target structure is in $\uparrow A$, then the algorithm accepts. This shows that (2) implies (1).

We have thus shown that all of the conditions are equivalent. □

Using the language of conjunctive queries, this shows that a CQ q has an equivalent adaptive right unbounded query algorithm if and only if q is equivalent to a Berge-acyclic CQ. It is also good to note that since every CQ can be written as a Datalog program, this also shows that there are Datalog-definable classes that are not decided by any adaptive right unbounded query algorithm over $\mathbb{B}$.

The list of conditions that were proved to be equivalent in the preceding proof can be extended, as in the case of left query algorithms. It is a well-known result by Foniok, Nešetřil, and Tardif [18] that a structure A is homomorphically equivalent to an acyclic structure if and only if it is the left side of a duality, i.e. there exists a set $\mathcal{D}$ such that $(\{A\}, \mathcal{D})$ is a duality. It is also known that A is the left side of a duality if and only if it has a downwards frontier.

Chapter 5

When Does Counting Help?

In this chapter, we connect the two previous chapters by comparing the expressive power of query algorithms over $\mathbb{B}$ with that of algorithms where the counting is done over $\mathbb{N}$. The question of whether counting helps amounts to asking whether a certain type of query algorithm over $\mathbb{N}$ is more expressive than the same type over $\mathbb{B}$. The question is uninteresting for classes that are not closed under homomorphic equivalence, since query algorithms over $\mathbb{B}$ can never decide such classes. We therefore restrict our attention to classes that are closed under homomorphic equivalence.

5.1 Does counting help for non-adaptive right query algorithms?

As mentioned in the introduction, the main result proved in the paper by ten Cate, Dalmau, Kolaitis, and Wu [8] is the fact that non-adaptive left query algorithms over $\mathbb{N}$ are not more expressive than non-adaptive left query algorithms over $\mathbb{B}$ among the classes closed under homomorphic equivalence. In other words, for non-adaptive left query algorithms, counting does not help. Specifically, they prove the following.

Theorem 5.1 (Theorem 5.2 in ten Cate et al. [8]). *Let $\mathcal{C}$ be a class of structures closed under homomorphic equivalence. For every finite set $\mathcal{F}$ of connected structures, the following statements are equivalent.*

(i) $\mathcal{C}$ admits a non-adaptive left query algorithm over $\mathbb{N}$ of the form $(\mathcal{F}, X)$ for some set X.

(ii) $\mathcal{C}$ admits a non-adaptive left query algorithm over $\mathbb{B}$ of the form $(\mathcal{F}, X')$ for some set X'.

Proof idea. The direction (ii)⇒(i) is trivial. We focus on showing (i)⇒(ii). Let $(\mathcal{F}, X)$ be a non-adaptive left query algorithm over $\mathbb{N}$ deciding a class $\mathcal{C}$ that is closed under homomorphic equivalence. The idea is to show that $\mathcal{C}$ is also decided by the non-adaptive left query algorithm over $\mathbb{B}$ defined with $(\mathcal{F}, X')$ where

$$X' := \{(\min(a_1, 1), \dots, \min(a_n, 1)) : \hom_{\mathbb{N}}(\mathcal{F}, A) = (a_1, \dots, a_n) \in X \text{ for some } A \in \mathcal{C} \}.$$

The class decided by $(\mathcal{F}, X')$ clearly contains $\mathcal{C}$. To prove the other inclusion, let B be a structure accepted by $(\mathcal{F}, X')$. Then there exists $A \in \mathcal{C}$ such that $F \to A$ if and only if $F \to B$ for all $F \in \mathcal{F}$ (then B has $\hom_{\mathbb{B}}(\mathcal{F}, B) = (\min(a_1, 1), \dots, \min(a_n, 1))$ where $\hom_{\mathbb{N}}(\mathcal{F}, A) = (a_1, \dots, a_n)$). To complete the proof, it suffices to show that $B \in \mathcal{C}$.

This is done by using the operations $\oplus$ and $\otimes$ and Proposition 1.1 to show that there exist structures A', B' such that $A' \leftrightarrow A$ and $B' \leftrightarrow B$ and $\hom_{\mathbb{N}}(\mathcal{F}, A') = \hom_{\mathbb{N}}(\mathcal{F}, B')$. Since $\mathcal{C}$ is closed under homomorphic equivalence, we have that $A' \in \mathcal{C}$. Since A' and B' have the same homomorphism profiles over $\mathbb{N}$ restricted to $\mathcal{F}$, then $(\mathcal{F}, X)$ must also accept B', so $B' \in \mathcal{C}$. Then finally, since $B' \leftrightarrow B$, we also get $B \in \mathcal{C}$. We will not go into the details of how A' and B' are constructed. We point the interested reader to the original paper. $\square$

The corresponding question of whether counting helps for non-adaptive right query algorithms was left open in the paper by ten Cate et al. We will not resolve this problem, but we will give an answer in a specific case.

Note that Theorem 5.1 shows something slightly stronger than only that non-adaptive left query algorithms over $\mathbb{N}$ and $\mathbb{B}$ have equal expressive power for classes closed under homomorphic equivalence. Namely, that a non-adaptive left query algorithm over $\mathbb{N}$ deciding a class closed under homomorphic equivalence can be turned into a non-adaptive left query algorithm over $\mathbb{B}$ that decides the same class *without changing the queries.* We show that this does not hold for right query algorithms.

Theorem 5.2. *There exists a finite set $\mathcal{F}$ of connected structures such that there is a non-adaptive right query algorithm $(\mathcal{F}, X)$ over $\mathbb{N}$ that decides a class $\mathcal{C}$ that is closed under homomorphic equivalence, but there is no non-adaptive right query algorithm over $\mathbb{B}$ of the form $(\mathcal{F}, X')$ that decides $\mathcal{C}$.*

Proof. Let $\mathcal{F} = \{F\}$ where $F = (\{0,1,2\}, \{(0,1),(1,0),(0,2),(2,0)\})$ (see Figure 5.1).

F is clearly connected. Define $X = \{3^n : n \in \mathbb{N}\}$.

Claim: The non-adaptive right query algorithm $(\mathcal{F}, X)$ over $\mathbb{N}$ decides the class of directed graphs that have no edges.

Proof of claim. Let B be a directed graph. If B we has no edges then $\hom(B, F) = 3^{|B|}$ so the algorithm accepts B. If B has an edge, we have two cases:

- If B is non-bipartite then $B \not\to F$ since $F \to C_2$ so otherwise B would be bipartite. Thus $\hom(B, F) = 0$ and the algorithm rejects B.

- If B is bipartite, we can write $B = B_1 \oplus \ldots \oplus B_k$ where B_1 is bipartite, has an edge, and is connected. We show that $\hom(B_1, F)$ is even. It is easy to see that a connected bipartite digraph has a unique bipartition. Since B_1 has an edge, we have that V_1 and V_2 are non-empty in the unique bipartition $\{V_1, V_2\}$ of B_1. Let $g : F \to \mathsf{C}_2$ be the homomorphism mapping 0 to 0, and 1 and 2 to 1. Then $f : B_1 \to F$ is a homomorphism if and only if $g \circ f$ is a homomorphism, which holds if and only if $\{(g \circ f)^{-1}(0), (g \circ f)^{-1}(1)\}$ is a bipartition of B_1. It is thus clear that $f : B_1 \to F$ is a homomorphism if and only if $f^{-1}(\{0\}) = V_1$ and $f^{-1}(\{1,2\}) = V_2$ or $f^{-1}(\{0\}) = V_2$ and $f^{-1}(\{1,2\}) = V_1$. There exist $2^{|V_2|}$ functions satisfying the former conditions and $2^{|V_1|}$ functions satisfying the latter.

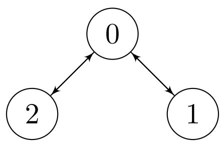

Figure 5.1: The digraph F.

Thus $\hom(B_1, A) = 2^{|V_1|} + 2^{|V_2|}$ which is even since V_1 and V_2 are non-empty. Finally, we can now deduce that

$$\hom(B, A) = \hom(B_1 \oplus \ldots, B_n, A) = \hom(B_1, A) \cdots \hom(B_n, A)$$

is even and thus $(\mathcal{F}, X)$ rejects B.

We have now proved the claim.

To finish the proof of the theorem, we note that a non-adaptive right query algorithm $(\mathcal{F}, X')$ over $\mathbb{B}$ has $X' \subseteq \{0, 1\}$. If $1 \in X'$, then the algorithm accepts the bipartite digraphs (some of which contain an edge), and if $1 \notin X'$, then the algorithm does not accept any digraphs that contain no edges. In both cases, it does not accept the same digraphs as $(\mathcal{F}, X)$. □

Note that this does not disprove the weaker statement that counting does not help for non-adaptive right query algorithms, which remains an open question. This is because the class of digraphs with no edges is decided by the non-adaptive right query algorithm over $\mathbb{B}$ given by $(\{F'\}, \{1\})$ where $F' = (\{0\}, \varnothing)$ is the singleton digraph with no edges. This result does, however, provide some indication that counting might help for non-adaptive right query algorithms.

5.2 Counting helps for adaptive query algorithms

Our work in the previous chapters allows us to answer the question of whether counting helps for every type of adaptive query algorithm. Indeed, in all of these cases, counting does help, even for classes closed under homomorphic equivalence.

Theorem 5.3.

(i) There exists a class closed under homomorphic equivalence that is decided by an adaptive left 2-query algorithm over $\mathbb{N}$ but not by an adaptive left k-query algorithm over $\mathbb{B}$ for any k.

(ii) There exists a class closed under homomorphic equivalence that is decided by an adaptive left unbounded query algorithm over $\mathbb{N}$ but not by an adaptive left unbounded query algorithm over $\mathbb{B}$.

(iii) There exists a class closed under homomorphic equivalence that is decided by an adaptive right 2-query algorithm over $\mathbb{N}$ but not by an adaptive right k-query algorithm over $\mathbb{B}$ for any k.

(iv) There exists a class closed under homomorphic equivalence that is decided by an adaptive right unbounded query algorithm over $\mathbb{N}$ but not by an adaptive right unbounded query algorithm over $\mathbb{B}$.

Proof.

(i) Consider the class of all digraphs that contain a directed cycle. This class is closed under homomorphisms and is thus closed under homomorphic equivalence. In Theorem 3.17 we show that this class is decided by an adaptive left 2-query algorithm over $\mathbb{N}$ and in Theorem 4.1 we show that it is not decided by an adaptive left k-query algorithm over $\mathbb{B}$ for any k.

(ii) As shown in Example 2.6, every class of structures is decided by an adaptive left unbounded query algorithm over $\mathbb{N}$. However, not all classes closed under homomorphic equivalence are decided by an adaptive left unbounded query algorithm over $\mathbb{B}$. One such example is given in Theorem 4.10.

(iii)-(iv) In Theorem 3.26, it is shown that every class is decided by an adaptive right 2-query algorithm. However, the equivalences shown in the proof of Theorem 4.21 tell us that the class $[\mathsf{C}_1]_{\leftrightarrow}$ is not decided by an adaptive right unbounded query algorithm over $\mathbb{B}$, since C_1 is not acyclic.

$\square$

This result further shows how surprising and fragile Theorem 5.1 is by demonstrating that it does not generalize to any kind of adaptive query algorithm we have studied.

Conclusions and Further Research

In this thesis, we studied the expressive power of various types of homomorphism query algorithms. Over the course of these studies, we obtained several new results.

We began, in Chapter 2, by introducing the previously unexplored notion of an adaptive *unbounded* query algorithm.

In Chapter 3 we studied query algorithms over $\mathbb{N}$. In contrast to the case of simple graphs, we showed that there exists a class of relational structures that cannot be decided by an adaptive left bounded query algorithm over $\mathbb{N}$. We provided a complete description of the comparative expressive power of adaptive and non-adaptive left k-query algorithms over $\mathbb{N}$. We also introduced the notion of an $f(n)$-query algorithm and established upper and lower bounds for the asymptotic growth of a function f having the property that all classes of structures can be decided by an adaptive left $f(n)$-query algorithm over $\mathbb{N}$. There remains a significant gap between these upper and lower bounds, the reduction of which is an interesting direction for future research. Finally, we used previous work by Wu [30] to show that every class can be decided with only two adaptive right homomorphism queries over $\mathbb{N}$.

In Chapter 4 we explored query algorithms over $\mathbb{B}$. We thoroughly studied the expressive power of adaptive left unbounded query algorithms over $\mathbb{B}$, showing that they are more expressive than their bounded counterparts, provided a characterization of the classes they can express, and studied how the characterization simplifies in the special cases of homomorphic equivalence classes, CSPs, and Datalog-definable classes. We shed some light on the decidability question for the problem of existence of an equivalent adaptive left unbounded query algorithm for a given Datalog program, though the problem remains unresolved. We also briefly discussed the expressive power of adaptive right query algorithm over $\mathbb{B}$ and showed that analogous results can be obtained in that case.

Finally, in Chapter 5, we use the results from Chapters 3 and 4 to show that for all the types of adaptive query algorithms studied, counting does help, that is, their expressive power increases when the homomorphism counting is done over $\mathbb{N}$ instead of $\mathbb{B}$. We also partially resolved the question of whether counting helps for non-adaptive right bounded query algorithms, which was a question left open by ten Cate et al. [8]. However, the full question remains open.

Several aspects related to the topics discussed in this thesis remain unstudied.

One such example involves homomorphism-density queries, which have been studied, for example, by Böker [5]. A homomorphism density query asks for the fraction of functions between two structures that are homomorphisms, instead of the number of homomorphisms. Homomorphism density query algorithms are then at least as expressive as query algorithms over $\mathbb{B}$, since the fraction of homomorphisms reveals whether a homomorphism exists or not. They are also at most as expressive as query algorithms over $\mathbb{N}$, since the fraction of homomorphisms can be reconstructed from the homomorphism count and the number of vertices of the two structures. However, their exact expressive power remains unclear.

Another variation of query algorithms is obtained by replacing the homomorphism count queries with queries that count the number of distinct ways to map a tuple of *marked* elements in a homomorphism. Specifically, given structures B, A, and a tuple $\mathbf{b} = (b_1, \dots, b_n)$ of elements of B we let $\hom(B_{\mathbf{b}}, A)$ denote the number

$$|\{(a_1, \dots, a_n) : \text{ there exists } \varphi : B \to A \text{ such that } \varphi(b_i) = a_i \text{ for each } i\ \}|.$$

This counts the number of distinct ways the tuple $\mathbf{b}$ can be mapped to A by a homomorphism. This setting arises naturally in the study of conjunctive queries and has been examined by Bielecki and Van den Bussche [4]. Note that these queries generalize both homomorphism count queries over $\mathbb{N}$ and $\mathbb{B}$: choosing $\mathbf{b}$ to include all elements of B yields $\hom(B_{\mathbf{b}}, A) = \hom_{\mathbb{N}}(B, A)$, while choosing $\mathbf{b}$ as the empty tuple yields $\hom(B_{\mathbf{b}}, A) = \hom_{\mathbb{B}}(B, A)$. Most of the questions we explored for the other types of query algorithms remain unaddressed in this setting; we leave their investigation to future work.

We can also look at *hybrid* query algorithms, which can use both left and right queries. Adaptive hybrid query algorithms over $\mathbb{N}$ have been studied by Wu [29], but there has been no research on such algorithms over $\mathbb{B}$. Every class of structures that is closed under homomorphic equivalence can be decided by an adaptive hybrid unbounded query algorithm over $\mathbb{B}$. On input B, the algorithm simply queries for $\hom_{\mathbb{B}}(A, B)$ and $\hom_{\mathbb{B}}(B, A)$ for each structure A. At some point, both of these numbers are 1, then the homomorphic equivalence class of B is decided, which suffices to classify it correctly. This evokes the question of what the expressive power of hybrid k-query algorithms is: Do we get a strict hierarchy as in Figure 1? What is the relation between the expressive power of non-adaptive and adaptive hybrid query algorithms over $\mathbb{B}$?

Finally, it is natural to consider restrictions on the size of the queries used. For example, in Theorem 3.26, the second query is exponentially large relative to the input size. The question of whether, for every class of structures, there exists an adaptive right unbounded query algorithm over $\mathbb{N}$ that uses a polynomial amount of polynomially large queries remains open and may be very difficult to resolve. The corresponding question for left query algorithms is likewise open and may prove equally challenging.

Bibliography

[1] Albert Atserias. On digraph coloring problems and treewidth duality. *European Journal of Combinatorics*, 29(4):796–820, 2008. Homomorphisms: Structure and Highlights. URL: `https://www.sciencedirect.com/science/article/pii/S0195669807002004`, `doi:10.1016/j.ejc.2007.11.004`.

[2] Albert Atserias, Phokion G Kolaitis, and Wei-Lin Wu. On the expressive power of homomorphism counts. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2021.

[3] Pablo Barceló, Floris Geerts, Juan Reutter, and Maksimilian Ryschkov. Graph neural networks with local graph parameters. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 25280–25293. Curran Associates, Inc., 2021. URL: `https://proceedings.neurips.cc/paper_files/paper/2021/file/d4d8d1ac7e00e9105775a6b660dd3cbb-Paper.pdf`.

[4] Michał Bielecki and Jan Van den Bussche. Database interrogation using conjunctive queries. In *International Conference on Database Theory*, pages 259–269. Springer, 2002.

[5] Jan Böker. Graph similarity and homomorphism densities. *arXiv preprint arXiv:2104.14213*, 2021.

[6] Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–330, 2017. `doi:10.1109/FOCS.2017.37`.

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

[8] Balder ten Cate, Victor Dalmau, Phokion G Kolaitis, and Wei-Lin Wu. When do homomorphism counts help in query algorithms? In *27th International Conference on Database Theory (ICDT 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[9] Balder ten Cate, Víctor Dalmau, and Jakub Opršal. Right-Adjoints for Datalog Programs. In Graham Cormode and Michael Shekelyan, editors, *27th International Conference on Database Theory (ICDT 2024)*, volume 290 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:20, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICDT.2024.10`, `doi:10.4230/LIPIcs.ICDT.2024.10`.

[10] Surajit Chaudhuri. Finding nonrecursive envelopes for Datalog predicates. In *Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 135–146, 1993.

[11] Surajit Chaudhuri and Phokion G Kolaitis. Can Datalog be approximated? In *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 86–96, 1994.

[12] Surajit Chaudhuri and Moshe Y Vardi. Optimization of real conjunctive queries. In *Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 59–70, 1993.

[13] Yijia Chen, Jörg Flum, Mingjun Liu, and Zhiyang Xun. On algorithms based on finitely many homomorphism counts. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.MFCS.2022.32`, `doi:10.4230/LIPIcs.MFCS.2022.32`.

[14] Jesse A Comer. Homomorphism counts, database queries, and modal logics, 2023.

[15] Zdeněk Dvořák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64(4):330–342, 2010.

[16] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer Berlin Heidelberg, 1995.

[17] Péter L. Erdős, Dömötör Pálvölgyi, Claude Tardif, and Gábor Tardos. Regular families of forests, antichains and duality pairs of relational structures. *Combinatorica*, 37(4):651–672, 2017. `doi:10.1007/s00493-015-3003-4`.

[18] Jan Foniok, Jaroslav Nešetřil, and Claude Tardif. Generalised dualities and maximal finite antichains in the homomorphism order of relational structures. *European Journal of Combinatorics*, 29(4):881–899, 2008.

[19] Haim Gaifman, Harry Mairson, Yehoshua Sagiv, and Moshe Y. Vardi. Undecidable optimization problems for database logic programs. *J. ACM*, 40(3):683–713, July 1993. `doi:10.1145/174130.174142`.

[20] Martin Grohe. Counting bounded tree depth homomorphisms. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 507–520, 2020.

[21] GH Hardy. *An introduction to the theory of numbers*. Oxford Science Publication, 1979.

[22] Pavol Hell and Jaroslav Nesetril. *Graphs and homomorphisms*, volume 28. OUP Oxford, 2004.

[23] Benoit Larose, Cynthia Loten, and Claude Tardif. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Computer Science*, 3, 2007.

[24] László Lovász. Operations with structures. *Acta Mathematica Hungarica*, 18(3-4):321–328, 1967.

[25] Laura Mančinska and David E. Roberson. Quantum isomorphism is equivalent to equality of homomorphism counts from planar graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 661–672, 2020. `doi:10.1109/FOCS46700.2020.00067`.

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

[27] Benjamin Rossman. Existential positive types and preservation under homomorphisms. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)*, pages 467–476. IEEE, 2005.

[28] Tim Seppelt. Logical equivalences, homomorphism indistinguishability, and forbidden minors. *Information and Computation*, 301:105224, 2024. URL: `https://www.sciencedirect.com/science/article/pii/S0890540124000890`, `doi:10.1016/j.ic.2024.105224`.

[29] Wei-Lin Wu. Query algorithms based on homomorphism counts. In *Structure Meets Power Workshop (Contributed Talks)*, page 24, 2022.

[30] Wei-Lin Wu. *A Study of the Expressive Power of Homomorphism Counts.* University of California, Santa Cruz, 2023.

[31] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL: `https://openreview.net/forum?id=ryGs6iA5Km`.

[32] Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 331–342, 2017. `doi:10.1109/FOCS.2017.38`.