The Fine-Grained Complexity of Constraint Satisfaction Problems

László Egri

Doctor of Philosophy

School of Computer Science

McGill University, Montréal

December, 2012

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements of the degree of Ph.D.

Abstract

Constraint satisfaction problems (CSPs) provide a unified framework for studying a wide variety of computational problems naturally arising in combinatorics, artificial intelligence and database theory. To any finite domain D and any constraint language Γ (a finite set of relations over D), we associate the constraint satisfaction problem CSP(Γ): an instance of CSP(Γ) consists of a list of variables $x_1, x_2, \ldots, x_n$ and a list of constraints of the form "$(x_7, x_2, ..., x_5) \in R$" for some relation R in Γ. The goal is to determine whether the variables can be assigned values in D such that all constraints are simultaneously satisfied. The computational complexity of CSP(Γ) is entirely determined by the structure of the constraint language Γ and, thus, one wishes to identify classes of Γ such that CSP(Γ) belongs to a particular complexity class.

In recent years, combined logical and algebraic approaches to understand the complexity of CSPs within the complexity class P have been especially fruitful. In particular, precise algebraic conditions on Γ have been conjectured to be sufficient and necessary for the membership of CSP(Γ) in the

complexity classes L and NL (under standard complexity theoretic assumptions, e.g. $\mathsf{L} \neq \mathsf{NL}$). These algebraic conditions are known to be necessary, and from the algorithmic side, a promising body of evidence is fast-growing. The main tools to establish membership of CSPs in L and NL are the logic programming fragments symmetric and linear Datalog, respectively.

This thesis is centered around the above algebraic conjecture for CSPs in L, and most of the technical work is devoted to establishing the membership of several large classes of CSPs in L. Among other results, we characterize all graphs for which the list homomorphism problem is in L, a well-studied and natural class of CSPs. We also extend this result to obtain a complete characterization of the complexity of the list homomorphism for graphs. We develop new tools (dualities for symmetric Datalog) to show membership of CSPs in L, prove an $\mathsf{L} - \mathsf{NL}$ dichotomy for the list homomorphism problem for oriented paths, provide results about the structure and polymorphisms of Maltsev digraphs, and also contribute to the conjecture of Dalmau that every CSP in NL is in fact in linear Datalog.

Résumé

Les problèmes de satisfaction de contraintes (ou CSP) forment un cadre particulièrement riche permettant de formaliser de façon uniforme un grand nombre de problèmes algorithmiques tirés de l'optimisation combinatoire, de l'intelligence artificielle et de la théorie des bases de données. À chaque domaine D et chaque langage de contraintes Γ (i.e. un ensemble de relations sur D), on associe le problème CSP(Γ) suivant. Une instance du problème est constituée d'une liste de variables $x_1, \ldots, x_n$ et d'une liste de contraintes de la forme $(x_7, x_2, \ldots, x_5) \in R$, où $R \in \Gamma$. On cherche à déterminer si des valeurs de D peuvent être assignées aux variables de telle sorte que les contraintes soient toutes satisfaites simultanément. La complexité algorithmique de CSP(Γ) est entièrement fonction de la structure du langage de contraintes Γ et on cherche alors à identifier des classes de contraintes pour lesquelles CSP(Γ) appartient à une classe de complexité spécifique.

Récemment, la combinaison des approches logique et algébrique a porté fruits dans la compréhension de la complexité des CSP à l'intérieur de la classe P. En particulier, on a conjecturé des conditions algébriques nécessaires

et suffisantes précises pour l'appartenance de CSP(Γ) dans les classes L et NL (sous les hypothèses habituelles en théorie de la complexité, e.g. $\mathsf{L} \neq \mathsf{NL}$). Ces conditions algébriques sont sues être nécessaires, et d'un point de vue algorithmique, les indications en faveur du résultat s'accumulent rapidement. Les outils principaux pour établir l'appartenance d'un CSP à L ou NL sont respectivement les fragments "symmetric Datalog" et "linear Datalog" en programmation logique.

Notre thèse est centrée sur la conjecture algébrique ci-haut mentionnée pour les CSP dans L, et la majeure partie du travail technique est dédiée à montrer l'appartenance de plusieurs grandes familles de CSP dans L. Entre autres résultats, nous caractérisons tous les graphes pour lesquels le problème de "list homomorphism" est dans L, une famille naturelle et bien étudiée de CSP. Nous étendons aussi ce résultat pour obtenir une caractérisation complète de la question pour les graphes. Nous développons de nouveaux outils (les dualités pour "symmetric Datalog") pour montrer l'appartenance de CSP dans L, nous prouvons une dichotomie L-NL pour les problèmes de "list homomorphism" pour les chemins orientés, nous donnons des résultats sur la structure et les polymorphismes des digraphes de Maltsev, et nous contribuons à la conjecture de Dalmau à l'effet que chaque CSP dans NL est en fait dans "linear Datalog".

Acknowledgments

First and foremost, I wish to express my deepest gratitude to my supervisor Denis Thérien. I am grateful to Denis for getting me interested in complexity theory during his two fantastic courses and giving me the opportunity to work in this field. I also thank him for inviting me to his terrific Barbados workshops, and the financial support he provided me.

I am particularly grateful to Benoit Larose for letting me benefit from his expertise on constraint satisfaction problems, and for many fruitful discussions and stimulating collaborations. Further thanks go to my other collaborators: Catarina Carvalho, Marcel Jackson, Andrei Krokhin, Todd Niven, and Pascal Tesson. I thank Catarina Carvalho (twice), Martin Grohe, and Nicole Schweikardt for hosting me at their universities.

Special thanks go to Anil Ada for constituting the other half of the complexity lab from the very beginnings. I also thank Arkadev Chattopadhyay for being part of the fun working environment for the first year of my PhD.

I also want to thank NSERC and FQRNT for their financial support.

Finally, I thank my parents for their encouragement, and Minori Yokota

for her patient support.

Contents

Chapter 1

Introduction

1.1 The Constraint Satisfaction Problem

Constraint satisfaction problems (CSP) constitute a unifying framework to study various computational problems [32, 60, 71, 72, 67, 80, 75]. More specifically, CSPs naturally arise in artificial intelligence, database theory, graph theory, etc. Loosely speaking, an instance of a CSP consists of a list of variables and a set of constraints, each specified by an ordered tuple of variables and a constraint relation over some specified domain. The goal is then to determine whether variables can be assigned domain values such that all constraints are simultaneously satisfied.

The CSP is NP-complete, and therefore it is widely believed that no efficient (i.e. polynomial-time) general-purpose algorithm exists to solve it. However, in many practical applications the instances that arise have a spe-

cial form that allow for efficient heuristics, see for example [15, 25, 65, 76].

Another way to obtain polynomial time solvable CSPs is not by making assumptions about the form of the input instances, but rather, by restricting the type of constraints that are allowed. That is, a finite set of finite constraint relations Γ is fixed (also called the *template* of the CSP), and then an instance is specified as before except that all constraints have the form $(x_{i_1}, \ldots, x_{i_k}) \in R$ for some $R \in \Gamma$. This model is referred to as *nonuniform* CSP, and when Γ is given, it is denoted by CSP(Γ). Examples of nonuniform CSPs include k-Sat, Horn-3Sat, Graph H-Coloring, and many others. This thesis is focused on the nonuniform CSP, and therefore from now on when we say CSP, we *always* mean nonuniform CSP.

Much recent effort has focused on the complexity of the (nonuniform) CSP, i.e. to make precise statements about the complexity of CSP(Γ) *given only* Γ. In what follows, we outline the main directions and specify our goals.

1.2 The Dichotomy Conjecture

The well-known CSP dichotomy conjecture of Feder and Vardi [43] forecasts that every CSP is either solvable in polynomial time (tractable) or NP-complete, and progress toward this conjecture has been steady during the last fifteen years. An early breakthrough in the development occurred when Jeavons, Cohen and Gyssens [57] announced an algebraic approach to the problem. Their work, refined later by Bulatov, Jeavons and Krokhin

[17, 15], showed that the complexity of a non-uniform CSP is fully determined by a set of operations, the *polymorphisms* of the template of the CSP. This algebraic approach allowed to conjecture the precise boundary between NP-complete and tractable problems [17], and led to results unreachable before. For example, using the algebraic approach, Bulatov proved the dichotomy conjecture for CSPs with domain size 3 [13], and for *conservative* CSPs [16]. Another important application of the algebraic approach is a theorem of Barto, Kozik and Niven [8, 9] stating that the dichotomy conjecture holds for digraphs with no *sources* and no *sinks*.

A different, descriptive complexity theoretic approach consists in classifying the CSPs according to the sophistication of the logical apparatus required to define the set of negative instances of the given CSP. (Why we work with the negative instances of a CSP is a technicality that will be explained in the next chapter, Subsection 2.2.5; the set of negative instances of a CSP will be denoted by co-CSP.) It was noted early on that when the negative instances of a CSP are definable in the database-inspired logic programming language Datalog then the CSP lies in polynomial time[1] (P) [43], and this provides a unifying explanation for a number of (but not all) tractable cases.

There are two known major "islands" of tractable CSPs. Problems in the first class are solvable using the *few subpowers algorithm* [55, 29]. Problems in the other class are solvable using *local consistency checking*, which is definitely the widest known and most natural algorithm for solving CSPs. In fact,

[1] This is because a Datalog program can be evaluated in polynomial time.

these CSPs are precisely the ones whose negative instances can be defined in Datalog. Understanding the applicability of local consistency checking was considered as an important step towards establishing the dichotomy conjecture, and this was recently achieved by Barto and Kozik [6]. Some believe that the dichotomy conjecture might be settled in the near future.

1.3 Fine-Grained Complexity of CSPs

1.3.1 CSPs and Datalog Fragments

From a complexity-theoretic perspective, the classification of CSP(Γ) as in P or being NP-complete is rather coarse and therefore somewhat unsatisfactory. Consequently, understanding the fine-grained complexity of CSPs gained considerable attention during the last few years. Ultimately, one would like to know the precise complexity of a CSP lying in P, i.e. to identify a "standard" complexity class for which a given CSP is complete. Towards this, it was established that Schaefer's $\mathsf{P} - \mathsf{NP}$ dichotomy for Boolean CSPs [78] can indeed be refined: each CSP over the Boolean domain is either definable in first order logic, or complete for one of the classes L, NL, $\oplus\mathsf{L}$, P or NP under AC^0-*reductions* [3]. The question whether some form of the above fine-grained classification extends to non-Boolean domains is rather natural.

Strong connections between expressibility in symmetric Datalog and linear Datalog, and CSP solvability in logarithmic space (L) and nondeterministic logspace (NL), respectively, have been indicated in [38] and [28]. These

papers also conjectured that a CSP is in L if and only if its negative instances are expressible in symmetric Datalog, and it is in NL if and only if its negative instances are expressible in linear Datalog. Note that a symmetric (linear) Datalog program can be evaluated in L (NL), and therefore only one direction of the above conjectures has to be proven. An important evidence that the converse might also hold in the case of symmetric Datalog is provided by [38], which shows that over the Boolean domain, the negative instances of a CSP are expressible in symmetric Datalog if and only if the CSP is in L, assuming that $\mathsf{L} \neq \mathsf{NL}$ and $\mathsf{L} \neq \oplus\mathsf{L}$.[2] In fact, the two most important tools to study CSPs whose complexity is below P are symmetric Datalog and linear Datalog.

1.3.2 A Few Words on Tame Congruence Theory

As mentioned above, the polymorphisms of the template Γ of a CSP completely determine the complexity of CSP(Γ). These polymorphisms can be thought of as the operations of an *algebra*, and we denote this algebra by $\mathbb{A}(\Gamma)$ (precise definitions will be given in the next chapter).

Tame Congruence Theory, a deep universal-algebraic framework devel-

[2]The class $\mathsf{Mod_pL}$ was defined in [20]. The results mentioned below are also from the same paper. For a prime p, $\mathsf{Mod_pL}$ denotes the class of languages recognized by Mod_p-counting nondeterministic logspace machines. Formally, $K \in \mathsf{Mod_pL}$ if there exists a nondeterministic logspace machine M such that $w \in K$ if and only if the number of accepting paths of M on w is not divisible by p. The class $\mathsf{Mod_pL}$ is closed under complement. When $p = 2$, the corresponding class is usually denoted as $\oplus\mathsf{L}$. These classes contain a number of natural problems related to modular arithmetic such as solving systems of linear equations over $\mathbb{Z}_p$.

oped by Hobby and McKenzie in the mid 80s [54], classifies the local behaviour of finite algebras into five types (unary, affine, Boolean, lattice and semilattice). It was recently shown (see [19, 18, 63]) that there is a strong connection between the computational and descriptive complexity of co-CSP(Γ) and the set of types that appear in $\mathbb{A}(\Gamma)$ and its subalgebras. There are strong conditions involving types which are sufficient for NL-hardness, P-hardness and NP-hardness of CSP(Γ) as well as for inexpressibility of co-CSP(Γ) in Datalog, linear Datalog and symmetric Datalog. These sufficient conditions are also suspected (and in some cases proved) to be necessary, under natural complexity-theoretic assumptions. For example, (a) the presence of unary type is known to imply NP-completeness, while its absence is conjectured to imply tractability (see [19]); (b) the absence of the unary and affine types was recently proved to be (unconditionally) equivalent to definability in Datalog [6]; (c) the absence of the unary, affine, and semilattice types is proved necessary, and suspected to be sufficient, for membership in NL and definability in linear Datalog [63]; (d) the absence of all types but Boolean is proved necessary, and suspected to be sufficient, for membership in L and definability in symmetric Datalog [63]. The strength of evidence varies from case to case and, in particular, the conjectured algebraic conditions concerning CSPs in NL and L (and, as mentioned above, linear and symmetric Datalog) still rest on relatively limited evidence [18, 21, 22, 28, 24, 31, 63]. Our thesis is mostly centered around those CSP(Γ)-s for which the typeset of $\mathbb{A}(\Gamma)$ contains only the Boolean type, i.e. those CSP which are conjectured to be in L.

1.3.3 A Note on Conjectures Related to Symmetric and Linear Datalog

The linear Datalog conjecture (Conjecture 1) states that if CSP(Γ) is in NL, then in fact, co-CSP(Γ) is in linear Datalog [28]. (This conjecture is hard because combined with some known results [1], it would easily imply that NL $\neq$ P.[3]) The algebraic linear Datalog conjecture (Conjecture 2) states that the variety (a certain class of algebras) associated with Γ omits the unary, affine and semilattice types if and only if co-CSP(Γ) is in linear Datalog [63]. One direction of this conjecture is known: if the variety associated with Γ admits one of the unary, affine or semilattice types then co-CSP(Γ) is not in linear Datalog [63].

Note that if we assume that NL is different from P and $\mathsf{Mod}_p\mathsf{L}$ for any prime p, then Conjecture 2 implies Conjecture 1: Consider any Γ such that CSP(Γ) is in NL and assume for contradiction that co-CSP(Γ) is not in linear Datalog. Then by the assumption that Conjecture 2 holds the variety associated with Γ admits at least one of the unary, affine or semilattice types. By results of [63], this implies that CSP(Γ) is hard for NP, P or $\mathsf{Mod}_p\mathsf{L}$ for some prime p, which implies that NL is equal to P or $\mathsf{Mod}_p\mathsf{L}$ for some prime p.

The symmetric Datalog conjecture (Conjecture 3) states that if CSP(Γ)

[3] Assume that the P-complete CSP Horn-3Sat is in NL. If Conjecture 1 holds, then Horn-3Sat is in linear Datalog. But we know that Horn-3Sat is not in linear Datalog by [1].

is in L, then in fact, co-CSP(Γ) is in symmetric Datalog. (This conjecture is also hard because combined with some known results [39], it would imply that $\mathsf{L} \neq \mathsf{NL}$.[4]) The algebraic symmetric Datalog conjecture (Conjecture 4) states that the variety associated with Γ omits the unary, affine, lattice and semilattice types if and only if co-CSP(Γ) is in symmetric Datalog. Just as above, one direction is known: If the variety associated with Γ admits one of the unary, affine, lattice or semilattice types, then co-CSP(Γ) is not in symmetric Datalog [63].

A similar argument to the one given for Conjectures 1 and 2 shows that under the assumption that L is different from NL and $\mathsf{Mod_pL}$ for any prime p, Conjecture 4 implies Conjecture 3.

Note that we usually speak about showing that "every CSP in NL is also in linear Datalog". Strictly speaking, this is Conjecture 1, but because showing Conjecture 1 would separate complexity classes, we assume that those complexity classes are different, and *then* try to show Conjecture 1. One way to think about Conjecture 2 is as a tool to show Conjecture 1 in the presence of standard complexity theoretic assumptions.

[4] Assume that the NL-complete CSP directed *st*-connectivity (actually, a slightly modified version of it) is in L. If Conjecture 3 holds, then directed *st*-connectivity is in symmetric Datalog. But we know that directed *st*-connectivity is not in symmetric Datalog by [34, 39].

1.3.4 Summary of Contributions and Thesis Structure

The unifying theme of this thesis is CSPs within the complexity class P. We are mostly concerned with CSPs of logarithmic space complexity. Nevertheless, CSPs in NL also show up, for example, Chapter 7 is a modest contribution to our understanding of CSPs in NL. We briefly outline the chapters of the thesis.

In the next chapter, we present the necessary background material. Chapter 3 introduces new techniques that can be used to establish the membership of CSPs in the complexity class L. Applications are discussed and, in particular, a new short proof of the main result of [31] is presented. This work was published in [35] (first part of the paper).

The *list homomorphism problem* for graphs is a well-studied and natural class of CSPs. In Chapter 4, we completely classify the computational complexity of the list homomorphism problem for graphs (with possible loops) in combinatorial and algebraic terms: for every graph H, the problem is either NP-complete, NL-complete, L-complete or is first-order definable; descriptive complexity equivalents are given as well via Datalog and its fragments. Most of the work in the chapter is focused on those list homomorphism problems which are in L. Our algebraic characterisations match important conjectures in the study of constraint satisfaction problems. This is joint work with Andrei Krokhin, Benoit Larose, and Pascal Tesson [36, 37].

In Chapter 5, we show that the list homomorphism problem for oriented paths admits an $\mathsf{L} - \mathsf{NL}$ dichotomy. That is, for any oriented path, the list

homomorphism problem can be solved in L, or is complete for NL.

In Chapter 6, we study digraphs preserved by a Maltsev operation, Maltsev digraphs. We show that these digraphs retract either onto a directed path or to the disjoint union of directed cycles, showing that the CSP for Maltsev digraphs is in L. (This was observed in [58] using an indirect argument.) We then generalize results in [58] to show that a Maltsev digraph is preserved not only by a majority operation, but by a class of other operations (e.g., minority, Pixley) and obtain a $O(|V_G|^4)$-time algorithm to recognize Maltsev digraphs. We also prove analogous results for digraphs preserved by conservative Maltsev operations which we use to establish that the list homomorphism problem for Maltsev digraphs is in L. Finally, we give a simple inductive construction of directed acyclic digraphs preserved by a Maltsev operation. The chapter contains both individual and joint work with Catarina Carvalho, Marcel Jackson and Todd Niven. These results (and some others) appear in [23].

In Chapter 7 we contribute to the conjecture that if CSP(Γ) is in NL then co-CSP(Γ) is in *linear Datalog*, a strict subclass of NL. More specifically, we define a non-trivial subclass $\mathcal{C}$ of NL that contains NL-complete problems and also problems not in linear Datalog, and show that if co-CSP(Γ) is in $\mathcal{C}$ then co-CSP(Γ) is also in linear Datalog. This result constitutes the second part of [35].

Finally, in Chapter 8, we conclude with some open problems.

Chapter 2

Background

2.1 The CSP and Examples

We begin with the formal definition of a CSP.

Definition 1. A *constraint language* over a set D is a subset of all finitary relations over D. For any set D and any constraint language Γ over D, CSP(Γ) is the combinatorial decision problem:

Instance: A triple $\langle V, D, C \rangle$, where

- V is a set of *variables*;
- C is a set of *constraints*, $\{C_1, ..., C_q\}$;
- Each constraint $C_i \in C$ is a pair $\langle s_i, R_i \rangle$, where
  - s_i is a tuple of variables of length n_i, called the *constraint scope*;

– $R_i \in \Gamma$ is an n_i-ary relation over D, called the *constraint relation.*

Question: Does there exist a *solution*, that is, a function $f : V \to D$ such that for each constraint $\langle s, R\rangle \in C$ with $s = \langle v_1, ..., v_n\rangle$ the tuple $\langle f(v_1), ..., f(v_n)\rangle$ belongs to R? The set of solutions of a CSP instance $\mathcal{P} = \langle V, D, C\rangle$ will be denoted by $\mathrm{Sol}(\mathcal{P})$.

To give a feel for CSPs, we present a number of examples and then link CSPs to the homomorphism problem and discuss some more basic concepts.

Example 2. 3-Sat is the decision problem in which we are given a 3CNF-formula ϕ with variables $x_1, ..., x_n$ and clauses $c_1, ..., c_m$ and we have to decide whether the clauses can be satisfied simultaneously. Now we express 3-Sat as a CSP. For each possible type of clause we create a relation in the following way:

Clause Type	Relation
$(x \vee y \vee z)$	$R_0 = \{0,1\}^3 \backslash \{(0,0,0)\}$
$(\neg x \vee y \vee z)$	$R_1 = \{0,1\}^3 \backslash \{(1,0,0)\}$
$(\neg x \vee \neg y \vee z)$	$R_2 = \{0,1\}^3 \backslash \{(1,1,0)\}$
$(\neg x \vee \neg y \vee \neg z)$	$R_3 = \{0,1\}^3 \backslash \{(1,1,1)\}$

Let $\Gamma = \{R_0, R_1, R_2, R_3\}$. Then the CSP instance $\langle V, D, C\rangle$ has a solution if and only if ϕ is satisfiable, where $V = \{x_1, ..., x_n\}, D = \{0, 1\}, C = \{c_1, ..., c_m\}$ and in $c_i = \langle (x, y, z), R_j\rangle$, where R_j is the relation corresponding to (x, y, z) as given in the table above.

Example 3. A *Boolean* constraint language is a constraint language over a two element domain $D = \{d_0, d_1\}$. Similarly to the encoding used in Example 2, we can express SATISFIABILITY [73] as a CSP. It was established by Schaefer in 1978 in a seminal paper [78] that a Boolean constraint language Γ is tractable if (at least) one of the following six conditions holds:

1. Every relation in Γ contains the tuple in which all entries are equal to d_0;
2. Every relation in Γ contains the tuple in which all entries are equal to d_1;
3. Every relation in Γ is definable by a conjunction of clauses, where each clause has at most one positive literal (i.e., a conjunction of Horn clauses);
4. Every relation in Γ is definable by a conjunction of clauses, where each clause has at most one negative literal (i.e., a conjunction of anti-Horn clauses);
5. Every relation in Γ is definable by a conjunction of clauses, where each clause contains at most two literals;
6. Every relation in Γ is the set of solutions of a system of linear equations over the finite field with two elements, GF(2).

Otherwise Γ is NP-complete. This result is often called *Schaefer's dichotomy theorem* [78]. Relying on universal algebra, Allender et al. refined Schaefer's

theorem [2] by showing that any Boolean CSP is either definable in first order logic, or complete for one of the classes L, NL, $\oplus\mathsf{L}$, P or NP.

Example 4. Let $D = \{0, 1\}$, and E be the less-than-or-equal-to relation, i.e. $E = \{(0, 0), (0, 1), (1, 1)\}$. Furthermore, let $S = \{1\}$ and $T = \{0\}$. It is not difficult to see that the CSP defined by E, S, and T corresponds to the following decision problem: given a directed graph G and two subsets of the vertices of G, S^G and S^T, is there a directed path from a vertex in S^G to a vertex in S^T? This CSP is NL-complete.

Example 5. Let D be a finite domain. A binary relation $R \subseteq D^2$ is said to be *implicational* or a *0/1/all constraint* if it is one of the following forms (see [59] and [26]):

1. $R = B \times C$ for some $B, C \subseteq D$;
2. $R = \{\langle b, f(b)\rangle : b \in B\}$ where $B \subseteq D$ and f is an injective function;
3. $R = \{b\} \times C \cup B \times \{c\}$ for some $B, C \subseteq D$ with $b \in B$ and $c \in C$.

Dalmau showed [28] that CSPs defined by implicational constraints are in NL.

In the following examples, we exceptionally allow Γ to be infinite (both the universe and the number of relations can be infinite). We need the following definition:

Definition 6. A constraint language Γ is called:

- *Tractable* if CSP(Γ') can be solved in polynomial time, for each subset $\Gamma' \subseteq \Gamma$;
- *NP-complete* if CSP(Γ') is NP-complete for some finite subset $\Gamma' \subseteq \Gamma$.

Example 7. The binary *less than* relation over an ordered set D is defined as:

$$<_D = \{\langle d_1, d_2\rangle \in D^2 : d_1 < d_2\}$$

Let D be $\mathbb{N}$, the set of natural numbers. Then the class of CSP instances CSP($\{<_{\mathbb{N}}\}$) corresponds to the Acyclic Digraph problem [11] (the problem is to decide whether a graph is acyclic). Note that a directed graph is acyclic if and only if its vertices can be numbered in such a way that every arc leads from a vertex with smaller number to a vertex with a greater one. Since the Acyclic Digraph problem is tractable, $\{<_{\mathbb{N}}\}$ is tractable.

Example 8. Let *disequality* be the following relation over D:

$$\neq_D = \{\langle d_1, d_2\rangle \in D^2 : d_1 \neq d_2\}$$

CSP($\{\neq_D\}$) corresponds to the Graph Colorability problem [45, 73] with $|D|$ colors. This problem is in polynomial time if $|D| \leq 2$ or $|D| = \infty$ and NP-complete if $3 \leq |D| < \infty$.

Example 9. Let $\mathbb{F}$ be any finite field and Γ_{Lin} be the constraint language containing all those relations over $\mathbb{F}$ which consist of all the solutions to some system of linear equations over $\mathbb{F}$. Any relation from Γ_{Lin}, and therefore any

instance of $\mathrm{CSP}(\Gamma_{Lin})$ can be represented by a system of linear equations over $\mathbb{F}$ and this system of equations can be computed from the relations in polynomial time [15]. A system of linear equations can be solved in polynomial time (e.g., by Gaussian elimination) and therefore Γ_{Lin} is tractable.

2.2 Basic Notions

2.2.1 Relational Structures, Homomorphisms

For any set A (called *domain* or *universe*) and any non-negative integer n, the set of all n-tuples of elements of A is denoted by A^n. A subset of A^n is called an n-ary *relation* over A. A signature is a (finite) set of relation symbols. Each symbol R has an associated arity, which is denoted with $\mathrm{ar}(R)$. A relational structure $\mathbf{A}$ of signature τ consists of a set A called the universe (or domain) of $\mathbf{A}$, and a relation for each symbol in τ of the corresponding arity. Given a symbol $R \in \tau$ we use superscripts to denote the corresponding relation $R^{\mathbf{A}}$ in structure $\mathbf{A}$. All structures in this thesis are assumed to be finite. In the following, we denote the universe of a structure $\mathbf{A}$ with its lightface equivalent, e.g. A.

Let $\mathbf{B}$ be a structure of the same signature as $\mathbf{A}$. A homomorphism from $\mathbf{A}$ to $\mathbf{B}$ is a map f from A to B such that $f(R^{\mathbf{A}}) \subseteq R^{\mathbf{B}}$ for each $R \in \tau$, i.e. we have $(f(a_1), \ldots, f(a_r)) \in R^{\mathbf{B}}$ whenever $(a_1, \ldots, a_r) \in R^{\mathbf{A}}$. In this case we write $f : \mathbf{A} \to \mathbf{B}$. When it is clear from the context, we often say "maps" instead of "homomorphically maps". A *retract* of a structure $\mathbf{B}$ is an induced

substructure $\mathbf{B}'$ of $\mathbf{B}$ such that there is a homomorphism $g : \mathbf{B} \to \mathbf{B}'$ with $g(b) = b$ for every $b \in B'$. A retract of $\mathbf{B}$ that has minimal size among all retracts of $\mathbf{B}$ is called a *core* of $\mathbf{B}$. It is well known that all cores of a structure are isomorphic, and so one speaks of the core of a structure $\mathbf{B}$, core($\mathbf{B}$). We denote by CSP($\mathbf{B}$) the class of all τ-structures $\mathbf{A}$ that admit a homomorphism to $\mathbf{B}$, and by co-CSP($\mathbf{B}$) the complement of this class. Note that in this thesis, $\mathbf{B}$ in CSP($\mathbf{B}$) is always assumed to be finite.

Notice that in Section 2.1 we defined the CSP differently. To see that our original CSP definition is the same as the homomorphism variant, consider CSP($\mathbf{B}$) and an input structure $\mathbf{A}$: think of the elements in A as variables, the elements in B as values, tuples in the relations of $\mathbf{A}$ as constraint scopes, and the relations of $\mathbf{B}$ as constraint relations. With this correspondence, the solutions to this CSP instance are precisely the homomorphisms from $\mathbf{A}$ to $\mathbf{B}$.

2.2.2 Tuple Structures

The following non-standard definition will be used in Chapters 3 and 7, and later in this chapter. A *tuple structure* $\tilde{\mathbf{A}}$ over a vocabulary τ is a set of pairs $(R, \mathbf{t})$ where $R \in \tau$ and $\mathbf{t}$ is an ar(R)-tuple. We associate a domain $\tilde{A}$ with a tuple structure: $\tilde{A}$ contains every element that appears in some tuple in $\tilde{\mathbf{A}}$, and possibly some other elements. Clearly, tuple structures are equivalent to relational structures. If $\mathbf{A}$ is a relational structure, we denote the equivalent tuple structure with $\tilde{\mathbf{A}}$, and vice versa. For convenience, sometimes we use

the two notations interchangeably.

2.2.3 Algebra

An n-ary operation on a set A is a map $f : A^n \to A$. For example, a *projection* operation is an operation of the form $e_n^i(x_1, \ldots, x_n) = x_i$ for some $1 \leq i \leq n$. Given an h-ary relation θ and an n-ary operation f on the same set A, we say that f *preserves* θ or that θ is *invariant* under f if the following holds: given any matrix M of size $h \times n$ whose columns are in θ, applying f to the rows of M will produce an h-tuple in θ. A *polymorphism* of a structure $\mathbf{T}$ is an operation f that preserves each relation in $\mathbf{T}$; in this case we also say that $\mathbf{T}$ *admits* f. For the special case of graphs, this means that if there is an edge between a_i and b_i for each $1 \leq i \leq n$ (where the a_i's and b_i's are not necessarily distinct) then there is an edge between $f(a_1, \ldots, a_n)$ and $f(b_1, \ldots, b_n)$. More specific algebraic definitions will be given in Chapter 4, where these definitions will be explicitly used. To give a flavor of the power of the algebraic approach to study the complexity of the CSP, we mention some results about CSPs involving polymorphisms.

Definition 10. Let A be a nonempty set. An n-ary operation $f : A^n \to A$ is a *near unanimity operation* if it satisfies

$$f(\underbrace{x, \ldots, x}_{i-1}, y, \underbrace{x, \ldots, x}_{n-i}) = x \qquad \text{for all } x, y \in A, \text{ and for all } i \leq n. \quad (2.1)$$

In the special case when $n = 3$, f is called a *majority operation*.

Definition 11. Let A be a nonempty set. A ternary operation $f : A^3 \to A$ is a *Maltsev operation* if it satisfies

$$f(y,x,x) = f(x,x,y) = y \qquad \text{for all } x,y \in A \tag{2.2}$$

Theorem 12 ([30], 2008)**.** *Suppose the finite structure* $\mathbf{B}$ *admits a majority polymorphism. Then* CSP($\mathbf{B}$) *has bounded pathwidth duality and hence is in* NL.

Theorem 13 ([10], 2012)**.** *Suppose the finite structure* $\mathbf{B}$ *admits a* $(d+1)$*-ary near unanimity polymorphism. Then* CSP($\mathbf{B}$) *has bounded pathwidth duality and hence is in* NL.

Theorem 14 ([14])**.** *Suppose the finite structure* $\mathbf{B}$ *admits a Maltsev operation. Then there is a polynomial-time algorithm for* CSP($\mathbf{B}$).

More background on algebra is given in Subsection 4.2.2.

2.2.4 Datalog

We provide only the necessary concepts related to Datalog and its fragments, and the reader can find more details, for example, in [66, 27, 38]. Datalog is a database-inspired query language whose connection with CSP-complexity is now relatively well understood (see e.g. [6]). Let τ be some finite vocabulary. A Datalog program over τ is specified by a finite set of rules of the form $h \leftarrow b_1 \wedge \cdots \wedge b_t$, where h and the b_i are atomic formulas $R(x_1, \ldots, x_k)$. When

we specify the variables of an atomic formula, we always list the variables from left to right, or we simply provide a tuple $\mathbf{x}$ of variables whose i-th variable is $\mathbf{x}[i]$. We distinguish two types of relational predicates occurring in a Datalog program: predicates that occur at least once in the head of a rule (i.e., its left-hand side) are called *intensional database predicates* (*IDBs*) and are not in τ. The predicates which occur only in the body of a rule (its right-hand side) are called *extensional database predicates* (*EDBs*) and must all lie in τ. A rule that contains no IDB in the body is called a *nonrecursive rule*, and a rule that contains at least one IDB in the body is called a *recursive rule*. A Datalog program contains a distinguished IDB of arity 0 which is called the *goal predicate*; a rule whose head IDB is a goal IDB is called a *goal rule*.

Linear Datalog is a syntactic restriction of Datalog in which there is at most one IDB in the body of each rule. The class of linear Datalog programs that contain only rules with at most k variables and IDBs with at most $j \leq k$ variables is denoted with *linear* (j,k)*-Datalog*. We say that the *width* of such a linear Datalog program is (j,k).

Symmetric Datalog is a syntactic restriction of linear Datalog. A linear Datalog program $\mathcal{P}$ is symmetric if for any recursive rule $I(\mathbf{x}) \leftarrow J(\mathbf{y}) \wedge \bar{E}(\mathbf{z})$ of $\mathcal{P}$ (except for goal rules), where $\bar{E}(\mathbf{z})$ is a shorthand for the conjunction of the EDBs of the rule over variables in $\mathbf{z}$, the symmetric pair $J(\mathbf{y}) \leftarrow I(\mathbf{x}) \wedge \bar{E}(\mathbf{z})$ of that rule is also in $\mathcal{P}$. The *width* of a symmetric Datalog program is defined similarly to the width of a linear Datalog program.

2.2 Basic Notions

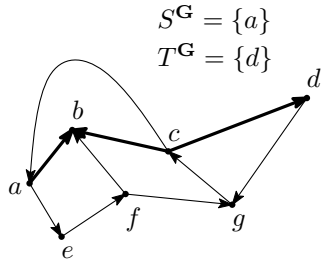
We explain the semantics of linear (symmetric) Datalog using derivations (it could also be explained with *fixed point operators*, but that would be inconvenient for the proofs). Let $\mathcal{P}$ be a linear Datalog program with vocabulary τ. A *$\mathcal{P}$-derivation with codomain D* is a sequence of pairs $\mathscr{D} = (\rho_1, \lambda_1), \ldots, (\rho_q, \lambda_q)$, where ρ_ℓ is a rule of $\mathcal{P}$, and λ_ℓ is a function from the variables V_ℓ of ρ_ℓ to D, $\forall \ell \in [q]$. The sequence $\mathscr{D}$ must satisfy the following properties. Rule ρ_1 is nonrecursive, and ρ_q is a goal rule. For all $\ell \in [q-1]$, the head IDB I of ρ_ℓ is the IDB in the body of $\rho_{\ell+1}$, and if the variables of I in the head of ρ_ℓ and the body of $\rho_{\ell+1}$ are $\mathbf{x}$ and $\mathbf{y}$, respectively, then $\lambda_\ell(\mathbf{x}[i]) = \lambda_{\ell+1}(\mathbf{y}[i])$, $\forall i \in [\mathrm{ar}(I)]$.

Let $R(\mathbf{z})$ be an EDB with variables in some rule ρ_ℓ of a derivation $\mathscr{D}$. Then we write $R(\mathbf{t})$ to denote that $\lambda_\ell(\mathbf{z}) = \mathbf{t}$, i.e. that λ_ℓ *instantiates* the variables of $R(\mathbf{z})$ to $\mathbf{t}$, and we say that $R(\mathbf{t})$ *appears* in ρ_ℓ, or less specifically, that $R(\mathbf{t})$ *appears* in $\mathscr{D}$. Given a structure $\mathbf{A}$ and a derivation $\mathscr{D}$ with codomain A for a program $\mathcal{P}$, we say that $\mathscr{D}$ *is a derivation for* $\mathbf{A}$ if for every $R(\mathbf{t})$ that appears in a rule of $\mathscr{D}$, $(R, \mathbf{t}) \in \tilde{\mathbf{A}}$. We denote a $\mathcal{P}$-derivation for a structure $\mathbf{A}$ by $\mathscr{D}_{\mathcal{P}}(\mathbf{A})$. A linear (symmetric) Datalog program $\mathcal{P}$ *accepts* an input structure $\mathbf{A}$ if there exists a $\mathcal{P}$-derivation for $\mathbf{A}$.

An example is given in Figure 2.1. The vocabulary is $\tau = \{E^2, S^1, T^1\}$, where the superscripts denote the arity of the symbols. Notice that in the symmetric Datalog program $\mathcal{P}$, rules 2.4 and 2.5 form a symmetric pair. It is not difficult to see that $\mathcal{P}$ accepts a τ-structure $\mathbf{A}$ if and only if there is an oriented path (oriented means that as the path is traversed, each edge is

oriented forward or backward, see Section 2.2.7) in $E^{\mathbf{A}}$ from an element in $S^{\mathbf{A}}$ to an element in $T^{\mathbf{A}}$.

$$I(x) \leftarrow S(x) \tag{2.3}$$
$$I(y) \leftarrow I(x) \wedge E(x,y) \tag{2.4}$$
$$I(x) \leftarrow I(y) \wedge E(x,y) \tag{2.5}$$
$$G \leftarrow I(x) \wedge T(x) \tag{2.6}$$



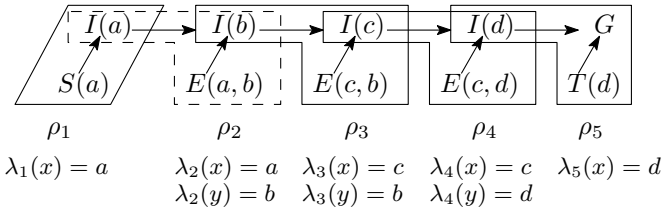


Figure 2.1: *Top left:* Symmetric Datalog program $\mathcal{P}$. *Top right:* Input structure $\mathbf{G}$ where the binary relation $E^{\mathbf{G}}$ is specified by the digraph. *Bottom:* Visualization of a $\mathcal{P}$-derivation $\mathscr{D}_{\mathcal{P}}(\mathbf{G}) = (\rho_1, \lambda_1), \ldots, (\rho_5, \lambda_5)$ for $\mathbf{G}$, where ρ_1 is nonrecursive, ρ_2, ρ_4 are rules of type 2.4, ρ_3 is a rule of type 2.5, and ρ_5 is the goal rule. For example, the dashed box corresponds to rule ρ_2, and it is the rule $I(y) \leftarrow I(x) \wedge E(x,y)$ of $\mathcal{P}$, where λ_2 assigns a to variable x and b to variable y.

2.2.5 Defining CSPs

The following discussion applies not just to Datalog but also to its symmetric and linear fragments. The class of structures accepted by a Datalog program is homomorphism-closed, i.e. if $\mathbf{A}$ belongs to the class and $\mathbf{A} \to \mathbf{B}$, then $\mathbf{B}$ also belongs to the class. To see this, assume that there is derivation for the structure $\mathbf{A}$, and a homomorphism h from $\mathbf{A}$ to $\mathbf{B}$. We obtain a

derivation for $\mathbf{B}$ by "replacing" every occurrence of an element a of $\mathbf{A}$ in the derivation with $h(a)$. On the other hand, unless CSP($\mathbf{T}$) is the set of all structures, CSP($\mathbf{T}$) is not homomorphism-closed. Therefore (with the exception of the trivial case), it is not possible to define CSP($\mathbf{T}$) in Datalog. However, co-CSP($\mathbf{T}$) is closed under homomorphisms, and in fact, it is often possible to define co-CSP($\mathbf{T}$) in Datalog.

2.2.6 Obstruction Sets

It is well known and easy to see that for any structure $\mathbf{B}$, there is a set of structures $\mathcal{O}$, called an *obstruction set*, such that a structure $\mathbf{A}$ *homomorphically maps to* $\mathbf{B}$ if and only if there is no structure in $\mathcal{O}$ that homomorphically maps to $\mathbf{A}$. In fact, there are many possible obstruction sets for any structure $\mathbf{B}$. We say that $\mathbf{B}$ has *duality* X, if $\mathbf{B}$ has an obstruction set which has the special property X. We formally define an obstruction set as:

Definition 15 (Obstruction Set)**.** A set $\mathcal{O}$ of τ-structures is called an *obstruction set* for $\mathbf{B}$, if for any τ-structure $\mathbf{A}$, $\mathbf{A} \not\to \mathbf{B}$ if and only if there exists $\mathbf{S} \in \mathcal{O}$ such that $\mathbf{S} \to \mathbf{A}$.

Note that given an obstruction set $\mathcal{O}$ for $\mathbf{B}$, $\mathbf{A} \in$ CSP($\mathbf{B}$) iff for all structures $\mathbf{O} \in \mathcal{O}$, $\mathbf{O} \not\to \mathbf{A}$.

Example 16. Let $\mathbf{B}$ be a structure with domain $\{0, 1\}$, and having a single relation $\{(0, 1), (1, 0)\}$. In graph-theoretic terms, $\mathbf{B}$ is just an undirected

edge. It is well known and easy to see that an input graph $\mathbf{A}$ homomorphically maps to $\mathbf{B}$ iff there is no cycle $\mathbf{C}$ of odd length such that $\mathbf{C} \to \mathbf{A}$. That is, the set of all odd cycles can be chosen as an obstruction set $\mathcal{O}$ for $\mathbf{B}$. See the illustration in Figure 2.2.

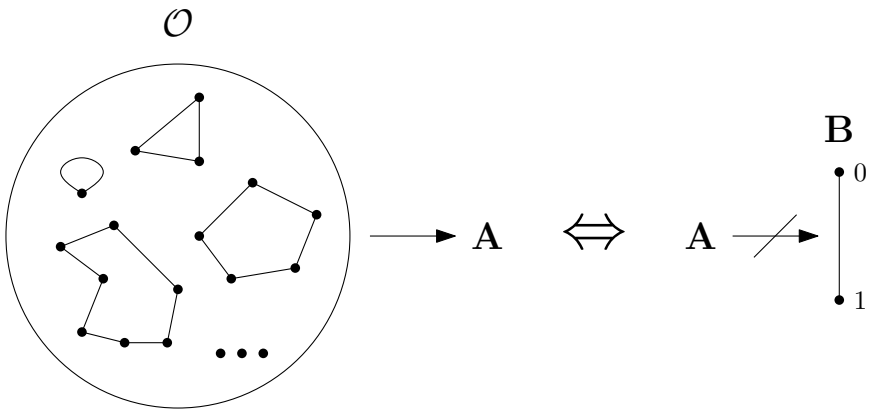


Figure 2.2: The set of all odd cycles is an obstruction set for the undirected edge (graphs are thought of as relational structures).

2.2.7 Graph Theory

Depending on the context, we denote the vertices of a graph G with V_G and its edge set with E_G. An *oriented path* is a digraph obtained by orienting the edges of an undirected path. In other words, an oriented path has vertices $v_0, \ldots, v_{q+1}$ and edges $e_0, \ldots, e_q$, where e_i is either (v_i, v_{i+1}), or (v_{i+1}, v_i). We call (v_i, v_{i+1}) a *forward edge* and (v_{i+1}, v_i) a *backward edge*. The *length* of an oriented path, $len(P)$, is the number of edges it contains. The *net length* of a path P, $net(P)$, is the number of forward edges minus the number

of backward edges in P. A *(reverse) dipath* is a sequence of, not necessarily distinct, vertices $v_1, \ldots, v_n$ such that for every $i \in [n-1]$, (v_i, v_{i+1}) $((v_{i+1}, v_i))$ is an edge. A *directed cycle* is a dipath $v_1, \ldots, v_n$ such that (v_n, v_1) is also an edge. A *cycle* is an oriented path with starting point v_1 and endpoint v_m such that either (v_m, v_1) or (v_1, v_m) is an edge. We use the term *simple* dipath or (directed) cycle to indicate that all vertices of the dipath or (directed) cycle are distinct.

A *component* of digraph G is a maximal subgraph H of G such that for every pair of vertices $u, v \in V_H$, there is an oriented path from u to v. A digraph with one component is said to be *connected*. A digraph is a *directed acyclic graph* (DAG) if it contains no directed cycles. A DAG G is *balanced* if there exists $q \in \mathbb{N}$ such that V_G can be partitioned into q levels $L_0, \ldots, L_{q-1}$, such that any edge of G goes from L_i to L_{i+1}, for some $i \in \{0, \ldots, q-2\}$. We always choose q to be minimal. Observe that any oriented path is balanced. The mapping level : $V_G \to \{0, 1, 2, \ldots, q-1\}$ assigns each vertex v a number i such that v is in the i-th level L_i. The *level of an edge* (a, b) of G is level(a), i.e. the level of the starting vertex of (a, b). The height of G (denoted with height(G)) is $\max_{a \in V_G}$ level(a).

Let P be an oriented path whose vertex with indegree 0 and outdegree 1 is u, and whose vertex with indegree 1 and outdegree 0 is v. We say that P is *minimal* if u is in L_0 (the bottommost level) and v is in $L_{\text{height}(G)}$ (the topmost level), and there are no other vertices of P in the bottommost or the topmost levels.

If a and b are vertices of G, $a \stackrel{k}{\to} b$ denotes the existence of a dipath from a to b of length k; $a \to b$ denotes $a \stackrel{1}{\to} b$. Let G be a digraph, and x a vertex of G. We define $x^{+1} = \{y \in V_G : (x, y) \in E_G\}$ (E_G is the edge set of G), and $x^{-1} = \{y \in V_G : (y, x) \in E_G\}$. We call a vertex v a *source* if $v^{-1} = \varnothing$, and a *sink* if $v^{+1} = \varnothing$. Oriented paths can be thought of as relational structures over the vocabulary $\{E^2\}$, so sometimes we denote them with boldface letters, e.g. $\mathbf{P}$.

Chapter 3

Dualities for Symmetric Datalog[1]

3.1 Introduction

As mentioned in Section 1.3 (Chapter 1), strong connections between membership of CSP($\mathbf{B}$) in L and expressibility of co-CSP($\mathbf{B}$) in symmetric Datalog have been indicated and conjectured [38, 63]. Symmetric Datalog programs can be evaluated in logarithmic space (L) [38] using Reingold's logspace algorithm for undirected *st*-connectivity [77], and it is conjectured that if CSP($\mathbf{B}$) is in L then co-CSP($\mathbf{B}$) can be defined in symmetric Datalog [38]. Therefore providing tools to show whether co-CSP($\mathbf{B}$) can be defined in symmetric Datalog is an important task. In this chapter, we introduce two *dualities* for

[1]The contents of this chapter were published as the first part of [35].

symmetric Datalog and study expressibility of CSPs in symmetric Datalog by means of these new dualities.

Recall Section 2.2.6 from Chapter 2 and the definitions there. The following two well-known theorems relate definability of co-CSP($\mathbf{B}$) in Datalog and linear Datalog to $\mathbf{B}$ having *bounded treewidth* and *bounded pathwidth* duality, respectively:

1. co-CSP($\mathbf{B}$) is definable in Datalog if and only if $\mathbf{B}$ has bounded treewidth duality [43];

2. co-CSP($\mathbf{B}$) is definable in linear Datalog if and only if $\mathbf{B}$ has bounded pathwidth duality [27].

It was stated as an open problem in [18] to find a duality for symmetric Datalog in the spirit of the previous two theorems. We provide two such dualities: *symmetric bounded pathwidth duality* (SBPD) and *piecewise symmetric bounded pathwidth duality* (PSBPD). We note that SBPD is a special case of PSBPD. For both bounded treewidth and bounded pathwidth duality, the structures in the obstruction sets are restricted to have some special form. For SBPD and PSBPD the situation is a bit more subtle. In addition we require the obstruction sets to contain structures only of a special form (they must have bounded pathwidth), the obstruction sets must also possess a certain "symmetric closure" property. To the best of our knowledge, this is the first instance of a duality where in addition to the local requirement that each structure must be of a certain form, the set must also satisfy an

interesting global requirement.

Using SBPD, we give a short and simple new proof of the main result of [31] that "Maltsev + Datalog ⇒ symmetric Datalog". Considering the simplicity of this proof, we suspect that SBPD (or PSBPD) could be a useful tool in an attempt to prove the *algebraic symmetric Datalog conjecture* [63], a conjecture that proposes an algebraic characterization of all CSPs lying in L. An equivalent form of this conjecture is that "Datalog + n-permutability ⇒ symmetric Datalog" (by combining results from [54, 6, 64]), where n-permutability is a generalization of Maltsev.

One way to gain more insight into the dividing line between CSPs in L or NL is through studying the complexity of CSPs corresponding to oriented paths. The only known thing regarding the complexity of these CSPs is that they are all in NL (by combining results from [40, 30, 27]). To make progress in this direction, it is natural to ask whether there are oriented paths for which the CSP is NL-complete and L-complete. We provide two classes of oriented paths, $\mathcal{C}_1$ and $\mathcal{C}_2$, such that for any $\mathbf{B}_1 \in \mathcal{C}_1$, the corresponding CSP is NL-complete, and for any $\mathbf{B}_2 \in \mathcal{C}_2$, the corresponding CSP is in L. In fact, it can be seen with the help of [63] that for most $\mathbf{B}_2 \in \mathcal{C}_2$, CSP($\mathbf{B}_2$) is L-complete. To prove the membership of CSP($\mathbf{B}_2$) in L (for $\mathbf{B}_2 \in \mathcal{C}_2$), we use PSBPD in an essential way. One can hope to build on this work to achieve an L-NL dichotomy for oriented paths.

3.2 Preliminaries

3.2.1 Path Decompositions and Derivations

Definition 17. [Path-Decomposition] Let $\mathbf{S}$ be a τ-structure. A (j,k)-*path decomposition* (or path decomposition of *width* (j,k)) of $\mathbf{S}$ is a sequence $S_0,\ldots,S_{n-1}$ of subsets of S such that

1. For every $(R,(a_1,\ldots,a_{\mathrm{ar}(R)})) \in \tilde{\mathbf{S}}$, $\exists \ell \in \{0,\ldots,n-1\}$ such that $\{a_1,\ldots,a_{\mathrm{ar}(R)}\} \subseteq S_\ell$;
2. If $a \in S_i \cap S_{i'}$ $(i<i')$ then $a \in S_\ell$ for all $i<\ell<i'$;
3. $\forall \ell \in \{0,\ldots,n-1\}$, $|S_\ell| \leq k$, and $\forall \ell \in \{0,\ldots,n-2\}$, $|S_\ell \cap S_{\ell+1}| \leq j$.

For ease of notation, it will be useful to introduce a concept closely related to path decompositions. Let τ be a vocabulary. Let $\mathbf{S}$ be a τ-structure that can be expressed as $\mathbf{S}=\mathbf{S}_0\cup\cdots\cup\mathbf{S}_{n-1}$, where the $S_0,\ldots,S_{n-1}$ (the universes of the $\mathbf{S}_i$) satisfy properties 2 and 3 above. Note that $\cup$ here denotes union, *not* disjoint union of τ-structures. We say that $\mathbf{S}$ is a (j,k)-*path*, and that $(\mathbf{S}_0,\ldots,\mathbf{S}_{n-1})$ is a (j,k)-*path representation* of $\mathbf{S}$. We denote (j,k)-path representations with script letters, e.g. $\mathscr{S}=(\mathbf{S}_0,\ldots,\mathbf{S}_{n-1})$. The substructure $\mathbf{S}_i\cup\cdots\cup\mathbf{S}_{i'}$ of $\mathbf{S}$ (assuming a (j,k)-representation is fixed) is denoted with $\mathbf{S}_{[i,i']}$. We call n the *length* of the representation. Obviously, a structure is a (j,k)-path if and only if it admits a (j,k)-path decomposition.

Let $\mathscr{D}=(\rho_1,\lambda_1),\ldots,(\rho_q,\lambda_q)$ be a derivation for some linear or symmetric program $\mathcal{P}$ with vocabulary τ. We can extract from $\mathscr{D}$ a τ-structure $\mathrm{Ex}(\mathscr{D})$

such that $\mathscr{D}$ is a derivation for $\mathrm{Ex}(\mathscr{D})$. We specify $\mathrm{Ex}(\mathscr{D})$ as a tuple structure $\tilde{\mathbf{A}}$: for each $R(\mathbf{t})$ that appears in $\mathscr{D}$ ($R \in \tau$), we add the pair $(R, \mathbf{t})$ to $\tilde{\mathbf{A}}$, and set $\tilde{A}$ to be the set of those elements that appear in a tuple.

Let $\mathscr{D} = (\rho_1, \lambda_1), \ldots, (\rho_q, \lambda_q)$ be a derivation. For each x that is in a rule ρ_ℓ for some $\ell \in [q]$, call x^ℓ the *indexed version of* x. We define an equivalence relation $\mathrm{Eq}(\mathscr{D})$ on the set of indexed variables of $\mathscr{D}$. First we define a graph $G = (V, E)$ as:

- V is the set of all indexed versions of variables in $\mathscr{D}$;
- $(x^\ell, y^{\ell'}) \in E$ if $\ell' = \ell + 1$, x is the i-th variable of the head IDB I of ρ_ℓ, and y is the i-th variable of the body IDB I of $\rho_{\ell+1}$.

Two indexed variables x^ℓ and $y^{\ell'}$ are related in $\mathrm{Eq}(\mathscr{D})$ if they are connected in G. Observe that if $C = \{x_1^{\ell_1}, x_2^{\ell_2}, \ldots, x_c^{\ell_c}\}$ is a connected component of G, then it must be that $\lambda_{\ell_1}(x_1) = \lambda_{\ell_2}(x_2) = \cdots = \lambda_{\ell_c}(x_c)$.

Definition 18 (Free Derivation)**.** Let $\mathcal{P}$ be a linear Datalog program and $\mathscr{D} = (\rho_0, \lambda_0), \ldots, (\rho_q, \lambda_q)$ be a derivation for $\mathcal{P}$. Then $\mathscr{D}$ is said to be *free* if for any two $(x^\ell, y^{\ell'}) \notin \mathrm{Eq}(\mathscr{D})$, $\lambda_\ell(x) \neq \lambda_{\ell'}(y)$.

Intuitively, this definition says that $\mathscr{D}$ is free if any two variables in $\mathscr{D}$ which are not "forced" to have the same value are assigned different values.

3.2.2 Canonical Programs

Fix a τ-structure $\mathbf{B}$ and $j \leq k$. Let $Q_1, \ldots, Q_n$ be all possible at most j-ary relations over B. The *canonical linear* (j,k)*-Datalog program for* $\mathbf{B}$ *(*(j,k)-

CanL($\mathbf{B}$)*)* contains an IDB I_m of the same arity as Q_m for each $m \in [n]$. The rule $I_c(\mathbf{x}) \leftarrow I_d(\mathbf{y}) \wedge \bar{E}(\mathbf{z})$ belongs to the canonical program if it contains at most k variables, and the implication $Q_c(\mathbf{x}) \leftarrow Q_d(\mathbf{y}) \wedge \bar{E}(\mathbf{z})$ is true for all possible instantiations of the variables to elements of B. The goal predicate of this program is the 0-ary IDB I_g, where $Q_g = \emptyset$. In this case, the implication $Q_g \leftarrow Q_d(\mathbf{y}) \wedge \bar{E}(\mathbf{z})$ means that $Q_d(\mathbf{y}) \wedge \bar{E}(\mathbf{z})$ is false for any instantiation of $\mathbf{y}p$ and $\mathbf{z}$.

The *canonical symmetric* (j,k)*-Datalog program for* $\mathbf{B}$ *(*(j,k)-CanS($\mathbf{B}$)*)* has the same definition as (j,k)-CanL($\mathbf{B}$), except that it has less rules due to the following additional restriction. If $I_c(\mathbf{x}) \leftarrow I_d(\mathbf{y}) \wedge \bar{E}(\mathbf{z})$ is in the program, then both $Q_c(\mathbf{x}) \leftarrow Q_d(\mathbf{y}) \wedge \bar{E}(\mathbf{z})$ and $Q_d(\mathbf{y}) \leftarrow Q_c(\mathbf{x}) \wedge \bar{E}(\mathbf{z})$ must hold for all possible instantiations of the variables to elements of B. The program (j,k)-CanS($\mathbf{B}$) is obviously symmetric. When it is clear from the context, we write CanL($\mathbf{B}$) and CanS($\mathbf{B}$) instead of (j,k)-CanL($\mathbf{B}$) and (j,k)-CanS($\mathbf{B}$), respectively.

3.2.3 The Main Goal

Recall from Chapter 2 that an obstruction set $\mathcal{O}$ defines co-CSP($\mathbf{B}$) implicitly as $\mathbf{A} \in$ co-CSP($\mathbf{B}$) if and only if there exists $\mathbf{S} \in \mathcal{O}$ such that $\mathbf{S} \to \mathbf{A}$. If $\mathcal{O}$ above can be chosen to have property X, then we say that $\mathbf{B}$ has X-duality. Our main goal is to show that co-CSP($\mathbf{B}$) is definable in symmetric Datalog if and only if $\mathbf{B}$ has *symmetric bounded pathwidth* duality, defined below after going through the necessary background.

3.2.4 The Zigzag Operator

A *zigzag operator* ξ takes a (j,k)-path representation $\mathscr{S} = (\mathbf{S}_0, \ldots, \mathbf{S}_{n-1})$ of a (j,k)-path $\mathbf{S}$ and a minimal oriented path $\mathbf{P} = e_0, \ldots, e_q$ such that $\text{height}(\mathbf{P}) = n$, and it returns another (j,k)-path $\xi(\mathscr{S}, \mathbf{P})$. Intuitively, $\xi(\mathscr{S}, \mathbf{P})$ is the (j,k)-path $\mathbf{S}$ "modulated" by $\mathbf{P}$ such that the forward and backward edges e_i of $\mathbf{P}$ are mimicked in $\xi(\mathscr{S}, \mathbf{P})$ by "forward and backward" copies of $\mathbf{S}_{\text{level}(e_i)}$. Before the formal definition, it could help the reader to look at the right side of Figure 3.1, where the oriented path used to modulate the (j,k)-path over the vocabulary E^2 (i.e. digraphs) with representation $(\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2)$ is $\mathbf{P}$ on the left side. The left side is a more abstract example, and the reader might find it useful after reading the definition.

We inductively define the (j,k)-path $\xi(\mathscr{S}, \mathbf{P})$ as $(\mathbf{S}_{e_0}, \mathbf{S}_{e_1}, \ldots, \mathbf{S}_{e_q})$ together with a sequence of isomorphisms $\varphi_{e_0}, \varphi_{e_1}, \ldots, \varphi_{e_q}$, where φ_{e_i} is an isomorphism from $\mathbf{S}_{e_i}$ to $\mathbf{S}_{\text{level}(e_i)}$, $0 \leq i \leq q$. For the base case, we define $\mathbf{S}_{e_0}$ to be an isomorphic copy of $\mathbf{S}_0$, and φ_{e_0} to be the isomorphism that maps $\mathbf{S}_{e_0}$ back to $\mathbf{S}_0$. Assume inductively that $\mathbf{S}_{e_0}, \ldots, \mathbf{S}_{e_{i-1}}$ and $\varphi_{e_0}, \ldots, \varphi_{e_{i-1}}$ are already defined. Let $\mathbf{S}'_{e_i}$ be an isomorphic copy of $\mathbf{S}_{\text{level}(e_i)}$ with domain disjoint from $S_{e_0} \cup \cdots \cup S_{e_{i-1}}$, and fix φ'_{e_i} to be the isomorphism that maps back S'_{e_i} to $S_{\text{level}(e_i)}$. We "glue" $\mathbf{S}'_{e_i}$ to $\mathbf{S}_{e_{i-1}}$ by renaming some elements of $\mathbf{S}'_{e_i}$ to elements of $\mathbf{S}_{e_{i-1}}$. To facilitate understanding, we can think of the already constructed structures $\mathbf{S}_{e_0}, \ldots, \mathbf{S}_{e_{i-1}}$ as labels of the edges $e_0, \ldots, e_{i-1}$ of $\mathbf{P}$, respectively, and we want to determine $\mathbf{S}_{e_i}$, the label of the next edge. The connection between $\mathbf{S}_{e_{i-1}}$ and $\mathbf{S}_{e_i}$ will be defined such that $\mathbf{S}_{e_{i-1}}$ and $\mathbf{S}_{e_i}$

"mimic" the orientation of the edges e_{i-1} and e_i.

We resume our formal definition. Set $\ell = \text{level}(e_i)$, and let $\ell' = \ell - 1$ if e_i is a forward edge, and $\ell' = \ell + 1$ if e_i is a backward edge. If an element $x \in S'_{e_i}$ and an element $y \in S_{e_{i-1}}$ are both copies of the same element $a \in S_\ell \cap S_{\ell'}$, then rename x to y in S'_{e_i}. After all such elements are renamed, $\mathbf{S}'_{e_i}$ becomes $\mathbf{S}_{e_i}$. That is, for all $a \in S_\ell \cap S_{\ell'}$, rename $\varphi'^{-1}_{e_i}(a)$ in $\mathbf{S}'_{e_i}$ to $\varphi^{-1}_{e_{i-1}}(a)$ to obtain $\mathbf{S}_{e_i}$.

We define the isomorphism φ_{e_i} from $\mathbf{S}_{e_i}$ to $\mathbf{S}_{\text{level}(e_i)}$ as:

$$\varphi_{e_i}(x) = \begin{cases} \varphi'_{e_i}(x) & \text{if } x \in S_{e_i} \text{ and } x \notin S_{e_{i-1}} \\ \varphi_{e_{i-1}}(x) & \text{if } x \in S_{e_i} \cap S_{e_{i-1}}. \end{cases}$$

3.3 Two Dualities for Symmetric Datalog

The two main theorems (Theorems 22 and 29) of this section can be combined to obtain:

Theorem 19. *For a finite structure* $\mathbf{B}$*, TFAE:*

1. *There is a symmetric Datalog program that defines* co-CSP($\mathbf{B}$)*;*
2. $\mathbf{B}$ *has symmetric bounded pathwidth duality (for some parameters);*
3. $\mathbf{B}$ *has piecewise symmetric bounded pathwidth duality (for some parameters);*

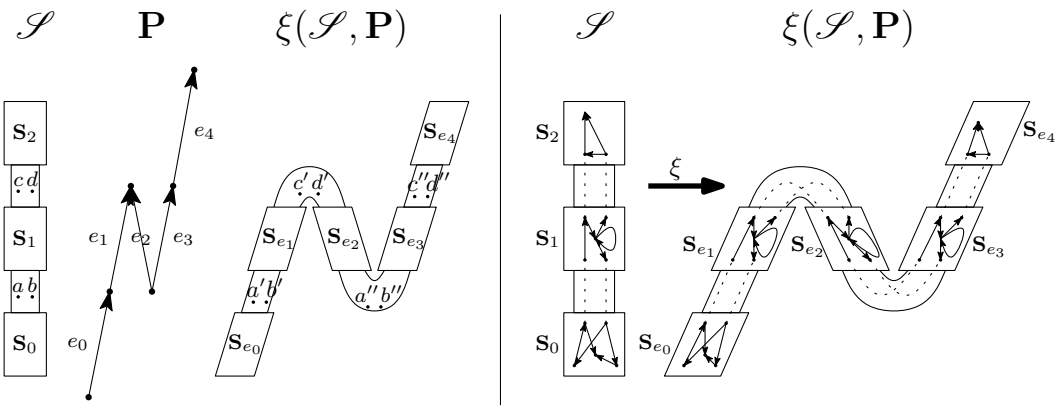


Figure 3.1: *Left:* Applying a zigzag operator to the (j,k)-path $\mathbf{S}$ with the (j,k)-representation $\mathscr{S} = (\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2)$. Suppose that $S_0 \cap S_1 = \{a, b\}$ and $S_1 \cap S_2 = \{c, d\}$. We demonstrate how $\mathbf{S}_{e_0}$ and $\mathbf{S}_{e_2}$ are obtained. $\mathbf{S}_{e_0}$ is a disjoint copy of $\mathbf{S}_0$ (and the copy of a and b in $\mathbf{S}_{e_0}$ are a' and b', respectively). To obtain $\mathbf{S}_{e_2}$, first make a disjoint copy $\mathbf{S}'_{e_2}$ of $\mathbf{S}_{\text{level}(e_2)} = \mathbf{S}_1$. Set $\ell = \text{level}(e_2) = 1$. Since e_1 is a forward edge and e_2 is a backward edge, $\ell' = \ell + 1 = 2$. Therefore to "glue" $\mathbf{S}'_{e_2}$ to $\mathbf{S}_{e_1}$, we need to look at $S_\ell \cap S_{\ell'} = \{c, d\}$. Assume that the copy of c and d in $\mathbf{S}_{e_1}$ are c' and d', respectively. Furthermore, assume that the copy of c and d in $\mathbf{S}'_{e_2}$ are $\tilde{c}$ and $\tilde{d}$, respectively. To obtain $\mathbf{S}_{e_2}$, we rename $\tilde{c}$ to c', and $\tilde{d}$ to d' in $\mathbf{S}'_{e_2}$. *Right:* A specific example when $\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2$ are the digraphs in the boxes. The dashed lines indicate identification of vertices. The level of $(\mathbf{S}_{e_2}, \mathbf{S}_{e_3})$, for example, is 0 since e_2 and e_3 share a vertex at vertex level 1 (defined below).

3.3.1 Symmetric Bounded Pathwidth Duality

Definition 20 ((j,k)-symmetric)**.** Assume that $\mathcal{O}$ is a set of (j,k)-paths. Suppose furthermore that a (j,k)-path representation can be fixed for each structure in $\mathcal{O}$ such that the following holds. For every $\mathbf{S} \in \mathcal{O}$ with representation $\mathscr{S}$ of some length n, and every minimal oriented path $\mathbf{P}$ of height n, it holds that $\xi(\mathscr{S}, \mathbf{P}) \in \mathcal{O}$. Then $\mathcal{O}$ is said to be (j,k)-*symmetric*.

Definition 21 (SBPD)**.** A structure $\mathbf{B}$ has (j,k)-*symmetric bounded path-width duality* ((j,k)-*SBPD*) if there is an obstruction set $\mathcal{O}$ for $\mathbf{B}$ that consists of (j,k)-paths, and in addition, $\mathcal{O}$ is (j,k)-symmetric.

The following is our main duality theorem for symmetric Datalog:

Theorem 22. *For a finite structure* $\mathbf{B}$, co-CSP($\mathbf{B}$) *can be defined by a symmetric* (j,k)*-Datalog program if and only if* $\mathbf{B}$ *has* (j,k)*-SBPD.*

We will use Lemma 23 in the proof of Theorem 22. Lemma 23 can be proved using the standard canonical Datalog argument. Lemma 24 is also used in the proof of Theorem 22 and it is the main technical lemma of the section.

Lemma 23. *If* CanS($\mathbf{B}$) *accepts a structure* $\mathbf{A}$*, then* $\mathbf{A} \not\to \mathbf{B}$.

Proof. Structure $\mathbf{B}$ is not accepted by CanS($\mathbf{B}$) because a derivation could be translated into a valid chain of implications, which is not possible by the definition of CanS($\mathbf{B}$). If CanS($\mathbf{B}$) accepts $\mathbf{A}$ and $\mathbf{A} \to \mathbf{B}$, then CanS($\mathbf{B}$) accepts $\mathbf{B}$, a contradiction. □

Lemma 24. *For any* τ*-structures* $\mathbf{A}$ *and* $\mathbf{B}$*, if there exists a structure* $\mathbf{S}$ *with a* (j,k)*-path representation* $\mathscr{S}$ *of some length* n *such that* $\mathbf{S} \to \mathbf{A}$*, and for any minimal oriented path* $\mathbf{P}$ *of height* n*, it holds that* $\xi(\mathscr{S}, \mathbf{P}) \not\to \mathbf{B}$*, then* (j,k)-CanS($\mathbf{B}$) *accepts* $\mathbf{A}$.

To prove Lemma 24 we need to define an additional concept related to the zigzag operator. Once the (j,k)-path $\xi(\mathscr{S}, \mathbf{P}) = (\mathbf{S}_{e_0}, \ldots, \mathbf{S}_{e_q})$ is defined,

where $\mathbf{P}$ is the path $e_0, \ldots, e_q$, each pair $(\mathbf{S}_{e_i}, \mathbf{S}_{e_{i+1}})$, $\forall i \in \{0, \ldots, q-1\}$ is assigned a *level*: level$(\mathbf{S}_{e_i}, \mathbf{S}_{e_{i+1}})$ is the level of the vertex v minus 1, where v is the vertex that e_i and e_{i+1} share (see Figure 3.1).

Proof of Lemma 24. For the rest of this proof, let $\mathcal{CS}$ denote (j,k)-CanS$(\mathbf{B})$, and $\mathcal{CL}$ denote (j,k)-CanL$(\mathbf{B})$. If program $\mathcal{CS}$ accepts structure $\mathbf{S}$ then because $\mathbf{S} \to \mathbf{A}$, $\mathcal{CS}$ also accepts $\mathbf{A}$. So it is sufficient to show that program $\mathcal{CS}$ accepts structure $\mathbf{S}$.

First we specify how to associate a $\mathcal{CL}$-derivation with $\xi(\mathscr{S}, \mathbf{P})$, where $\mathbf{P}$ is a minimal oriented path of height n. Assume that $\xi(\mathscr{S}, \mathbf{P}) = \mathbf{S}_0 \cup \cdots \cup \mathbf{S}_q$. For each $i \in \{0, \ldots, q-1\}$, fix an arbitrary order on the elements of $S_i \cap S_{i+1}$. Assume that $|S_i \cap S_{i+1}| = j' (\leq j)$, and define the j'-tuple $\mathbf{s}_i$ such that $\mathbf{s}_i[\ell]$ is the ℓ-th element of $S_i \cap S_{i+1}$. We define $\mathbf{s}_q$ to be the empty tuple. It is good to keep in mind that later, $\mathbf{s}_i$ will be associated with the IDB J_i.

The derivation will be $\mathscr{D}_{\mathcal{CL}}(\xi(\mathscr{S}, \mathbf{P})) = (\rho_0, \lambda_0), \ldots, (\rho_q, \lambda_q)$. We specify ρ_i as

$$J_i(\mathbf{x}_i) \leftarrow J_{i-1}(\mathbf{x}_{i-1}) \wedge \bar{E}(\mathbf{y}_i) \quad \text{if } i \in [q] \qquad\qquad J_0(\mathbf{x}_0) \leftarrow \bar{E}(\mathbf{y}_0) \quad \text{if } i = 0.$$

We begin with describing the EDBs of a rule ρ_i together with their variables. Assume that $S_i = \{d_1, \ldots, d_t\}$, and observe that $t \leq k$. The variables of ρ_i are $v_1, \ldots, v_t$. For every $R \in \tau$, and every tuple $(d_{f(1)}, \ldots, d_{f(r)}) \in R^{\mathbf{S}_i}$, where $r = \text{ar}(R)$, $R(v_{f(1)}, \ldots, v_{f(r)})$ is an EDB of ρ_i.

We describe the variables of the IDBs J_{i-1} and J_i, $i \geq 1$ (the case when $i = 0$ is handled similarly). Assume that $\mathbf{s}_{i-1} = (d_{g(1)}, \ldots, d_{g(j_1)})$ and $\mathbf{s}_i = (d_{h(1)}, \ldots, d_{h(j_2)})$. Then the IDB in the body of ρ_i together with its variables is $J_{i-1}(v_{g(1)}, \ldots, v_{g(j_1)})$, and the head IDB together with its variables is $J_i(v_{h(1)}, \ldots, v_{h(j_2)})$. The function λ_i simply assigns the value d_g to the variable v_g, $\forall g \in [t]$.

It remains to specify the IDBs, i.e. which IDBs of $\mathcal{CL}$ the J_i-s correspond to. For each $i \in \{0, \ldots, q\}$, J_i denotes $J_{M_i^{\mathbf{P}}}$, where $M_i^{\mathbf{P}}$ is a subset of $B^{j'}$ for some $j' \leq j$. We define the sequence $M_0^{\mathbf{P}}, M_1^{\mathbf{P}}, \ldots, M_q^{\mathbf{P}}$ inductively. To define $M_0^{\mathbf{P}}$, consider the nonrecursive rule $J_0(\mathbf{x}_0) \leftarrow \bar{E}(\mathbf{y}_0)$. Assume that the arity of J_0 is j', and that $\mathbf{y}_0$ contains k' variables. (Note that the variables in $\mathbf{x}_0$ and $\mathbf{y}_0$ are not necessarily disjoint.) For all possible functions $\alpha : \mathbf{x}_0[1], \ldots, \mathbf{x}_0[j'], \mathbf{y}_0[1], \ldots, \mathbf{y}_0[k'] \to B$ such that the conjunction of EDBs $\bar{E}(\alpha(\mathbf{y}_0[1]), \ldots, \alpha(\mathbf{y}_0[k']))$ is true, place the tuple $(\alpha(\mathbf{x}_0[1]), \ldots, \alpha(\mathbf{x}_0[j']))$ into $M_0^{\mathbf{P}}$. In the special case when $\mathbf{x}_0[1], \ldots, \mathbf{x}_0[j'], \mathbf{y}_0[1], \ldots, \mathbf{y}_0[k']$ is the empty string ϵ, $\alpha(\epsilon)$ is defined to be ϵ. Note that a 0-ary relation R is either empty or it contains ϵ. If R is empty, then R false, and if R contains ϵ, then R is true.

Assume that $M_{i-1}^{\mathbf{P}}$ is already defined. Then similarly to the base case, for each possible instantiation α of the variables of ρ_i over B with the restriction that $\alpha(\mathbf{x}_{i-1}) \in M_{i-1}^{\mathbf{P}}$, if the conjunction of the EDBs of ρ_i is true, then add the tuple $\alpha(\mathbf{x}_i)$ to $M_i^{\mathbf{P}}$. It is not difficult to see that if $M_q^{\mathbf{P}} \neq \emptyset$ (i.e. $M_q^{\mathbf{P}}$ contains ϵ), then we can construct a homomorphism from $\xi(\mathscr{S}, \mathbf{P})$ to $\mathbf{B}$ which would

be a contradiction.

For each $i \in \{0, \ldots, q-1\}$, assume that $(\mathbf{S}_i, \mathbf{S}_{i+1})$ has level ℓ_i. Then we say that the IDB J_i has level ℓ_i and we write $\text{level}(J_i) = \ell_i$.

We proceed to construct a $\mathcal{CS}$-derivation $\mathscr{D}_{\mathcal{CS}}(\mathbf{S})$ for $\mathbf{S}$. Let $\mathbf{Q}$ be a directed path of height n. We construct $\mathscr{D}_{\mathcal{CS}}(\mathbf{S})$ just like we would construct $\mathscr{D}_{\mathcal{CL}}(\xi(\mathscr{S}, \mathbf{Q}))$ above, except that we will define the subscripts of the IDBs, $M_0^{\mathbf{Q}}, \ldots, M_{n-1}^{\mathbf{Q}}$, differently, so that every rule of the resulting derivation belongs to $\mathcal{CS}$. From now on we write $M_0, \ldots, M_{n-1}$ instead of $M_0^{\mathbf{Q}}, \ldots, M_{n-1}^{\mathbf{Q}}$.

To define $M_0, \ldots, M_{n-1}$, let $\mathbf{P}_0, \mathbf{P}_1, \ldots$ be an enumeration of all (finite) minimal oriented paths of height n. Intuitively, we will collect in $\mathcal{N}_m^\ell$ all subscripts (recall that a subscript is a relation) of all those IDBs which have the same level ℓ in $\mathscr{D}_{\mathcal{CL}}(\xi(\mathscr{S}, \mathbf{P}_m))$. Formally, for each $\ell \in \{0, \ldots, n-1\}$ define $\mathcal{N}_m^\ell = \{M_t^{\mathbf{P}_m} \mid \text{level}(J_t) = \ell\}$. Then we collect the subscripts at a fixed level ℓ in $\mathcal{O}_\ell$ over all derivations corresponding to $\mathbf{P}_0, \mathbf{P}_1, \ldots$. Formally, for each $\ell \in \{0, \ldots, n-1\}$, we define $\mathcal{O}_\ell = \mathcal{N}_0^\ell \cup \mathcal{N}_1^\ell, \ldots$. We are ready to define $M_0, \ldots, M_{n-1}$. For each $s \in \{0, \ldots, n-1\}$, define $M_s = \bigcup_{W \in \mathcal{O}_s} W$.

It remains to show that every rule of the derivation we defined is in $\mathcal{S}$ and that the last IDB is the goal IDB. If the last IDB is not the goal IDB of $\mathcal{S}$, then $M_{n-1} \neq \emptyset$. By definition, it must be that for some minimal oriented path $\mathbf{P}_m$ of height n and length q_m, $\mathbf{M}_{q_m-1}^{\mathbf{P}_m} \neq \emptyset$ (note that the last IDB of $\mathscr{D}_{\mathcal{CL}}(\xi(\mathscr{S}, \mathbf{P}_m))$ has subscript $\mathbf{M}_{q_m-1}^{\mathbf{P}_m}$). As noted before, this would mean that $\xi(\mathscr{S}, \mathbf{P}_m) \to \mathbf{B}$, a contradiction.

We show that each rule of $\mathscr{D}_{\mathcal{CS}}(\mathbf{S})$ as defined above belongs to $\text{CanS}(\mathbf{B})$.

Suppose $\mathscr{D}_{\mathcal{CS}}(\mathbf{S})$ contains a rule ρ

$$J_i(\mathbf{x}_i) \leftarrow J_{i-1}(\mathbf{x}_{i-1}) \wedge \bar{E}(\mathbf{y}_i)$$

that is not in CanS($\mathbf{B}$). By definition, there cannot be an instantiation α of variables of ρ to elements of B such that $\alpha(\mathbf{x}_{i-1}) \in M_{i-1}$, the conjunction of EDBs holds, but $\alpha(\mathbf{x}_i) \notin M_i$. Assume then that there is an α such that $\alpha(\mathbf{x}_i) \in M_i$, the conjunction of EDBs holds, but $\alpha(\mathbf{x}_{i-1}) \notin M_{i-1}$. It is also not difficult to see that this is not possible because we used *all* minimal oriented paths in the construction of $\mathscr{D}_{\mathcal{CS}}(\mathbf{S})$. □

Proof of Theorem 22. If co-CSP($\mathbf{B}$) is defined by a symmetric (j,k)-Datalog program $\mathcal{P}$, then using the symmetric property of $\mathcal{P}$, it is laborious but straightforward to show that

$$\mathcal{O} = \bigcup_{\substack{\mathscr{D} \text{ is a free} \\ \text{derivation of } \mathcal{P}}} \{\mathrm{Ex}(\mathscr{D})\}$$

is a (j,k)-symmetric obstruction set for $\mathbf{B}$.

For the converse, assume that $\mathbf{B}$ has (j,k)-SBPD. Let $\mathcal{O}$ be a symmetric obstruction set of width (j,k) (i.e. the path decomposition of every structure in $\mathcal{O}$ has width (j,k)) for $\mathbf{B}$. We claim that (j,k)-CanS($\mathbf{B}$) defines co-CSP($\mathbf{B}$). Assume that $\mathbf{A} \to \mathbf{B}$. Then by Lemma 23, (j,k)-CanS($\mathbf{B}$) does not accept $\mathbf{A}$. Suppose now that $\mathbf{A} \not\to \mathbf{B}$. Then by assumption, there exists a (j,k)-path $\mathbf{S} \in \mathcal{O}$ with a representation $\mathscr{S}$ of length n such that $\mathbf{S} \to \mathbf{A}$.

Furthermore, since $\mathcal{O}$ is symmetric, for any minimal oriented path $\mathbf{P}$ of height n, $\xi(\mathscr{S}, \mathbf{P}) \not\to \mathbf{B}$. It follows from Lemma 24 that CanS($\mathbf{B}$) accepts $\mathbf{A}$. □

From the above proof it is obvious that:

Corollary 25 ([31])**.** *If a symmetric* (j,k)*-Datalog program defines* co-CSP($\mathbf{B}$)*, then so does* (j,k)-CanS($\mathbf{B}$)*.*

3.3.2 Piecewise Symmetric Bounded Pathwidth Duality

Piecewise symmetric bounded pathwidth duality (PSBPD) for symmetric Datalog is less stringent than SBPD; however, the price is larger program width. Although the following definitions might seem technical, the general idea is simple: a piecewise symmetric obstruction set $\mathcal{O}$ does not need to contain all (j,k)-paths obtained by "zigzagging" (j,k)-paths in $\mathcal{O}$ in all possible ways. It is sufficient to zigzag a (j,k)-path $\mathbf{S}$ using only oriented paths which "avoid" certain segments of $\mathbf{S}$: some constants c and d are fixed for $\mathcal{O}$, and there are at most c fixed segments of $\mathbf{S}$ that are avoided by the zigzag operator, each of size at most d. We give the formal definitions.

Definition 26 ((c,d)-filter)**.** Let $\mathbf{S}$ be a (j,k)-path with a representation $\mathscr{S} = \mathbf{S}_0, \ldots, \mathbf{S}_{n-1}$. A (c,d)-*filter* $\mathscr{F}$ for $\mathscr{S}$ is a set of intervals

$$\{[s_1, t_1], [s_2, t_2], \ldots, [s_{c'}, t_{c'}]\}$$

such that

1. $c' \leq c$; $0 \leq s_1$; $t_{c'} \leq n-1$; $s_i \leq t_i, \forall i \in [c']$; and $t_\ell + 2 \leq s_{\ell+1}, \forall \ell \in [c'-1]$;
2. $|\bigcup_{i \in [s_\ell, t_\ell]} S_i| \leq d, \forall \ell \in [c']$.

Elements of $\mathscr{F}$ are called *delimiters*. An oriented path $\mathbf{P}$ of height n *obeys* a (c,d)-filter $\mathscr{F}$ if for any delimiter $[s_i, t_i] \in \mathscr{F}$, the set of edges e of $\mathbf{P}$ such that $s_i \leq \text{level}(e) \leq t_i$ form a (single) directed path. A demonstration is given in Figure 3.2.

Remark: The first condition says that we have at most c delimiters (i.e. intervals), and that two consecutive delimiters are not right next to each other. The second condition limits the size of the substructure of $\mathbf{S}$ associated with a delimiter.

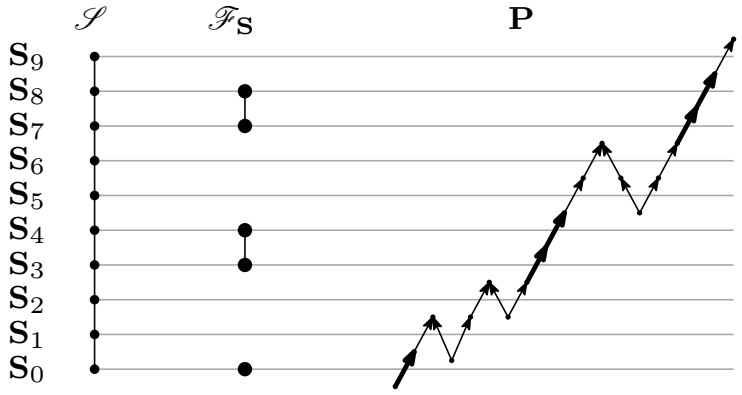


Figure 3.2: $\mathscr{S}$ is a (j,k)-path representation of $\mathbf{S}$. $\mathscr{F}_\mathbf{S}$ is the $(3, 2k)$-filter $\{[0,0], [3,4], [7,8]\}$ for $\mathscr{S}$. $\mathbf{P}$ is an oriented path that obeys the filter. For example, observe that the edges at levels 3 and 4 form a directed subpath, and that "zigzagging" happens only at those parts of $\mathbf{P}$ that do not fall into the intervals of the filter.

Definition 27 (Piecewise Symmetric)**.** Assume that $\mathcal{O}$ is a set of (j,k)-paths, and c and d are nonnegative integers. Suppose furthermore that for each $\mathbf{S} \in \mathcal{O}$, there is a (j,k)-path representation $\mathscr{S}$, and a (c,d)-filter $\mathscr{F}_{\mathbf{S}}$ such that the following holds. For every $\mathbf{S} \in \mathcal{O}$ of some length n, and every minimal oriented path $\mathbf{P}$ of height n that obeys the filter $\mathscr{F}_{\mathbf{S}}$, it holds that $\xi(\mathscr{S}, \mathbf{P}) \in \mathcal{O}$. Then $\mathcal{O}$ is (j,k,c,d)-*piecewise symmetric*.

Roughly speaking, an oriented path $\mathbf{P}$ is allowed to modulate only those segments of $\mathscr{S}$ which do not correspond to any delimiters in $\mathscr{F}_{\mathbf{S}}$. Compare Definition 27 with Definition 20, and observe that the only difference is that in the piecewise case, oriented paths must be of a restricted form. Therefore a set that is (j,k)-symmetric is also (j,k,c,d)-piecewise symmetric for any c and d. We simply associate the empty (c,d)-filter with each structure.

Definition 28 (PSBPD)**.** A structure $\mathbf{B}$ has (j,k,c,d)-*piecewise symmetric bounded pathwidth duality* (*(j,k,c,d)-PSBPD*) if there is an obstruction set $\mathcal{O}$ for $\mathbf{B}$ that consists of (j,k)-paths, and in addition, $\mathcal{O}$ is (j,k,c,d)-piecewise symmetric.

Theorem 29. *For a finite structure* $\mathbf{B}$*,* $\mathbf{B}$ *has SBPD (for some parameters) if and only if* $\mathbf{B}$ *has PSBPD (for some parameters).*

We need the corollary of the following lemma in the proof of the above theorem.

Lemma 30. *Let* $\mathbf{P}$ *be a minimal oriented path* $e_0, \ldots, e_{n-1}$ *with the* $(1,2)$*-path representation* $\mathscr{P} = (\mathbf{e_0}, \ldots, \mathbf{e_{n-1}})$*, where* $\mathbf{e_i}$ *is a structure with two*

domain elements and a binary relation that contains the tuple (edge) e_i. Let $\mathbf{Q}$ *be a minimal oriented path $f_0, \ldots, f_m$ with n edge levels. Then the oriented path $\xi(\mathscr{P}, \mathbf{Q})$ is minimal and has the same height as* $\mathbf{P}$.

Proof. It is obvious that $\xi(\mathscr{P}, \mathbf{Q})$ is an oriented path. Furthermore the map that assigns every vertex of $\xi(\mathscr{P}, \mathbf{Q})$ to its original in $\mathbf{P}$ is a homomorphism. It is easy to check that this homomorphism maps the edges of $\xi(\mathscr{P}, \mathbf{Q})$ back to their originals and the level of an edge in $\xi(\mathscr{P}, \mathbf{Q})$ is the same as the level of the original of that edge. Checking the minimality of $\xi(\mathscr{P}, \mathbf{Q})$ is also straightforward. □

Corollary 31. *Let $\mathcal{O}$ be a set of (j,k)-paths, where a (j,k)-representation is fixed for each path. Let $\mathcal{O}'$ be the set that contains all (j,k)-paths that can be obtained from a (j,k)-path in $\mathcal{O}$ by applying some zigzag operator. Then $\mathcal{O}'$ is (j,k)-symmetric.*

Remark: A similar statement holds in the piecewise symmetric case.

Proof. The formal proof is straightforward but laborious. We explain the basic idea. Let $\mathbf{S}'$ be an element of $\mathcal{O}'$. If we can show that applying an arbitrary zigzag operator to $\mathbf{S}'$ yields a (j,k)-path in $\mathcal{O}'$, then we are clearly done. So assume that $\mathbf{S}'$ was obtained from $\mathbf{S} \in \mathcal{O}$ by applying a zigzag operator. The (j,k)-path $\mathbf{S}'$ inherits the (j,k)-representation of $\mathbf{S}$ in a natural way. Then we apply any zigzag operator to $\mathbf{S}'$ to obtain $\mathbf{S}''$, and we need to show that $\mathbf{S}''$ is in $\mathcal{O}'$.

We get from $\mathbf{S}$ to $\mathbf{S}'$ using a zigzag operator and from $\mathbf{S}'$ to $\mathbf{S}''$ another zigzag operator. Using Lemma 30, these two zigzag operators can be replaced by a single zigzag operator to obtain $\mathbf{S}''$ from $\mathbf{S}$ directly. □

Proof of Theorem 29. Let $\mathcal{O}$ be a (j,k)-symmetric obstruction set for $\mathbf{B}$. As observed above, for any c and d, $\mathcal{O}$ is also (j,k,c,d)-piecewise symmetric.

For the converse, let $\mathcal{O}_{ps}$ be a (j,k,c,d)-piecewise symmetric obstruction set. Our goal is to construct a (j',k')-symmetric obstruction set $\mathcal{O}_{sym}$ for $\mathbf{B}$ as follows. For each structure $\mathbf{S} \in \mathcal{O}_{ps}$, let $\mathscr{S} = \mathbf{S}_0 \cup \cdots \cup \mathbf{S}_{n-1}$ be the corresponding (j,k)-path representation. Using the filter for $\mathbf{S}$, we "regroup" $\mathbf{S}_0, \ldots, \mathbf{S}_{n-1}$ to obtain a (j',k')-path representation $\mathscr{S}' = \mathbf{T}_0 \cup \cdots \cup \mathbf{T}_m$ of $\mathbf{S}$. We add each $\mathbf{S}$ together with its new representation to $\mathcal{O}_{sym}$, and also add every structure that is needed to ensure that $\mathcal{O}_{sym}$ is symmetric. Finally, we show that $\mathcal{O}_{sym}$ is a symmetric obstruction set for $\mathbf{B}$. We begin with the regrouping procedure.

Let $\mathbf{S} \in \mathcal{O}_{ps}p$, $\mathscr{S} = \mathbf{S}_0 \cup \cdots \cup \mathbf{S}_{n-1}$ be the corresponding (j,k)-path representation, and $\{[s_1,t_1],[s_2,t_2],\ldots,[s_{c'},t_{c'}]\}$ be the (c,d)-filter $\mathscr{F}_{\mathbf{S}}$. The regrouping procedure is quite pictorial and it is demonstrated in Figure 3.3. We define

$$\mathbf{T}_0 = \bigcup_{\substack{\ell \in [a,b]: \\ [a,b] \in \mathscr{F}_{\mathbf{S}}}} \mathbf{S}_\ell.$$

This places all substructures in $\mathscr{S}$ which correspond to delimiters of $\mathscr{F}_{\mathbf{S}}$ into one big initial structure. Note though that $|T_0| \leq c \cdot d$. Define the *complement*

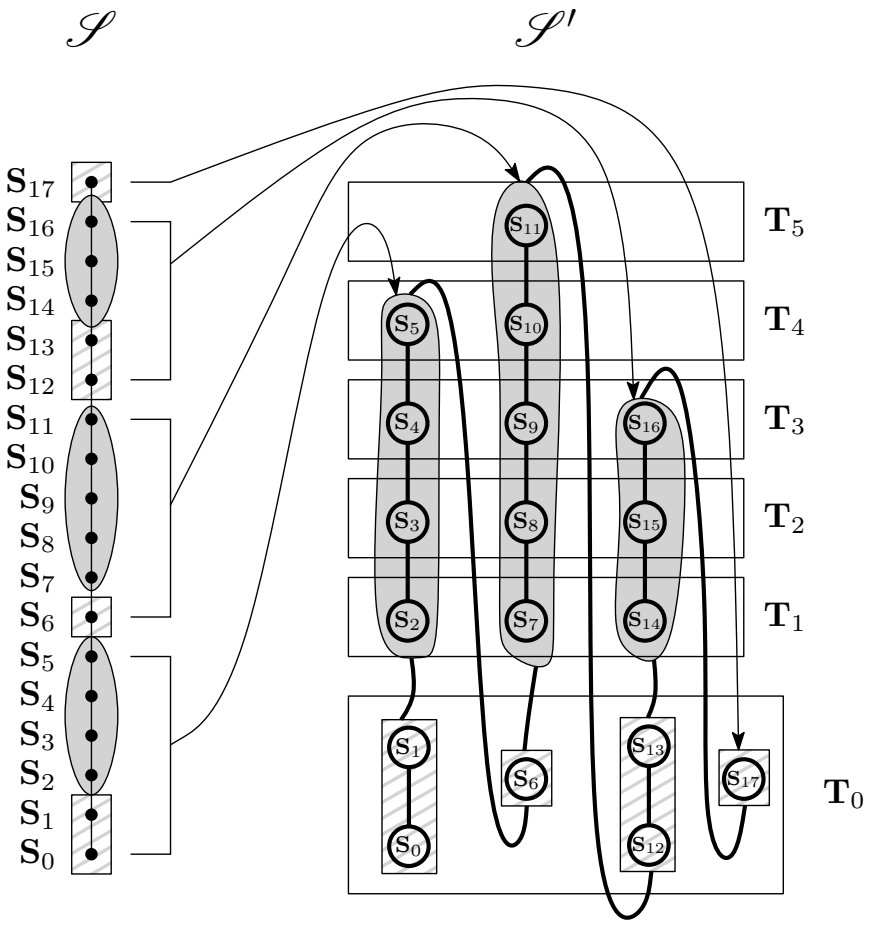


Figure 3.3: An example regrouping for the proof of Theorem 29. The filter $\mathcal{F}_{\mathbf{S}} = \{[0,1],[6],[12,13],[17]\}$. The structures corresponding to the filter are laying inside the rectangles with lines. The complement of the filter is $\bar{\mathcal{F}}_{\mathbf{S}} = \{[2,3,4],[7,8,9,10,11],[14,15,16]\}$. The structure corresponding to $\bar{\mathcal{F}}_{\mathbf{S}}$ lay in the gray ovals. The new (j',k')-path representation $\mathscr{S}'$ of $\mathbf{S}$ is on the right. Notice the following pattern: the segments of $\mathscr{S}$ determined by $\mathscr{F}_{\mathbf{S}}$ are placed next to each other in $\mathscr{S}'$.

of $\mathscr{F}_{\mathbf{S}}$ as

$$\bar{\mathscr{F}}_{\mathbf{S}} = \{[0, s_1 - 1], [t_1 + 1, s_2 - 1], [t_2 + 1, s_3 - 1], \ldots, [t_{c'}, n - 1]\},$$

and set

$$m = \max_{[a,b]\in\bar{\mathscr{F}}_{\mathbf{S}}} (b-a).$$

Intuitively, m is the length of the longest interval in $\mathscr{S}$ between any two delimiters.

We define $\mathbf{T}'_\ell$ as follows. For each interval $[a,b] \in \bar{\mathscr{F}}_{\mathbf{S}}$ take the $(\ell-1)$-th structure $\mathbf{S}_{a+\ell-1}$ in that interval and define $\mathbf{T}'_\ell$ to be the union of these structures. Formally, for every $\ell \in \{1,\dots,m\}$, set

$$\mathbf{T}'_\ell = \bigcup_{\substack{i=a+\ell-1\leq b:\\ [a,b]\in\bar{\mathscr{F}}_{\mathbf{S}}}} \mathbf{S}_i.$$

Observe that $|T'_\ell| \leq k\cdot(c+1)$. We need to ensure property 2 in Definition 17, so we need to place some additional elements into the domains of the $\mathbf{T}'_\ell$.

Let $[x,y] \in \mathscr{F}_{\mathbf{S}}$ and $[z,w] \in \bar{\mathscr{F}}_{\mathbf{S}}$ be such that $z = y+1$. Then the set of elements $S_x \cup \cdots \cup S_w$ is called a *column*. (For the beginning and end of $\mathscr{S}$ a column is defined in the natural "truncated" way.) Because $\mathscr{S}$ is a (j,k)-path representation, it follows from the definition that the intersection of any pair of columns has size at most j. Let $C_1,\dots,C_r$ be an enumeration of all the columns. Set $D = \bigcup_{\ell\neq\ell'} C_\ell \cap C_{\ell'}$ and observe that $|D| \leq j\cdot\binom{r}{2}$. We add D to the domain of $\mathbf{T}_0$, and also to the domain of $\mathbf{T}'_i$ to obtain $\mathbf{T}_i$, $\forall i \in \{1,\dots,m\}$. It is straightforward to see that the new representation $\mathscr{T} = (\mathbf{T}_0,\dots,\mathbf{T}_m)$ satisfies property 2 of Definition 17. Using the remarks about the sizes of the sets, we observe that $\mathscr{T}$ is a (j',k')-path decomposition

of $\mathbf{S}$, where j' and k' are functions of j, k, c and d.

We place all structures $\mathbf{S} \in \mathcal{O}_{ps}$ into $\mathcal{O}_{sym}$ but we associate the new representation with $\mathbf{S}$. For a structure $\mathbf{S} \in \mathcal{O}_{sym}$, we also apply all valid zigzag operators to $\mathbf{S}$ (with respect to the new representation) and add all these structures to $\mathcal{O}_{sym}$. By Corollary 31, $\mathcal{O}_{sym}$ is a (j', k')-symmetric set. We need to establish that $\mathcal{O}_{sym}$ is an obstruction set. Because $\mathcal{O}_{ps} \subseteq \mathcal{O}_{sym}$, it is sufficient to show that no structure in $\mathcal{O}_{sym}$ maps to $\mathbf{B}$. To do that we show that for any structure in $\mathcal{O}_{sym}$, there is a structure in $\mathcal{O}_{ps}$ that homomorphically maps to it.

Giving a formal proof would lead to unnecessary notational complications and therefore we give an example that is easier to follow and straightforward to generalize. The example is represented in Figure 3.4. Let $\mathbf{S} \in \mathcal{O}_{sym}$ such that $\mathbf{S}$ is also in $\mathcal{O}_{ps}$. Assume that the (j', k')-representation of $\mathbf{S}$ in $\mathcal{O}_{sym}$ is $\mathscr{T}$. We consider $\xi(\mathscr{T}, \mathbf{P})$ for some minimal oriented path and show how to find a minimal oriented path $\mathbf{Q}$ such that $\xi(\mathscr{S}, \mathbf{Q}) \to \xi(\mathscr{T}, \mathbf{P})$. To construct $\mathbf{Q}$, we make a copy of $\mathbf{P}$ aligned with $\mathbf{S}_0, \mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3, \mathbf{S}_4$ in $\mathscr{S}$. This is represented by the dashed lines in Figure 3.4. We also make a copy of $\mathbf{P}$ aligned with $\mathbf{S}_5, \mathbf{S}_6, \mathbf{S}_7, \mathbf{S}_8, \mathbf{S}_9$. This is represented with the dash dotted lines. Note that the resulting minimal oriented path respects the delimiters, i.e. the zigzag operator will not "zigzag" $\mathbf{S}_0$ and $\mathbf{S}_5$. (In general, we never need to "zigzag" structures that were placed into $\mathbf{T}_0$, i.e. the structures that correspond to the delimiters, because $\mathbf{P}$ is minimal.)

In $\xi(\mathscr{T}, \mathbf{P})$ we denote the copies of the $\mathbf{S}_i$ with $\widetilde{\mathbf{S}}_i$ and primed $\widetilde{\mathbf{S}}_i$. Using

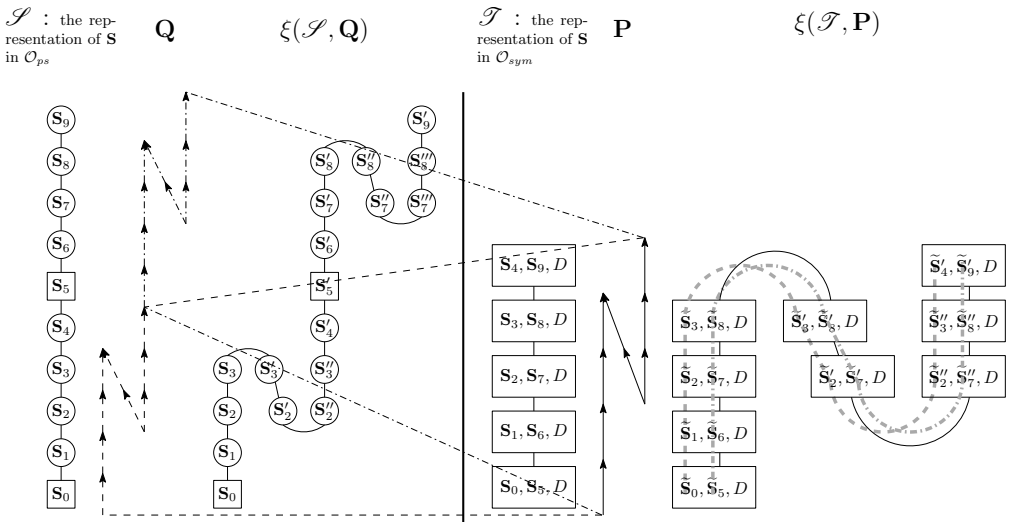


Figure 3.4: Example in the proof of Theorem 29. See the proof for details.

the definition of the zigzag operator, it follows that the function f that maps an element of $\mathbf{S}_0 \cup \mathbf{S}_1 \cup \mathbf{S}_2 \cup \mathbf{S}_3 \cup \mathbf{S}'_3 \cup \mathbf{S}'_2 \cup \mathbf{S}''_3 \cup \mathbf{S}'_4$ in $\xi(\mathscr{S}, \mathbf{Q})$ to the corresponding element in $\widetilde{\mathbf{S}}_0 \cup \widetilde{\mathbf{S}}_1 \cup \widetilde{\mathbf{S}}_2 \cup \widetilde{\mathbf{S}}_3 \cup \widetilde{\mathbf{S}}'_3 \cup \widetilde{\mathbf{S}}'_2 \cup \widetilde{\mathbf{S}}''_3 \cup \widetilde{\mathbf{S}}'_4$ is a homomorphism. We similarly define a homomorphism h from $\mathbf{S}'_5 \cup \mathbf{S}'_6 \cup \mathbf{S}'_7 \cup \mathbf{S}'_8 \cup \mathbf{S}''_8 \cup \mathbf{S}''_7 \cup \mathbf{S}'''_7 \cup \mathbf{S}'''_8 \cup \mathbf{S}'_9$ in $\xi(\mathscr{S}, \mathbf{Q})$ to $\widetilde{\mathbf{S}}'_5 \cup \widetilde{\mathbf{S}}'_6 \cup \widetilde{\mathbf{S}}'_7 \cup \widetilde{\mathbf{S}}'_8 \cup \widetilde{\mathbf{S}}''_8 \cup \widetilde{\mathbf{S}}''_7 \cup \widetilde{\mathbf{S}}'''_7 \cup \widetilde{\mathbf{S}}'''_8 \cup \widetilde{\mathbf{S}}'_9$ in $\xi(\mathscr{T}, \mathbf{P})$.

If for every x that is in the domain of both f and h, $f(x) = h(x)$, then we can combine f and h to get the desired homomorphism. Assume for example that the element x appears in $\mathbf{S}_2$ and also in $\mathbf{S}'''_8$ in $\xi(\mathscr{S}, \mathbf{Q})$, and suppose that $f(x) = y$ and $h(x) = y'$. Let the originals of y and y' be z and z' in $\mathscr{T}$, respectively. We also find z and z' in $\mathbf{S}_2$ and $\mathbf{S}_8$ in $\mathscr{S}$. Observe that x in $\mathbf{S}_2$ in $\xi(\mathscr{S}, \mathbf{Q})$ is a copy of z and x in $\mathbf{S}'''_8$ in $\xi(\mathscr{S}, \mathbf{Q})$ is a copy of z'. If $z \neq z'$ (in $\mathscr{S}$) then x could not appear both in $\mathbf{S}_2$ and $\mathbf{S}'''_8$ by the definition of the zigzag operator. Therefore $z = z'$, $z \in D$, and by definition, z is in every bag

of $\mathscr{T}$. The elements y and y' are copies of z, and because z appears in every "bag" of $\mathscr{T}$, all copies of z in $\xi(\mathscr{T}, \mathbf{P})$ are identified to be the same element. In particular, $f(x) = y = y' = h(x)$. □

3.4 Applications

3.4.1 Datalog + Maltsev ⇒ Symmetric Datalog

Using SBPD, we give a short and simple re-proof of the main result of [31]:

Theorem 32 ([31])**.** *Let* $\mathbf{B}$ *be a finite core structure. If* $\mathbf{B}$ *is invariant under a Maltsev operation and* co-CSP($\mathbf{B}$) *is definable in Datalog, then* co-CSP($\mathbf{B}$) *is definable in symmetric Datalog (and therefore* CSP($\mathbf{B}$) *is in* L *by [38]).*

We only need to show that if co-CSP($\mathbf{B}$) is in linear Datalog and $\mathbf{B}$ is preserved by a Maltsev operation, then co-CSP($\mathbf{B}$) is in symmetric Datalog. The "jump" from Datalog to linear Datalog essentially follows from already established results, as observed in [31]. For completeness' sake, we outline the argument but we omit the technical definitions.

Lemma 33 ([31])**.** *Let* $\mathbf{B}$ *be a core structure such that the variety corresponding to the algebra associated with* $\mathbf{B}$ *(*$\mathcal{V}(\mathbb{A}(\mathbf{B}))$*) is congruence-permutable and* co-CSP($\mathbf{B}$) *is in Datalog. Then* $\mathcal{V}(\mathbb{A}(\mathbf{B}))$ *is arithmetical.*

Lemma 34 ([74])**.** *A variety is arithmetical if and only if it has a majority term and a Maltsev term.*

Therefore if co-CSP($\mathbf{B}$) is in Datalog and $\mathbf{B}$ is invariant under a Maltsev operation (i.e. is congruence-permutable), then $\mathbf{B}$ is also closed under a majority operation. By [30], invariance of $\mathbf{B}$ under a majority operation implies that co-CSP($\mathbf{B}$) is in linear Datalog.

To re-prove Theorem 32, we show the following lemma using an SBPD argument.

Lemma 35. *If* co-CSP($\mathbf{B}$) *is definable by a linear Datalog program and* $\mathbf{B}$ *is invariant under a Maltsev operation* m*, then* co-CSP($\mathbf{B}$) *is definable by a symmetric Datalog program.*

To get ready for the proof of Lemma 35, we define an N of size s as an oriented path that consists of s forward edges, followed by s backward edges, followed by another s forward edges. Proposition 36 is easy to prove, and the Maltsev properties are used in Lemma 37.

Proposition 36. *A minimal oriented path is either a directed path, or it contains a subpath which is an* N*.*

Lemma 37. *Let* $\mathbf{B}$ *be a structure invariant under a Maltsev operation* m*,* $\mathbf{S}$ *be a* (j,k)*-path with a* (j,k)*-representation* $\mathscr{S} = (\mathbf{S}_0, \ldots, \mathbf{S}_{n-1})$*, and* $\mathbf{P} = e_0, \ldots, e_q$ *be a minimal oriented path of height* n*. If* $\xi(\mathscr{S}, \mathbf{P}) \to \mathbf{B}$*, then* $\mathbf{S} \to \mathbf{B}$*.*

Proof. Using Proposition 36, there is a t such that $\mathbf{Q} = e_t, e_{t+1}, \ldots, e_{t+(3s-1)}$ is an N of size s in $\mathbf{P}$. Assume that the first and last vertices of $\mathbf{Q}$ are v and

w, respectively. Let $\mathbf{P}'$ be the oriented path obtained from $\mathbf{P}$ by removing $\mathbf{Q}$, and adding a directed path $\mathbf{Q}' = f_t, f_{t+1}, \ldots, f_{t+(s-1)}$ of length s from v to w. We claim that there is a homomorphism γ from $\xi(\mathscr{S}, \mathbf{P}')$ to $\mathbf{B}$. Once this is established, repeating the argument sufficiently many times clearly yields that $\mathbf{S} \to \mathbf{B}$.

Let $\xi(\mathscr{S}, \mathbf{P}) = (\mathbf{S}_{e_0}, \ldots, \mathbf{S}_{e_q})$, and $\varphi_{e_0}, \ldots, \varphi_{e_q}$ be the corresponding isomorphisms (recall the zigzag operator definition in Section 3.2.4). Similarly, let $\xi(\mathscr{S}, \mathbf{P}') = (\mathbf{S}_{f_0}, \ldots, \mathbf{S}_{f_{q-2s}})$, and $\psi_{f_0}, \ldots, \psi_{f_{q-2s}}$ be the corresponding isomorphisms. Because $\mathbf{S}_{[e_0, e_{t-1}]}$ and $\mathbf{S}_{[e_{t+3s}, e_q]}$ are isomorphic to $\mathbf{S}_{[f_0, f_{t-1}]}$ and $\mathbf{S}_{[f_{t+s}, f_{q-2s}]}$, respectively, γ for elements in $S_{[f_0, f_{t-1}]} \cup S_{[f_{t+s}, e_{q-2s}]}$ is defined in the natural way. It remains to define γ for every $d \in S_{[f_t, f_{t+(s-1)}]}$.

Assume that $d \in S_{f_{t+\ell}}$ for some $\ell \in \{0, \ldots, s-1\}$. Find the original of d in $\mathbf{S}$ and let it be d_o, i.e. $d_o = \psi_{f_{t+\ell}}(d)$. Then we find the three copies d_1, d_2, d_3 of d_o in $\mathbf{S}_{[f_t, f_{t+(3s-1)}]}$. That is, first we find the three edges $e_{\ell_1}, e_{\ell_2}, e_{\ell_3}$ of $\mathbf{Q}$ which have the same level as $f_{t+\ell}$ (all levels are with respect to $\mathbf{P}$ and $\mathbf{P}'$). Then $d_i = \varphi^{-1}_{e_{\ell_i}}(d_o)$, $i \in [3]$. We define $\gamma(d) = m(d_1, d_2, d_3)$. By the Maltsev properties of m, γ is well-defined. As $\mathbf{B}$ is invariant under m, $\xi(\mathscr{S}, \mathbf{P}') \xrightarrow{\gamma} \mathbf{B}$. □

Proof of Lemma 35. If co-CSP($\mathbf{B}$) can be defined by a linear (j,k)-Datalog program, then there is an obstruction set $\mathcal{O}$ for $\mathbf{B}$ in which every structure is a (j,k)-path by [27]. We construct a symmetric obstruction set $\mathcal{O}_{sym}$ for $\mathbf{B}$ as follows. For every (j,k)-path $\mathbf{S}$ with a (j,k)-representation $\mathscr{S} = \mathbf{S}_0, \ldots, \mathbf{S}_{n-1}$ in $\mathcal{O}$ and for every minimal oriented path $\mathbf{P}$ of height n, place $\xi(\mathscr{S}, \mathbf{P})$ into

$\mathcal{O}_{sym}$. By Corollary 31, $\mathcal{O}_{sym}$ is (j,k)-symmetric.

Observe that $\mathcal{O} \subseteq \mathcal{O}_{sym}$, so it remains to show that no element of $\mathcal{O}_{sym}$ maps to $\mathbf{B}$. But if $\mathbf{T} \in \mathcal{O}_{sym}$, then $\mathbf{T} = \xi(\mathscr{S}, \mathbf{P})$ for some $\mathbf{S} \in \mathcal{O}$ and $\mathbf{P}$. By Lemma 37, if $\xi(\mathscr{S}, \mathbf{P}) \to \mathbf{B}$, then $\mathbf{S} \to \mathbf{B}$. This contradicts the assumption that $\mathcal{O}$ is an obstruction set for $\mathbf{B}$. □

3.4.2 A class of oriented paths for which the CSP is in L, and a class for which the CSP is NL-complete

In this section we define a class $\mathcal{C}$ of oriented paths such that if $\mathbf{B} \in \mathcal{C}$ then co-CSP($\mathbf{B}$) is in symmetric Datalog. Our strategy is to find an obstruction set $\mathcal{O}$ for $\mathbf{B} \in \mathcal{C}$, and then to show that our obstruction set is piecewise symmetric. We need some notation.

We say that a directed path is *forward* to mean that its first and last vertices are the vertices with indegree zero and outdegree zero, respectively. Let $\mathbf{P}$ be an oriented path with first vertex v and last vertex w. Then the *reverse* of $\mathbf{P}$, denoted with $\bar{\mathbf{P}}$, is a copy of the oriented path $\mathbf{P}$ in the reverse direction, i.e. the first vertex of $\bar{\mathbf{P}}$ is a copy of w and its last vertex is a copy of v. Let $\mathbf{Q}$ be another oriented path. The *concatenation* of $\mathbf{P}$ and $\mathbf{Q}$ is the oriented path $\mathbf{PQ}$ in which the last vertex of $\mathbf{P}$ is identified with the first vertex of $\mathbf{Q}$. For a nonnegative integer r, $\mathbf{P}^r$ denotes $\mathbf{P}_1\mathbf{P}_2\cdots\mathbf{P}_r$, where the $\mathbf{P}_\ell$ are disjoint copies of $\mathbf{P}$. Given two vertices v and w, we denote the presence of an edge from v to w with $v \to w$.

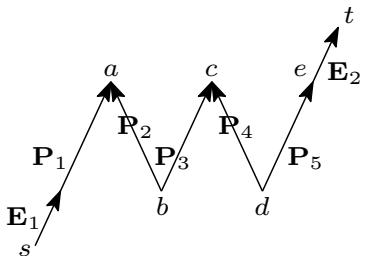


Figure 3.5: 2-wave in the proof of Theorem 39.

Definition 38 (Wave)**.** If an oriented path $\mathbf{Q}$ can be expressed as $\mathbf{E}_1(\mathbf{P}\bar{\mathbf{P}})^r\mathbf{P}\mathbf{E}_2$, where $\mathbf{E}_i$ ($i \in [2]$) denotes the forward directed path that is a single edge, $\mathbf{P}$ is a forward directed path of length ℓ, and $r \geq 0$, then $\mathbf{Q}$ is called an *ℓ-wave*. A 2-wave is shown in Figure 3.7, *1*.

Theorem 39. *Let* $\mathbf{Q}$ *be a wave. Then* $\mathbf{Q}$ *has PSBPD,* co-CSP($\mathbf{Q}$) *is definable in symmetric Datalog, and* CSP($\mathbf{Q}$) *is in* L.

Proof. We prove the case when $\mathbf{Q}$ is an ℓ-wave for $\ell = 2$. For larger ℓ-s, the proof generalizes in a straightforward manner. Let $\mathbf{P}$ be a directed path of length h, $\mathbf{P}_1, \mathbf{P}_3, \mathbf{P}_5$ be disjoint copies of $\mathbf{P}$, and $\mathbf{P}_2, \mathbf{P}_4$ be copies of the reverse of $\mathbf{P}$. Let $\mathbf{E}_1$ and $\mathbf{E}_2$ be forward edges. Assume the 2-wave $\mathbf{Q}$ is $\mathbf{E}_1\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3\mathbf{P}_4\mathbf{P}_5\mathbf{E}_2$ (Figure 3.5). We will provide a piecewise symmetric obstruction set $\mathcal{O}_{ps}$ for $\mathbf{Q}$, such that every element of $\mathcal{O}_{ps}$ is an oriented path. To do this, first we observe that by [53], $\mathbf{Q}$ has path duality, i.e. we can assume that the set $\mathcal{O}$ of all oriented paths that do not homomorphically map to $\mathbf{Q}$ form an obstruction set for $\mathbf{Q}$. To construct $\mathcal{O}_{ps}$ from $\mathcal{O}$, we will place certain elements of $\mathcal{O}$ into $\mathcal{O}_{ps}$ such that $\mathcal{O}_{ps}$ is still an obstruction set

for $\mathbf{Q}$.

We begin with some simple observations. Any oriented path that has height at most $h+1$ maps to $\mathbf{Q}$, so these oriented paths can be neither in $\mathcal{O}$ nor in $\mathcal{O}_{ps}$. Any oriented path that has height strictly larger than $h+2$ obviously does not map to $\mathbf{Q}$, so all such paths are in $\mathcal{O}$ and we also place these paths into $\mathcal{O}_{ps}$. Assume that $\mathbf{P} \in \mathcal{O}$ has height exactly $h+2$. It is easy to see that if $\mathbf{P}$ is not minimal, then it contains a minimal subpath that does not map to $\mathbf{Q}$. Therefore, it is sufficient to place only those oriented paths from $\mathcal{O}$ of height $h+2$ into $\mathcal{O}_{ps}$ which are minimal.

Let $\mathbf{P} \in \mathcal{O}_{ps}$ of height $h+2$ (then $\mathbf{P}$ is minimal). Intuitively, any attempt to homomorphically map the vertices of $\mathbf{P}$ to $\mathbf{Q}$ starting by first mapping the first vertex of $\mathbf{P}$ to the first vertex of $\mathbf{Q}$ and then progressively finding the image of the vertices of $\mathbf{P}$ from left to right would get stuck at a or c.

Formally, assume that the vertices of $\mathbf{P}$ are $v_1, \ldots, v_n$. Let $\mathbf{P}_{[i]}$ denote the subpath of $\mathbf{P}$ on the first i vertices. Choose i to be the largest index such that $\mathbf{P}_{[i]} \xrightarrow{\varphi} \mathbf{Q}$ and $\varphi(v_1) = s$. Then φ cannot be extended to v_{i+1} for one of the following reasons. Clearly, φ must map v_i to a source or a sink other than s or t, i.e. to a,b,c or d. Furthermore, we can assume that v_i is not mapped to b or d. This is because if v_i is mapped to b or d, then $\text{level}(v_i) = 1$, so the edge between v_i and v_{i+1} is from v_i to v_{i+1}, and therefore φ can be extended. So we can assume that v_i is mapped to a or c. Because we cannot extend φ, v_{i+1} must be at level $\ell + 2$, so it must be that v_{i+1} is the last vertex v_n of $\mathbf{P}$. Because $\mathbf{P} \not\to \mathbf{Q}$, $\mathbf{P}_{[n-1]}$ must be an oriented path

such that any homomorphism φ from $\mathbf{P}_{[n-1]}$ to $\mathbf{Q}$ such that $\varphi(v_1) = s$ maps v_{n-1} to a or c but not to e.

We assume first that any homomorphism φ from $\mathbf{P}_{[n-1]}$ to $\mathbf{Q}$ maps v_{n-1} to a. We follow the vertices of $\mathbf{P}_{[n-1]}$ from left to right. Let w_a be the first vertex that is at level $h+1$. If there is a vertex to the right of w_a at level 1, then because $\mathbf{P}_{[n-1]}$ will have to reach level $h+1$ again, we will be able to map v_{n-1} to c, and that is not possible by assumption. So $\mathbf{P}$ must have the following form (Form 1): $(w_1 \to w_2)\mathbf{X}(w_3 \to w_4)\mathbf{Y}(w_5 \to w_6)$, where $\mathbf{X}$ is any oriented path of height $h-1$ with first vertex at the bottom and last vertex at the top level of $\mathbf{X}$, and $\mathbf{Y}$ is any oriented path of height $h-1$ with both its first and last vertices being in the top level of $\mathbf{Y}$. See Figure 3.6, left.

For the second case, we assume that $\mathbf{P}_{[n-1]}$ is such that v_{n-1} can be mapped to c. Again, we follow the vertices of $\mathbf{P}_{[n-1]}$ from left to right. Let w_a be the first vertex that is at level $h+1$. We must have a vertex going back to level 1 (otherwise we could not "pass" b and could not map v_{n-1} to c). Let w_b be the first such vertex. We will have to go back to level $h+1$ again, so let w_c be the first vertex at that level. Finally, we cannot go back to level 1 again, since then the last vertex of $\mathbf{P}_{[n-1]}$ can be mapped to e. We can "go down" to at most level 2 of $\mathbf{P}_{[n-1]}$. So $\mathbf{P}$ must have the form (Form 2) $(w_1 \to w_2)\mathbf{X}(w_3 \to w_4)\mathbf{Y}(w_5 \leftarrow w_6)\mathbf{Z}(w_7 \to w_8)\mathbf{W}(w_9 \to w_{10})$, where $\mathbf{X}$ ($\mathbf{Z}$) is any oriented path of height $h-1$ with first vertex at the bottom and last vertex at the top level of $\mathbf{X}$ ($\mathbf{Z}$), $\mathbf{Y}$ is any oriented path of height $h-1$

with first vertex at the top and last vertex at the bottom level of $\mathbf{Y}$, and $\mathbf{W}$ is any oriented path of height $h-1$ with both its first and last vertices being in the top level of $\mathbf{W}$. See Figure 3.6, right.

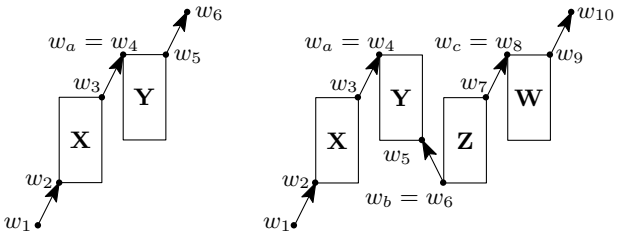


Figure 3.6: Obstructions of height $h+2$ for a 2-wave.

Because $\mathcal{O}_{ps} \subseteq \mathcal{O}$ and for any structure $\mathbf{S} \in \mathcal{O}$, there is a structure $\mathbf{S}' \in \mathcal{O}_{ps}$ such that $\mathbf{S}' \to \mathbf{S}$, $\mathcal{O}_{ps}$ is an obstruction set for $\mathbf{Q}$.

It remains to show that $\mathcal{O}_{ps}$ is piecewise symmetric. Let $\mathbf{S}$ be an oriented path of height more than $h+2$, and assume the vertex set of $\mathbf{S}$ is $v_1, \ldots, v_n$. We need to define a representation $\mathscr{S}$, and a filter $\mathscr{F}_{\mathbf{S}}$ for $\mathbf{S}$. The representation $(\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_{n-2})$ is $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$ (width $(1, 2)$). The filter $\mathscr{F}_{\mathbf{S}}$ is the empty filter. Note that if we apply a zigzag operation to $\mathbf{S}$, we get an oriented path of the same height as $\mathbf{S}$, so $\mathcal{O}_{ps}$ is closed under zigzagging of obstructions of height greater than $h+2$.

Let $\mathbf{S}$ be an oriented path of height $h+2$ of Form 1, and assume the vertex set of $\mathbf{S}$ is $v_1, \ldots, v_n$. The representation $\mathscr{S} = (\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_{n-2})$ is constructed as in the previous paragraph. We specify $\mathscr{F}_{\mathbf{S}}$ to be the following $(3, 6)$-filter. Assume that the edge (w_3, w_4) is structure $\mathbf{S}_i$. Then $\mathcal{F}_{\mathbf{S}} = \{[0, 0], [i, i], [n-2, n-2]\}$. Using the definitions it is easy to see that if $\mathbf{P}$ obeys

the filter $\mathscr{F}_{\mathbf{S}}$, then $\xi(\mathscr{S}, \mathbf{P})$ is also an oriented path of Form 1. Therefore $\mathcal{O}_{ps}$ is closed under zigzagging of obstructions of Form 1. Obstructions of Form 2 can be handled similarly. □

We state the following generalization of waves.

Definition 40 (Staircase)**.** A *monotone wave* is an oriented path of the form $(\bar{\mathbf{P}}\mathbf{P})^r\bar{\mathbf{P}}$, where $\mathbf{P}$ is a forward directed path and $r \geq 0$. We call the vertices of a monotone wave in the topmost level *peaks*, and the vertices in the bottommost level *troughs*.

If a minimal oriented path $\mathbf{Q}$ can be expressed as

$$\mathbf{P}_1\mathbf{W}_1\mathbf{P}_2\mathbf{W}_2 \dots \mathbf{P}_{n-1}\mathbf{W}_{n-1}\mathbf{P}_n,$$

where $\mathbf{P}_1, \dots, \mathbf{P}_n$ are forward directed paths, $\mathbf{W}_1, \dots, \mathbf{W}_{n-1}$ are monotone waves, and for any $i \in [n-1]$, the troughs of $\mathbf{W}_i$ are in a level strictly below the level of the troughs of $\mathbf{W}_{i+1}$, and also, the peaks of $\mathbf{W}_i$ are in a level strictly below the level of the peaks of $\mathbf{W}_{i+1}$, then $\mathbf{Q}$ is called a *staircase*. An example is given in Figure 3.7, *2*.

Theorem 41. *Let* $\mathbf{Q}$ *be a staircase. Then* $\mathbf{Q}$ *has PSBPD,* co-CSP($\mathbf{Q}$) *is definable in symmetric Datalog, and* CSP($\mathbf{Q}$) *is in* L.

Proof. Assume that the height of p$\mathbf{Q}$ is h. As for waves, we use [53] to conclude that $\mathbf{Q}$ has path duality. We will construct a piecewise symmetric obstruction set $\mathcal{O}_{ps}$ for $\mathbf{Q}$ by placing three classes of oriented paths into $\mathcal{O}_{ps}$.

First, $\mathcal{O}_{ps}$ contains all oriented paths which have height strictly greater than h. These oriented paths obviously do not map to $\mathbf{Q}$.

The next class of oriented paths we place into $\mathcal{O}_{ps}$ are those which have height precisely h. Recall that $\mathbf{Q}$ consists of waves patched together with directed paths in between. Let the wave subpaths of $\mathbf{Q}$ be $\mathbf{W}_1, \ldots, \mathbf{W}_n$, from left to right. For each $\mathbf{W}_i$, we construct a class of oriented paths. Assume that $\mathbf{W}_i$ has height h_i and let $\mathcal{O}_i$ be the set of minimal oriented paths of height h_i which do not map to $\mathbf{W}_i$. For each $\mathbf{R} \in \mathcal{O}_i$, we construct $\mathbf{C} = \mathbf{B}_1\mathbf{R}\mathbf{B}_2$, where $\mathbf{B}_1$ and $\mathbf{B}_2$ are oriented paths (possibly empty) such that $\mathbf{C}$ has height h, and the level of $\mathbf{R}$ in $\mathbf{C}$ matches the level of $\mathbf{W}_i$. Observe that there cannot be a homomorphism from $\mathbf{C}$ to $\mathbf{Q}$. We place all such constructed $\mathbf{C}$ into $\mathcal{O}_{ps}$.

Let ℓ be the length of the longest directed subpath of $\mathbf{Q}$. The third class of oriented paths are those that have height h', where $\ell < h' < h$. For every such h', we produce a set of obstructions. (*Remark:* we set $\ell < h'$ because any oriented path of length ℓ or less maps to $\mathbf{Q}$.)

Assume inductively (the base case is trivial) that we already have a piecewise symmetric obstruction set for every staircase of height strictly less than h. Consider every subpath $\mathbf{Q}_1, \ldots, \mathbf{Q}_m$ of $\mathbf{Q}$ of height h'. Notice that core($\mathbf{Q}_i$) is a staircase which is not a directed path. By the inductive hypothesis we have a piecewise symmetric obstruction set $\mathcal{U}_i$ for $\mathbf{Q}_i$. We keep only those oriented paths in $\mathcal{U}_i$ which have height at most h'; observe that $\mathcal{U}_i \neq \emptyset$. Construct $\mathbf{D} = \mathbf{B}_1\mathbf{T}_1 \cdots \mathbf{B}_m\mathbf{T}_m\mathbf{B}_{m+1}$, where $(\mathbf{T}_1, \ldots, \mathbf{T}_m) \in \mathcal{U}_1 \times \cdots \times \mathcal{U}_m$

and the $\mathbf{B}_j$ are arbitrary oriented paths such that the height of $\mathbf{D}$ is h'. Place all these $\mathbf{D}$-s into $\mathcal{O}_{ps}$.

Notice that $\mathbf{D}$ does not map to $\mathbf{Q}$ for the following reason. Assume for contradiction that $\mathbf{D}$ maps to a subpath $\mathbf{S}$ of $\mathbf{Q}$. Then $\mathbf{D}$ also maps to the core of $\mathbf{S}$ which is a staircase. But by construction $\mathbf{D}$ contains a subpath that does not map to $\mathbf{S}$.

We show that $\mathcal{O}_{ps}$ is an obstruction set for $\mathbf{Q}$. If a structure $\mathbf{Z} \in \mathcal{O}_{ps}$ homomorphically maps to an input structure $\mathbf{A}$, then obviously, there cannot be a homomorphism from $\mathbf{A}$ to $\mathbf{Q}$. Assume for contradiction that no structure in $\mathcal{O}_{ps}$ maps to $\mathbf{A}$ but $\mathbf{A}$ does not map to $\mathbf{Q}$. Then $\mathcal{O}$ contains an oriented path $\mathbf{P}$ that maps to $\mathbf{A}$. So if we show the following claim then we are done.

Claim. *For any oriented path $\mathbf{P}$ that does not homomorphically map to $\mathbf{Q}$, there is an oriented path $\mathbf{Z} \in \mathcal{O}_{ps}$ that homomorphically maps to $\mathbf{P}$.*

Proof of Claim. Assume that $\mathbf{P}$ has height precisely h. We show that there exists $\mathbf{Z} \in \mathcal{O}_{ps}$ of height h such that $\mathbf{Z} \to \mathbf{P}$. Assume for contradiction that none of the oriented paths of height h in $\mathcal{O}_{ps}$ map to $\mathbf{P}$. As before, let $\mathbf{W}_1, \ldots, \mathbf{W}_n$ be the wave segments of $\mathbf{Q}$, from left to right, and assume without loss of generality that none of the $\mathbf{W}_i$ is a directed path. Let the initial and final vertices of $\mathbf{W}_i$ be a_i and b_i respectively, $i \in [n]$. For each $i \in [n]$, find the minimal oriented subpaths of $\mathbf{P}$ whose initial vertices have the same level as a_i, and final vertices have the same level as b_i, or vice versa (note that because of the structure of $\mathbf{Q}$, no such oriented path could contain

another as a subpath, however, these oriented paths could overlap). For any such subpath $\mathbf{R}$ of $\mathbf{P}$ associated with $\mathbf{W}_i$, map the lowest vertex of $\mathbf{R}$ to a_i, and the highest vertex of $\mathbf{R}$ to b_i. *Remark 1: In fact there is no other choice.* The rest of the vertices of $\mathbf{R}$ can be mapped to $\mathbf{Q}$ as follows. If $\mathbf{R}$ does not map to $\mathbf{W}_i$ with first and last vertices matched then by definition, $\mathbf{P}$ is in $\mathcal{O}_{ps}$ and we have a contradiction. Therefore let the homomorphism for $\mathbf{R}$ be $\varphi_{\mathbf{R}}$. *Remark 2: Also observe that $\varphi_{\mathbf{R}}$ maps the inner vertices of $\mathbf{R}$ to vertices of the staircase which are between a_i and b_i.*

We show that the partial homomorphisms $\varphi_{\mathbf{R}}$ map the same vertex of $\mathbf{P}$ to the same vertex in $\mathbf{Q}$, and furthermore we can also map those vertices of $\mathbf{P}$ to an element of $\mathbf{Q}$ that are not mapped anywhere by the $\varphi_{\mathbf{R}}$. This way we obtain a homomorphism from $\mathbf{P}$ to $\mathbf{Q}$ and this would be a contradiction.

First, any vertex v is assigned to a vertex of $\mathbf{Q}$ by at most two homomorphisms which correspond to consecutive wave segments of $\mathbf{Q}$. This is because in $\mathbf{Q}$, $\mathbf{W}_i$ and $\mathbf{W}_j$ are disjoint unless $j = i + 1$. Using Remarks 1 and 2, we can see that if a vertex v of $\mathbf{P}$ is in the domain of two "non-consecutive" homomorphisms, then because those homomorphisms could not agree on where to map v, it is not possible that $\mathbf{P} \to \mathbf{Q}$. This is a contradiction.

Let $\varphi_{\mathbf{R}_1}$ and $\varphi_{\mathbf{R}_2}$ (assume without loss of generality that $\mathbf{R}_1$ and $\mathbf{R}_2$ correspond to $\mathbf{W}_1$ and $\mathbf{W}_2$, respectively) be two partial homomorphisms such that their domains overlap. Then the markers a_1, b_1, a_2, b_2 appear in the order a_1, a_2, b_1, b_2 when traversing $\mathbf{P}$ from left to right. The vertices that are in the domain of both homomorphisms are the ones from a_2 to b_1. By the

choice of a_1, b_1, a_2, b_2, the segment of $\mathbf{P}$ from a_2 to b_1 is a minimal oriented path. Checking the images of the vertices going back from b_1 to a_2 under the map $\varphi_{\mathbf{R}_1}$, we see that these vertices are mapped to the rightmost directed path segment of $\mathbf{W}_1$. Similarly, the image of these vertices under $\varphi_{\mathbf{R}_1}$ is the leftmost directed path of the $\mathbf{W}_2$. That is, the two homomorphisms coincide for the vertices from a_2 to b_1.

Furthermore, some vertices of $\mathbf{P}$ are not in the domain of any partial homomorphisms. Consider the two minimal oriented paths $\mathbf{S}$ and $\mathbf{S}'$ on the two sides of such a maximal continuous sequence of vertices in $\mathbf{P}$. There are two cases. First, assume that $\mathbf{S}$ and $\mathbf{S}'$ both correspond to the same $\mathbf{W}_i$. Let the markers for $\mathbf{S}$ be a and b and the markers for $\mathbf{S}'$ be a' and b'. Then following $\mathbf{P}$ from left to right, the markers appear in the order a, b, b', a'. The images of the vertices from b to b' are not defined. (Observe that b and b' are mapped to the same vertex.) Consider the last directed path segment of $\mathbf{W}_i$ together with the first directed path segment of $\mathbf{W}_{i+1}$ (or just the last edges of $\mathbf{Q}$ if $i = n$). Observe that the vertices from b to b' can be mapped to this directed path. The case when $\mathbf{S}$ and $\mathbf{S}'$ correspond to different waves of $\mathbf{Q}$ is handled similarly.

Suppose lastly that $\mathbf{P}$ has height $h' < h$. Because $\mathbf{P}$ does not map to any of the subpaths of $\mathbf{Q}$ of height h', for each subpath $\mathbf{Q}_1, \ldots, \mathbf{Q}_m$ of $\mathbf{Q}$ of height h', $\mathbf{P}$ contains a subpath $\mathbf{S}_i$ such that $\mathbf{S}_i \not\to \mathbf{Q}_i$, $i \in [m]$. If $\mathbf{S}_i \not\to \mathbf{Q}_i$ then $\mathbf{S}_i \not\to \mathit{core}(\mathbf{Q}_i)$. Recall that core$(\mathbf{Q}_i)$ is a staircase and by definition, $\mathcal{U}_i$ contains an oriented path $\mathbf{S}'_i$ such that $\mathbf{S}'_i \to \mathbf{S}_i$. It is clear that we can choose

oriented paths $\mathbf{B}_1, \ldots, \mathbf{B}_{m+1}$ such that $\mathbf{B}_1\mathbf{S}'_1\mathbf{B}_2 \ldots \mathbf{B}_m\mathbf{S}'_m\mathbf{B}_{m+1} \to \mathbf{P}$. □

Finally, it is not hard to see from the construction how to associate filters with the elements of $\mathcal{O}_{ps}$ to establish that $\mathcal{O}_{ps}$ is piecewise symmetric. □

We also give a large class of oriented paths for which the CSP is NL-complete. We need the following lemma and proposition to prove Theorem 44.

Lemma 42 ([49])**.** *Let* $\mathbf{P}_1$ *and* $\mathbf{P}_2$ *be two minimal oriented paths of the same height* h*. Then there is a minimal oriented path* $\mathbf{Q}$ *of height* h *such that* $\mathbf{Q} \to \mathbf{P}_1, \mathbf{P}_2$.

Proposition 43. *A core oriented path has a single automorphism, i.e. it is rigid.*

Proof. Let $\mathbf{P}$ be a core oriented path and $\mathbf{P}$ be an isomorphic copy of $\mathbf{P}'$. There are at most two isomorphisms from $\mathbf{P}'$ to $\mathbf{P}$ (because a vertex with indegree 0 must be mapped to a vertex with indegree 0, and similarly for a vertex with outdegree 0). One possibility is to map the first vertex of $\mathbf{P}'$ to the first vertex of $\mathbf{P}$ and the last vertex of $\mathbf{P}'$ to the last vertex of $\mathbf{P}$. For contradiction, assume that the second possibility happens, i.e. there is an isomorphism φ that maps the first vertex of $\mathbf{P}'$ to the last vertex of $\mathbf{P}$ and the last vertex of $\mathbf{P}'$ to the first vertex of $\mathbf{P}$. Assume that both the first vertex v and last vertex w of $\mathbf{P}'$ have indegree zero (the other case is similar). Then the level(v) = level(w). This implies that the number of forward and

backward edges in $\mathbf{P}$ is the same, so $\mathbf{P}$ has $2q$ edges. By the existence of φ, $\mathbf{P}$ must have the form $\mathbf{Q}\bar{\mathbf{Q}}$, and such an oriented path is clearly not a core. □

Theorem 44. *Let* $\mathbf{P}$ *be a core oriented path that contains a subpath* $\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3$ *of some height* h *with the following properties:* $\mathbf{P}_1, \mathbf{P}_2$ *and* $\mathbf{P}_3$ *are minimal oriented paths, they all have height* h*, and there is a minimal oriented path* $\mathbf{Q}$ *of height* h *such that* $\mathbf{Q} \to \mathbf{P}_1$*,* $\mathbf{Q} \to \mathbf{P}_3$ *but* $\mathbf{Q} \not\to \mathbf{P}_2$*. Then* CSP($\mathbf{P}$) *is* NL*-complete.*

An example is given in Figure 3.7, *3* and *4*.

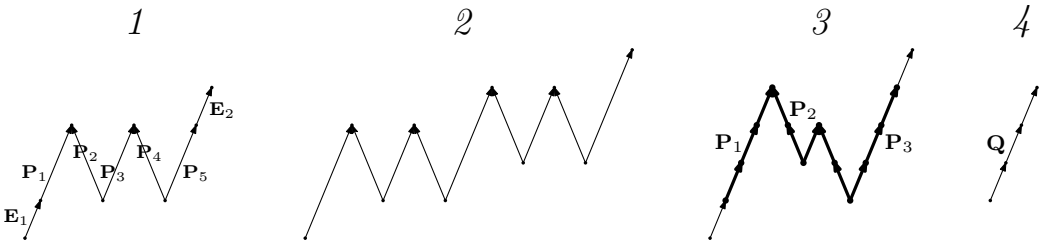


Figure 3.7: *1:* A 2-wave. *2:* A staircase. *3:* An example oriented path for which the CSP is NL-complete. *4:* The oriented path $\mathbf{Q}$ in Theorem 44 corresponding to the oriented path in *3*.

Proof of Theorem 44. We show that the less-than-or-equal-to relation on two elements, $\mathbf{R}_{\leq} = \{(0,0),(0,1),(1,1)\}$, and the relations $\{0\}$ and $\{1\}$ can be expressed from $\mathbf{P}$ using primitive positive (pp) formulas (i.e. a first order formulas with only existential quantification, conjunction and equality). This will prove the theorem because it is easy to see and well known that co-CSP($\{\mathbf{R}_{\leq}, \{0\}, \{1\}\}$) is equivalent to the NL-complete directed *st*-Conn problem.

Since $\mathbf{P}$ is a core, it is rigid by Proposition 43. Assume that the first vertex of $\mathbf{P}_1$ is in a level lower than the level of the last vertex of $\mathbf{P}_1$ (the other case can be handled similarly). See the illustration in Figure 3.8. Assume that the first vertex of $\mathbf{P}_1$ is 0 and the first vertex of $\mathbf{P}_3$ is 1. We construct a structure $\mathbf{G}$ with two special vertices x and y such that $\{(h(x), h(y)) \mid \mathbf{G} \xrightarrow{h} \mathbf{P}\} = \mathbf{R}_{\leq}$. It is well known and easy to show that then $\mathbf{R}_{\leq}$ can also be expressed from $\mathbf{P}$ using a pp-formula. Let $\mathbf{P}'$ be an isomorphic copy of $\mathbf{P}$. We refer to copies of

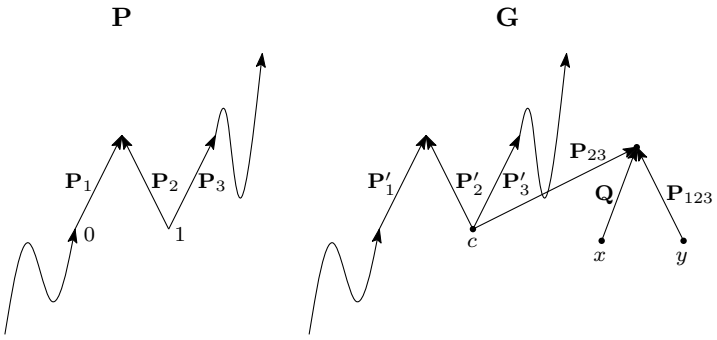


Figure 3.8: Construction of the gadget $\mathbf{G}$.

$\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ as $\mathbf{P}'_1, \mathbf{P}'_2, \mathbf{P}'_3$, respectively. Using Lemma 42, we find a minimal oriented path $\mathbf{P}_{23}$ of height h that maps to both $\mathbf{P}_2$ and $\mathbf{P}_3$. Similarly, we find a minimal oriented path $\mathbf{P}_{123}$ that maps to each of $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$. We rename the first vertex of $\mathbf{Q}$ to x, and the first vertex of $\mathbf{P}_{123}$ to y. To construct $\mathbf{G}$, we identify the topmost vertices of the oriented paths $\mathbf{P}_{23}, \mathbf{Q}$ and $\mathbf{P}_{123}$. Then we identify the first vertex of $\mathbf{P}_{23}$ with the vertex c of $\mathbf{P}'$ that is shared by $\mathbf{P}'_2$ and $\mathbf{P}'_3$. Observe that any homomorphism from $\mathbf{G}$ to $\mathbf{P}$, must map c to 1. It is straightforward to verify that $\{(h(x), h(y)) \mid \mathbf{G} \xrightarrow{h} \mathbf{P}\} = \mathbf{R}_{\leq}$.

Because $\mathbf{P}$ is rigid, any relation of the form $\{v\}$ where $v \in P$ can be expressed by a pp-formula. $\square$

Chapter 4

The Complexity of the List Homomorphism Problem for Graphs[1]

4.1 Introduction

Homomorphisms of graphs, i.e. edge-preserving mappings, generalize graph colorings, and can model a wide variety of combinatorial problems dealing with mappings and assignments [50]. Because of the richness of the homomorphism framework, many computational aspects of graph homomorphisms have recently become the focus of much attention. In the *list* **H**-*colouring* problem (for a fixed graph **H**), one is given a graph **G** and a list L_v of vertices

[1]The contents of this chapter were obtained in joint work with Andrei Krokhin, Benoit Larose, and Pascal Tesson, published in [36, 37].

of $\mathbf{H}$ for each vertex v in $\mathbf{G}$, and the goal is to determine whether there is a homomorphism h (for graphs this is just an edge-preserving map) from $\mathbf{G}$ to $\mathbf{H}$ such that $h(v) \in L_v$ for all v. The complexity of such problems has been studied by combinatorial methods, e.g., in [41, 42]. In this chapter, we study the complexity of the list homomorphism problem for graphs in the wider context of classifying the complexity of CSPs, see [9, 43, 50].

The aim of the present chapter is to show that the algebraic conditions explained in Section 1.3.2 are indeed sufficient and necessary in the special case of list $\mathbf{H}$-colouring for undirected graphs with possible loops, and to characterise, in this special case, the dividing lines in graph-theoretic terms. Note that our results provide the first complete classification of CSPs with a fixed template for a reasonably large class of structures outside the Boolean case [2]. One can view the list $\mathbf{H}$-colouring problem as a CSP where the template is the structure $\mathbf{H}^L$ consisting of the binary (edge) relation of $\mathbf{H}$ and all unary relations on H (i.e. every subset of H). Tractable list homomorphism problems for general structures were characterised in [16, 4] in algebraic terms. The tractable cases for graphs were described in [42] in both combinatorial and (more specific) algebraic terms; the latter implies, when combined with a recent result [30], that in these cases co-CSP($\mathbf{H}^L$) is definable in linear Datalog and therefore CSP($\mathbf{H}^L$) is in fact in NL. We complete the picture by refining this classification and showing that CSP($\mathbf{H}^L$) is either NP-complete, or NL-complete, or L-complete or in AC^0 (and in fact first-order definable). We also remark that the problem of recognising into which case

the problem CSP($\mathbf{H}^L$) falls can be solved in polynomial time.

As we mentioned above, the distinction between NP-complete cases and those in NL follows from earlier work [30, 42], and the situation is similar with distinction between L-hard cases and those leading to membership in AC^0 [61, 62]. Therefore, the main body of technical work in the chapter concerns the distinction between NL-hardness and membership in L. We give two equivalent characterisations of the class of graphs $\mathbf{H}$ such that CSP($\mathbf{H}^L$) is in L. One characterisation is via forbidden induced subgraphs (for example, the reflexive graphs in this class are exactly the (P_4, C_4)-free graphs[2], while the irreflexive ones are exactly the bipartite (P_6, C_6)-free graphs), while the other characterisation is via an inductive definition. The first characterisation is used to show that graphs outside of this class give rise to NL-hard problems; we do this by providing constructions witnessing the presence of a non-Boolean type in the algebras associated with the graphs. The second characterisation is used to prove positive results. We first provide operations in the associated algebra which satisfy certain identities; this allows us to show that the necessary condition on types is also sufficient in our case. We also use the inductive definition to demonstrate that the class of negative instances of the corresponding CSP is definable in symmetric Datalog, which implies membership of the CSP in L.

[2] P_k is a path on k vertices, and C_k is the chordless cycle on k vertices. A graph is $(P_k, C_{k'})$-free if it does not contain a P_k or a $C_{k'}$ as an induced subgraph.

4.2 Preliminaries

4.2.1 Graphs and relational structures

The *direct n-th power* of a τ-structure $\mathbf{T}$, denoted $\mathbf{T}^n$, is defined to have universe T^n and, for any (say m-ary) $R \in \tau$, $(\mathbf{a}_1, \ldots, \mathbf{a}_m) \in R(\mathbf{T}^n)$ if and only if $(\mathbf{a}_1[i], \ldots, \mathbf{a}_m[i]) \in R(\mathbf{T})$ for each $1 \leq i \leq n$. For a subset $I \subseteq T$, the *substructure induced by* I *on* $\mathbf{T}$ is the structure $\mathbf{I}$ with universe I and such that $R(\mathbf{I}) = R(\mathbf{T}) \cap I^m$ for every m-ary $R \in \tau$.

For the purposes of this chapter, a *graph* is a relational structure $\mathbf{H} = \langle H; \theta \rangle$ where θ is a symmetric binary relation on H. The graph $\mathbf{H}$ is *reflexive* (*irreflexive*) if $(x, x) \in \theta$ ($(x, x) \notin \theta$) for all $x \in H$. Given a graph $\mathbf{H}$, let $S_1, \ldots, S_k$ denote all subsets of H; let $\mathbf{H}^L$ be the relational structure obtained from $\mathbf{H}$ by adding all the S_i as unary relations; more precisely, let τ be the signature that consists of one binary relational symbol θ and unary symbols R_i, $i = 1, \ldots, k$. The τ-structure $\mathbf{H}^L$ has universe H, $\theta(\mathbf{H}^L)$ is the edge relation of $\mathbf{H}$, and $R_i(\mathbf{H}^L) = S_i$ for all $i = 1, \ldots, k$. It is easy to see that $\mathbf{H}^L$ is a core; in fact its only self-map which is a homomorphism is the identity. We call CSP($\mathbf{H}^L$) the *list homomorphism problem for* $\mathbf{H}$. Note that if $\mathbf{G}$ is an instance of this problem then $\theta(\mathbf{G})$ can be considered as a digraph, but the directions of the arcs are unimportant because $\mathbf{H}$ is undirected. Also, if an element $v \in G$ is in $R_i(\mathbf{G})$ then this is equivalent to v having S_i as its list, so $\mathbf{G}$ can be thought of as a digraph with $\mathbf{H}$-lists. Note that an element of $\mathbf{G}$ can in principle have several $\mathbf{H}$-lists, which is equivalent to having their

intersection as a single list.

In [42], a dichotomy result was proved, identifying bi-arc graphs as those whose list homomorphism problem is tractable, and others as giving rise to NP-complete problems. Bi-arc graphs are defined as follows. Fix a circle with two distinct specified points p and q. A bi-arc is a pair of arcs (N, S) on the circle such that N contains p but not q and S contains q but not p. A graph $\mathbf{H}$ is a *bi-arc graph* if there is a family of bi-arcs $\{(N_x, S_x) : x \in H\}$ such that, for every $x, y \in H$, the following conditions hold: (i) if x and y are adjacent, then neither N_x intersects S_y nor N_y intersects S_x, and (ii) if x is not adjacent to y then both N_x intersects S_y and N_y intersects S_x. Equivalently, $\mathbf{H}$ is a bi-arc graph if and only if the complement of the graph $\mathbf{H} \times \mathbf{K}_2$ is a circular arc graph (i.e., can be represented by arcs on a circle so that two vertices are adjacent if and only if the corresponding arcs intersect) [42].

4.2.2 Algebra

An algebra is a pair $\mathbb{A} = \langle A; F \rangle$ where A is a non-empty set, and F is a family of finitary operations on A. With any structure $\mathbf{T}$, one associates an algebra $\mathbb{A}_{\mathbf{T}}$ whose universe is T and whose operations are all polymorphisms of $\mathbf{T}$. Given a graph $\mathbf{H}$, we let, for the ease of notation, $\mathbb{H}$ denote the algebra associated with $\mathbf{H}^L$. An operation on a set is called *conservative* if it preserves all subsets of the set (as unary relations). So, the operations of $\mathbb{H}$ are the conservative polymorphisms of $\mathbf{H}$. Polymorphisms can provide a convenient language when defining classes of graphs. For example, it was shown in [12]

that a graph is a bi-arc graph if and only if it admits a conservative majority operation, where recall that a majority operation is a ternary operation m satisfying the identities $m(x,x,y) = m(x,y,x) = m(y,x,x) = x$ (for all x, y).

In order to state some of our results, we need the following basic notions from universal algebra (see textbooks [54, 70] for more universal-algebraic background and [15, 24] for the basics of the connection between universal algebra and CSP). Let I be a signature, i.e. a set of operation symbols f each of a fixed arity; we use the term "signature" for both structures and algebras, this will cause no confusion. An *algebra of signature* I is a pair $\mathbb{A} = \langle A; F \rangle$ where A is a non-empty set, the *universe of* $\mathbb{A}$, and $F = \{f^{\mathbb{A}} : f \in I\}$ is the set of *basic* operations (for each $f \in I$, $f^{\mathbb{A}}$ is an operation on A of the corresponding arity). The *term operations* of $\mathbb{A}$ are the operations built from the operations in F and projections by using composition. The *polynomial operations* of $\mathbb{A}$ are the operations built from the operations in F, the constant operations and projections by using composition. An algebra all of whose (basic or term) operations are conservative is called a *conservative algebra*. A *subalgebra* $\mathbb{B}$ of an algebra $\mathbb{A}$ consists of a subset B of A that is invariant under all operations of $\mathbb{A}$ and the restrictions of the operations of $\mathbb{A}$ to B. A *homomorphic image* of an algebra $\mathbb{A}$ is an algebra $\mathbb{C}$ which is similar to $\mathbb{A}$ (i.e. with the same signature) and such that there is a surjective mapping $\psi : A \to C$ with $\psi(f^{\mathbb{A}}(a_1, \ldots, a_r)) = f^{\mathbb{C}}(\psi(a_1), \ldots, \psi(a_r))$ for all operations $f \in I$ and all tuples of elements of A. Direct products and powers of algebras are defined in a natural way, by taking direct product of universes

and defining the operations to act componentwise. A class of similar algebras which is closed under formation of homomorphic images, subalgebras and direct products is called a *variety*. The *variety generated* by an algebra $\mathbb{A}$, denoted by $\mathcal{V}(\mathbb{A})$, is the smallest variety containing $\mathbb{A}$, it coincides with the class of all homomorphic images of subalgebras of direct powers of $\mathbb{A}$.

Tame Congruence Theory, as developed in [54], is a powerful tool for the analysis of finite algebras. Every finite algebra has a *typeset*, which describes (in a certain specified sense) the local behaviour of the algebra. It contains one or more of the following 5 *types*: (1) the *unary* type, (2) the *affine* type, (3) the *Boolean* type, (4) the *lattice* type and (5) the *semilattice* type. The numbering of the types is fixed, and they are often referred to by their numbers. The typeset of a variety $\mathcal{V}$, denoted $typ(\mathcal{V})$, is simply the union of typesets of all finite algebras in it. We note that there is a very tight connection between the kind of identities that are satisfied by the algebras in a variety and the types that are *admitted* or *omitted* by a variety, i.e. those types that do or do not appear in the typesets of algebras in the variety [54]. We will be mostly interested in type-omitting conditions for varieties of the form $\mathcal{V}(\mathbb{A}_{\mathbf{T}})$, and Corollary 3.2 of [79] says that in this case it is enough to consider the typesets of $\mathbb{A}_{\mathbf{T}}$ and its subalgebras. On the intuitive level, if $\mathbf{T}$ is a core structure then the typeset $typ(\mathcal{V}(\mathbb{A}_{\mathbf{T}}))$ contains crucial information about the kind of relations that $\mathbf{T}$ can or cannot simulate, thus implying lower/upper bounds on the complexity of CSP($\mathbf{T}$).

The definitions of the types are rather technical in general, but they are

simple enough for conservative algebras, and all algebras in this chapter are conservative. Let $\mathbb{A} = \langle A, F \rangle$ be a conservative algebra and let $X = \{a, b\}$ be a two-element subset of A. By conservativity, every operation in F preserves X, so X is the universe of a subalgebra $\mathbb{X}$ of $\mathbb{A}$. Identify a with 0 and b with 1, and think of operations on X as Boolean operations. Then X satisfies exactly one of the following five conditions (see [54]):

- The type of X (in $\mathbb{A}$) is *unary*, or **1**, if $f|_X$ is a projection for each $f \in F$.
- The type of X is *affine*, or **2**, if it is not unary and $f|_X$ is a linear operation for each $f \in F$. Equivalently, the type of X is affine if the polynomial operations of $\mathbb{X}$ are all linear Boolean operations.
- The type of X is *semilattice*, or **5**, if it is not unary and either each operation $f|_X$, $f \in F$, is the minimum of some of its arguments or each operation $f|_X$, $f \in F$, is the maximum of some of its arguments.
- The type of X is *lattice*, or **4**, if it is not semilattice, but all operations $f|_X$, $f \in F$, are monotone. Equivalently, the polynomial operations of $\mathbb{X}$ are all monotone Boolean operations.
- The type of X is *Boolean*, or **3**, in all other cases, that is, if the family $\{f|_X \mid f \in F\}$ contains a non-linear operation and a non-monotone operation. Equivalently, the polynomial operations of $\mathbb{X}$ are all possible Boolean operations.

4.2 Preliminaries

The typeset of a conservative algebra $\mathbb{A}$ admits all types of two-element subsets of A, and possibly some other types. Consider the following ordering of the types: $\mathbf{1} < \mathbf{2} < \mathbf{3} > \mathbf{4} > \mathbf{5} > \mathbf{1}$. It follows from Corollary 3.2 of [79] that, for any type $\mathbf{i}$, the variety $\mathcal{V}(\mathbb{A})$ omits all types below $\mathbf{i}$ (with respect to the above ordering) if and only if none of the two-element subsets of A has type below $\mathbf{i}$. Thus, for a structure of the form $\mathbf{H}^L$, the knowledge of how conservative polymorphisms of $\mathbf{H}$ behave on two-element subsets of H gives us all necessary information about the typeset of $\mathcal{V}(\mathbb{H})$.

We will use the ternary operations $f_1, \dots, f_n$ satisfying the following identities:

$$x = f_1(x, y, y) \tag{4.1}$$

$$f_i(x, x, y) = f_{i+1}(x, y, y) \text{ for all } i = 1, \dots n-1 \tag{4.2}$$

$$f_n(x, x, y) = y. \tag{4.3}$$

The following lemma from [54] contains some type-omitting results that will be important later.

Lemma 45. *1. A finite algebra $\mathbb{A}$ has term operations $f_1, \dots, f_n$, for some $n \geq 1$, satisfying identities (4.1)–(4.3) if and only if the variety $\mathcal{V}(\mathbb{A})$ omits the unary, lattice, and semilattice types.*

2. If a finite algebra $\mathbb{A}$ has a majority term operation then $\mathcal{V}(\mathbb{A})$ omits the unary, affine, and semilattice types.

We remark in passing that operations satisfying identities (4.1)–(4.3) are also known to characterise a certain algebraic (congruence) condition called $(n+1)$-permutability [54].

4.3 Main results and proof outline

In this section we state our main results, Theorems 46, 48, and 49. Theorem 46 follows from known results (with a little help from Lemma 60), the proof of Theorem 49 is a relatively simple application of a result from [61], and the proof of Theorem 48 constitutes most of this chapter.

Theorem 46. *Let* $\mathbf{H}$ *be a graph.*

- *If* $\mathcal{V}(\mathbb{H})$ *admits the unary type, then* co-CSP$(\mathbf{H}^L)$ *is not expressible in Datalog and* CSP$(\mathbf{H}^L)$ *is* NP*-complete (under first-order reductions);*
- *if* $\mathcal{V}(\mathbb{H})$ *omits the unary but admits the lattice type, then* co-CSP$(\mathbf{H}^L)$ *is not expressible in symmetric Datalog but is expressible in linear Datalog, and* CSP$(\mathbf{H}^L)$ *is* NL*-complete (under first-order reductions).*

Proof. The first statement is shown in [62]. If $\mathcal{V}(\mathbb{H})$ omits the unary type, then $\mathbf{H}^L$ admits a majority operation by Lemma 60 in Section 4.5 and then co-CSP$(\mathbf{H}^L)$ is expressible in linear Datalog by [30]; in particular the problem is in NL. If, furthermore, the variety admits the lattice type, then co-CSP$(\mathbf{H}^L)$ is not expressible in symmetric Datalog and is NL-hard by results in [62]. □

By Lemma 45, the presence of a majority operation in $\mathbb{H}$ implies that $typ(\mathcal{V}(\mathbb{H}))$ can contain only the Boolean and lattice types. The lattice type is dealt with in Theorem 46, so it remains to investigate graphs $\mathbf{H}$ with $\mathcal{V}(\mathbb{H})$ admitting only the Boolean type. We will now define the class of graphs that plays a central role in this chapter.

Definition 47. The class $\mathcal{F}$ consists of all graphs $\mathbf{H}$ that contain none of the following as an induced subgraph:

1. the reflexive path of length 3 and the reflexive 4-cycle;
2. the irreflexive cycles of length 3, 5 and 6, and the irreflexive path of length 5;
3. **B1**, **B2**, **B3**, **B4**, **B5** and **B6** (see Figure 4.1.)

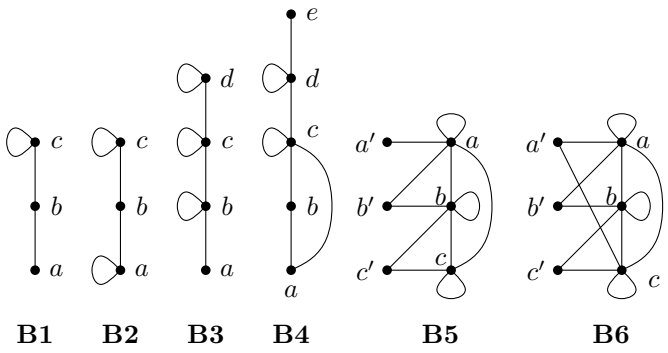


Figure 4.1: The forbidden graphs.

Notice that when only reflexive or only irreflexive graphs are of interest, then the only relevant forbidden subgraphs are those in Definition 47(1) or

in Definition 47(2), respectively. Observe that all irreflexive graphs in $\mathcal{F}$ are bipartite.

The next theorem is the main result of this chapter.

Theorem 48. *Let* $\mathbf{H}$ *be a graph. Then the following conditions are equivalent:*

1. $\mathbf{H}$ *admits conservative operations satisfying (4.1)–(4.3) for* $n = 3$*;*
2. $\mathbf{H}$ *admits conservative operations satisfying (4.1)–(4.3) for some* $n \geq 1$*;*
3. $\mathcal{V}(\mathbb{H})$ *admits only the Boolean type;*
4. $\mathbf{H} \in \mathcal{F}$*;*
5. co-CSP$(\mathbf{H}^L)$ *is definable in symmetric Datalog.*

If the above holds then CSP$(\mathbf{H}^L)$ *is in the complexity class* L*.*

Proof. (1) trivially implies (2). If (2) holds then by Lemma 45 $\mathcal{V}(\mathbb{H})$ omits the unary, lattice, and semilattice types. By Lemma 60 in Section 4.5, $\mathbf{H}$ admits a majority operation, so Lemma 45 implies that $\mathcal{V}(\mathbb{H})$ also omits the affine type; hence (3) holds. Implication (3)⇒(4) is the content of Lemma 61 in Section 4.5, and (5) implies (3) by a result of [62]. We give an inductive characterisation of the class $\mathcal{F}$ in Theorem 58 in Section 4.4, and then use it to show that (4) implies both (1) and (5), in Subsection 4.5.1 and Section 4.6, respectively. Finally, definability in symmetric Datalog implies membership in L by [38]. □

For completeness' sake, we describe graphs whose list homomorphism problem is definable in first-order logic (equivalently, is in AC^0, see [18].) By results in [62], any problem CSP($\mathbf{T}$) is either first-order definable or L-hard under FO reductions. Hence, it follows from Theorem 48 that, for a graph $\mathbf{H} \in \mathcal{F}$, the list homomorphism problem for $\mathbf{H}$ is either first-order definable or L-complete.

Theorem 49. *Let $\mathbf{H}$ be a graph. Then* CSP($\mathbf{H}^L$) *is first-order definable if and only if $\mathbf{H}$ has the following form: H is the disjoint union of two sets R and I such that (i) R is the set of loops of $\mathbf{H}$ and induces a complete graph, (ii) I is the set of non-loops of $\mathbf{H}$ and induces a graph with no edges, and (iii) $I = \{x_1, \ldots, x_m\}$ can be ordered so that the neighbourhood of x_i is contained in the neighbourhood of x_{i+1} for all $1 \leq i \leq m-1$.*

Remark 50. *Given a graph $\mathbf{H}$, it can be decided in polynomial time which of the different cases delineated in Theorems 46, 48, 49 the list homomorphism problem for $\mathbf{H}$ satisfies. Indeed, it is known (see [42]) that $\mathbf{H}$ is a bi-arc graph if and only if the complement of $\mathbf{H} \times \mathbf{K}_2$ is a circular arc graph which can be recognised in linear time [69]. Assume that $\mathbf{H}$ is a bi-arc graph: the definition of $\mathcal{F}$ (Definition 47) gives a polynomial-time (in fact, even AC^0) algorithm to recognise them; and those graphs whose list homomorphism problem is first-order definable can be recognised in polynomial time by results of [61].*

The remaining sections are devoted to proving the lemmas used in the proof of the above theorems. Section 4.4 deals with the graph-theoretic

proofs, Section 4.5 presents the proofs of the algebraic results, and Section 4.6 provides the symmetric Datalog expressibility proofs. Finally, Section 4.7 contains the proof of Theorem 49.

4.4 Combinatorial graph characterisations

In this section, we give an inductive characterisation of the class $\mathcal{F}$ defined in the previous section. This characterisation is stated in Theorem 58. Before proving Theorem 58, we provide inductive characterisations for the reflexive and the irreflexive subclasses of $\mathcal{F}$ in Lemmas 53 and 55, respectively. These lemmas will facilitate the proof of Theorem 58.

Let $\mathcal{F}_{\text{re}}$ denote the reflexive graphs in $\mathcal{F}$ (i.e. reflexive graphs that do not contain graphs in Definition 47(1) as induced subgraphs), and $\mathcal{F}_{\text{ir}}$ the irreflexive graphs in $\mathcal{F}$ (i.e. irreflexive graphs that do not contain graphs in Definition 47(2) as induced subgraphs).

We need the following two operations on graphs:

Definition 51. Let $\mathbf{H}_1$ and $\mathbf{H}_2$ be bipartite irreflexive graphs, with colour classes B_1, T_1 and B_2 and T_2 respectively, with T_1 and B_2 non-empty. We define the *special sum* $\mathbf{H}_1 \odot \mathbf{H}_2$ (which depends on the choice of the B_i and T_i)[3] as follows: it is the graph obtained from the disjoint union of $\mathbf{H}_1$ and $\mathbf{H}_2$ by adding all possible edges between the vertices in T_1 and B_2. We say that an irreflexive graph $\mathbf{H}$ is *a special sum* or *expressed as a special sum* if

[3]Notice that one can often divide a bipartite graph into parts in several ways, and even choose B_1 and/or T_2 to be empty.

there exist two bipartite graphs and a choice of colour classes on each such that $\mathbf{H}$ is isomorphic to the special sum of these two graphs.

Definition 52. Given two vertex-disjoint graphs $\mathbf{H}_1$ and $\mathbf{H}_2$, the *adjunction of* $\mathbf{H}_1$ *to* $\mathbf{H}_2$ is the graph $\mathbf{H}_1 \oslash \mathbf{H}_2$ obtained by taking the disjoint union of the two graphs, and adding every edge of the form (x, y) where x is a loop in $\mathbf{H}_1$ and y is a vertex of $\mathbf{H}_2$.

We begin with the simple case of reflexive graphs.

4.4.1 The reflexive graphs in $\mathcal{F}$

Lemma 53. *$\mathcal{F}_{\mathrm{re}}$ is the smallest class of reflexive graphs $\mathcal{I}_{\mathrm{re}}$ such that:*

1. *$\mathcal{I}_{\mathrm{re}}$ contains the one-element graph;*
2. *$\mathcal{I}_{\mathrm{re}}$ is closed under disjoint union;*
3. *if $\mathbf{H}_1$ is a single loop and $\mathbf{H}_2 \in \mathcal{I}_{\mathrm{re}}$ then $\mathbf{H}_1 \oslash \mathbf{H}_2 \in \mathcal{I}_{\mathrm{re}}$.*

Proof. It is easy to see that $\mathcal{I}_{\mathrm{re}} \subseteq \mathcal{F}_{\mathrm{re}}$. Suppose that $\mathcal{F}_{\mathrm{re}} \not\subseteq \mathcal{I}_{\mathrm{re}}$, and let $\mathbf{H}$ be a graph of smallest size such that $\mathbf{H} \in \mathcal{F}_{\mathrm{re}}$ and $\mathbf{H} \notin \mathcal{I}_{\mathrm{re}}$, i.e. $\mathbf{H}$ cannot be obtained from the one-element graph using the operations of disjoint union and adjunction of a loop. By minimality, $\mathbf{H}$ is connected, contains no universal vertex (a vertex that is a neighbour of every other vertex including itself), it contains more than one vertex, and all of its proper induced connected subgraphs contains a universal vertex. Pick some edge (x, y) in $\mathbf{H}$; since there

is no universal vertex there exists some t not adjacent to y. Let $\mathbf{G}$ be the subgraph induced by $H \setminus \{x\}$.

Assume first that $\mathbf{G}$ *is connected.* Let u be a universal vertex of $\mathbf{G}$; we have edges $(x, y), (y, u), (u, t)$. Since $\mathbf{H}$ has no universal vertex then x is not adjacent to u. Thus $\{x, y, t, u\}$ is either a reflexive path of length 3 or a reflexive 4-cycle, a contradiction.

Suppose now that $\mathbf{G}$ *is not connected.* Let C and D be distinct components of $\mathbf{G}$; since x is not universal in $\mathbf{H}$ there exists some z not adjacent to x, and without loss of generality suppose that $z \in C$. Since $\mathbf{H}$ is connected there exists a path from z to some element in D, in particular we can find edges $(z, w), (w, x), (x, v)$, where $z, w \in C$ and w is a neighbour of x, z is not adjacent to x, and $v \in D$. It is easy to verify that $\{z, w, x, v\}$ induces a reflexive path of length 3, a contradiction. □

Remark 54. *Lemma 53 states that the reflexive graphs avoiding the path of length 3 and the 4-cycle are precisely those constructed from the one-element loop using disjoint union and adjunction of a universal vertex. These graphs (with no self-loops) have been studied before. Namely, our inductive definition corresponds to the definition of the so-called NLCT width 1 graphs. In [47], it is shown that NLCT width 1 graphs are precisely the (C4, P4)-free graphs, which are known to be the trivially perfect graphs. Our result provides an alternative proof of the equivalence of these conditions.*

4.4.2 The irreflexive graphs in $\mathcal{F}$

The following result gives an inductive characterisation of the class of graphs $\mathcal{F}_{\text{ir}}$.

Lemma 55. *$\mathcal{F}_{\text{ir}}$ is the smallest class of irreflexive graphs $\mathcal{I}_{\text{ir}}$ such that:*

1. *$\mathcal{I}_{\text{ir}}$ contains the one-element graph;*
2. *$\mathcal{I}_{\text{ir}}$ is closed under disjoint union;*
3. *$\mathcal{I}_{\text{ir}}$ is closed under special sum.*

Proof. We show that $\mathcal{I}_{\text{ir}} \subseteq \mathcal{F}_{\text{ir}}$. The class $\mathcal{F}_{\text{ir}}$ obviously contains the one-element graph. In order to prove the inclusion, it is sufficient to show that if $\mathbf{H}_1$ and $\mathbf{H}_2$ are graphs that do not contain any cycles of length 3, 5 or 6, or a path of length 5 as an induced subgraph, then neither the disjoint union of $\mathbf{H}_1$ and $\mathbf{H}_2$, nor the special sum of $\mathbf{H}_1$ and $\mathbf{H}_2$ contain any cycles of length 3, 5 or 6, or a path of length 5 as an induced subgraph. This is clearly the case for disjoint union, so now we concentrate on the special sum of $\mathbf{H}_1$ and $\mathbf{H}_2$.

As it was observed after Definition 47, if an irreflexive graph does not contain cycles of length 3, 5 or a path of length 5, then it must be bipartite. It follows that $\mathbf{H}_1 \odot \mathbf{H}_2$ must be bipartite, so $\mathbf{H}_1 \odot \mathbf{H}_2$ contains no induced cycles of length 3 or 5. Assume then that $\mathbf{C}$ is an induced subgraph of $\mathbf{H}_1 \odot \mathbf{H}_2$, where $\mathbf{C}$ is a 6-cycle or a 5-path. We shall obtain a contradiction by showing that $\mathbf{C}$ must be contained either in $\mathbf{H}_1$ or $\mathbf{H}_2$. By assumption

and definition of special sum, it is clear that, since $\mathbf{C}$ is connected, it must contain at least one vertex in T_1 and at least one in B_2; on the other hand, since $\mathbf{C}$ contains no induced 4-cycle, $\mathbf{C}$ can have at most 2 vertices in T_1 and at most 1 in B_2, without loss of generality. Suppose first that there is exactly one vertex of $\mathbf{C}$ in T_1. Since every vertex of $\mathbf{C}$ has degree at most 2, it follows that no more than 1 vertex of $\mathbf{C}$ can be in B_1, and similarly no more than 1 vertex of $\mathbf{C}$ can be in T_2. Therefore $\mathbf{C}$ cannot contain vertices both in T_1 and B_2, so $\mathbf{C}$ is either in $\mathbf{H}_1$ or $\mathbf{H}_2$, a contradiction. On the other hand if $\mathbf{C}$ has 2 vertices in T_1, then $\mathbf{C}$ has no vertex in T_2 and at most 2 in B_1, so again, $\mathbf{C}$ cannot contain vertices both in T_1 and B_2, a contradiction. Hence, we conclude that $\mathcal{I}_{\text{ir}} \subseteq \mathcal{F}_{\text{ir}}$.

For the reverse inclusion, $\mathcal{F}_{\text{ir}} \subseteq \mathcal{I}_{\text{ir}}$, suppose for a contradiction that there exists a graph $\mathbf{H} \in \mathcal{F}_{\text{ir}}$ such that $\mathbf{H} \notin \mathcal{I}_{\text{ir}}$. Choose $\mathbf{H}$ so that its set of vertices is of minimal size. Obviously $\mathbf{H}$ is connected. We denote the usual graph distance between vertices x and y by $d(x, y)$, i.e. the length of a shortest path in the graph between x and y. Let $N(x)$ denote the set of neighbours of x in $\mathbf{H}$, and let $N_2(x) = \{t \in T_1 : d(x, t) = 2\}$.

Claim 1. For every $x \in H$ there exists $y \in H$ such that $d(x, y) = 3$.

Proof. Otherwise, since $\mathbf{H}$ is connected, we'd have some $x \in H$ with $d(x, y) \leq 2$ for all $y \in H$. Now let B_2 denote the set of all vertices adjacent to x, and let $T_2 = H \setminus (B_2 \cup \{x\})$. Furthermore let $B_1 = \emptyset$ and $T_1 = \{x\}$. Since $\mathbf{H}$ is bipartite, (B_2, T_2) is a bipartition, and hence $\mathbf{H}$ is expressed as a special sum, a contradiction. □

Claim 2. There exists $x \in H$ such that the subgraph induced by $H \setminus \{x\}$ is connected.

Proof. Notice first that if for some x the subgraph $\mathbf{G}$ induced by $H \setminus \{x\}$ is not connected, then it contains at most one connected component with 2 or more vertices. Indeed, by Claim 1 let $y \in H$ such that $d(x, y) = 3$; let y, w, z, x be an induced path of length 3 from y to x. Note that the connected component of y has size at least 2. Now choose a different connected component $\mathbf{C}$ of $\mathbf{G}$ that contains at least two vertices. Since $\mathbf{H}$ is connected, $\mathbf{C}$ clearly contains adjacent vertices u and v with u adjacent to x. But then the vertices y, w, z, x, u, v induce a path of length 5 in $\mathbf{H}$, a contradiction.

Now choose any vertex x in $\mathbf{H}$. If the subgraph induced by $H \setminus \{x\}$ is connected we are done; otherwise, one of its components must be trivial, i.e. $\mathbf{H}$ has a vertex x' dangling from x. Then the subgraph induced by $H \setminus \{x'\}$ is connected. □

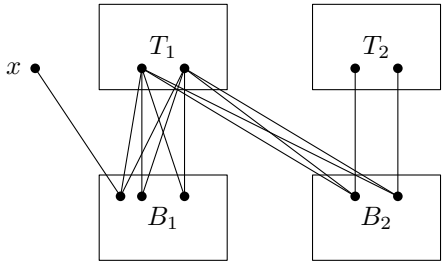


Figure 4.2: The graph $\mathbf{H}$.

So we may now suppose that $\mathbf{H}$ has the following structure: there is some vertex x such that the subgraph $\mathbf{G}$ induced by $H \setminus \{x\}$ is connected;

by induction hypothesis, **G** is a special sum, with subsets B_i, T_i, $(i = 1, 2)$ where T_1 and B_2 are non-empty. We suppose without loss of generality that x is adjacent to some vertex in $B_1 \cup B_2$ (see Figure 4.2.)

Case 1: T_2 is non-empty.

Claim 3. There exists an edge (u, v) with $u \in B_2$ and $v \in T_2$ such that u is not adjacent to x.

Proof. Suppose for a contradiction that this is not the case: then $B_2 = B_2^0 \cup B_2^1$ where B_2^0 consists of all elements of B_2 not adjacent to any vertex in T_2, and since T_2 is non-empty, the set B_2^1 is non-empty, and contains by hypothesis only vertices adjacent to x. Then define a decomposition of H as follows: let $B_1' = B_1 \cup B_2^0$, $T_1' = T_1 \cup \{x\}$, $B_2' = B_2^1$ and $T_2' = T_2$. But then **H** is a special sum, a contradiction. □

Claim 4. The subgraph induced by $N(x) \cup N_2(x)$ is complete bipartite.

Proof. Otherwise, we may find elements $t \in N_2(x)$ and $z \in N(x)$ which are not adjacent. Let y be a vertex on a path of length 2 between x and t. Let u and v be the elements whose existence is guaranteed by the last claim: then it is easy to see that the sequence z, x, y, t, u, v is an induced path of length 5 in **H**, a contradiction. □

Consider the following decomposition of H: let $B_1' = B_1 \setminus (N(x) \cap B_1)$, $T_1' = N_2(x)$, $B_2' = (N(x) \cap B_1) \cup B_2$ and $T_2' = (T_1 \setminus N_2(x)) \cup \{x\} \cup T_2$. By Claim 4 this is a decomposition of **H** as a special sum, unless there exists some edge

(y, z) with $y \in B_1'$ and $z \in T_2'$, i.e. with $y \in B_1 \setminus N(x)$ and $z \in T_1 \setminus N_2(x)$. Suppose this occurs. Then we have the following:

Claim 5. (y, t) is an edge for every $t \in N_2(x)$.

Proof. If this is not the case, then choose some $t \in N_2(x)$ not adjacent to y; let $n \in N(x)$ be adjacent to t. By Claim 3 we can find $u \in B_2$ not adjacent to x. Then the sequence y, z, u, t, n, x is an induced path of length 5 in $\mathbf{H}$, a contradiction. □

It follows from Claim 5 that we can modify our last decomposition as follows: simply remove from B_1' all the offending vertices such as y. More precisely, let Y be the set of all $y \in B_1'$ that have some neighbour $z \in T_1 \setminus N_2(x)$, and let $B_1'' = B_1' \setminus Y$, $T_1'' = T_1'$, $B_2'' = Y \cup B_2'$ and $T_2'' = T_2'$. By Claim 5, this shows that $\mathbf{H}$ is a special sum, a contradiction.

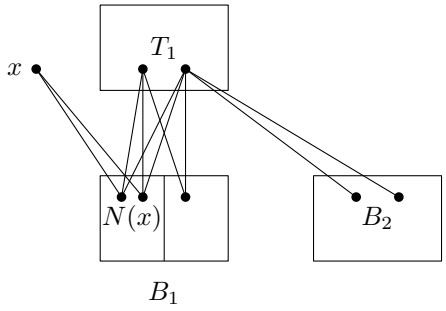


Figure 4.3: The graph $\mathbf{H}$ with T_2 empty.

Case 2: T_2 is empty.

Notice that in this case we may assume that $N(x) \subseteq B_1$, by simply decomposing $H \setminus \{x\}$ if necessary as $B_1' = B_1 \cup N(x)$, $B_2' = B_2 \setminus N(x)$, and

$T_i' = T_i$ for $i = 1, 2$. (Of course, if $\mathbf{H}$ is not a special sum then there is at least one vertex in $B_2 \setminus N(x)$ for otherwise we could set $T_1' = T_1 \cup \{x\}$.)

Claim 6. For every $y, z \in N(x)$ we have $N(y) \subseteq N(z)$ or $N(z) \subseteq N(y)$. Similarly, for every $u, v \in T_1$, either $N(u) \cap N(x) \subseteq N(v) \cap N(x)$ or $N(v) \cap N(x) \subseteq N(u) \cap N(x)$.

Proof. Suppose this is not the case: then we may find $y, z \in N(x)$ and $u \in N(y)$ and $v \in N(z)$ such that u is not adjacent to z and v is not adjacent to y. Let $b \in B_2$. Then clearly the subgraph of $\mathbf{H}$ induced by $\{x, y, z, u, v, b\}$ is a 6-cycle, a contradiction. The argument for the second statement is identical. □

By Claim 6, there exists an ordering of $N(x) = \{b_0, \ldots, b_m\}$ such that $N(b_i) \subseteq N(b_j)$ if $i \leq j$, and an ordering of $T_1 = \{t_0, \ldots, t_M\}$ such that $N(t_i) \cap N(x) \subseteq N(t_j) \cap N(x)$ if $i \leq j$. Since $\mathbf{H}$ is connected, it is easy to see that b_m must be adjacent to t_M; and by Claim 1, b_m cannot be adjacent to t_0.

Claim 7. For every $t \in T_1$, either $N(t) \cap N(x) = N(x)$ or $N(t) \cap N(x) = \emptyset$.

Proof. Suppose this is not the case. Then there exists some $t \in T_1$ such that t is adjacent to b_m but not to b_0. Then for any $b \in B_2$ the sequence b_0, x, b_m, t, b, t_0 is an induced path of length 5, a contradiction. □

Let F denote the set of vertices $t \in T_1$ such that $N(t) \cap N(x) = N(x)$ and let E denote the set of vertices $t \in T_1$ such that $N(t) \cap N(x) = \emptyset$.

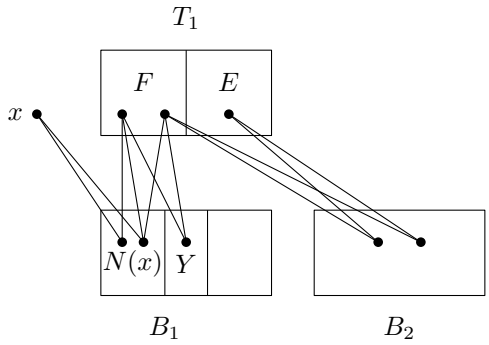


Figure 4.4: The graph **H**.

Claim 8. For every $y \in B_1 \setminus N(x)$, if y is adjacent to some vertex in E then it is adjacent to every vertex in F.

Proof. Otherwise we can find $t \in E$ and $t' \in F$ and $y \in B_1 \setminus N(x)$ such that (y, t) is an edge but (y, t') is not. Then for any $b \in B_2$ the sequence x, b_m, t', b, t, y is an induced path of length 5, a contradiction. □

Let Y denote the set of vertices in $B_1 \setminus N(x)$ that are adjacent to some vertex in E. By the last claim, the subgraph induced by $(Y \cup B_2 \cup N(x)) \cup F$ is complete bipartite. Consider the following decomposition: let $B_2' = Y \cup N(x) \cup B_2$, $B_1' = B_1 \setminus B_2'$, $T_1' = F$ and $T_2' = E \cup \{x\}$. By the above argument, this shows that **H** is in $\mathcal{I}_{\text{ir}}$, a contradiction. □

4.4.3 The case of general graphs

In this section we shall prove Theorem 58 which provides an inductive characterisation of $\mathcal{F}$, our main family of graphs.

Call graphs in $\mathcal{I}_{\text{ir}}$ (see Lemma 55) *basic irreflexive*.

Definition 56. A connected graph $\mathbf{H}$ is *basic* if

1. $\mathbf{H}$ is a single loop, or
2. $\mathbf{H}$ is a basic irreflexive graph, or
3. $\mathbf{H}$ is obtained from a basic irreflexive graph $\mathbf{H}_{\text{ir}}$ with colour classes B and T by adding every edge (including loops) of the form (t, t') where $t, t' \in T$.

Definition 57. Let $\mathcal{I}$ be the smallest class of graphs such that:

1. $\mathcal{I}$ contains the basic graphs;
2. $\mathcal{I}$ is closed under disjoint union;
3. if $\mathbf{H}_1$ is a basic graph and $\mathbf{H}_2 \in \mathcal{I}$ then $\mathbf{H}_1 \oslash \mathbf{H}_2 \in \mathcal{I}$.

Theorem 58. *The graph classes $\mathcal{F}$ and $\mathcal{I}$ coincide.*

Proof. To establish the inclusion $\mathcal{I} \subseteq \mathcal{F}$, we start by showing that every basic graph is in $\mathcal{F}$, i.e. that a basic graph does not contain any of the forbidden graphs. If $\mathbf{H}$ is a single loop or a basic irreflexive graph, then this is immediate. Otherwise $\mathbf{H}$ is obtained from a basic irreflexive graph $\mathbf{H}_{\text{ir}}$ with colour classes B and T by adding every edge of the form (t_1, t_2) where $t_i \in T$. In particular, the loops form a clique and no edge connects two non-loops; it is clear in that case that $\mathbf{H}$ contains none of **B1**, **B2**, **B3**, **B4**. On the other hand if $\mathbf{H}$ contains **B5** or **B6**, then $\mathbf{H}_{\text{ir}}$ contains the path of length 5 or the 6-cycle, contradicting the fact that $\mathbf{H}_{\text{ir}}$ is basic.

Next we show that $\mathcal{F}$ is closed under disjoint union and adjunction of basic graphs. It is obvious that the disjoint union of graphs that avoid the forbidden graphs will also avoid these. So suppose that an adjunction $\mathbf{H}_1 \oslash \mathbf{H}_2$, where $\mathbf{H}_1$ is a basic graph, contains an induced forbidden graph $\mathbf{B}$ whose vertices are neither all in H_1 nor H_2; without loss of generality H_1 contains at least one loop, its loops form a clique and none of its edges connects two non-loops. It is then easy to verify that $\mathbf{B}$ contains both loops and non-loops. Because the other cases are similar, we prove only that $\mathbf{B}$ is not **B3**. Observe that every loop in $\mathbf{H}_1$ is adjacent to every loop in $\mathbf{H}_1 \oslash \mathbf{H}_2$. So b, c, and d (see Figure 4.1) must be in H_2. But if a is in H_1, then it cannot be adjacent to a loop in H_2, so a is also in $\mathbf{H}_2$, a contradiction.

Now we must show that $\mathcal{F} \subseteq \mathcal{I}$, i.e. every graph in $\mathcal{F}$ can be obtained from the basic graphs by disjoint union and adjunction of basic graphs. Suppose this is not the case. If $\mathbf{H}$ is a counterexample of minimum size, then obviously it is connected, and it contains at least one loop for otherwise it is a basic irreflexive graph. By Lemma 53, $\mathbf{H}$ also contains at least one non-loop.

Let $\mathbf{R}(\mathbf{H})$ denote the subgraph of $\mathbf{H}$ induced by its set $R(H)$ of loops, and let $\mathbf{J}(\mathbf{H})$ denote the subgraph induced by $J(H)$, the set of non-loops of $\mathbf{H}$. Since $\mathbf{H}$ is connected and neither **B1** nor **B2** is an induced subgraph of $\mathbf{H}$, the graph $\mathbf{R}(\mathbf{H})$ is also connected, and furthermore every vertex in $J(H)$ is adjacent to some vertex in $R(H)$. By Lemma 53, we know that $\mathbf{R}(\mathbf{H})$ contains at least one universal vertex: let U denote the (non-empty) set of universal vertices of $\mathbf{R}(\mathbf{H})$. Let J denote the set of all $a \in J(H)$ such that

$N(a) \cap R(H) \subseteq U$. Let us show that $J \neq \emptyset$. For every $u \in U$, there is $w \in J(H)$ not adjacent to u because otherwise $\mathbf{H}$ is obtained by adjoining u to the rest of $\mathbf{H}$, a contradiction with the choice of $\mathbf{H}$. If this w has a neighbour $r \in R(H) \setminus U$ then there is some $s \in R(H) \setminus U$ not adjacent to r, and the graph induced by $\{w, u, s, r\}$ contains **B2** or **B3**, a contradiction. Hence, $w \in J$. Let $\mathbf{S}$ denote the subgraph of $\mathbf{H}$ induced by $U \cup J$. The graph $\mathbf{S}$ is connected. We claim that the following properties also hold:

1. if a and b are adjacent non-loops, then $N(a) \cap U = N(b) \cap U$;
2. if a is in a connected component of the subgraph of $\mathbf{S}$ induced by J with more than one vertex, then for any other $b \in J$, one of $N(a) \cap U, N(b) \cap U$ contains the other.

The first statement holds because **B1** is forbidden, and the second follows from the first because **B4** is also forbidden. Let $J_1, \ldots, J_k$ denote the different connected components of J in $\mathbf{S}$. By (1) we may let $N(J_i)$ denote the set of common neighbours of members of J_i in U. By (2), we can re-order the J_i's so that for some $1 \leq m \leq k$ we have $N(J_i) \subseteq N(J_j)$ for all $i \leq m$ and all $j > m$, and, in addition, we have $m = 1$ or $|J_i| = 1$ for all $1 \leq i \leq m$. Let $\mathbf{B}$ denote the subgraph of $\mathbf{S}$ induced by $B = \bigcup_{i=1}^{m} (J_i \cup N(J_i))$, and let $\mathbf{C}$ be the subgraph of $\mathbf{H}$ induced by $H \setminus B$. We claim that $\mathbf{H} = \mathbf{B} \oslash \mathbf{C}$. For this, it suffices to show that every element in $\bigcup_{i=1}^{m} N(J_i)$ is adjacent to every non-loop $c \in C$. By construction this holds if $c \in J \cap C$. Now suppose this does not hold: then some $x \in J(H) \setminus J$ is not adjacent to some $y \in N(J_i)$ for some

$i \leq m$. Since $x \notin J$ we may find some $z \in R(H) \setminus U$ adjacent to x; it is of course also adjacent to y. Since $z \notin U$ there exists some $z' \in R(H) \setminus U$ that is not adjacent to z, but it is of course adjacent to y. If x is adjacent to z', then $\{x, z, z'\}$ induces a subgraph isomorphic to **B2**, a contradiction. Otherwise, $\{x, z, y, z'\}$ induces a subgraph isomorphic to **B3**, also a contradiction.

If every J_i with $i \leq m$ contains a single element, notice that $\mathbf{B}$ is a basic graph: indeed, removing all edges between its loops yields a bipartite irreflexive graph which contains neither the path of length 5 nor the 6-cycle, since $\mathbf{B}$ contains neither **B5** nor **B6**. Since this contradicts our hypothesis on $\mathbf{H}$, we conclude that $m = 1$. But this means that $N(J_1)$ is a set of universal vertices in $\mathbf{H}$. Let u be such a vertex and let D denote its complement in $\mathbf{H}$: clearly $\mathbf{H}$ is obtained as the adjunction of the single loop u to D, contradicting our hypothesis. This concludes the proof. □

4.5 Algebraic results

We need the following well-known auxiliary result. Note that the assumption of conservativity of the algebra $\mathbb{A}_{\mathbf{T}}$ in it is not essential. Note also that the assumptions of this lemma effectively say that CSP($\mathbf{T}$) can simulate the graph k-colouring problem (with $k = |U|$) or the directed st-connectivity problem.

Lemma 59. *Let $\mathbf{S}, \mathbf{T}$ be structures such that the algebra $\mathbb{A}_{\mathbf{T}}$ is conservative, let $s_1, s_2 \in S$, and let $R = \{(f(s_1), f(s_2)) \mid f : \mathbf{S} \to \mathbf{T}\}$.*

1. *If* $R = \{(x, y) \in U^2 \mid x \neq y\}$ *for some subset* $U \subseteq T$ *with* $|U| \geq 3$ *then* $\mathcal{V}(\mathbb{A}_{\mathbf{T}})$ *admits the unary type.*

2. *If* $R = \{(t, t), (t, t'), (t', t')\}$ *for some distinct* $t, t' \in T$ *then* $\mathcal{V}(\mathbb{A}_{\mathbf{T}})$ *admits at least one of the following types: unary, lattice, semilattice.*

Proof. The assumption of this lemma implies that $\mathbb{A}_{\mathbf{T}}$ has a subalgebra (with universe U and $\{t, t'\}$, respectively) such that all operations of the subalgebra preserve the relation R. It is well known (see, e.g., [50]) that all conservative operations preserving the disequality relation on U are projections which proves the first statement, while it is easy to check that the order relation on a 2-element set (such as the relation R from the second statement) cannot admit operations satisfying identities (4.1)–(4.3), so one can use Lemma 45 to prove the second statement. □

The following lemma connects the characterisation of bi-arc graphs given in [12] with a type-omitting condition.

Lemma 60. *Let* $\mathbf{H}$ *be a graph. Then the following conditions are equivalent:*

1. *the variety* $\mathcal{V}(\mathbb{H})$ *omits the unary type;*

2. *the graph* $\mathbf{H}$ *admits a conservative majority operation;*

3. *the graph* $\mathbf{H}$ *is a bi-arc graph.*

Proof. The equivalence of (2) and (3) is from [12], and (2) implies (1) by Lemma 45, so the rest of this proof shows that (1) implies (3). We shall

use the following construction from [42]. Given a graph $\mathbf{H}$, let $\mathbf{K}$ denote the irreflexive bipartite graph obtained from $\mathbf{H}$ as follows: its vertices consist of two copies of the vertex set of $\mathbf{H}$, say $H' = \{x' : x \in H\}$ and $H'' = \{x'' : x \in H\}$, with edges (x', y'') iff (x, y) is an edge of $\mathbf{H}$. In other words, $\mathbf{K} = \mathbf{H} \times \mathbf{K}_2$ where $\mathbf{K}_2$ is the irreflexive edge. Let $\mathbb{K}$ denote the algebra associated with $\mathbf{K}^L$.

By putting together Proposition 3.1 of [42] and Corollary 4.6 of [41], one immediately obtains that $\mathbf{H}$ is a bi-arc graph if and only if $\mathbf{K}$ is chordal bipartite and contains no special edge-asteroids. We need not know what these two conditions on $\mathbf{K}$ mean - it is shown in (proofs of) Theorems 3.1 and 3.2 of [41] that if $\mathbf{K}$ fails to satisfy either of them then $\mathbf{T} = \mathbf{K}^L$ satisfies the conditions of Lemma 59(1) for suitable $\mathbf{S}$, s_1, s_2, and so $\mathcal{V}(\mathbb{K})$ admits the unary type. Hence, it only remains to show that the variety $\mathcal{V}(\mathbb{K})$ omits the unary type whenever $\mathcal{V}(\mathbb{H})$ does so.

It is well known (see, e.g., Corollary 3.3 in [79]), that the unary type is present in the variety generated by a conservative algebra $\mathbb{A}$ if and only if there exist elements a, b in the algebra such that each operation of $\mathbb{A}$ is a projection when restricted to $\{a, b\}$. So we may assume now that, for every 2-element subset $\{a, b\}$ of H, there is an operation of $\mathbb{H}$ that is not a projection when restricted to $\{a, b\}$, and we need to show the same for $\mathbb{K}$.

For each operation (say k-ary) operation f of $\mathbb{H}$, introduce a $(2k-1)$-ary operation g_f on K, as follows. Let $x = (x_1^{e_1}, \ldots, x_{2k-1}^{e_{2k-1}})$ be an element of K^{2k-1}, where $x_1, \ldots, x_{2k-1} \in H$ and $e_1, \ldots, e_{2k-1} \in \{', ''\}$. Then obviously

exactly one of $'$ or $''$ appears at least k times; let ϵ denote this symbol; let $i_1, \ldots, i_k$ denote the first k positions where it appears in the tuple x; then define

$$g_f(x) = f(x_{i_1}, \ldots, x_{i_k})^\epsilon.$$

It is clear that this is a well-defined operation on K, and it is easy to see that it preserves edges of $\mathbf{K}$; since f is conservative, so is g_f. Hence, g_f is an operation of $\mathbb{K}$.

Let $\{x^u, y^v\}$ be a 2-element subset of K. Suppose first that x^u and y^v belong to different colour classes of K: then the restriction of g_f to this subset satisfies the property

$$g_f(x^u, \ldots, x^u, y^v, x^u \ldots, x^u) = f(x, \ldots, x)^u = x^u$$

and similarly for $g(y^v, \ldots, y^v, x^u, y^v, \ldots, y^v)$. On the other hand if x^u and y^v are in the same colour class, then the restriction of g_f to $\{x^u, y^v\}$ coincides with that of f (with $k-1$ additional fictitious variables). It follows that in either case, the restriction of g_f is not a projection whenever the restriction of f is not. □

The following lemma establishes the implication (3)⇒(4) in Theorem 48.

Lemma 61. *If* $\mathbf{H} \notin \mathcal{F}$ *then* $\mathcal{V}(\mathbb{H})$ *admits a non-Boolean type.*

Proof. By Theorem 9.15 of [54], $\mathcal{V}(\mathbb{H})$ admits only the Boolean type if and only if $\mathbf{H}$ admits a sequence of conservative operations satisfying certain

identities in the spirit of (4.1)–(4.3). (Note that Theorem 9.15 of [54] applies to the so-called locally finite varieties, but every variety generated by a single finite algebra, such as $\mathcal{V}(\mathbb{H})$, has this property [70]). By conservativity, such operations can be restricted to any subset of H while satisfying the same identities, so the property of having only the Boolean type in the variety generated by their conservative algebra is inherited by induced subgraphs of $\mathbf{H}$. It follows that it is enough to prove this lemma for the forbidden graphs from Definition 47.

For the irreflexive odd cycles, the lemma follows immediately from the main results of [9, 68]. The proof of Theorem 3.1 of [41] shows that the conditions of Lemma 59(1) are satisfied by (some $\mathbf{S}, s_1, s_2$ and) $\mathbf{T} = \mathbf{F}^L$ where $\mathbf{F}$ is the irreflexive 6-cycle. One can easily check that the reflexive 4-cycle is not a bi-arc graph, so we can apply Lemma 60 in this case.

For the remaining forbidden graphs $\mathbf{F}$ from Definition 47, we use Lemma 59(2) with $\mathbf{T} = \mathbf{F}^L$. In each case, the binary relation of the structure $\mathbf{S}$ will be a short undirected path, and s_1, s_2 will be the endpoints of the path. We will represent such a structure $\mathbf{S}$ by a sequence of subsets of F (indicating lists assigned to vertices of the path). It can be easily checked that, in each case, the relation R defined as in Lemma 59(2) is of the required form.

If $\mathbf{F}$ is the reflexive path of length 3, say $a-b-c-d$, then $\mathbf{S} = ac-bc-ad-ac$. If $\mathbf{F}$ is the irreflexive path of length 5, say $a-b-c-d-e-f$, then $\mathbf{S} = ae-bd-ce-bf-ae$. For graphs $\mathbf{B1}-\mathbf{B6}$, we use notation from Figure 4.1. For $\mathbf{B1}$, $\mathbf{S} = bc-bc-ab-ab-bc$. For $\mathbf{B2}$, $\mathbf{S} = bc-ac-ab-bc$.

For **B3**, $\mathbf{S} = bc - ad - bd - bc$. For **B4**, $\mathbf{S} = ae - bd - cd - ae$. Finally, for both **B5** and **B6**, $\mathbf{S} = ac - b'c' - ab - a'c' - ac$. □

4.5.1 Implication (4) ⇒ (1) in Theorem 48

We prove this implication in two steps: first for irreflexive graphs and then in general.

Recall the definition of basic irreflexive graphs from Lemma 55 and Definition 51.

Lemma 62. *If* $\mathbf{H}$ *is a basic irreflexive graph then* $\mathbf{H}$ *admits conservative operations satisfying (4.1)–(4.3) for* $n = 3$.

Proof. We shall show by induction on the size of $\mathbf{H}$ that there exist conservative operations f_1, f_2, f_3 preserving the graph $\mathbf{H}$, obeying the identities (4.1)–(4.3) and furthermore that satisfy the following condition (D):

For every $x, y, z, n, m \in H$ *such that* n *is adjacent to* x *and* m *is adjacent to* z, $f_1(x, y, z)$ *is adjacent to* n *and* $f_3(x, y, z)$ *is adjacent to* m.

The result is trivial for a one-element graph. If $\mathbf{H}$ is not connected, then $\mathbf{H}$ is the disjoint union of proper subgraphs $\mathbf{H}_1$ and $\mathbf{H}_2$. Let f_1, f_2, f_3 and g_1, g_2, g_3 be the desired operations on $\mathbf{H}_1$ and $\mathbf{H}_2$ respectively; we define operations h_1, h_2, h_3 on $\mathbf{H}$ as follows:

For every $1 \leq s \leq 3$, let $h_s(x,y,z) = f_s(x,y,z)$ if $(x,y,z) \in H_1^3$ and let $h_s(x,y,z) = g_s(x,y,z)$ if $(x,y,z) \in H_2^3$; if $(x,y,z) \in H_i \times H_j \times H_k$ with i,j,k not all equal, then let $h_1(x,y,z) = x$ and $h_3(x,y,z) = z$, and finally let $h_2(x,y,z) = z$ if $(i,j,k) \in \{(1,1,2),(2,2,1)\}$ and let $h_2(x,y,z) = x$ otherwise.

It is immediate that identities (4.1) and (4.3) are satisfied and that each h_s is a conservative homomorphism. For (4.2): we may assume that $x \neq y$; if x and y are in the same H_i then (4.2) follows from the fact that the f_i and g_i satisfy it; otherwise we have that $h_1(x,x,y) = x = h_2(x,y,y)$ and $h_2(x,x,y) = y = h_3(x,y,y)$. It is easy to see that condition (D) is satisfied by h_1 and h_3.

Now suppose that the basic graph $\mathbf{H}$ is connected, and hence is the special sum of two smaller graphs. For the moment, it will be convenient to denote the colour classes of $\mathbf{H}$ by C_1 and C_2; our first task is to show it suffices to define our operations on $C_1^3 \cup C_2^3$. Indeed, suppose that we have functions $F_1', F_2', F_3' : C_1^3 \cup C_2^3 \to H$ that satisfy all the required identities, are edge-preserving and conservative. Then we may extend these to full operations $F_1, F_2, F_3 : H^3 \to H$ as follows: let

$$F_1(x,y,z) = \begin{cases} F_1'(x,y,z), & \text{if } (x,y,z) \in C_1^3 \cup C_2^3; \\ x, & \text{otherwise.} \end{cases}$$

$$F_3(x,y,z) = \begin{cases} F_3'(x,y,z), & \text{if } (x,y,z) \in C_1^3 \cup C_2^3; \\ z, & \text{otherwise.} \end{cases}$$

$$F_2(x,y,z) = \begin{cases} F_2'(x,y,z), & \text{if } (x,y,z) \in C_1^3 \cup C_2^3; \\ z, & \text{if } (x,y,z) \in C_i \times C_i \times C_j \text{ for some } i \neq j; \\ x, & \text{otherwise.} \end{cases}$$

Notice that distinct sets $C_i \times C_j \times C_k$ and $C_{i'} \times C_{j'} \times C_{k'}$ are in different connected components of $\mathbf{H}^3$, unless $i \neq i'$, $j \neq j'$ and $k \neq k'$; it follows immediately that the F_i are edge-preserving; they are also clearly conservative. It is a simple matter to verify that all the required identities are satisfied. Hence, from now on, we assume without mention that in all triples (x,y,z) considered all the entries come from the same colour class of the graph under consideration.

So let $\mathbf{H}$ be the special sum of two smaller graphs $\mathbf{H}_i$ with colour classes B_i and T_i, $i = 1,2$; by induction hypothesis $\mathbf{H}_1$ admits the required operations f_1, f_2, f_3 and $\mathbf{H}_2$ admits operations g_1, g_2, g_3 satisfying the necessary conditions. We define operations F_1, F_2, F_3 on $\mathbf{H}$ as follows. For convenience, let $S = T_1 \cup B_2$, $B = B_1 \cup B_2$, $T = T_1 \cup T_2$. Notice that by definition of special sum S induces a complete bipartite graph in $\mathbf{H}$.

$$F_1(x,y,z) = \begin{cases} f_1(x,y,z), & \text{if } x,y,z \in H_1, \text{ else} \\ g_1(x,y,z), & \text{if } x,y,z \in H_2, \text{ else} \\ x, & \text{if } y = z \text{ or } x \in S, \text{ else} \\ u, & \text{where } u \text{ is the leftmost of } \{y,z\} \cap S. \end{cases}$$

$$F_3(x,y,z) = \begin{cases} f_3(x,y,z), & \text{if } x,y,z \in H_1, \text{ else} \\ g_3(x,y,z), & \text{if } x,y,z \in H_2, \text{ else} \\ z, & \text{if } x = y \text{ or } z \in S, \text{ else} \\ v, & \text{where } v \text{ is the leftmost of } \{x,y\} \cap S. \end{cases}$$

$$F_2(x,y,z) = \begin{cases} F_1(x,x,z), & \text{if } y = z, \text{ else} \\ F_3(x,z,z), & \text{if } x = y, \text{ else} \\ f_2(x,y,z), & \text{if } x,y,z \in H_1, \text{ else} \\ g_2(x,y,z), & \text{if } x,y,z \in H_2, \text{ else} \\ w, & \text{where } w \text{ is the leftmost of } \{x,y,z\} \cap S. \end{cases}$$

Obviously all three operations are conservative, and by definition they obey all the required identities. Now we verify that F_1 satisfies condition (D): let (x,n) be an edge of $\mathbf{H}$: we show that $F_1(x,y,z)$ is adjacent to n. If $x,y,z \in H_i$ for some $i = 1,2$ then this follows by induction hypothesis,

and it is clearly true if $F_1(x, y, z) = x$. Otherwise, $F_1(x, y, z) = u$ for some $u \in S$; if $x, y, z \in B$ then $x \in B_1$ so $n \in T_1$ is adjacent to u. Otherwise $x, y, z \in T$, so $x \in T_2$ hence $n \in B_2$ is adjacent to u. The proof that F_3 satisfies (D) is identical. It remains to show that each F_i is edge-preserving. Let (x, y, z) be adjacent to (x', y', z') and suppose without loss of generality that $x, y, z \in B$ and $x', y', z' \in T$. We start with F_1. If $x, y, z \in B_1$ then $x', y', z' \in T_1$ and hence F_1 coincides with f_1 on both tuples and we are done by induction hypothesis. If $F_1(x, y, z) = x$ then by (D) we have $F_1(x', y', z')$ adjacent to x. Otherwise, we have that $x \in B_1$ (and thus $x' \in T_1$) and $F_1(x, y, z) = u \in B_2$; in any case $F_1(x', y', z') \in T_1$ so it is adjacent to u. The argument for F_3 is identical. Now we consider F_2. Notice that by induction hypothesis and definition of the F_i, we have that F_2 coincides with f_2 (or with g_2) on tuples whose coordinates all lie in H_1 (respectively H_2). If $x, y, z \in B_1$, then certainly $x', y', z' \in T_1$, and then the result follows by induction hypothesis and the last remark. Now we require the following claim:

Claim. Suppose that a, b, c do not all lie in the same H_i. If $b = c$ or $a = b$ then $F_2(a, b, c) \in S$.

Proof. Suppose that $b = c$, so that $F_2(a, b, c) = F_1(a, a, c)$. By hypothesis, a and c do not lie in the same H_i, and in particular they are distinct, hence by definition of F_1 we have that $F_1(a, a, c) = a$ if $a \in S$ or $F_1(a, a, c) = u$ for some $u \in S$ (here $u = c$ of course). The proof for the case $a = b$ is

identical. □

Now we can finish the proof. Suppose first that x, y, z are not all in the same H_i; by the claim $F_2(x, y, z) \in S$. If x', y', z' are not all in the same H_i then $F_2(x', y', z') \in S$ also and we are done. Otherwise, x', y', z' all lie in T_1 (since one of them is a neighbour of an element of B_1) and hence $F_2(x', y', z') = f_2(x', y', z') \in S$ and we are done. Now suppose that x, y, z are all in B_2 (we dealt with the case B_1 earlier.) Then $F_2(x, y, z) = g_2(x, y, z) \in S$, so if x', y', z' are not all in the same H_i we are done by the claim again. Otherwise either $x', y', z' \in T_1$ so $F_2(x', y', z') = f_2(x', y', z') \in S$, or else $x', y', z' \in T_2$: then $F_2(x', y', z') = g_2(x', y', z')$ and we are done by induction hypothesis. □

Lemma 63. *If* $\mathbf{H} \in \mathcal{F}$ *then* $\mathbf{H}$ *admits conservative operations satisfying (4.1)–(4.3) for* $n = 3$.

Proof. We invoke the characterisation of $\mathcal{F}$ from Theorem 58. We will prove that $\mathbf{H}$ has the required polymorphisms when $\mathbf{H}$ is a basic graph, and show that this property is preserved under disjoint union and adjunction of basic graphs.

Let $\mathbf{H}$ be a basic graph. The result is trivial if $\mathbf{H}$ is a single loop, and if $\mathbf{H}$ is a basic irreflexive graph then we invoke Lemma 62. So now assume that $\mathbf{H}$ is obtained from some basic irreflexive graph $\mathbf{H}_1$ with colour classes B and T by adding all edges (t, t') with $t, t' \in T$. By Lemma 62 there exist operations f_1, f_2, f_3 on $\mathbf{H}_1$ satisfying the required identities; furthermore recall that we

can assume that the f_i satisfy condition (D):

For every $x, y, z, n, m \in H$ such that n is adjacent to x and m is adjacent to z, $f_1(x, y, z)$ is adjacent to n and $f_3(x, y, z)$ is adjacent to m.

For convenience of notation, define, on triples (x_1, x_2, x_3) such that $\{x_1, x_2, x_3\}$ intersects the set T, two ternary operations μ^L and μ^R by $\mu^L(x_1, x_2, x_3) = x_j$ where $j = \min\{i : x_i \in T\}$ and $\mu^R(x_1, x_2, x_3) = x_k$ where $k = \max\{i : x_i \in T\}$. Notice that both of these operations trivially preserve the edges of $\mathbf{H}$. We define operations F_1, F_2 and F_3 on $\mathbf{H}$ as follows:

$$F_1(x_1, x_2, x_3) = \begin{cases} x_1, & \text{if } x_2 = x_3, \text{ else} \\ f_1(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \text{ intersects only one of } B \text{ and } T, \\ \mu^L(x_1, x_2, x_3), & \text{otherwise.} \end{cases}$$

$$F_3(x_1, x_2, x_3) = \begin{cases} x_3, & \text{if } x_1 = x_2, \text{ else} \\ f_3(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \text{ intersects only one of } B \text{ and } T, \\ \mu^R(x_1, x_2, x_3), & \text{otherwise.} \end{cases}$$

$$F_2(x_1,x_2,x_3) = \begin{cases} f_2(x_1,x_2,x_3), & \text{if } \{x_1,x_2,x_3\} \text{ intersects only one of } B \text{ and } T, \\ \mu^L(x_1,x_2,x_3), & \text{otherwise.} \end{cases}$$

It is clear that all three operations are conservative, and that identities (4.1) and (4.3) are satisfied. To prove (4.2), suppose without loss of generality that $x \neq y$: if $\{x,y\}$ intersects only one of B and T then $F_1(x,x,y) = f_1(x,x,y) = f_2(x,y,y) = F_2(x,y,y)$; on the other hand if $\{x,y\}$ intersects both B and T then $F_1(x,x,y) = F_2(x,y,y)$ is the unique element in $\{x,y\}$ that belongs to T. The proof that $F_2(x,x,y) = F_3(x,y,y)$ is similar.

Next we prove that property (D) holds for F_1 (the proof for F_3 is identical.) Let n be a neighbour of x_1. If $F_1(x_1,x_2,x_3) = x_1$ the result is trivial, and if $F_1(x_1,x_2,x_3) = f_1(x_1,x_2,x_3)$ then we are done because f_1 satisfies (D). If $F_1(x_1,x_2,x_3) = \mu^L(x_1,x_2,x_3)$, there are two cases: if $x_1 \in T$ then $F_1(x_1,x_2,x_3) = x_1$, otherwise $x_1 \in B$ forces $n \in T$ so n is necessarily adjacent to $\mu^L(x_1,x_2,x_3) \in T$.

Finally we show that F_1 is edge-preserving (the proof for F_2 and F_3 is identical.) Let (x_1,x_2,x_3) and (y_1,y_2,y_3) be adjacent. Suppose first that $x_2 = x_3$; then $F_1(x_1,x_2,x_3) = x_1$. If $y_2 = y_3$ there is nothing to show so we may assume that $y_2 \neq y_3$. Since f_1 has property (D) we may also assume with no loss of generality that $\{y_1,y_2,y_3\}$ intersects B and T and hence $F_1(y_1,y_2,y_3)$ is the leftmost y_i in T. If this is y_1 we're done, otherwise x_1 must be in the clique T and we are also done. So now suppose that $x_2 \neq x_3$

and $y_2 \neq y_3$. If the x_i all lie in B or all in T and the same holds for the y_i, then we are done since f_1 is a homomorphism and T is a clique. Otherwise suppose without loss of generality that $\{y_1, y_2, y_3\}$ intersects both B and T; then some x_i must be in T, and then in any case the values of F_1 on both triples lie in the clique T and hence are adjacent. This completes the proof for all basic graphs.

The proof for disjoint union is identical to the one in the irreflexive case (Lemma 62).

Finally we show that the property of admitting conservative operations satisfying (4.1)–(4.3) for $n = 3$ is preserved under adjunction of a basic graph. Let $\mathbf{H}_1$ be a basic graph, where L_1 and N_1 denote the set of loops and non-loops of $\mathbf{H}_1$ respectively, and let $\mathbf{H}_2$ satisfy our induction hypothesis, and let L_2 and N_2 denote the set of loops and non-loops of $\mathbf{H}_2$ respectively. We may assume that L_1 is non-empty, and hence it is a clique. Let g_1, g_2, g_3 be operations on $\mathbf{H}_2$ that satisfy all required identities and property (D). By our earlier analysis, we know there exist operations f_1, f_2, f_3 on the basic graph $\mathbf{H}_1$ that satisfy all required identities and property (D), and moreover satisfy the following condition (E):

$$\begin{gathered} \textit{If } \{x, y, z\} \textit{ intersects } L_1 \textit{ and } y \neq z \textit{ then} \\ f_1(x, y, z) \textit{ and } f_3(y, z, x) \textit{ belong to } L_1. \end{gathered}$$

For convenience of notation, define two ternary operations λ^L, λ^R on

triples (x_1, x_2, x_3) such that $\{x_1, x_2, x_3\}$ intersects the set L_1 by $\lambda^L(x_1, x_2, x_3) = x_j$ where $j = \min\{i : x_i \in L_1\}$ and $\lambda^R(x_1, x_2, x_3) = x_k$ where $k = \max\{i : x_i \in L_1\}$. Define two ternary operations ν^L and ν^R on triples (x_1, x_2, x_3) such that $\{x_1, x_2, x_3\}$ intersects the set H_2 by $\nu^L(x_1, x_2, x_3) = x_j$ where $j = \min\{i : x_i \in H_2\}$ and $\nu^R(x_1, x_2, x_3) = x_k$ where $k = \max\{i : x_i \in H_2\}$. Notice that λ^L and λ^R are trivially edge-preserving, and so are ν^L and ν^R if we restrict them to triples (x_1, x_2, x_3) such that $\{x_1, x_2, x_3\} \subseteq N_1 \cup H_2$.

We define operations F_1, F_2 and F_3 on $\mathbf{H}$ as follows:

$$F_1(x_1, x_2, x_3) = \begin{cases} x_1, & \text{if } x_2 = x_3, \text{ else} \\ f_1(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \subseteq H_1, \text{ else} \\ g_1(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \subseteq H_2, \text{ else} \\ \lambda^L(x_1, x_2, x_3), & \text{if } \{x_1, x_2, x_3\} \text{ intersects both } L_1 \text{ and } H_2, \\ \nu^L(x_1, x_2, x_3), & \text{otherwise.} \end{cases}$$

$$F_3(x_1,x_2,x_3) = \begin{cases} x_3, & \text{if } x_1 = x_2, \text{ else} \\ f_3(x_1,x_2,x_3), & \text{if } \{x_1,x_2,x_3\} \subseteq H_1, \text{ else} \\ g_3(x_1,x_2,x_3), & \text{if } \{x_1,x_2,x_3\} \subseteq H_2, \text{ else} \\ \lambda^R(x_1,x_2,x_3), & \text{if } \{x_1,x_2,x_3\} \text{ intersects both } L_1 \text{ and } H_2, \\ \nu^R(x_1,x_2,x_3), & \text{otherwise.} \end{cases}$$

$$F_2(x_1,x_2,x_3) = \begin{cases} f_2(x_1,x_2,x_3), & \text{if } \{x_1,x_2,x_3\} \subseteq H_1, \text{ else} \\ g_2(x_1,x_2,x_3), & \text{if } \{x_1,x_2,x_3\} \subseteq H_2, \text{ else} \\ \lambda^L(x_1,x_2,x_3), & \text{if } \{x_1,x_2,x_3\} \text{ intersects both } L_1 \text{ and } H_2, \\ \nu^L(x_1,x_2,x_3), & \text{otherwise.} \end{cases}$$

It is clear that each F_i is conservative and that identities (4.1) and (4.3) are satisfied. To prove (4.2), suppose without loss of generality that $x \neq y$: if $\{x,y\}$ is contained in H_1 or contained in H_2 then the result follows from the fact that the f_i and g_i satisfy (4.2); if $\{x,y\}$ intersects both L_1 and H_2 then $F_1(x,x,y) = F_2(x,y,y)$ is the unique element in $\{x,y\}$ that belongs to L_1; if $\{x,y\}$ intersects both N_1 and H_2 then $F_1(x,x,y) = F_2(x,y,y)$ is the unique element in $\{x,y\}$ that belongs to H_2. The proof that $F_2(x,x,y) = F_3(x,y,y)$ is similar.

Next we prove that property (D) holds for F_1 (the proof for F_3 is identi-

cal.) Let n be a neighbour of x_1. If $F_1(x_1, x_2, x_3) = x_1$ the result is trivial, and if $\{x_1, x_2, x_3\}$ is contained in H_1 or contained in H_2 then the result follows from the fact that both f_i and g_i satisfy (D). Suppose now that $\{x_1, x_2, x_3\}$ intersects both L_1 and H_2. Then $F_1(x_1, x_2, x_3) \in L_1$; in particular if $n \in H_2 \cup L_1$ we are done. If on the other hand $n \in N_1$ then $x_1 \in L_1$ so $F_1(x_1, x_2, x_3) = \lambda^L(x_1, x_2, x_3) = x_1$. If $F_1(x, y, z) = \nu^L(x, y, z) \neq x$ then $x \in N_1$, $n \in L_1$, and $\nu^L(x, y, z) \in H_2$, so we are done.

Finally we show that F_1 is edge-preserving (the proof for F_2 and F_3 is identical.) Let (x_1, x_2, x_3) and (y_1, y_2, y_3) be adjacent. We analyse the different cases. Without loss of generality we may assume throughout that $y_2 \neq y_3$.

(1) Suppose first that $x_2 = x_3$ so that $F_1(x_1, x_2, x_3) = x_1$. (a) If $\{y_1, y_2, y_3\} \subseteq H_1$, then $F_1(y_1, y_2, y_3) = f_1(y_1, y_2, y_3)$; either $x_1 \in H_2$, forcing $y_1 \in L_1$ so by property (E) we have that $F_1(y_1, y_2, y_3) \in L_1$ adjacent to x_1, or else $x_1 \in H_1$ and so property (D) guarantees $F_1(y_1, y_2, y_3)$ adjacent to x_1. (b) If $\{y_1, y_2, y_3\} \subseteq H_2$, then $F_1(y_1, y_2, y_3) = g_1(y_1, y_2, y_3)$; if $x_1 \in H_1$ then it is in L_1 and is adjacent to $F_1(y_1, y_2, y_3)$; otherwise $x_1 \in H_2$ and property (D) applies. (c) If $\{y_1, y_2, y_3\}$ intersects both L_1 and H_2, then $F_1(y_1, y_2, y_3)$ returns the leftmost entry which is in L_1; hence if x_1 is not in N_1 then it is adjacent to $F_1(y_1, y_2, y_3)$. If $x_1 \in N_1$ then $y_1 \in L_1$ so $F_1(y_1, y_2, y_3) = y_1$ and we are done. (d) Suppose that $\{y_1, y_2, y_3\}$ intersects both H_1 and H_2 but not L_1: then $F_1(y_1, y_2, y_3)$ returns the leftmost entry in H_2; if x_1 is in H_1 then it must be in L_1; otherwise $x_1 \in H_2$ forces $y_1 \in H_2$; in both cases x_1 is adjacent

to $F_1(y_1, y_2, y_3)$.

From now on we may assume that $x_2 \neq x_3$.

(2) Suppose $\{x_1, x_2, x_3\} \subseteq H_1$. (a) If $\{y_1, y_2, y_3\} \subseteq H_2$, then $x_i \in L_1$ for all i, so $F_1(x_1, x_2, x_3) \in L_1$ by property (E), so we are done. (b) If $\{y_1, y_2, y_3\}$ intersects both L_1 and H_2, then $F_1(y_1, y_2, y_3)$ returns the leftmost entry which is in L_1. There is some i such that $y_i \in H_2$, hence $x_i \in L_1$; by (E) it follows that $F_1(x_1, x_2, x_3) \in L_1$ and is adjacent to $F_1(y_1, y_2, y_3)$. (c) Suppose that $\{y_1, y_2, y_3\}$ intersects both H_1 and H_2 but not L_1; then $F_1(y_1, y_2, y_3)$ returns the leftmost entry in H_2. There is some i such that $y_i \in H_2$, hence $x_i \in L_1$; by (E) again $F_1(x_1, x_2, x_3) \in L_1$ and is adjacent to $F_1(y_1, y_2, y_3)$.

(3) Suppose $\{x_1, x_2, x_3\} \subseteq H_2$. By the previous cases we may assume that $\{y_1, y_2, y_3\}$ intersects both H_1 and H_2; in this case it must also intersect L_1, hence $F_1(y_1, y_2, y_3)$ returns the leftmost entry which is in L_1, which is adjacent to every vertex in H_2.

(4) Suppose $\{x_1, x_2, x_3\}$ intersects both L_1 and H_2. If the same holds for $\{y_1, y_2, y_3\}$, the result follows from the fact that λ^L is edge-preserving. We may now assume that $\{y_1, y_2, y_3\}$ intersects both H_1 and H_2 but not L_1. Then by definition $F_1(x_1, x_2, x_3) \in L_1$ and $F_1(y_1, y_2, y_3) \in H_2$, and hence are adjacent.

(5) Finally, suppose that each of $\{x_1, x_2, x_3\}$ and $\{y_1, y_2, y_3\}$ intersects both H_1 and H_2, but not L_1. Then the result follows from the fact that ν^L is edge-preserving. □

4.6 Symmetric Datalog constructions

The goal of this section is to prove the following lemma.

Lemma 64. *If* $\mathbf{H} \in \mathcal{F}$ *then* co-CSP($\mathbf{H}^L$) *is in symmetric Datalog.*

Recall that $\mathcal{F} = \mathcal{I}$ by Theorem 58, and we will use the inductive definition of this class in this section. We start by describing a method to compose symmetric Datalog programs.

4.6.1 Composing symmetric Datalog programs

The output of a Datalog program over τ with a set of IDBs I is a structure over the extended signature $\tau \cup I$. Such a structure can naturally be fed as input to another Datalog program working over this extended signature and using a set J of IDBs disjoint from I. The result is a structure over the signature $\tau \cup I \cup J$. Of course, the effect of this composition can be obtained by simply merging the list of rules of the two programs. However, this naive construction does not preserve the linearity or symmetricity of the programs. The following lemma shows that in fact symmetricity can be preserved at the cost of an increase in the arity of the IDBs.

Lemma 65. *Let* $\mathscr{D}$ *be a symmetric Datalog program over the signature* $\tau = \{E_1, \ldots, E_k\}$ *that outputs IDBs* $I_1, \ldots, I_t$ *of respective arities* $a_1, \ldots, a_t$*. Let* $\ulcorner I_{j_1} \wedge \ldots \wedge I_{j_s} \urcorner$ *denote the relation of arity* $a_{j_1} + \ldots + a_{j_s}$ *which is the Cartesian product of the (not necessarily distinct) relations* $I_{j_1}, \ldots, I_{j_s}$*. For any* c*, there*

exists a symmetric program $\mathscr{D}_c$ over the signature τ which correctly derives all the $\ulcorner I_{j_1} \wedge \ldots \wedge I_{j_s} \urcorner$ with $s \leq c$.

Proof. It suffices to show that this holds when $c = 2$ since the more general statement can be obtained by iterating the construction detailed below. Note that we consider $\ulcorner I_k \wedge I_\ell \urcorner$ as a single[4] IDB. We use $\bar{x}, \bar{y}$ and so on to denote tuples of variables and say that $\bar{x}$ and $\bar{y}$ are *disjoint* if they share no variable. We also use $E(\bar{w})$ to denote some conjunction of EDBs. We construct $\mathscr{D}_2$ with the following rules.

1. Original rules of $\mathscr{D}$ are kept.

2. If $I_j(\bar{x}) \leftarrow E(\bar{w})$ is a non-recursive rule in $\mathscr{D}$, we include for any $1 \leq q \leq t$ the rule

$$\ulcorner I_j(\bar{x}) \wedge I_q(\bar{y}) \urcorner \leftarrow I_q(\bar{y}) \wedge E(\bar{w})$$

   where $\bar{y}$ is disjoint from $\bar{x}$ and $\bar{w}$. We also include the symmetric rule

$$I_q(\bar{y}) \leftarrow \ulcorner I_j(\bar{x}) \wedge I_q(\bar{y}) \urcorner \wedge E(\bar{w}).$$

3. Finally, if $I_j(\bar{x}) \leftarrow I_k(\bar{y}) \wedge E(\bar{w})$ is a recursive rule of $\mathscr{D}$, we include for any $1 \leq q \leq t$ the rule

$$\ulcorner I_q(\bar{z}) \wedge I_j(\bar{x}) \urcorner \leftarrow \ulcorner I_q(\bar{z}) \wedge I_k(\bar{y}) \urcorner \wedge E(\bar{w})$$

[4] As stated, the lemma distinguishes the IDBs $\ulcorner I_j \wedge I_k \urcorner$ and $\ulcorner I_k \wedge I_j \urcorner$. However, the two are clearly equivalent from a computational perspective. To simplify our description of $\mathscr{D}_2$, we thus implicitly assume that any rule involving $\ulcorner I_j \wedge I_k \urcorner$ is accompanied by the counterpart rule using $\ulcorner I_k \wedge I_j \urcorner$.

where $\bar{z}$ is disjoint from $\bar{x}, \bar{y}$ and $\bar{w}$. Because $\mathscr{D}$ is symmetric, we know that the symmetric of the above rule also appears in $\mathscr{D}_2$.

By construction $\mathscr{D}_2$ is symmetric. We claim that it computes the product relations correctly. It can be easily seen using induction that all rules above are sound, i.e. there is a derivation in $\mathscr{D}_2$ for any $\ulcorner I_q(\bar{z}) \wedge I_j(\bar{x})\urcorner$ only if there are derivations for $I_q(\bar{z})$ and $I_j(\bar{x})$ in $\mathscr{D}$. For example, assume that $\ulcorner I_q(\bar{z}) \wedge I_j(\bar{x})\urcorner$ is derived in $\mathscr{D}_2$ using the rule $\ulcorner I_q(\bar{z}) \wedge I_j(\bar{x})\urcorner \leftarrow \ulcorner I_q(\bar{z}) \wedge I_k(\bar{y})\urcorner \wedge E(\bar{w})$ of type (3), and we already established that $I_q(\bar{z})$ and $I_k(\bar{y})$ are derived in $\mathscr{D}$. Then $I_j(\bar{x})$ (and $I_q(\bar{z})$) can be derived in $\mathscr{D}$ by the definition of rules of type (3). The same argument shows that $\mathscr{D}$ and $\mathscr{D}_2$ derive the same IDBs $I_1, \ldots, I_t$. In fact, it is convenient to view the execution of $\mathscr{D}_2$ as a two-stage process where the original IDBs are derived first.

It remains to show that if there are derivations in $\mathscr{D}$ for $I_j(\bar{x})$ and $I_k(\bar{y})$ then there is a derivation of $\ulcorner I_j(\bar{x}) \wedge I_k(\bar{y})\urcorner$ in $\mathscr{D}_2$. Note first that since there is a derivation of $I_k(\bar{y})$ in $\mathscr{D}$, that same derivation exists in $\mathscr{D}_2$ (this is the purpose of rules of type (1)). Now let

$$\rightarrow I_{j_1}(\bar{x}_1) \rightarrow \ldots \rightarrow I_{j_n}(\bar{x}_n) \rightarrow I_j(\bar{x})$$

denote the sequence of IDBs used in the derivation of $I_j(\bar{x})$ in $\mathscr{D}$. Suppose that $I_{j_1}(\bar{x}_1)$ is derived in $\mathscr{D}$ by instantiating a first-order rule

$$I_{j_1}(\bar{x}_1) \leftarrow E(\bar{w}_0).$$

The rules of type (2) provide a corresponding derivation of $\ulcorner I_{j_1}(\bar{x}_1) \wedge I_k(\bar{y})\urcorner$ in $\mathscr{D}_2$ through

$$\ulcorner I_{j_1}(\bar{x}_1) \wedge I_k(\bar{y})\urcorner \leftarrow I_k(\bar{y}) \wedge E(\bar{w}_0).$$

Similarly, suppose that the derivation of $I_{j_2}(\bar{x}_2)$ in $\mathscr{D}$ is given by

$$I_{j_2}(\bar{x}_2) \leftarrow I_{j_1}(\bar{x}_1) \wedge E(\bar{w}_1).$$

The rules of type (3) provide a corresponding derivation of $\ulcorner I_{j_2}(\bar{x}_2) \wedge I_k(\bar{y})\urcorner$ in $\mathscr{D}_2$ through

$$\ulcorner I_{j_2}(\bar{x}_2) \wedge I_k(\bar{y})\urcorner \leftarrow \ulcorner I_{j_1}(\bar{x}_1) \wedge I_k(\bar{y})\urcorner \wedge E(\bar{w}_1).$$

Thus, we can successively derive $\ulcorner I_{j_{t+1}}(\bar{x}_{t+1}) \wedge I_k(\bar{y})\urcorner$, from $\ulcorner I_{j_t}(\bar{x}_t) \wedge I_k(\bar{y})\urcorner$ and ultimately obtain a derivation of $\ulcorner I_j(\bar{x}) \wedge I_k(\bar{y})\urcorner$. □

This construction is used to prove the following lemma which, intuitively, proves that symmetric Datalog programs can be composed in a way that preserves the symmetry.

Lemma 66. *Let $\mathscr{D}$ be a symmetric Datalog program over the signature $\tau = \{E_1, \ldots, E_q\}$, and assume that the set of IDBs of $\mathscr{D}$ is $I = \{I_1, \ldots, I_s\}$, with respective arities $a_1, \ldots, a_s$. Further, let $\mathscr{E}$ be a symmetric Datalog program over the signature $\tau' = \tau \cup I$, and assume that the set of IDBs of $\mathscr{E}$ is $J = \{J_1, \ldots, J_t\}$, with respective arities $b_1, \ldots, b_t$. For a τ-structure $\mathbf{H}$, let $\mathbf{H}'$ denote the τ'-structure defined by $\mathscr{D}(\mathbf{H})$. One can construct a symmetric*

Datalog program $\mathscr{F}$ over the original signature τ with the following properties:

a) the IDBs I and J of $\mathscr{D}$ and $\mathscr{E}$ are IDBs in $\mathscr{F}$;

b) $I_k(\bar{x})$ is derived in $\mathscr{F}(\mathbf{H})$ iff $I_k(\bar{x})$ is derived in $\mathscr{D}(\mathbf{H})$;

c) $J_\ell(\bar{x})$ is derived in $\mathscr{F}(\mathbf{H})$ iff $J_\ell(\bar{x})$ is derived in $\mathscr{E}(\mathbf{H}')$.

Proof. Let c be such that each rule of $\mathscr{E}$ uses at most c EDBs in $\tau' - \tau$, i.e. at most c of the IDBs of $\mathscr{D}$. By the previous lemma, we can assume that $c = 1$ since we can otherwise construct $\mathscr{D}_c$ and thus get relations that represent any conjunction of the I_j.

The IDBs of our new program $\mathscr{F}$ include the IDBs of $\mathscr{D}$ and $\mathscr{E}$ (i.e. the IDBs in $I \cup J$) as well as auxiliary IDBs of the form $\ulcorner J_\ell \wedge I_k \urcorner$ for any $1 \leq k \leq s$ and $1 \leq \ell \leq t$. The rules of $\mathscr{F}$ are constructed as follows.

1. Every rule of $\mathscr{D}$ is kept.

2. All rules of $\mathscr{E}$ which use only EDBs of the original signature τ are kept.

3. For any non-recursive rule of $\mathscr{D}$, say $I_k(\bar{x}) \leftarrow E(\bar{w})$, we include for each $1 \leq \ell \leq t$ the symmetric pair of rules

$$\ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner \leftarrow J_\ell(\bar{y}) \wedge E(\bar{w})$$

$$J_\ell(\bar{y}) \leftarrow \ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner \wedge E(\bar{w})$$

where $\bar{y}$ is disjoint from $\bar{x}$ and $\bar{w}$.

4. For any recursive rule of $\mathscr{D}$, say $I_{k_1}(\bar{x}_1) \leftarrow I_{k_2}(\bar{x}_2) \wedge E(\bar{w})$, we include for each $1 \leq \ell \leq t$ the rule

$$\ulcorner J_\ell(\bar{y}) \wedge I_{k_1}(\bar{x}_1) \urcorner \leftarrow \ulcorner J_\ell(\bar{y}) \wedge I_{k_2}(\bar{x}_2) \urcorner \wedge E(\bar{w})$$

where $\bar{y}$ is disjoint from $\bar{x}_1, \bar{x}_2$ and $\bar{w}$. The symmetricity of $\mathscr{D}$ ensures that $\mathscr{F}$ contains the symmetric rule:

$$\ulcorner J_\ell(\bar{y}) \wedge I_{k_2}(\bar{x}_2) \urcorner \leftarrow \ulcorner J_\ell(\bar{y}) \wedge I_{k_1}(\bar{x}_1) \urcorner \wedge E(\bar{w}).$$

5. For any non-recursive rule of $\mathscr{E}$ that uses one of the I_k as an EDB, say $J_\ell(\bar{y}) \leftarrow I_k(\bar{x}) \wedge E(\bar{w})$, we include the symmetric pair of (recursive) rules

$$\ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner \leftarrow I_k(\bar{x}) \wedge E(\bar{w})$$

$$I_k(\bar{x}) \leftarrow \ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner \wedge E(\bar{w}).$$

6. For any non-recursive rule of $\mathscr{E}$ that does not use any of the I_k as an EDB, say $J_\ell(\bar{y}) \leftarrow E(\bar{w})$, we include for each $1 \leq k \leq s$ the symmetric pair of (recursive) rules

$$\ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner \leftarrow I_k(\bar{x}) \wedge E(\bar{w})$$

$$I_k(\bar{x}) \leftarrow \ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner \wedge E(\bar{w}).$$

7. For any recursive rule of $\mathscr{E}$ that use some I_k as an EDB, say $J_{\ell_1}(\bar{y}_1) \leftarrow J_{\ell_2}(\bar{y}_2) \wedge I_k(\bar{x}) \wedge E(\bar{w})$, we include the rule

$$\ulcorner J_{\ell_1}(\bar{y}_1) \wedge I_k(\bar{x}) \urcorner \leftarrow \ulcorner J_{\ell_2}(\bar{y}_2) \wedge I_k(\bar{x}) \urcorner \wedge E(\bar{w}).$$

Because $\mathscr{E}$ is symmetric, we know that $\mathscr{F}$ also includes the rule

$$\ulcorner J_{\ell_2}(\bar{y}_2) \wedge I_k(\bar{x}) \urcorner \leftarrow \ulcorner J_{\ell_1}(\bar{y}_1) \wedge I_k(\bar{x}) \urcorner \wedge E(\bar{w}).$$

8. For any recursive rule of $\mathscr{E}$ that does not use an I_j as an EDB, say $J_{\ell_1}(\bar{y}_1) \leftarrow J_{\ell_2}(\bar{y}_2) \wedge E(\bar{w})$, we include for each $1 \leq k \leq s$ the rule

$$\ulcorner J_{\ell_1}(\bar{y}_1) \wedge I_k(\bar{x}) \urcorner \leftarrow \ulcorner J_{\ell_2}(\bar{y}_2) \wedge I_k(\bar{x}) \urcorner \wedge E(\bar{w})$$

where $\bar{x}$ is disjoint from $\bar{y}_1, \bar{y}_2$ and $\bar{w}$. Because $\mathscr{E}$ is symmetric, we know that $\mathscr{F}$ also includes the rule

$$\ulcorner J_{\ell_2}(\bar{y}_2) \wedge I_k(\bar{x}) \urcorner \leftarrow \ulcorner J_{\ell_1}(\bar{y}_1) \wedge I_k(\bar{x}) \urcorner \wedge E(\bar{w}).$$

We claim that $\mathscr{F}$ has the desired properties. Again, we first note that all the rules are sound: if there is a derivation in $\mathscr{F}(\mathbf{H})$ for $I_k(\bar{x})$ (resp. $J_\ell(\bar{y})$; $\ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner$) then there is a derivation for $I_k(\bar{x})$ in $\mathscr{D}(\mathbf{H})$ (resp. a derivation for $J_\ell(\bar{y})$ in $\mathscr{E}(\mathbf{H}')$; derivations for $I_k(\bar{x})$ in $\mathscr{D}(\mathbf{H})$ and for $J_\ell(\bar{y})$ in $\mathscr{E}(H')$). In other words, none of the above rules can produce unwanted

tuples.

It remains to show that $\mathscr{F}$ is complete, i.e. that if there exists a derivation for $I_k(\bar{x})$ in $\mathscr{D}(\mathbf{H})$ and a derivation for $J_\ell(\bar{y})$ in $\mathscr{E}(\mathbf{H}')$ then there are derivations in $\mathscr{F}(\mathbf{H})$ for $I_k(\bar{x})$, $J_\ell(\bar{y})$ and $\ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner$. This is obvious for $I_k(\bar{x})$ since the original rules of $\mathscr{D}$ are also rules of $\mathscr{F}$. Similarly, rules of type (2) yield the claim if the derivation of $J_\ell(\bar{y})$ in $\mathscr{E}(\mathbf{H}')$ never uses one of the I_j as an EDB.

The non-trivial case consists of derivations of $\mathscr{E}(\mathbf{H}')$ which use the I_j as EDBs. The crux of the argument rests on the possibility of "inverting" any sequence of derivation steps in a symmetric program. Consider the sequence of IDBs in a valid derivation path in $\mathscr{D}(\mathbf{H})$:

$$I_{k_1}(\bar{x}_1) \rightarrow I_{k_2}(\bar{x}_2) \rightarrow \ldots \rightarrow I_{k_n}(\bar{x}_n).$$

If the ith step is obtained as $I_{k_{i+1}}(\bar{x}_{i+1}) \leftarrow I_{k_i}(\bar{x}_i) \wedge E(\bar{w}_i)$ then the symmetricity of $\mathscr{D}$ ensures that $I_{k_i}(\bar{x}_i) \leftarrow I_{k_{i+1}}(\bar{x}_{i+1}) \wedge E(\bar{w}_i)$ is also a valid derivation step in $\mathscr{D}(\mathbf{H})$ and the sequence

$$I_{k_n}(\bar{x}_n) \rightarrow I_{k_{n-1}}(\bar{x}_{k_{n-1}}) \rightarrow \ldots \rightarrow I_{k_1}(\bar{x}_1)$$

also corresponds to a valid derivation path. In other words, if $\mathscr{D}(\mathbf{H})$ can produce a derivation of $I_{k_n}(\bar{x}_n)$ from $I_{k_1}(\bar{x}_1)$ then it can also produce a derivation of $I_{k_1}(\bar{x}_1)$ from $I_{k_n}(\bar{x}_n)$.

We begin with the following claim.

Claim 1. Assume that there exists a derivation of $I_k(\bar{x})$ in $\mathscr{D}(H)$. Then there exists a derivation of $J_\ell(\bar{y})$ in $\mathscr{F}(H)$ iff there exists a derivation of $\ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner$ in $\mathscr{F}(H)$.

Proof. In the left to right implication, we assume that $J_\ell(\bar{y})$ is derived in $\mathcal{F}(H)$ and use a simple induction on the length of the derivation of $I_k(\bar{x})$ in $\mathcal{D}(H)$. If $I_k(\bar{x})$ is derived from a non-recursive rule then the derivation of $\ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner$ in $\mathcal{F}(H)$ is obtained through a rule of type (3). The induction step is obtained through rules of type (4).

The right to left implication is established through the same basic idea but using the inverse path of derivation. Assume that $\ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner$ can be derived in $\mathscr{F}(\mathbf{H})$. If $I_k(\bar{x})$ is derived from a non-recursive rule then we can derive $J_\ell(\bar{y})$ from $\ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner$ because of the symmetric rule of type (3). If the derivation of $I_k(\bar{x})$ in $\mathscr{D}(\mathbf{H})$ is of length at least 2 then consider the last derivation step, say:

$$I_k(\bar{x}) \leftarrow I_{k'}(\bar{x}') \wedge E(\bar{w}).$$

By induction, there exists a derivation from $\ulcorner J_\ell(\bar{y}) \wedge I_{k'}(\bar{x}') \urcorner$ to $J_\ell(\bar{y})$ and the symmetric rule of type (4) provides the missing derivation step:

$$\ulcorner J_\ell(\bar{y}) \wedge I_{k'}(\bar{x}') \urcorner \leftarrow \ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x}) \urcorner \wedge E(\bar{w}).$$

This completes the proof of the claim. □

Using Claim 1, we can complete the proof of the lemma by showing that any derivation of $J_\ell(\bar{y})$ in $\mathscr{E}(\mathbf{H}')$ has a corresponding derivation of $J_\ell(\bar{y})$ in $\mathscr{F}(\mathbf{H})$. Suppose that the derivation of $J_\ell(\bar{y})$ in $\mathscr{E}(\mathbf{H}')$ has length n. If $n = 1$ then $J_\ell(\bar{y})$ is derived in $\mathscr{E}(\mathbf{H}')$ from a non-recursive rule which may or may not use one of the I_j as an EDB. By using a rule of type (5) or (6), we can obtain in $\mathscr{F}(\mathbf{H})$ a derivation for some $\ulcorner J_\ell(\bar{y}) \wedge I_k(\bar{x})\urcorner$ where $I_k(\bar{x})$ has a derivation[5] in $\mathscr{D}(\mathbf{H})$. It then follows from Claim 1 that we can obtain in $\mathscr{F}(\mathbf{H})$ a derivation for $J_\ell(\bar{y})$ as well as derivations for *any* of the $\ulcorner J_\ell(\bar{y}) \wedge I_{k'}(\bar{x}')\urcorner$ when $I_{k'}(\bar{x}')$ has a derivation in $\mathscr{D}(\mathbf{H})$.

For the induction step, take $n \geq 2$ and suppose the last step in the derivation of $J_\ell(\bar{y})$ in $\mathscr{E}(\mathbf{H}')$ is given by

$$J_\ell(\bar{y}) \leftarrow J_{\ell_n}(\bar{y}_n) \wedge I_{k_n}(\bar{x}_n) \wedge E(\bar{w}_n).$$

We know by the induction hypothesis that there is a derivation in $\mathscr{F}(\mathbf{H})$ for $\ulcorner J_{\ell_n}(\bar{y}_n) \wedge I_{k_n}(\bar{x}_n)\urcorner$. A rule of type (7) can now complete the derivation of $\ulcorner J_\ell(\bar{y}) \wedge I_{k_n}(\bar{x}_n)\urcorner$ in $\mathscr{F}(\mathbf{H})$:

$$\ulcorner J_\ell(\bar{y}) \wedge I_{k_n}(\bar{x}_n)\urcorner \leftarrow \ulcorner J_{\ell_n}(\bar{y}_n) \wedge I_{k_n}(\bar{x}_n)\urcorner \wedge E(\bar{w}_n).$$

Finally, Claim 1 ensures that $J_\ell(\bar{y})$ itself can be derived in $\mathscr{F}(\mathbf{H})$. The case

[5]Note that we can assume without loss of generality that there is at least one such $I_k(\bar{x})$. If all the I_k are empty in $\mathscr{D}(\mathbf{H})$ then derivations in $\mathscr{E}(\mathbf{H}')$ can only be constructed from rules that do not use the I_k as EDBs and, as we remarked earlier, the rules of type (2) cover this case.

where the last derivation step of $J_\ell(\bar{y})$ in $\mathscr{E}(\mathbf{H}')$ does not rely on one of the I_j as an EDB is covered by rules of type (8). □

4.6.2 Symmetric programs for the list-homomorphism problem for graphs in $\mathcal{F}$

Our objective is to show that for any undirected graph $\mathbf{H}$ in $\mathcal{F}$ the set co-CSP($\mathbf{H}^L$) of digraphs with $\mathbf{H}$-lists that do not map homomorphically to $\mathbf{H}^L$ is definable in symmetric Datalog. We proceed by induction on the structure of $\mathbf{H}$, i.e. using the inductive definition of $\mathcal{F}$. If $\mathbf{H}$ consists of a single loop or non-loop, co-CSP($\mathbf{H}^L$) is trivially definable in symmetric Datalog and it remains to show that this property is preserved by the operators disjoint union, basic graph adjunction, formation of a basic graph by completion of a colour class and special sum.

We begin with simple but useful observations that allow more concise and intuitive descriptions of our constructions. These remarks and basic tricks all rely on Lemma 66.

1. In a number of constructions below, we want to obtain from two symmetric Datalog programs $\mathscr{D}_1$ and $\mathscr{D}_2$ with goal predicates T_1 and T_2, respectively, a new symmetric program $\mathscr{D}$ which accepts an input $\mathbf{G}$ if $\mathbf{G}$ is accepted by $\mathscr{D}_1$ or if $\mathbf{G}$ is accepted by $\mathscr{D}_2$. This can be done effortlessly since we can simply take the union of the rules of $\mathscr{D}_1$ and $\mathscr{D}_2$, create a new goal predicate T and include the rules $T \leftarrow T_1$ and

$T \leftarrow T_2$.

If instead we want $\mathscr{D}$ to accept $\mathbf{G}$ if both $\mathscr{D}_1$ *and* $\mathscr{D}_2$ accept $\mathbf{G}$, we can use Lemma 66 as follows. Consider the relational structure output by $\mathscr{D}_1$: this structure includes the relation T_1 which we can now use as an EDB in $\mathscr{D}$. It now suffices to add T_1 to the body of any rule of $\mathscr{D}_2$ which has T_2 as its head.

2. When analysing programs we always assume that the input structure $\mathbf{G}$ is connected[6]. This is possible without loss of generality. Indeed, consider Datalog programs over a signature τ that contains a binary relation E (this is the case in all our constructions). It is straightforward to define a k-ary relation C_E in symmetric Datalog which contains the tuple $(x_1, \ldots, x_k)$ iff all x_i are in the same connected component of E. Moreover, Lemma 66 allows us to assume that any program $\mathscr{D}$ has access to this relation as an EDB.

   Suppose that the body of each rule in $\mathscr{D}$ includes the EDB condition $C_E(x_1, \ldots, x_k)$ where the x_i are the variables occurring in the rule. Note that if $\mathbf{G}$ is a digraph given as input to $\mathscr{D}$, then any derivation of $\mathscr{D}(\mathbf{G})$ must now take place within a single connected component of $\mathbf{G}$. So the digraphs accepted by $\mathscr{D}$ are disjoint unions of connected digraphs (recall Footnote 6), one of which is accepted by $\mathscr{D}$. In our case, we

[6]Note that because the target graph is an undirected graph, the direction of the edges of $\mathbf{G}$ can be ignored. Therefore we say that two vertices u and v in $\mathbf{G}$ are *connected* if u and v are connected in $\mathbf{G}$ once the direction of the edges are ignored.

construct Datalog programs which accept a digraph $\mathbf{G}$ with $\mathbf{H}$-lists iff there is no homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$ (here we understand that $\mathbf{G}$ includes the unary relations encoding the lists) and this of course holds iff there is some connected component of $\mathbf{G}$ that does not map to $\mathbf{H}^L$. In proving the correctness of a given program, we can therefore assume connectivity of the input structure without loss of generality.

3. If T is a subset of vertices of $\mathbf{H}$ and if $\mathbf{G}$ is a digraph with $\mathbf{H}$-lists, we can construct in symmetric Datalog a digraph $\mathbf{G}^{\cap T}$ in which every vertex v is bound by a list L'_v which is the intersection of the original L_v with T.

   Furthermore, in the constructions below we typically assume that symmetric programs $\mathscr{D}_1, \mathscr{D}_2$ exist for co-CSP($\mathbf{H}_1^L$) and co-CSP($\mathbf{H}_2^L$) and construct a symmetric program $\mathscr{D}$ for co-CSP($\mathbf{H}^L$) where $\mathbf{H}$ is a graph obtained by combining $\mathbf{H}_1$ and $\mathbf{H}_2$ through some operator. Strictly speaking, the inputs of $\mathscr{D}$ are digraphs with $\mathbf{H}$-lists and thus cannot be fed as inputs to $\mathscr{D}_1$ or $\mathscr{D}_2$ since the latter only deal with lists contained in $\mathbf{H}_1$ and $\mathbf{H}_2$, respectively. Note however that $\mathbf{G}^{\cap H_1}$ *can* be used as an input to $\mathscr{D}_1$ and we use this trick repeatedly. If it is needed, we can also use symmetric programs to construct digraphs $\mathbf{G}_1$ and $\mathbf{G}_2$ with, respectively, $\mathbf{H}_1$-lists and $\mathbf{H}_2$-lists and new edge relations denoted E_1 and E_2, respectively. We can further modify the rules of $\mathscr{D}_1$ and $\mathscr{D}_2$ by replacing the occurrences of E by E_1 and E_2 and by our first remark,

we can then construct a symmetric program $\mathscr{D}$ that accepts $\mathbf{G}$ iff $\mathscr{D}_1$ accepts $\mathbf{G}_1$ and $\mathscr{D}_2$ accepts $\mathbf{G}_2$ or a program $\mathscr{D}$ that accepts $\mathbf{G}$ iff $\mathscr{D}_1$ accepts $\mathbf{G}_1$ or $\mathscr{D}_2$ accepts $\mathbf{G}_2$.

Let $\mathbf{G}$ be a digraph with $\mathbf{H}$-lists. Any $v \in G$ is potentially bound by more than one unary predicate but of course this amounts to imposing a list on v which is the intersection of all such unary predicates. We call this intersection the *minimal list of* v. By the inductive definition of $\mathcal{F}$, i.e. the definition of $\mathcal{I}$, the following lemmas complete the proof of Lemma 64.

Lemma 67. *Suppose* co-CSP($\mathbf{H}_1^L$) *and* co-CSP($\mathbf{H}_2^L$) *are definable in symmetric Datalog and let* $\mathbf{H}$ *be the disjoint union of* $\mathbf{H}_1$ *and* $\mathbf{H}_2$. *Then* co-CSP($\mathbf{H}^L$) *is also definable in symmetric Datalog.*

Proof. Suppose co-CSP($\mathbf{H}_1^L$) and co-CSP($\mathbf{H}_2^L$) are recognised by symmetric Datalog programs $\mathscr{D}_1$ and $\mathscr{D}_2$ with respective goal predicates K_1 and K_2. If $\mathbf{H}$ is the disjoint union of $\mathbf{H}_1$ and $\mathbf{H}_2$, it is clear that a connected digraph $\mathbf{G}$ with $\mathbf{H}$-lists maps into $\mathbf{H}^L$ iff $\mathbf{G}$ maps to $\mathbf{H}_1^L$ or to $\mathbf{H}_2^L$. Of course $\mathbf{G}$ maps to $\mathbf{H}_i^L$ iff $\mathbf{G}^{\cap H_i}$ does. In other words, $\mathbf{G}$ does not map into $\mathbf{H}^L$ iff $\mathscr{D}_1$ accepts $\mathbf{G}^{\cap H_1}$ and $\mathscr{D}_2$ accepts $\mathbf{G}^{\cap H_2}$. As we noted in the above remarks, this can be tested with a symmetric program $\mathscr{D}$. □

Lemma 68. *Let* $\mathbf{H}_1$ *be an irreflexive basic graph with colour classes* B *and* T *and assume that* co-CSP($\mathbf{H}_1^L$) *is recognised by a symmetric Datalog program* $\mathscr{D}_1$. *If* $\mathbf{H}$ *is the graph obtained by transforming* T *into a reflexive clique then* co-CSP($\mathbf{H}^L$) *is also definable in symmetric Datalog.*

Proof. Let $\mathbf{G}$ be a digraph with $\mathbf{H}$-lists. We produce a graph $\mathbf{G}'$ from $\mathbf{G}$ by removing any edge (u, v) of $\mathbf{G}$ if both L_u and L_v contain at least one vertex from T. If there is a homomorphism h from $\mathbf{G}$ to $\mathbf{H}^L$, then h is clearly a homomorphism from $\mathbf{G}'$ to $\mathbf{H}_1^L$. If there is a homomorphism g from $\mathbf{G}'$ to $\mathbf{H}_1^L$, then we can produce a homomorphism g' from $\mathbf{G}$ to $\mathbf{H}^L$ as follows:

$$g'(w) = \begin{cases} t, & \text{if } L_w \text{ contains a vertex } t \in T \text{ (any } t \text{ in } T \cap L_w \text{ suffices);} \\ g(w), & \text{otherwise.} \end{cases}$$

To check the validity of g', we first notice that g can be assumed to have a certain property. Specifically, observe that any vertex w of $\mathbf{G}'$ whose list contains at least one element t from T can be assumed to be mapped by g to t, and this is just what g' is doing. Then we add back the edges of $\mathbf{G}'$ to obtain back $\mathbf{G}$, and notice that if (u, v) is added back, then $(g'(u), g'(v))$ is an edge of $\mathbf{G}$. Therefore to check the existence of a homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$, we can just check if there is a homomorphism from $\mathbf{G}'$ to $\mathbf{H}^L$. It remains to construct a symmetric Datalog program $\mathscr{D}$ that outputs $\mathbf{G}'$ given $\mathbf{G}^L$.

We define a unary relation G_B that contains all vertices of $\mathbf{G}$ whose minimal lists are subsets of B. This can be achieved by using all rules of the form $G_B(x) \leftarrow \bigwedge_{S \in \mathcal{S}} U_S(x)$, where $\mathcal{S}$ is a collection subsets of B. We also define a unary relation G_T that contains all vertices of $\mathbf{G}$ whose minimal lists contain at least one element of T and possibly some other vertices. This is implemented using the rules $G_T(x) \leftarrow \bigwedge_{S \in \mathcal{S}} U_S(x)$, for all possible

collections $\mathcal{S}$ of subsets of T such that $\bigcap_{S\in\mathcal{S}}$ contains at least one element of T. Note that a vertex in G_B could also be present in G_T. If there is an edge between two vertices in G_B, there can be no homomorphism. This can be easily indicated by the rule $I \leftarrow E(x,y) \wedge G_B(x) \wedge G_B(y)$ (note that we are implicitely using Lemma 66 to be able to view G_B and G_T as EDBs). Finally, we define E', the edge relation of $\mathbf{G}'$ with the rule

$$E'(x,y) \leftarrow E(x,y) \wedge G_B(x) \wedge G_T(y). \tag{4.4}$$

Let's check that $\mathcal{D}$ has the desired output. Let (u,v) be an edge of $\mathbf{G}$. If $L_v, L_u \subseteq B$, then $\mathcal{D}$ indicates that there can be no homomorphism. If $L_v \subseteq B$ and L_u contains an element of T (or the other way around), then $(u,v) \in E'$, as desired. If both L_v and L_u contain an element of T, then neither u nor v is in G_B, so there is no way rule 4.4 could have put (u,v) into E'. □

Lemma 69. *Let* $\mathbf{H}_1$ *be a basic graph and assume that* co-CSP($\mathbf{H}_1^L$) *and* co-CSP($\mathbf{H}_2^L$) *are recognised by symmetric Datalog programs* $\mathscr{D}_1$ *and* $\mathscr{D}_2$*, respectively. If* $\mathbf{H}$ *is the result of adjoining* $\mathbf{H}_1$ *to* $\mathbf{H}_2$ *then* co-CSP($\mathbf{H}^L$) *is also definable in symmetric Datalog.*

Proof. For $i \in \{1,2\}$, let R_i, I_i respectively denote the set of loops and non-loops of $\mathbf{H}_i$. Recall that the adjunction of $\mathbf{H}_1$ to $\mathbf{H}_2$ is the graph obtained by taking the disjoint union of the two graphs and adding every edge from R_1 to H_2. Moreover, because $\mathbf{H}_1$ is basic, its loops R_1 form a clique and there

are no edges between I_1 and H_2.

Let $\mathbf{G}$ be a digraph with $\mathbf{H}$-lists. WLOG, we can also assume that $\mathbf{G}^L$ is not the graph with a single vertex (without self-loop). We begin by defining a few sets:

- i_1 : set of all vertices whose minimal lists are subsets of I_1;
- r_1 : set of all vertices whose minimal lists contain at least one vertex from R_1, and r_1 could possibly contain any other vertex (see later);
- h_2 : set of all vertices whose minimal lists do not contain any vertex from R_1 but contain at least one vertex from H_2, and h_2 could possibly contain some vertices whose minimal lists are subsets of I_1 (see later).

Claim. $\mathbf{G}$ *maps to* $\mathbf{H}^L$ *iff the following conditions hold:*

1. *There is no edge* (u, v) *in* $\mathbf{G}$ *such that* $u, v \in i_1$*;*
2. *There is no edge* (u, v) *in* $\mathbf{G}$ *such that* $u \in i_1$ *and* $v \in h_2$ *(or the other way around);*
3. *Let* $\mathbf{G}_1$ *be the subgraph of* $\mathbf{G}$ *that contains an edge* (u, v) *of* $\mathbf{G}$ *iff* $u \in i_1$ *and* $v \in r_1$ *(or the other way around). Then* $\mathbf{G}_1$ *(with lists restricted to* H_1*) maps to* $\mathbf{H}_1^L$*;*
4. *Let* $\mathbf{G}_2$ *be the subgraph of* $\mathbf{G}$ *that contains an edge* (u, v) *of* $\mathbf{G}$ *iff* $u, v \in h_2$*. Then* $\mathbf{G}_2$ *(with lists restricted to* H_2*) maps to* $\mathbf{H}_2^L$*;*

Proof of Claim. If conditions 1 or 2 fail then there is obviously no homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$. So from now on we assume that both conditions 1 and 2 are satisfied. Observe the following: if there is any homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$ then there is one such that any vertex w such that $L_w \subseteq H_2 \cup I_1$ and L_w contains an element of H_2 is mapped to a vertex in H_2. Let's see what $\mathbf{G}_2$ is in condition 4: if (u, v) is an edge of $\mathbf{G}$ such that both L_u and L_v contain an element of H_2, then (u, v) is in $\mathbf{G}_2$. Edge (u, v) cannot be an edge such that L_u or L_v is a subset of I_1 by assuming conditions 1 and 2. But then if there is a homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$, then the image of (u, v) can be assumed to be in $\mathbf{H}_2$. So if condition 4 is violated, there can be no homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$.

Assume that (u, v) is an edge of $\mathbf{G}_1$ in condition 3. Observe that if $u \in i_1$, then L_v must contain an element of R_1. If it does not, then v also belongs to i_1 or h_2 and therefore at least one of conditions 1 or 2 would be violated. Notice that if there is a homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$ then v can be assumed to be mapped to a vertex in R_1. So if there is a homomorphism $\mathbf{G}$ to $\mathbf{H}^L$, then we can assume that $\mathbf{G}_1$ is mapped to $\mathbf{H}_1$ and therefore if condition 3 fails, there can be no homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$.

Conversely, assume that all the above conditions are satisfied. Let φ_1 be a homomorphism from $\mathbf{G}_1$ to $\mathbf{H}_1^L$ and φ_2 be a homomorphism from $\mathbf{G}_2$ to $\mathbf{H}_2^L$. Just like before, we can assume that φ_1 maps a vertex whose list contains an element of R_1 to a vertex in R_1. We construct a homomorphism φ from $\mathbf{G}$

to $\mathbf{H}^L$:

$$\varphi(w) = \begin{cases} \varphi_1(w), & \text{if } L_w \subseteq I_1 \text{ or } L_w \text{ contains an element of } R_1; \\ \varphi_2(w), & \text{otherwise, i.e. if } L_w \text{ contains an element} \\ & \text{of } H_2 \text{ and no element of } R_1. \end{cases}$$

Let's check that the map φ is defined for every vertex of $\mathbf{G}$ and that it is actually a homomorphism. Recall that $\mathbf{G}$ is connected and it is not a single vertex, so every vertex of $\mathbf{G}$ is in an edge. So let (u, v) be any edge of $\mathbf{G}$.

Assume first that L_u contains an element of R_1. Then u is mapped to a vertex in R_1 by φ_1. If L_v also contains an element of R_1, then (u, v) is clearly mapped to an edge. If L_v contains no element of R_1 but contains an element of H_2, φ_2 maps u to a vertex in H_2 and therefore (u, v) is mapped to an edge (using the definition of adjunction). If L_v contains only vertices in I_1, then $(\varphi_1(u), \varphi_1(v))$ is obviously an edge of $\mathbf{H}_1$.

Suppose that L_u contains an element of H_2 but no element of R_1. If L_v has the same properties, then the image of (u, v) is $(\varphi_2(u), \varphi_2(v))$. If $L_v \subseteq I_1$, then condition 1 or 2 would be violated. Lastly if both $L_u, L_v \subseteq I_1$, then condition 1 is violated. □

Now we just have to construct a symmetric Datalog program $\mathcal{D}$ verifying the conditions in the claim. Notice that the relations i_1, r_1 and h_2 are defined so that we can define them in symmetric Datalog. For example, to define h_2, we use the rules $h_2(x) \leftarrow \bigwedge_{S \in \mathcal{S}} U_S(x)$, where $\mathcal{S}$ is any collection of subsets of

$H_2 \cup I_1$ such that $\bigcap_{S \in \mathcal{S}} S$ contains at least one element of H_2 (note that this could add a vertex to h_2 whose minimal list does not contain any element of H_2). Conditions 1 and 2 are trivial to check. To check condition 3, we produce $\mathbf{G}_1$ using the rule $E_1(x,y) \leftarrow i_1(x) \wedge r_1(x) \wedge E(x,y)$. Obtaining the lists of the vertices of $\mathbf{G}_1$ reduced to H_1 can be achieved by using rules of the form $U_Q^{\cap H_1}(x) \leftarrow U_S(x) \wedge E_1(x,y)$ and $U_Q^{\cap H_1}(x) \leftarrow U_S(x) \wedge E_1(y,x)$, where S is any subset of H such that $S \cap H_1 = Q$. Condition 4 can be checked similarly. □

Lemma 70. *Let* $\mathbf{H}_1$ *and* $\mathbf{H}_2$ *be irreflexive graphs such that* co-CSP($\mathbf{H}_1^L$) *and* co-CSP($\mathbf{H}_2^L$) *are recognised by symmetric Datalog programs* $\mathscr{D}_1$ *and* $\mathscr{D}_2$*, respectively. If* $\mathbf{H}$ *is the special sum of* $\mathbf{H}_1$ *and* $\mathbf{H}_2$ *then* co-CSP($\mathbf{H}^L$) *is also definable in symmetric Datalog.*

Proof. Recall that the special sum of bipartite irreflexive graphs $\mathbf{H}_1$ and $\mathbf{H}_2$ with colour classes B_i, T_i consists of the disjoint union of the graphs in which all edges between T_1 and B_2 are added. Note first that $\mathbf{G}$ must be bipartite in order to map to $\mathbf{H}$ and since bipartiteness can be checked in symmetric Datalog, we can assume that any input $\mathbf{G}$ is indeed bipartite. For now, we assume that we have access to a unary relation U_c that contains a single arbitrary vertex c of the input graph $\mathbf{G}$ and later we will get rid of this relation. Also recall that we are assuming that $\mathbf{G}$ is connected, so we can construct a symmetric Datalog program that outputs a unary relation B_G that contains all vertices in the same partition as c. This is achieved by putting any vertex that can be reached from c using an even length walk into

B_G. Similarly, T_G denotes the other color class. We use B_G and T_G as EDBs in the sequel.

Set $T = T_1 \cup T_2$ and $B = B_1 \cup B_2$. We define four unary relations over the vertices of $\mathbf{G}$:

1. t_1: contains all vertices whose minimal lists are subsets of T and contains at least one element of T_1, and possibly some other vertices whose minimal lists are subsets of T;

2. t_2: contains all vertices whose minimal lists are subsets of T_2;

3. b_1: contains all vertices whose minimal lists are a subsets of B_1;

4. b_2: contains all vertices whose minimal lists are subsets of B and contains at least one element of B_2, and possibly some other vertices whose minimal lists are subsets of B.

The following claim has a somewhat similar proof to the claim in the previous lemma, except that this one is simpler:

Claim. *There is a homomorphism from* $\mathbf{G}$ *to* $\mathbf{H}^L$ *(with color classes matched as above) iff the following conditions hold:*

1. *There are no edges* (u, v) *such that* $v \in t_2$ *and* $u \in b_1$ *(or the other way around);*

2. *Let* $\mathbf{G}_1$ *be the subgraph of* $\mathbf{G}$ *obtained by taking the graph induced by* $t_1 \cup b_1$ *and removing any isolated vertices. Then* $\mathbf{G}_1^{\cap H_1}$ *homomorphically maps to* $\mathbf{H}_1^L$*;*

3. *Let* $\mathbf{G}_2$ *be the subgraph of* $\mathbf{G}$ *obtained by taking the graph induced by* $t_2 \cup b_2$ *and removing any isolated vertices. Then* $\mathbf{G}_2^{\cap H_2}$ *homomorphically maps to* $\mathbf{H}_2^L$*;*

Proof. If condition 1 is violated, there is trivially no homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$ (where the color classes are matched), so from now on we assume that condition 1 is satisfied. If there is a homomorphism h from $\mathbf{G}$ to $\mathbf{H}^L$, then for any vertex v such that $h(v) \in T$ and $L_v \cap T_1 \neq \emptyset$, we can assume that $h(v) \in T_1$. Using the assumption that condition 1 is met, it is easy to see that h (restricted to vertices of $\mathbf{G}_1$) is a homomorphism from $\mathbf{G}_1^{\cap H_1}$ to $\mathbf{H}_1^L$. A similar argument can be made for condition 3. Therefore the above conditions are necessary for the existence of a homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$.

Conversely, assume that all conditions above are met. Let the two homomorphisms from conditions 2 and 3 be φ_1 and φ_2. To obtain a homomorphism φ from $\mathbf{G}$ to $\mathbf{H}^L$, observe first that the domains of φ_1 and φ_2 are disjoint: if for a vertex v, $L_v \subseteq T$ and L_v contains a vertex from T_1, then v cannot be in the domain of φ_2. If L_v does not contain a vertex from T_1, then v cannot be in the domain of φ_1. Similarly for the bottom partition.

For any vertex v in the domain of φ_1, $\varphi(v)$ is defined to be $\varphi_1(v)$. Similarly, for any vertex v in the domain of φ_2, $\varphi(v)$ is defined to be $\varphi_2(v)$. Note that $\mathbf{G}$ could still contain some vertices which are neither in the domain of φ_1 nor in the domain φ_2. Let v be such a vertex. Observe that v cannot be in t_2 because $\mathbf{G}$ is connected, and therefore v must have a neighbor u. Vertex u can be only in t_2 and therefore the edge (v, u) belongs to $\mathbf{G}_2$, so v is in

the domain φ_2. Similarly, v cannot be in b_1. If v is in t_1 and not in t_2, then the list of v must contain at least one element w of T_1. By the properties of special sum, we can define $\varphi(v)$ to be w. We define φ similarly if v is in b_2. It is not hard to check that φ is a homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$. □

To construct the symmetric Datalog program, notice that we can output t_1, t_2, b_1 and b_2. We can also output $\mathbf{G}_1$ using the rules $E_1(x,y) \leftarrow E(x,y) \wedge t_1(x) \wedge b_2(y)$ and $E_1(x,y) \leftarrow E(x,y) \wedge t_1(y) \wedge b_2(x)$. We need to trim the lists of $\mathbf{G}_1$ to contain only elements of H_1. Note we cannot just take all vertices in t_1 and remove vertices not in H_1, because if t_1 contains a vertex v such that $L_v \subseteq T_2$, then this approach would produce a vertex with the empty list, which would indicate a false negative. Instead, first we produce the vertex set of $\mathbf{G}_1$ with the rules $V_{\mathbf{G}_1}(x) \leftarrow E_1(x,y)$ and $V_{\mathbf{G}_1}(x) \leftarrow E_1(y,x)$, and then we trim the lists. Now we can just use the programs for co-CSP($\mathbf{H}_1^L$) and co-CSP($\mathbf{H}_2^L$) to test conditions 2 and 3, and we can directly check condition 1.

Up to this point, our program $\mathcal{D}_1$ checks if there is a homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$ such that the color class of c is mapped to $B_1 \cup B_2$. To check the other possibility, we just write another program $\mathcal{D}_2$ with color classes reversed, and produce a third program $\mathcal{D}_3$ that accepts if both $\mathcal{D}_1$ and $\mathcal{D}_2$. We need to modify $\mathcal{D}_3$ so that it does not use the relation U_c. Produce $\mathcal{D}$ from $\mathcal{D}_3$ by adding a new variable x to every IDB of $\mathcal{D}_3$. For every rule $\mathcal{R}$ that contains $U_c(y)$, remove $U_c(y)$ from $\mathcal{R}$, and replace any occurrence of y in $\mathcal{R}$ with x. We claim that $\mathcal{D}_3$ accepts iff $\mathcal{D}$ accepts. Let P be a derivation

for $\mathcal{D}_3$. We obtain a $\mathcal{D}$-derivation for the same input structure by removing $U_c(a)$ from all rules, where a is the value assigned to the variable y if U_c. We add the new variables x to the IDBs and instantiate them to a. For the converse, we assume that the newly added variable takes on the value a everywhere (they are forced to be the same by definition), and make U_c be the relation $\{a\}$. Now we just to do obvious backward modification of rules to obtain a $\mathcal{D}_3$-derivation. □

4.7 List homomorphism problems definable in first-order logic

In this section we prove Theorem 49. We need the following characterisation of structures whose CSP is first-order definable [61]. Let $\mathbf{T}$ be a relational structure and let $a, b \in T$. We say that b *dominates* a in $\mathbf{T}$ if, for any relation $R(\mathbf{T})$, and any tuple $\bar{t} \in R(\mathbf{T})$, replacement of any occurrence of a by b in $\bar{t}$ will yield a tuple of $R(\mathbf{T})$. Recall the definition of a direct power of a structure from Subsection 4.2.1. If $\mathbf{T}$ is a relational structure, we say that the structure $\mathbf{T}^2$ *dismantles to the diagonal* if there exists a sequence of elements $\{a_0, \ldots, a_n\} = T^2 \setminus \{(a, a) : a \in T\}$ such that, for all $0 \leq i \leq n$, a_i is dominated in $\mathbf{T}_i$, where $\mathbf{T}_0 = \mathbf{T}^2$ and $\mathbf{T}_i$ is the substructure of $\mathbf{T}^2$ induced by $T^2 \setminus \{a_0, \ldots, a_{i-1}\}$ for $i > 0$.

Lemma 71 ([61])**.** *Let* $\mathbf{T}$ *be a core relational structure. Then* CSP($\mathbf{T}$) *is*

first-order definable if and only if $\mathbf{T}^2$ *dismantles to the diagonal.*

Proof. (of Theorem 49). We first prove that conditions (i) and (ii) are necessary. Notice that if CSP($\mathbf{H}^L$) is first-order definable then so is CSP($\mathbf{K}^L$) for any induced substructure $\mathbf{K}$ of $\mathbf{H}$. Let x and y be distinct vertices of $\mathbf{H}$ and let $\mathbf{K}^L$ be the substructure of $\mathbf{H}^L$ induced by $\{x, y\}$. If x and y are non-adjacent loops, then $\theta(\mathbf{K}) = \{(x, x), (y, y)\}$ is the equality relation on $\{x, y\}$; if x and y are adjacent non-loops, then $\theta(\mathbf{K}) = \{(x, y), (y, x)\}$, the adjacency relation of the complete graph on 2 vertices. It is well known (and can be easily derived from Lemma 71) that neither of these classes CSP($\mathbf{K}^L$) is first-order definable. It follows that the loops of $\mathbf{H}$ induce a complete graph and the non-loops induce a graph with no edges.

Now we prove (iii) is necessary. Suppose for a contradiction that there exist distinct elements x and y of I and elements n and m of R such that m is adjacent to x but not to y, and n is adjacent to y but not to x. Then CSP($\mathbf{G}$) is first-order definable, where $\mathbf{G}$ is the substructure of $\mathbf{H}^L$ induced by $\{x, y, m, n\}$. By Lemma 71, $\mathbf{G}^2$ dismantles to the diagonal. Then (x, y) must be dominated by one of (x, x), (y, x) or (y, y), since domination respects the unary relation $\{x, y\}^2$ (on G^2). But (m, n) is a neighbour of (x, y) and none of the other three, a contradiction.

For the converse: we show that we can dismantle $(\mathbf{H}^L)^2$ to the diagonal. Let $x \in H$: then (x_1, x) and (x, x_1) are dominated by (x, x). Suppose that we have dismantled every element containing a coordinate equal to x_i with $i \leq j - 1$: if x is any element of H such that the elements (x_j, x) and (x, x_j)

remain, then either x is a loop or $x = x_k$ with $k \geq j$; in any case the elements (x_j, x_k) and (x_k, x_j) are dominated by (x, x). In this way we can remove all pairs (x, y) with one of x or y a non-loop. For the remaining pairs, notice that if u and v are any loops then (u, v) is dominated (in what remains of $(\mathbf{H}^L)^2$) by (u, u). □

Chapter 5

A Dichotomy for the List Homomorphism Problem for Oriented Paths

5.1 Introduction

In this chapter, we further contribute to the understanding of the fine-grained complexity of the list homomorphism problem. We show that when $\mathbf{H}$ is an oriented path, CSP($\mathbf{H}^L$) is in L or is NL-complete. We use some notation from Chapter 4.

5.2 The L-NL Dichotomy

5.2.1 NL-hardness

We start with some definitions.

Definition 72. A *fuzzy-N* (see left of Figure 5.2) of height h is an oriented path $\mathbf{N}$ of the following form. Let $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ be minimal oriented paths of height h (recall that minimal means that the first vertex is in the bottom level and the last vertex is in top level and no other vertices are in the top and bottom levels). Let $\mathbf{U}$ be an oriented path of height less than h such that its first and last vertices are both in the top level. Let $\mathbf{D}$ be an oriented path of height less than h such that its first and last vertices are both in the bottom level. Then $\mathbf{N}$ is obtained by taking the union of $\mathbf{P}_1, \mathbf{U}, \mathbf{P}_2, \mathbf{D}$ and $\mathbf{P}_3$, and identifying the following pairs of vertices: the topmost vertex of $\mathbf{P}_1$ and the first vertex of $\mathbf{U}$, the last vertex of $\mathbf{U}$ and topmost vertex of $\mathbf{P}_2$, the bottommost vertex of $\mathbf{P}_2$ and first vertex of $\mathbf{D}$, the last vertex of $\mathbf{D}$ and bottommost vertex of $\mathbf{P}_3$.

Definition 73. $\mathbf{Z}_5$ is the oriented path:

Definition 74. The set $\mathcal{F}$ is the set of oriented paths that do not contain a fuzzy-N, or a $\mathbf{Z}_5$ as an (induced) subgraph.

Note that to show hardness results, our strategy is similar to the strategy in Chapter 4. In particular, see Lemma 61. Once we show that the variety associated with the oriented path $\mathbf{P}$ admits a unary, lattice or semilattice type, we know that CSP($\mathbf{P}^L$) is NL-hard by applying Theorem 4.1 of [63]. Therefore we only show how to construct $\mathbf{S}$ for Lemma 59(2) with $\mathbf{T} = \mathbf{P}^L$.

If we wished to avoid algebra, we can directly show that once the relation $\{(0,0),(0,1),(1,1)\}$ is defined from $\mathbf{P}^L$ (as in Lemma 59), we can reduce directed reachability to CSP($\mathbf{P}^L$) (as we already mentioned in the proof of Theorem 44, Chapter 3).

Lemma 75. *If an oriented path* $\mathbf{P}$ *contains a* $\mathbf{Z}_5$ *as a subgraph, then* CSP($\mathbf{H}^L$) *is* NL*-hard.*

Proof. In Figure 5.1 below, the gadget $\mathbf{S}$ is the oriented path with lists as indicated. It is straightforward to check that any homomorphism from $\mathbf{S}$ to $\mathbf{Z}_5$ must map (s_1, s_2) to $(0,0),(0,4)$ or $(4,4)$. The lemma follows from the above comments.

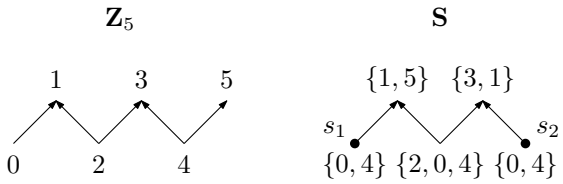


Figure 5.1: $\mathbf{Z}_5$ and gadget.

□

We need to recall Claim 1 within the Corollary of Theorem 1 of [49] to

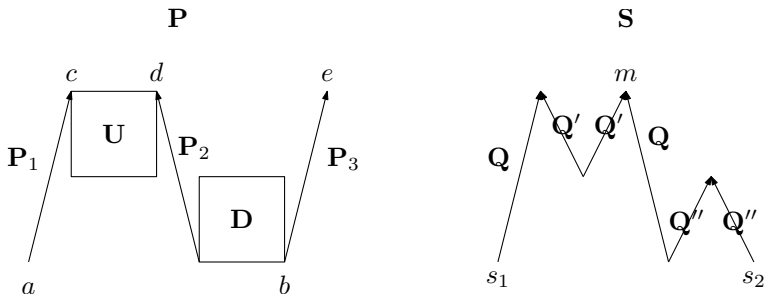


Figure 5.2: A fuzzy-N and a gadget.

prove the next lemma. The definition of minimal oriented path in [49] is slightly different from ours. We refer to their notion of minimality as *weak minimality*. An oriented path is weakly minimal if its first vertex is in the bottommost level and its last vertex is in the topmost level (or the other way around).

Claim 76 ([49])**.** *Let* $\mathbf{P} = p_0 p_1 \dots p_a$ *and* $\mathbf{P}' = p'_0 p'_1 \dots p'_b$ *be weakly minimal oriented paths of net length* k*. Then there exists an oriented path* $\mathbf{P}^* = p^*_0 p^*_1 \dots p^*_c$ *such that* $\mathbf{P}^* \xrightarrow{h} \mathbf{P}$ *and* $\mathbf{P}^* \xrightarrow{h'} \mathbf{P}'$*. Furthermore,* h *and* h' *can be chosen such that* $h(p^*_0) = p_0$*,* $h'(p^*_0) = p'_0$*,* $h(p^*_c) = p_a$*, and* $h'(p^*_c) = p'_b$*.*

Lemma 77. *If an oriented path* $\mathbf{P}$ *contains a fuzzy-N* $\mathbf{N}$ *as a subgraph, then* $\mathrm{CSP}(\mathbf{P}^L)$ *is* NL*-hard.*

Proof. We follow the same strategy as in Lemma 75 (and see the comments above). We simply produce the gadget $\mathbf{S}$ as before. Let the building blocks of $\mathbf{N}$ be $\mathbf{P}_1, \mathbf{U}, \mathbf{P}_2, \mathbf{D}, \mathbf{P}_3$, just like in the definition above. See left of Figure 5.2.

Using Claim 76, we can find a minimal oriented path $\mathbf{Q}$ of height h that

maps to $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$. Let $h_{\mathbf{U}}$ be the height of $\mathbf{U}$. Let v be a vertex of $\mathbf{U}$ in the bottom level, and "cut" $\mathbf{U}$ at vertex v, i.e. obtain weakly minimal oriented paths $\mathbf{U}_1$ and $\mathbf{U}_2$ of height $h_{\mathbf{U}}$ such that identifying the last vertex $\mathbf{U}_1$ with the first vertex of $\mathbf{U}_2$ gives back $\mathbf{U}$. Let $\mathbf{P}'_1$ be the shortest oriented subpath of $\mathbf{P}_1$ starting at the top of $\mathbf{P}_1$ that has height $h_{\mathbf{U}}$. Define $\mathbf{P}'_2$ and $\mathbf{P}'_3$ similarly. Using Claim 76, find an oriented path $\mathbf{Q}'$ that maps to $\mathbf{U}_1$, $\mathbf{U}_2$, $\mathbf{P}'_1$, $\mathbf{P}'_2$, and $\mathbf{P}'_3$ and furthermore for these homomorphisms, the first vertex of $\mathbf{Q}'$ is mapped to the first vertex of the target path, and the last vertex of $\mathbf{Q}'$ is mapped to the last vertex of the target path. In exactly the same way, find a $\mathbf{Q}''$ with respect to $\mathbf{D}$. Our gadget $\mathbf{S}$ is the oriented path constructed from $\mathbf{Q}$, $\mathbf{Q}'$, $\mathbf{Q}''$, shown in Figure 5.2 (right side).

Let $\mathbf{A}$ and $\mathbf{B}$ be oriented paths. Let $\mathbf{A}_{ij}$ denote the subpath of $\mathbf{A}$ from vertex i to j. Let $\mathbf{A}_{ij} \to \mathbf{B}_{kl}$ denote the existence of a homomorphism from $\mathbf{A}_{ij}$ to $\mathbf{B}_{kl}$ that maps i to k and j to l. Then by construction, the following homomorphisms exist: $\mathbf{S}_{s_1 m} \to \mathbf{P}_{ac}$, $\mathbf{S}_{s_1 m} \to \mathbf{P}_{ad}$, $\mathbf{S}_{s_1 m} \to \mathbf{P}_{be}$ (note that we can assume that these homomorphisms do not map anything to vertices in $\mathbf{P}_2$ or $\mathbf{D}$, except for vertices d and b.) We also have $\mathbf{S}_{s_2 m} \to \mathbf{P}_{ac}$, $\mathbf{S}_{s_2 m} \to \mathbf{P}_{bd}$, and $\mathbf{S}_{s_2 m} \to \mathbf{P}_{be}$. We add lists to restrict the possible homomorphisms. First, add lists so that s_1 and s_2 can map only to $\{a, b\}$. By adding lists, we restrict the possible images of vertices in $\mathbf{S}_{s_1 m}$ to those vertices of $\mathbf{P}$ which are in $\mathbf{P}_{ad}$ or $\mathbf{P}_3$. It is easy to check that there are homomorphisms that map $\mathbf{P}$ to $\mathbf{S}$ such that (s_1, s_2) is mapped to $(a, a), (a, b)$ or (b, b). The only remaining possibility would be to map (s_1, s_2) to (b, a), but that is not possible for the

following reason. If s_1 is mapped to b, then because no vertex in $\mathbf{S}_{s_1 s_2}$ can map to $\mathbf{D}$, vertex m must be mapped to e. Recall that $\mathbf{Q}$ is minimal and the height of $\mathbf{Q}''$ is strictly less than the height of $\mathbf{Q}$. But then $\mathbf{S}_{ms_2}$ contains only a single vertex in the top level, namely m. If s_2 was mapped to a, $\mathbf{S}_{ms_2}$ should have a vertex that is mapped to a vertex in $\mathbf{P}_{cd}$ in the top level of $\mathbf{P}$, a contradiction. □

5.2.2 Membership in L, NL

The following result is just putting together results from [40] and [30].

Theorem 78. *If* $\mathbf{P}$ *is an oriented path, then* CSP($\mathbf{P}^L$) *is in* NL.

Proof. In [40], a majority operation is defined for any oriented path. In fact, this majority operation is trivially conservative. Therefore the main result of [30] shows that co-CSP($\mathbf{P}^L$) is in linear Datalog and therefore in NL [28]. □

Now we focus on showing that if $\mathbf{P}$ is in $\mathcal{F}$, then CSP($\mathbf{P}^L$) is in L. We begin with defining a class of oriented paths constructed inductively using three operations. It can be easily verified that all three operators in Definition 79 are special cases of the special sum operator (see Definition 86 in the next section) or disjoint union followed by special sum.

Definition 79. We define three operators that produce new oriented paths from oriented paths:

1. The *growth operator* takes an oriented path in which the last vertex v is in the topmost level. We obtain a new oriented path by attaching a new arc (v, u). A similar operation is defined for the first vertex.

2. The *bulge operator* takes two oriented paths $\mathbf{P}_1$ and $\mathbf{P}_2$. Let u be a vertex of $\mathbf{P}_1$ in the top level that has indegree 1 and outdegree 0 (if such a vertex exists). Similarly, find such a vertex v in $\mathbf{P}_2$. We take the disjoint union of $\mathbf{P}_1$ and $\mathbf{P}_2$ and add a new vertex w, and the arcs (u, w) and (v, w) to obtain a new oriented path. A similar construction can be performed with vertices in the bottom levels.

3. The *dent operator* takes two oriented paths $\mathbf{P}_1$ and $\mathbf{P}_2$. Let u be the vertex of $\mathbf{P}_1$ that has indegree 1 and outdegree 0. Similarly, let v be such a vertex in $\mathbf{P}_2$. If u is the only vertex in the top level of $\mathbf{P}_1$ and v is the only vertex in the top level of $\mathbf{P}_2$, then we take the disjoint union of $\mathbf{P}_1$ and $\mathbf{P}_2$, add a new vertex w, and the arcs (w, u) and (w, v) to obtain a new oriented path. A similar construction can be performed with vertices in the bottom levels.

Lemma 80. *Let* $\mathbf{H}_1$ *and* $\mathbf{H}_2$ *be two oriented paths. Assume that there are logspace algorithms that decide* $\mathrm{CSP}(\mathbf{H}_1^L)$ *and* $\mathrm{CSP}(\mathbf{H}_2^L)$*. If* $\mathbf{H}$ *is obtained from* $\mathbf{H}_1$ *and* $\mathbf{H}_2$ *using the bulge or the dent operators, then there is a logspace algorithm for* $\mathrm{CSP}(\mathbf{H})$*. If* $\mathbf{H}$ *is obtained from* $\mathbf{H}_1$ *using the growth operator, then there is a logspace algorithm for* $\mathrm{CSP}(\mathbf{H})$*.*

Proof. As the above three operators are special cases of the special sum

operator, or a disjoint union operator followed by a special sum operator (see next section), the result follows from Lemmas 88 and 84. □

Definition 81. The set $\mathcal{I}$ is defined to be the set of all oriented paths which can be constructed using the operations in Definition 79 starting with a single vertex as the base case.

The following lemma is central.

Lemma 82. $\mathcal{I} = \mathcal{F}$.

Proof. Let $\mathbf{P}$ be an oriented path that does not contain a fuzzy-N or a $\mathbf{Z}_5$. If $\mathbf{P}$ contains a single edge in its top level, we can obviously obtain $\mathbf{P}$ from a smaller oriented path $\mathbf{P}'$ using the growth operator. A similar argument holds for the bottom level. If the topmost level contains only a single vertex v and it is not the last or the first vertex of $\mathbf{P}$, then we can remove v to obtain $\mathbf{P}_1$ and $\mathbf{P}_2$. We can clearly get back $\mathbf{P}_1$ and $\mathbf{P}_2$ with the bulge operator. A similar argument holds for the bottom level. Therefore we can assume that both the top and the bottom levels of $\mathbf{P}$ contain at least two vertices.

Let u and v be in the topmost level of $\mathbf{P}$ (see Figure 5.3). Assume first that the oriented path $\mathbf{M}$ from u to v has height at least 2. Then the path $\mathbf{Q}_1$ coming into u from left cannot have a vertex in the bottommost level because that would create a fuzzy-N. Similarly, the path leaving v to the right cannot have a vertex in the bottommost level. So the (at least) two vertices in the bottom level must actually be in $\mathbf{M}$. Let a be the first vertex in the bottommost level to the right of u, and let b be the last vertex in

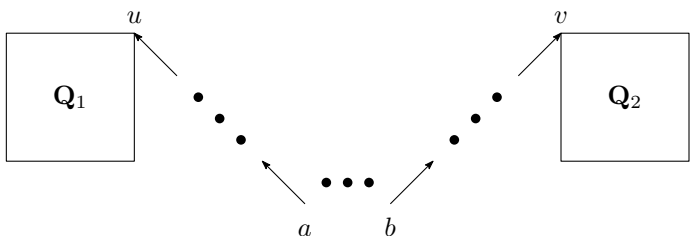


Figure 5.3

the bottommost level before v. The path $\mathbf{M}'$ from a to b must have height 1, otherwise a fuzzy-N is clearly present. To avoid $\mathbf{Z}_5$, $\mathbf{M}'$ must actually be the arcs (a, w) and (b, w), for some vertex w. Removing w creates two components which we can put back together using the dent operator.

Assume now that the height of $\mathbf{M}$ is 1. When $\text{height}(\mathbf{Q}_1) \leq 1$ and $\text{height}(\mathbf{Q}_2) \leq 1$, we check all cases individually. If $\text{height}(\mathbf{Q}_1) = \text{height}(\mathbf{Q}_2) = 0$ then the only possibility for $\mathbf{P}$ is shown in the left side Figure 5.4 (otherwise $\mathbf{P}$ contains a $\mathbf{Z}_5$). In this case, $\mathbf{P}$ can be decomposed into two smaller oriented paths by removing the vertex d, and $\mathbf{P}$ can be obtained back after an application of the dent operator. If $\text{height}(\mathbf{Q}_1) = 1$ and $\text{height}(\mathbf{Q}_2) = 0$ then the only possibilities for $\mathbf{P}$ are the graphs in the middle of Figure 5.4 (similarly for the symmetric case). These graphs can be easily decomposed and put back together with a dent operator. If $\text{height}(\mathbf{Q}_1) = 1$ and $\text{height}(\mathbf{Q}_2) = 1$ then the only possibilities for $\mathbf{P}$ is the graph on the right side in Figure 5.4.

Suppose therefore that at least one of $\text{height}(\mathbf{Q}_1) \geq 2$ or $\text{height}(\mathbf{Q}_2) \geq 2$ holds. We can also assume that $\mathbf{P}$ has a vertex z different from u and v in the top level, because otherwise we can decompose $\mathbf{P}$ by removing the only

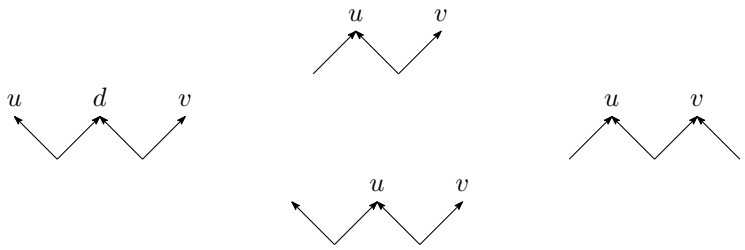


Figure 5.4

vertex between u and v. It is easy to see that z cannot be between u and v because otherwise $\mathbf{P}$ contains a $\mathbf{Z}_5$. Also, we can assume that z is in $\mathbf{Q}_m$ for an $m \in \{1, 2\}$ such that height$(\mathbf{Q}_m) \geq 2$. To see this, assume w.l.o.g. that height$(\mathbf{Q}_1) \geq 2$, height$(\mathbf{Q}_2) = 1$ and z in $\mathbf{Q}_2$ and observe that in this situation $\mathbf{P}$ contains a $\mathbf{Z}_5$.

Therefore assume w.l.o.g. that height$(\mathbf{Q}_1) \geq 2$ and z is in $\mathbf{Q}_1$. Assume the edge coming into u is (w, u). The situation is illustrated in Figure 5.5. If

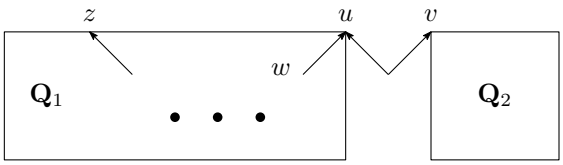


Figure 5.5

(w, z) is an edge of $\mathbf{P}$, then $h(\mathbf{Q}_1) \geq 2$ cannot be satisfied without having a $\mathbf{Z}_5$. So following the oriented path from w to the left, we go down at least one level, and eventually we get back to z in the top level. Suppose c is a vertex between u and z such that c is in the lowest level reached when following the path from u to z. Let the level of c be L_ℓ. We observe that there is no vertex

in level L_ℓ to the left of z because that would produce a fuzzy-N. We can also conclude that $h(\mathbf{Q}_2) < h(\mathbf{Q}_1)$ because otherwise $\mathbf{P}$ contains a fuzzy-N. So we can conclude that $\ell = 0$, and all vertices in level L_0 (i.e. bottom level of $\mathbf{P}$) are between u and z. Let i and j be two vertices in the bottom level. If the path from i to j has height more than 1, we have a fuzzy-N. This is true for *any* pair of vertices in the bottom level, so to avoid having a $\mathbf{Z}_5$, in fact we have *only* i and j in the bottom level. The only possibility left is having the edges (i, k) and (j, k). We can remove k to obtain two paths, and using the dent operator, we can get back $\mathbf{P}$.

For the other direction, it is easy to see that none of our three operations can create a $\mathbf{Z}_5$ or a fuzzy-N. □

Theorem 83. *Let* $\mathbf{P}$ *be an oriented path. If* $\mathbf{P}$ *contains a subpath that is a* $\mathbf{Z}_5$ *or a fuzzy-N then* $\mathrm{CSP}(\mathbf{P}^L)$ *is* NL*-complete. Otherwise* $\mathrm{CSP}(\mathbf{P}^L)$ *is in* L*. Furthermore, the previous condition, i.e. whether* $\mathbf{P}$ *contains a subpath that is a* $\mathbf{Z}_5$ *or a fuzzy-N, can be decided in polynomial time.*

Proof. For any oriented path $\mathbf{P}$, $\mathrm{CSP}(\mathbf{P}^L)$ is in NL by Theorem 78. If $\mathbf{P}$ contains a fuzzy-N or a $\mathbf{Z}_5$ then $\mathrm{CSP}(\mathbf{H}^L)$ is NL-hard by Lemmas 75 and 77. Otherwise, follow the inductive construction of $\mathbf{P}$ given by Lemma 82. Obviously, there is a logspace algorithm for $\mathrm{CSP}(\mathbf{H}^L)$ if $\mathbf{H}$ is a single vertex. At each construction step, apply Lemma 80 repeatedly until a logspace algorithm for $\mathrm{CSP}(\mathbf{P}^L)$ is produced.

Searching for a $\mathbf{Z}_5$ or a fuzzy-N in $\mathbf{P}$ can be easily done in polynomial time. □

5.3 Logspace Preserving Constructions

We begin by proving the following since it is used in the previous section.

Lemma 84 (Disjoint union lemma)**.** *Let* $\mathbf{H}_1$ *and* $\mathbf{H}_2$ *be digraphs. If there are logspace algorithms for* $\mathrm{CSP}(\mathbf{H}_1^L)$ *and* $\mathrm{CSP}(\mathbf{H}_2^L)$*, then there is a logspace algorithm for* $\mathrm{CSP}((\mathbf{H}_1 \sqcup \mathbf{H}_2)^L)$*. (The operator* $\sqcup$ *stands for disjoint union.)*

Proof. The components of a directed graph $\mathbf{G}$ can be output in L using Reingold's logspace algorithm for undirected st-connectivity [77]. We check for each component of $\mathbf{G}$ whether it homomorphically maps to $\mathbf{H}_1$ or $\mathbf{H}_2$. If there is a component $\mathbf{C}$ of $\mathbf{G}$ that does not map either to $\mathbf{H}_1$ nor to $\mathbf{H}_2$, then our algorithms rejects. Otherwise it accepts. □

The main part of this section is the definition of the special sum operator (*not* the same operator as in Chapter 4) and proving a logspace preservation lemma for it (Lemma 88). The basic idea of this construction is inspired by the special sum operator in Chapter 4 or [36, 37]. First we need the following definition.

Definition 85. Given a subset S of the vertices of a digraph $\mathbf{G}$, let $S^{i=0}$ denote the set of vertices in S that have in-degree zero (in $\mathbf{G}$), and $S^{i>0}$ denote the set of vertices in S that have in-degree at least one. Similarly define $S^{o=0}$ and $S^{o>0}$ with respect to out-degrees.

Definition 86. The *special sum* operator takes two balanced digraphs $\mathbf{L}$ and $\mathbf{R}$, a vertex level $K_{\mathbf{L}}$ of $\mathbf{L}$, and a vertex level $K_{\mathbf{R}}$ of $\mathbf{R}$, and two subsets

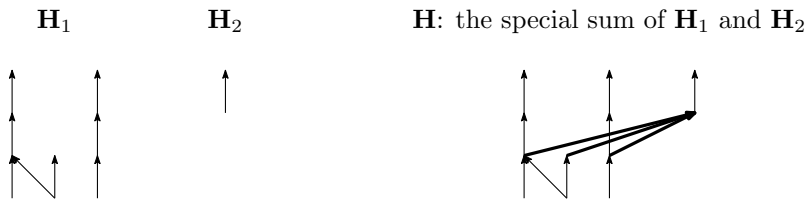


Figure 5.6: Example special sum construction.

$M_{\mathbf{L}} \subseteq K_{\mathbf{L}}^{i=0}$, and $M_{\mathbf{R}} \subseteq K_{\mathbf{R}}^{o=0}$. (If $\mathbf{L}$ has more than one component, then $K_{\mathbf{L}}$ is obtained by choosing a vertex level of each component and taking the union of these vertex levels. Similarly for $\mathbf{R}$ and $K_{\mathbf{R}}$.) The operator produces $\mathbf{H}$ which is obtained by taking the disjoint union of $\mathbf{L}$ and $\mathbf{R}$, and adding all edges of the form (u, v), where $u \in K_{\mathbf{R}}^{o>0} \cup M_{\mathbf{R}}$ and $v \in K_{\mathbf{L}}^{i>0} \cup M_{\mathbf{L}}$. (See the right side of Figure 5.7.)

Example 87. See Figure 5.6. The right side is obtained by applying the special sum operator to $\mathbf{H}_1$ and $\mathbf{H}_2$.

Lemma 88. *Let* $\mathbf{H}$ *be obtained as in Definition 86. Then if* $\mathrm{CSP}(\mathbf{L}^L)$ *and* $\mathrm{CSP}(\mathbf{R}^L)$ *are in logspace, then so is* $\mathrm{CSP}(\mathbf{H}^L)$.

Proof. Let $\mathbf{G}$ be a digraph with lists. If $\mathbf{G}$ maps to $\mathbf{H}^L$, then $\mathbf{G}$ must be balanced. We can check in logspace if a digraph is balanced (in fact, there is a simple symmetric Datalog program to do this). Let h be the height of $\mathbf{H}$. We can also output the height of $\mathbf{G}$ if it is at most h, and we can indicate if it is more than h, all in logspace. One way to do this is to write a symmetric program to check for each $\ell \leq h + 1$ if the input contains an oriented path of height ℓ. Run each symmetric program, and among those that accept,

choose the one which was checking for the largest height. If this number is at most h, output this number. Otherwise we can clearly reject the input. So now we can assume that $\mathbf{G}$ is balanced and has height at most h.

Using a similar argument to the one in Section 4.6.2 of Chapter 4, we can assume that the input is connected. It is obvious that if there is a homomorphism h from $\mathbf{G}$ to $\mathbf{H}^L$, then h pairs up the vertex levels of $\mathbf{G}$ and $\mathbf{H}$. That is, assume that the levels of $\mathbf{G}$ are $L_0, \ldots, L_n$, and the levels of $\mathbf{H}$ are $L'_0, \ldots, L'_m$. Then for any $0 \leq i \leq n$, there is a j such that every vertex in L_i is mapped to a vertex in L'_j. Furthermore, once two levels are matched, in our case L_i and L'_j, the rest of the pairings is forced in a trivial way. Therefore given our balanced input $\mathbf{G}$, we label its vertex levels by the levels of $\mathbf{H}$, and we check (later) the existence of a homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$ *with paired up levels* for all possible pairings. This overhead mechanism can be implemented in logspace, where we make use of Reingold's logspace algorithm for undirected st-connectivity [77]. So from now on, we assume that we already did all the above-described preprocessing and fixed which level of $\mathbf{G}$ maps to which level of $\mathbf{H}$.

We use the notation from Definition 86. In $\mathbf{H}$, let $U_{\mathbf{L}}$ be the set of vertices of $\mathbf{L}$ which are in the same level as the vertices in $K_{\mathbf{R}}$. Similarly, let $U_{\mathbf{R}}$ be the set of vertices of $\mathbf{R}$ that are in the same level as the vertices in $K_{\mathbf{L}}$. See the right side of Figure 5.7.

We give the proof referring to the example in Figure 5.7 (it is trivial to generalize the proof). Let I be the vertex level of $\mathbf{G}$ that is matched with

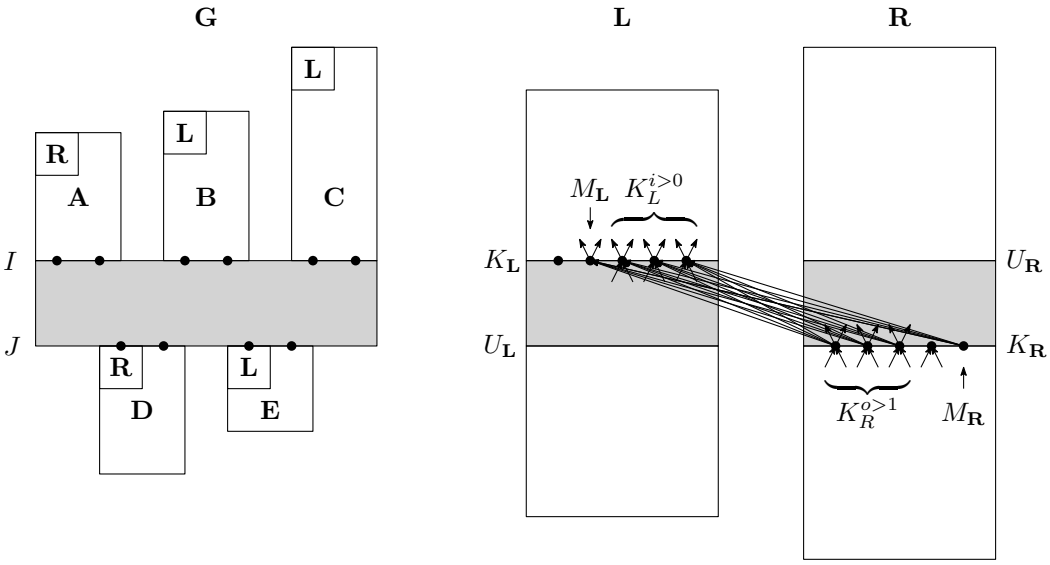


Figure 5.7: The special sum operator.

the level $K_{\mathbf{L}} \cup U_{\mathbf{R}}$ of $\mathbf{H}$, and let J be the level of $\mathbf{G}$ just below. Remove the edges induced by $I \cup J$ from $\mathbf{G}$ to yield components $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}$. We call the components above I ($\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$) *top components* and the components below J ($\mathbf{D}$, $\mathbf{E}$) *bottom components*. The following claim is easy to verify using the definition of the special sum operator:

Claim. *Let* $\mathbf{Q}$ *be a top component of* $\mathbf{G}$*, and* $\mathbf{Q}'$ *be* $\mathbf{Q}$ *modified in the following way. For any vertex* u *of* $\mathbf{Q}$ *in* I*, if the indegree of* u *is at least one (with respect to* $\mathbf{G}$*), then all elements of* $K_{\mathbf{L}}^{i=0}$ *are removed from the list of* u*. Assume there is a homomorphism* h *from* $\mathbf{G}$ *to* $\mathbf{H}^L$*. Then we can assume that* h *maps* $\mathbf{Q}$ *to* $\mathbf{L}$ *iff* $\mathbf{Q}'$ *homomorphically maps* $\mathbf{L}$*. A similar claim holds for bottom components.*

Using the above claim, we label a top component $\mathbf{Q}$ of $\mathbf{G}$ with $\mathbf{L}$ if $\mathbf{Q}'$ homomorphically maps to $\mathbf{L}$. To do this test, we use the algorithm for CSP($\mathbf{L}^L$) and input $\mathbf{Q}'$ where the lists of $\mathbf{Q}'$ are trimmed to contain only elements of L, and the lists of the vertices in the bottom level of $\mathbf{Q}'$ contain only vertices from $K_{\mathbf{L}}$. If $\mathbf{Q}'$ does not map to $\mathbf{L}$ but $\mathbf{Q}$ maps to $\mathbf{R}$ (the check is done similarly), we label the component with $\mathbf{R}$. If $\mathbf{Q}'$ does not map to $\mathbf{L}$ and $\mathbf{Q}$ does not map to $\mathbf{R}$, then there can be no homomorphism from $\mathbf{G}$ to $\mathbf{H}^L$ (recall again, this is only for one possible matching of the levels, so there could still be a homomorphism with some other matching). In an analogous way, we also label the bottom components. See the left side of Figure 5.7 where an example labeling is shown.

We produce the subgraph $\tilde{\mathbf{L}}$ of $\mathbf{G}$ which is the union the subgraphs that have label $\mathbf{L}$, in our case $\mathbf{B}, \mathbf{C}$ and $\mathbf{E}$, and all the edges from J to I that involve only vertices in $B \cup C \cup E$. We similarly construct $\tilde{\mathbf{R}}$. We check if there is a homomorphism from $\tilde{\mathbf{R}}$ to $\mathbf{R}$, and from $\tilde{\mathbf{L}}$ to $\mathbf{L}$ (levels matched, as before). We also check if there is no arc from J to I going from a vertex in $\tilde{\mathbf{L}}$ to $\tilde{\mathbf{R}}$. It is easy to check that $\mathbf{G}$ homomorphically maps to $\mathbf{H}^L$ iff all tests above pass. With a bit of work, all these can be implemented in logspace (using Reingold's undirected st-connectivity algorithm as a subroutine). □

Chapter 6

On Maltsev Digraphs and CSPs in L[1]

6.1 Introduction

The study of relational structures and, in particular, digraphs preserved by polymorphisms became extremely important during the last decade. For example, to establish the CSP dichotomy conjecture, it is sufficient to establish the conjecture restricted to digraph templates [43]. In this direction, [8] generalizes the graph dichotomy theorem of Hell and Nešetřil [51] to digraphs with no sinks and no sources by showing that digraph templates for which the CSP is tractable are very structured, in fact, they retract onto a disjoint union of directed cycles [8]. Other results relating the complexity of CSPs

[1]Some of the results in this chapter were obtained in collaboration with Catarina Carvalho, Marcel Jackson and Todd Niven, published in [23].

on digraphs to the existence of operations that preserve the digraph can be found, for example, in [7, 5]. The connection between the polymorphisms of a graph and its structure is widely studied, see for example [12, 42].

We study the structure of digraphs preserved by a Maltsev operation, that we call *Maltsev digraphs*. We show that these digraphs retract either onto the disjoint union of directed cycles or to a directed path. This gives a direct proof that the corresponding CSP is in constant width symmetric Datalog and therefore in L. (Membership of these CSPs in symmetric Datalog, without the constant width guarantee, was independently shown by Kazda [58], however, his proof is more indirect.)

We then generalize other results in [58] to show that a Maltsev digraph is preserved not only by a majority polymorphism, but also by a class of polymorphism obeying certain restrictions (e.g. minority, Pixley[2]). We also extend the results to the conservative setting, i.e. we show that a conservative Maltsev digraph is preserved by a class of conservative polymorphisms.

We apply our results to the list homomorphism problem discussed in Chapter 4. While we completely characterised the complexity of the list homomorphism problem for graphs, not much is known about the fine-grained complexity of the list homomorphism problem for digraphs. However, a P − NP dichotomy is known [52]. Toward a more refined classification, we show that the list homomorphism problem for Maltsev digraphs is in L. (See

[2]A Pixley operation m is a ternary operation satisfying $m(x,x,y) \approx m(y,x,x) \approx m(y,x,y) \approx y$.

also Chapter 5 for more results related to the directed case.)

A generalization of the rectangularity [14] property of digraphs is introduced. We call this rectangularity *total rectangularity*, and we establish that a digraph is preserved by a Maltsev operation iff it is totally rectangular. Similarly, we show that a digraph is preserved by a conservative Maltsev operation if and only if it is universally rectangular, a specific form of total rectangularity.

We also give an inductive construction of directed acyclic graphs preserved by a Maltsev operation. The main motivation behind this construction is that we suspect that extending this construction to "n-permutable digraphs" ("2-permutable digraphs" are precisely the Maltsev digraphs [48]) might make progress toward identifying all list homomorphism problems for digraphs in L. Recall that in Chapter 4, an inductive construction of "conservative n-permutable graphs" is key to the identification of all graphs whose list homomorphism problem is in L.

6.2 Retracts of Maltsev digraphs

Since all graphs in the rest of this chapter are directed, we use the terms graph and digraph interchangeably. Also, we often denote the vertex set of a digraph G with V_G and its edge set with E_G.

Definition 89 (totally rectangular)**.** A digraph G is *k-rectangular* if the

following implication holds for all vertices x, y, u, v:

$$x \xrightarrow{k} u \;\&\; y \xrightarrow{k} u \;\&\; y \xrightarrow{k} v \Rightarrow x \xrightarrow{k} v.$$

A digraph is *rectangular* if it is 1-rectangular, and *totally rectangular* if it is k-rectangular for every $k \in \mathbb{N}$.

It is not hard to verify that a Maltsev digraph must be totally rectangular, but in Section 6.3 (see Corollary 107) we show that the two properties are equivalent.

Example 90. The digraph in on the left in Fig. 6.1 is rectangular but not 2-rectangular, while the digraph on the right is totally rectangular.

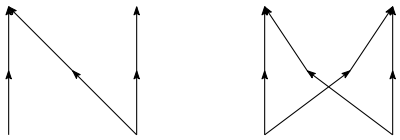


Figure 6.1: A rectangular digraph that is not 2-rectangular (left) and a digraph that is totally rectangular (right).

We now state the main result of this section.

Theorem 91. *Let G be a totally rectangular digraph. If G is acyclic then G retracts onto a simple dipath. Otherwise G retracts onto the disjoint union of simple directed cycles.*

The proof of Theorem 91 is a direct consequence of Lemma 98 below. We begin with some definitions and simple observations.

Lemma 92. *Let G be a DAG. Then G is balanced iff for every pair of vertices u, v in G, and any pair of oriented paths P and Q from u to v, it holds that $net(P) = net(Q)$.*

Definition 93. Let G be a digraph that contains a directed cycle. Let C be a shortest directed cycle in G and assume it has length m. We say that G is *inconsistent* if there exist two vertices u, v in G such that, there are two different oriented paths of net lengths ℓ_1 and ℓ_2 from u to v such that $\ell_1 \not\equiv \ell_2 \mod m$. Otherwise we say that G is *consistent*.

Proposition 94. *Let G be a digraph that contains a directed cycle. Let C be a shortest directed cycle in G. Then G retracts onto C iff G is consistent.*

Proof. Assume that C has length m. If G is consistent, then choose an arbitrary vertex of C and label it with 0. Call this vertex s. For each vertex $v \in G \backslash s$, find an oriented path P from s to v. Assume P has net length ℓ. Label v with the number $\ell \mod m$. Since G is consistent, this labeling is well-defined. It is easy to see that the mapping that sends every vertex with label i to the vertex of C that has label i is a retraction of G onto C.

Conversely, assume that G retracts to C through a homomorphism h. Label every vertex v with $h(v)$. It is not difficult to see that for any two vertices u and v of G such that there are two different (not necessarily simple) oriented paths of net lengths ℓ_1 and ℓ_2 from u to v, it must be that $\ell_1 \equiv \ell_2 \mod m$. □

Lemma 95. *Let G be a totally rectangular digraph and u, v be vertices in G. Let P and Q be two dipaths in G from u to v, such that $len(P) > len(Q)$. Set $k = len(P)$, $\ell = len(Q)$, and $d = k - \ell$. Then one of the following two cases occurs:*

1. *If $2\ell > k$, then G contains vertices u', v' and dipaths P', Q' from u' to v' with the following property: $len(P') = \ell$, $len(Q') = 2\ell - k$, and $len(P') - len(Q') = d$;*

2. *If $2\ell \leq k$, then G contains a directed cycle of length d.*

Proof. See Fig. 6.2. In the first case, let u' be the vertex of P such that the subpath $P_{u'v}$ of P from u' to v has length ℓ. Let v' be the vertex of P such that the subpath $P_{uv'}$ of P from u to v' has length ℓ. Applying the ℓ-rectangularity of G to $P_{u'v}$, Q, and $P_{uv'}$, we obtain the desired dipath P' with $len(P') = \ell$. The other required dipath Q' is the subpath of P from u' to v'. Since $k = 2\ell - len(Q')$ we have that $len(Q') = 2\ell - k$, and $len(P') - len(Q') = \ell - (2\ell - k) = k - \ell = d$.

In the second case, the two paths P' and Q' form a cycle of length $\ell + (k - 2\ell) = d$ because $2\ell \leq k$. □

Definition 96. Let P be an oriented path. Let $P_1, \ldots, P_n$ be all maximal (with respect to length) directed subpaths of P, such that P_i and P_{i+1}, $i \in [n-1]$ share a common vertex. We call $P_1, \ldots, P_n$ a *directed path decomposition of P*. We can obtain a *directed path decomposition for oriented*

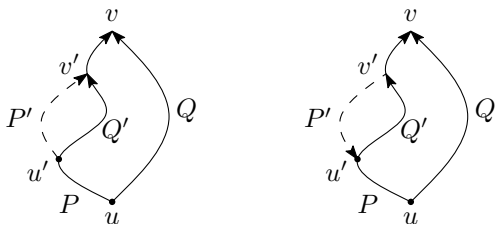


Figure 6.2: Case 1 (left) and Case 2 (right) of Lemma 95.

cycles in a similar way. Note that if an oriented cycle has two maximal directed subpaths, then these two subpaths share two vertices.

Lemma 97. *Let G be a totally rectangular digraph and $u, v \in V_G$. Let P_1 and P_2 be oriented paths in G from u to v. Assume that $net(P_1) > net(P_2)$, and set $d = net(P_1) - net(P_2)$. Then there are vertices $s, t \in V_G$ and dipaths Q_1 and Q_2 in G from s to t, such that $len(Q_1) - len(Q_2) = d$.*

Proof. Consider the oriented cycle C formed by P_1 and P_2. Let $R_0, \ldots, R_{n-1}$ be a directed path decomposition of C, and assume that $n \geq 2$. Find a shortest segment R_i in the decomposition and assume it has length k. Then both R_{i-1} and R_{i+1} have length at least k, where $i-1$ and $i+1$ are considered modulo n. Assume without loss of generality that R_{i-1} and R_i have the same endpoint b, and R_i and R_{i+1} have the same starting point c. Let a be the vertex of R_{i-1} at distance k from b (going backward on R_{i-1}), and d be the vertex of R_{i+1} at distance k from c. Then using the k-rectangularity of G, we can make a *shortcut* from a to d to obtain a new oriented cycle C' in G. We repeat these procedure until we obtain a cycle which has a directed path decomposition consisting only of two directed paths, Q_1 and Q_2. See

Figure 6.3 for an illustration.

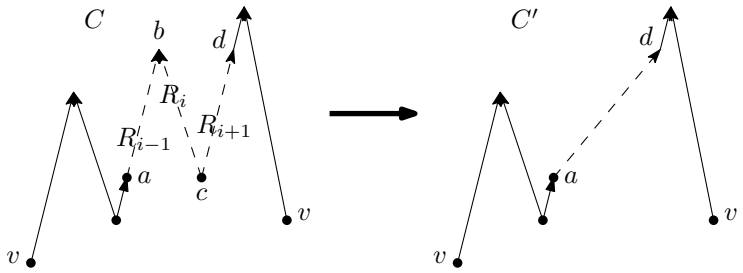


Figure 6.3: Making a shortcut.

Assume without loss of generality that Q_1 is longer. To see that $net(Q_1) - net(Q_2) = d$, observe that d is the difference of the forward and backward arcs in C. Applying the above shortcutting procedure to C to obtain C' does not change the difference of the forward and backward arcs in C. To see this, observe that we take out $2k$ forward arcs and k backward arcs from C, and then we add back k forward arcs to obtain C'. □

Lemma 98. *Let G be a connected totally rectangular digraph. If G is a DAG then G retracts onto a simple dipath. Otherwise G retracts onto a simple directed cycle.*

Proof. Assume first that G is a DAG. We claim that G must be balanced. Assume, for a contradiction, that G is not balanced. By Lemma 92, there exist $u, v \in V_G$ and oriented paths P and Q from u to v, such that $net(P) \neq net(Q)$. Using Lemma 97, we can assume that P and Q are dipaths of different length. Now we repeatedly apply Case 1 of Lemma 95 as long as it

is possible, and then applying Case 2 yields a cycle, a contradiction. So G is balanced.

Assume that G has levels $L_0, \ldots, L_{q-1}$. Fix vertices $s \in L_0$ and $t \in L_{q-1}$, and let O be any oriented path from s to t (such a path exists because G is connected). Applying the total rectangularity of G to appropriate subpaths of O, it is easy to see that there exists a dipath D of length $q-1$ from s to t in G. Clearly, G retracts onto D.

Suppose that G contains a directed cycle. By Proposition 94, it is enough to show that G is consistent. Assume this is not the case. Let C be a shortest directed cycle in G, and assume it has length m. Because G is inconsistent, we can find vertices $u, v \in V_G$ and oriented paths P_1 and P_2 from u to v, such that $net(P_1) \not\equiv net(P_2) \mod m$. Set $\ell_1 = net(P_1)$ and $\ell_2 = net(P_2)$. Assume w.l.o.g. that $\ell_1 > \ell_2$, and that u is a vertex of C. Note that if u is not a vertex of C, then we fix a vertex c of C and find any oriented path S from c to u. Then attaching S to P_1 and P_2 at vertex u gives us the desired oriented paths. Furthermore, we can assume that $\ell_1 - \ell_2 = d < m$, because if not, we can add C-loops from u to u to P_2 to increase its length by a multiple of m, until $\ell_1 - \ell_2 < m$. Using Lemma 97 we obtain directed paths Q_1 and Q_2 such that $len(Q_1) - len(Q_2) = d$, and then, by applying Lemma 95, we obtain a cycle of length d in G, a contradiction. □

By Lemma 98, each connected component of G retracts either onto a simple dipath or to a simple directed cycle. The trivial observation that a dipath homomorphically maps to a cycle completes the proof Theorem 91.

6.3 Characterisations, Polymorphisms and Algorithms

6.3.1 Rectangular Characterisations and Other Polymorphisms

In this section we generalise a technique of Kazda [58] to characterise digraphs that admit Maltsev and conservative Maltsev polymorphisms as those which are totally rectangular and universally rectangular respectively and to provide polynomial-time algorithms for recognising the relevant properties. Furthermore, we show that Maltsev digraphs also admit many other polymorphisms.

Definition 99 (conservatively k-rectangular, universally rectangular)**.** We say that a graph is *conservatively k-rectangular* if it satisfies the following sentence:

$$\left.\begin{array}{llllll} x \to & x_1 \to & \cdots \to & x_{k-1} \to & u \\ y \to & y_1 \to & \cdots \to & y_{k-1} \to & u \\ y \to & z_1 \to & \cdots \to & z_{k-1} \to & v \end{array}\right\} \Rightarrow \left\{\begin{array}{l} \text{There is a path } x \to w_1 \to \\ \cdots \to w_{k-1} \to v \text{ with } w_i \in \\ \{x_i, y_i, z_i\} \text{ for each } i. \end{array}\right. \tag{6.1}$$

A graph that is conservatively k-rectangular for all $k \geq 1$ will be called *universally rectangular*.

Example 100. The digraph on the right in Fig. 6.1 is conservatively rect-

angular but not conservatively 2-rectangular. While the digraph in Fig. 6.4 is universally rectangular.

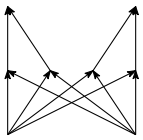


Figure 6.4: A universally rectangular digraph.

Definition 101. Let G be a digraph. Define the binary relations R^- on V_G by $x\ R^-\ y$ if $x^{-1} \cap y^{-1} \neq \varnothing$. The dual relation R^+ is defined by $x\ R^+\ y$ if $x^{+1} \cap y^{+1} \neq \varnothing$.

If G is rectangular, then the relation R^+ is an equivalence relation on the set $\{x \in V_G : x^+ \neq \varnothing\}$, the set of vertices of G that are not sinks. The relation R^- is an equivalence relation on the set $\{x \in V_G : x^- \neq \varnothing\}$, the set of vertices of G that are not sources. So it makes sense to consider the respective factor graphs, this was observed in [58]. We use the notation G/R^+ to denote the graph on the R^+-classes of G. Given R^+-classes A, B, we write $A \to B$ if there is some $a \in A$ and $b \in B$ with $a \to b$. Similarly, G/R^- denotes the same construction, but using the relation R^-. Note that G/R^+ is not strictly an actual graph quotient of G, only a quotient of an induced subgraph of G. Nevertheless, we sometimes refer to it as "the quotient of G by R^+".

Lemma 102. *Let G be a rectangular digraph and $k > 1$.*

1. *G is ℓ-rectangular for all $\ell = 1, \ldots, k$ if and only if G/R^+ is ℓ-rectangular for all $\ell = 1, \ldots, k-1$.*

2. *If G is conservatively ℓ-rectangular for all $\ell = 1, \ldots, k$ then G/R^+ is conservatively ℓ-rectangular for all $\ell = 1, \ldots, k-1$.*

Proof. The rectangularity of G ensures that G/R^+ is well-defined. Consider some $\ell \leq k-1$. We show that G is $(\ell+1)$-rectangular if and only if G/R^+ is ℓ-rectangular.

Consider vertices u, v, x, y such that $x/R^+ \stackrel{\ell}{\to} u/R^+$, $y/R^+ \stackrel{\ell}{\to} u/R^+$,and $y/R^+ \stackrel{\ell}{\to} v/R^+$ in G/R^+. This is equivalent to both the following properties holding:

1. there are $u_x, u_y \in u/R^+$ and $v_y \in v/R^+$ such that $x \stackrel{\ell}{\to} u_x$, $y \stackrel{\ell}{\to} u_x$, and $y \stackrel{\ell}{\to} v_y$ in G,

2. there are u', v' with $u \to u'$ and $v \to v'$ in G.

Now these two combined are equivalent to $x \stackrel{\ell+1}{\to} u', y \stackrel{\ell+1}{\to} u'$, and $y \stackrel{\ell+1}{\to} v'$ in G, a general instance of the premise of $(\ell+1)$-rectangularity of G. If G is $(\ell+1)$-rectangular then $x \stackrel{\ell+1}{\to} v'$ and as $v_x R^+ v$, we have that $x/R^+ \stackrel{\ell}{\to} v/R^+$ in G/R^+, showing that G/R^+ is ℓ-rectangular. Conversely, if G/R^+ is ℓ-rectangular then $x/R^+ \stackrel{\ell}{\to} v/R^+$ gives $x \stackrel{\ell+1}{\to} v'$, showing that G is $\ell+1$-rectangular. The second statement is proved similarly. □

For a totally rectangular graph G, define $G_0 = G$ and $G_{i+1} = G_i/R^+$, $i \geq 1$. From Lemma 102 it follows that G_i is defined for all positive integers

i, and eventually G_i will either be empty or a disjoint union of directed cycles (the only situations that R^+ can be trivial). We define $G_\infty = G_k$, where k is such that $G_k = G_{k+1}$ (i.e. G_∞ is either empty or a disjoint union of directed cycles).

The next lemma is easily obtained by applying the Maltsev property to the columns of the premise of (6.1) in Definition 99.

Lemma 103. *Let G be a digraph.*

1. *If G has a Maltsev polymorphism, then G is totally rectangular.*
2. *If G has a conservative Maltsev polymorphism, then G is universally rectangular.*

The next lemma is used in the proof of Theorem 106.

Lemma 104. *Let a, b be vertices in a totally rectangular digraph G satisfying conservative 2-rectangularity and assume that neither a nor b is a source or sink. If $a/R^+ \cap b/R^-$ is nonempty then either $b \in a/R^+$ or $a \in b/R^-$.*

Proof. Let $c \in a/R^+ \cap b/R^-$. There are vertices e, f, g, h such that $\{a, c\} \subseteq e^{-1}$, $b \in f^{-1}$, $a \in g^{+1}$ and $\{c, b\} \in h^{+1}$. However G is conservatively 2-rectangular so that there is either an edge from at least one of a, c to f or there is an edge from g to at least one of $\{b, c\}$. Then 1-rectangularity shows that either there is an edge from a to f or from g to b. □

To give details of the proof of Theorem 106, it is useful to recall the following collection of easy observations from [58].

Lemma 105. *Let G be a rectangular digraph. Then the following hold:*

1. *R^+ is an equivalence relation on $G \setminus S^+(G)$, where $S^+(G)$ is the set of sinks of G.*
2. *R^- is an equivalence relation on $G \setminus S^-(G)$, where $S^-(G)$ is the set of sources of G.*
3. *Whenever xR^+y, we have that $x^{+1} = y^{+1}$ and x^{+1} is an equivalence class of R^-.*
4. *Whenever xR^-y, we have that $x^{-1} = y^{-1}$ and x^{-1} is an equivalence class of R^+.*
5. *The mapping $\phi : X \to X^{+1}$ is a bijection from the set of equivalence classes of R^+ to the set of equivalence classes of R^-, and ϕ is an isomorphism from G/R^+ to G/R^-.*

Theorem 106. *Consider a property C of digraphs defined by the existence of polymorphisms $t_1, t_2, \ldots, t_k$ (not necessarily distinct) satisfying a single equational sequence*

$$t_1(x_{1,1}, x_{1,2}, \ldots, x_{1,n_1}) \approx \cdots \approx t_k(x_{k,1}, x_{n,2}, \ldots, x_{k,n_k}) \approx x,$$

where $\{x_{1,1}, \ldots, x_{1,n_1}\} = \cdots = \{x_{k,1}, \ldots, x_{k,n_k}\}$ and $x \in \{x_{1,1}, \ldots, x_{1,n_1}\}$. The following statements are true provided that the equation $x \approx y$ does not follow from C.

1. *Let G be a totally rectangular digraph. Then G has property C if and only if G_∞ has property C.*

2. *Let G be universally rectangular. Then G has property C with each t_i conservative if and only if G_∞ has property C with each of the t_i conservative.*

The same conclusions can be made without the requirement that $\approx x$ be included in the equational sequence, and if the polymorphisms are required to be idempotent.

Proof. Our proof is very similar to the main proof in [58]; we use Lemma 102 rather than the assumption of the Maltsev property directly. We focus only on the conservative case (not considered in [58]), as the non-conservative case is obtained by following this proof and missing some steps.

It is easy to see that if G has conservative property C then so does G/R^+ by defining $t_i(x_1/R^+, \ldots, x_n/R^+) = t_i(x_1, \ldots, x_n)/R^+$. Thus, it suffices to show that if G/R^+ satisfies some conservative property C then so does G. This uses only total rectangularity (to ensure that successive quotients are well defined).

We show the reverse direction by backward induction over successive quotients by R^+. It is useful to note that instead of explicit use of universal rectangularity, the argument uses only the fact that on each successive quotient by R^+, both rectangularity and the conclusion of Lemma 104 hold. Assume that G/R^+ has conservative polymorphisms $t_1^+, \ldots, t_k^+$ witnessing property C.

Let $t_i^-(x_1, \ldots, x_n)$ denote the operation given by $\phi(t_i^+(\phi^{-1}(x_1), \ldots, \phi^{-1}(x_n)))$ (ϕ is from Lemma 105). We show that one can construct conservative functions $t_1^G, \ldots, t_k^G$ on G such that the following hold for all $i = 1, \ldots, k$:

1. The equation sequence defining C holds;
2. If $x_1, \ldots, x_n \in V_G$ are not sinks then

$$t_i^G(x_1, \ldots, x_n)/R^+ = t_i^+(x_1/R^+, \ldots, x_n/R^+);$$

3. If $x_1, \ldots, x_n \in V_G$ are not sources then

$$t_i^G(x_1, \ldots, x_n)/R^- = t_i^-(x_1/R^-, \ldots, x_n/R^-).$$

We note that any function satisfying properties 1 and 2 is a polymorphism: this is proved in the case of majority function by Kazda [58] but his proof makes no use of the majority property, nor of the arity of the functions. Thus it remains to show that functions with all three properties stated above can be found.

We will assume that in the case where $\approx x$ is included in the definition of C, the values of the t_i are forced at many tuples. The assumption of idempotence on some of the terms in C similarly will force some constant tuples. Such forcing cannot result in the identification of distinct elements, as this enables a proof of the trivial equation $x \approx y$. Properties 2 and 3 hold trivially for such tuples.

We now fix some linear order $\leq_G$ on the vertices V_G and choose the values of $t^G(a_1, \ldots, a_n)$ which have not so far been forced. Assume that one of $a_1, \ldots, a_n \in V_G$ is a source. Choose the value of $t^G(a_1, \ldots, a_n)$ to be the smallest element a_l (under $\leq_G$) of $\{a_1, \ldots, a_n\}$ such that $a_l/R^+ = t^+(a_1/R^+, \ldots, a_n/R^+)$. This way condition 2 and 3 are satisfied. A dual statement holds if one of $x_1, \ldots, x_n$ is a sink. Note that if $t(a_1, \ldots, a_n)$ appears in the equation sequence definition C, then as the variables appearing throughout this sequence are the same and since G/R^+ satisfies the equations, the same $\leq_G$-earliest vertex will be selected in every case. Hence the equations are not violated by these selections.

Now consider the case where none of the $a_1, \ldots, a_n$ are sources or sinks. We show that the R^+-class $t^+(a_1/R^+, \ldots, a_n/R^+)$ intersects the R^--class $t^-(a_1/R^-, \ldots, a_n/R^-)$ and that one of the $a_1, \ldots, a_n$ lies in this intersection (this is the conservative part); this vertex will be chosen as the value of $t^G(a_1, \ldots, a_n)$.

Since t^+ and t^- are conservative functions, we have $t^+(a_1/R^+, \ldots, a_n/R^+) \in \{a_1/R^+, \ldots, a_n/R^+\}$ and $t^-(a_1/R^-, \ldots, a_n/R^-) \in \{a_1/R^-, \ldots, a_n/R^-\}$. Because any vertex $u \in V_G$ that is neither a source nor sink has $u \in u/R^+ \cap u/R^-$, we need to consider the situation where $t^+(a_1/R^+, \ldots, a_n/R^+)$ is the R^+-class of one of $\{a_1, \ldots, a_n\}$ but $t^-(a_1/R^-, \ldots, a_n/R^-)$ is the R^--class of a different vertex in $\{a_1, \ldots, a_n\}$. So assume that $t^+(a_1/R^+, \ldots, a_n/R^+) = a_l/R^+$ and $t^-(a_1/R^-, \ldots, a_n/R^-) = a_{l'}/R^-$, for some $l \neq l'$.

We now show that $a_l/R^+ \cap a_{l'}/R^- \neq \emptyset$. Since $a_i/R^+ \cap a_i/R^- \neq \emptyset$, for

each $i = 1, \ldots, n$, we have

$$\phi(\phi^{-1}(a_i/R^-)) \cap a_i/R^+ \neq \emptyset,$$

and therefore

$$(\phi^{-1}(a_i/R^-), a_i/R^+) \in E_{G/R^+}.$$

Thus

$$(t^+(\phi^{-1}(a_1/R^-), \ldots, \phi^{-1}(a_n/R^-)), t^+(a_1/R^+, \ldots, a_n/R^+)) \in E_{G/R^+},$$

from which it follows that

$$t^-(a_1/R^-, \ldots, a_n/R^-) \cap t^+(a_1/R^+, \ldots, a_n/R^+) = a_l/R^+ \cap a_{l'}/R^- \neq \emptyset.$$

As $a_l/R^+ \cap a_{l'}/R^- \neq \emptyset$, we can use Lemma 104 to show that there is $i \leq n$ with $a_i \in a_l/R^+ \cap a_{l'}/R^-$. Choose $t^G(a_1, \ldots, a_n)$ to be the $\leq_G$-earliest amongst the $a_i \in a_l/R^+ \cap a_{l'}/R^-$ (this takes care of any equations involving $t(a_1, \ldots, a_n)$). This completes the proof that there are $t_1^G, \ldots, t_k^G$ satisfying C and 2 and 3, as required.

□

Some instances of polymorphisms satisfying the conditions of Theorem 106 are majority, Maltsev and Pixley. In these cases G_∞ always has the desired polymorphism, giving the following corollary.

Corollary 107. *Let G be a digraph.*

1. *G admits a (conservative) Maltsev polymorphism iff it admits a (conservative) Pixley operation iff it is totally (universally) rectangular.*

2. *If G is totally (universally) rectangular then G admits a (conservative) minority polymorphism and a (conservative) majority polymorphism.*

Remark 108. *The first part of Corollary 107 strengthens the result given in Lemma 4 of [33], for the case of digraphs. The relational clone $\langle \mathbf{B} \rangle$ of a structure $\mathbf{B}$ is the set of all relations that can be expressed with primitive positive first-order formulas (i.e. only existential quantification, conjunction, and equality is allowed) from $\mathbf{B}$. When we restrict [33, Lemma 4] to digraphs, it can be stated as follows: A digraph G is preserved by a Maltsev operation iff every binary relation in $\langle G \rangle$ is rectangular. It is easy to see and well-known that every binary relation in $\langle G \rangle$ can be expressed as $B_G(S, a, b) = \{(h(a), h(b)) | h$ is a homomorphism from S to $G\}$ for a structure S with two distinguished vertices a and b. Then Corollary 107 implies that for a digraph G to be preserved by a Maltsev operation it is enough to require that only those binary relations in $\langle G \rangle$ that can be expressed as $B_G(S, a, b)$, where S is a directed path with initial vertex a and terminal vertex b, are rectangular.*

The above corollary yields an algorithm for verifying if a graph has a Maltsev (or Pixley) polymorphism. Indeed, the rectangularity of a digraph is equivalent to the following property of its adjacency matrix: when two rows (or two columns) share a common 1 they are identical. On an n-vertex

digraph this property may be verified in $O(n^3)$ steps. A digraph has a Maltsev polymorphism if and only if each (of at most n) successive quotient by R^+ is rectangular, with the process stopping once there are no R^+-classes of size more than 1 (which happens after at most n quotients). Overall this takes $O(n^4)$ steps (quadratic in terms of the size of the adjacency matrix).

Universal rectangularity (equivalently, the existence of a conservative Maltsev polymorphism) can also be verified in polynomial time by verifying total rectangularity and conservative 2-rectangularity at each successive quotient by R^+. In fact, the proof of Theorem 106 is sufficiently constructive to construct the desired polymorphisms (when they exist): simply work backwards from their definition of G_∞.

6.4 Constructing Maltsev DAGs

In this section, all graphs will be acyclic, i.e. DAGs. Recall that all DAGs preserved by a Maltsev polymorphism are balanced and retract onto a simple dipath, see Theorem 91.

We provide a simple inductive characterisation of totally rectangular DAGs. We note that in [58, Corollary 16] Kazda gives an inductive construction of Maltsev digraphs, however, this construction is not fully satisfying in the sense that it is non-deterministic, i.e. it does not specify how to obtain the desired preimages, and it is not clear if it can be made deterministic. The construction described below consists of repeated applications of

two straightforward steps (and their reverse versions) which clearly specify how to obtain a new Maltsev digraph from an already constructed one by a certain copying process. We need the following definitions.

Definition 109 ((Reverse) arborescence)**.** An *(reverse) arborescence* is a directed tree with root r such that every edge points away from (toward) r.

Definition 110 ($\nabla(r,h)$ and $\Delta(r,h)$)**.** Let G be a digraph, $r \in V_G$, and $h \in \mathbb{N}$. $\nabla(r,h)$ ($\Delta(r,h)$) is defined to be the subgraph of G whose vertices and edges are the vertices and edges of all (reverse) sub-dipaths of G which have initial vertex r and length h. A vertex $v \in \nabla(r,h) - \{r\}$ ($\Delta(r,h) - \{r\}$) is called an *endpoint* of $\nabla(r,h)$ ($\Delta(r,h)$) if there is a (reverse) dipath of length h from r to v. Otherwise v is called an *inner vertex* of $\nabla(r,h)$ ($\Delta(r,h)$).

Definition 111 (isolated $\nabla(r,h)$ and $\Delta(r,h)$)**.** Let G be a digraph. Consider $\nabla(r,h)$ ($\Delta(r,h)$) for some $r \in V_G$ and $h \in \mathbb{N}$. We say that $\nabla(r,h)$ ($\Delta(r,h)$) is *isolated in* G if for every inner vertex v of $\nabla(r,h)$, both the in-neighbourhood and the out-neighbourhood of v belongs to $\nabla(r,h)$ ($\Delta(r,h)$).

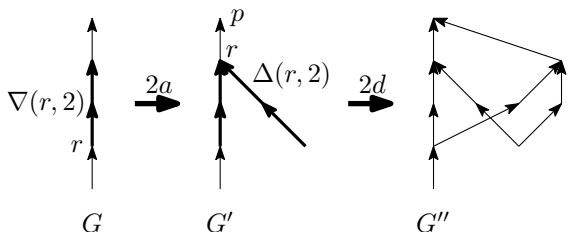


Figure 6.5: Construction of a totally rectangular DAG.

We are ready to define the construction formally in Fig. 6.6. This construction can be used, for example, to define a minority operation for a totally rectangular DAG, but this is a bit complicated and Theorem 106 is much more efficient.

1. $\mathcal{C}$ contains dipaths of all possible lengths $n \in \mathbb{N}_0$;
2. $\mathcal{C}$ is closed under applying the following operations:
   (a) Given a digraph G, let $r \in V_G$ and $h \in \mathbb{N}$ such that $\nabla(r, h)$ is an arborescence. Let ∇' be a copy of $\nabla(r, h)$. Join ∇' to G by identifying the corresponding endpoints of ∇' and $\nabla(r, h)$. Let the resulting graph be G';
   (b) Given a digraph G, let $r \in V_G$ be such that r has exactly one incoming edge (p, r), and $h \in \mathbb{N}$ such that $\nabla(r, h)$ is an isolated arborescence. Let ∇' be a copy of $\nabla(r, h)$ with root r'. Join ∇' to C by identifying the corresponding endpoints of ∇' and $\nabla(r, h)$, and adding the edge (p, r'). Let the resulting graph be G';
   (c) The reverse version of Step 2a (defined in the natural way);
   (d) The reverse version of Step 2b (defined in the natural way).
3. $\mathcal{M}$ is the set of digraphs that can be obtained by taking disjoint unions of digraphs in $\mathcal{C}$.

Figure 6.6: Inductive construction of the set $\mathcal{M}$ of totally rectangular DAGs.

Example 112. Consider the totally rectangular DAG G'' in Fig. 6.5. To construct it using the method in Fig. 6.6, we start with the dipath G and first apply Step 2a to G to obtain G'. Next we apply Step 2d to G' obtain

The class $\mathcal{M}$ is the set of all digraphs G such that every component C of G can be obtained as follows:

1. Initially, C is a directed path of arbitrary length;
   (a) Using Step 2c of the construction in Figure 6.6, build a balanced arborescence A.
   (b) Choose a vertex layer L_m such that each connected component of the subgraph between L_0 and L_m is a balanced arborescence (notice that these arborescences must have their endpoints in L_m).
      i. Choose a vertex $v \in L_k$, where $0 \leq k < m$. Apply Step 2a of the construction in Figure 6.6 with $\nabla(v, m-k)$ to obtain a new graph. Repeat this step as many times as you like.
      ii. Choose a vertex $v \in L_k$, where $0 < k < m$. Apply Step 2b of the construction in Figure 6.6 with $\nabla(v, m-k)$. Repeat this step as many times as you like.
      iii. Choose a vertex $v \in L_k$, where $0 < k < m$. Apply Step 2c of the construction in Figure 6.6.
   (c) Go back to Step 1b arbitrarily many times.

Figure 6.7: A more restricted construction of totally rectangular DAGs.

G''. The thick edges indicate the subgraphs $\nabla(r, 2)$ and $\Delta(r, 2)$, which are the subgraphs to be copied and attached appropriately.

In fact, we can extract from Lemma 119 a more restricted version of the construction in Figure 6.6 that is still sufficient. This restricted version is given in Figure 6.7.

Theorem 113. *The class of totally rectangular DAGs is the set of digraphs $\mathcal{M}$ defined in Fig. 6.6, or equivalently, the class of digraphs $\mathcal{M}$ defined in*

Figure 6.7.

6.4.1 Proof of Theorem 113

In the first part of this section, we prove one direction of the theorem stated in Corollary 115.

Lemma 114. *Let G be a connected digraph built using the construction in Figure 6.6 or Figure 6.7. Then G is totally rectangular.*

The corollary below easily follows:

Corollary 115. *Let G be a digraph built using the construction in Figure 6.6 or Figure 6.7. Then G is totally rectangular.*

Proof of Lemma 114. Because the construction in Figure 6.7 is a restricted version of the construction in Figure 6.6, from now on we focus only on the construction in Figure 6.6. We show that applying a construction step to a totally rectangular digraph G yields a totally rectangular digraph G'. Since a directed path is totally rectangular, this will prove the lemma.

Consider the case when G' was obtained from G by an application of Step 2a attaching ∇' to G. Take any 4 distinct vertices a, b, c, d such that there are directed paths P_{ab} from a to b, P_{cb} from c to b, and P_{cd} from c to d, each of some length ℓ. We show that there must be a directed path of length ℓ from d to a. Observe first that the construction has the property that there is no directed path from a vertex of G to a vertex of ∇', and vice versa,

except for the endpoints of ∇' (which are identified with the corresponding vertices of G). Let I be the common vertices of G and ∇'. We consider four cases:

1. *Vertices a and c are in G.* Then P_{ab}, P_{cd}, and P_{cp} are subpaths of G, so the total rectangularity of G provides a directed path of length ℓ from a to d.

2. *Vertex $a \in G$ and $c \in \nabla'$.* See Figure 6.8 for an illustration. Note that this is possible only if each path P_{ab}, P_{cd}, and P_{cb} has a vertex u, v and w, respectively, in I. Let P_{cv} denote the subpath of P_{cb} from c to v, and P_{vb} the subpath of P_{cb} from v to b. Similarly define the subpaths P_{cw} and P_{wd} of P_{cd}. Observe that by construction, P_{cv} is a copy of a dipath $P_{c'v}$ from a vertex c' to v in G. Similarly, P_{cw} is a copy of a dipath $P_{c'w}$ from a vertex c' to w in G. We have dipaths P_{ab} from a to b, $P_{c'v}P_{vb}$ from c' to b, and $P_{c'w}P_{wd}$ from c' to d, each of length ℓ and fully inside G. Therefore we have a dipath of length ℓ from a to d.

3. *Vertex $c \in G$ and $a \in \nabla'$.*

4. *Both vertices a and c are in ∇'.*

For the last two cases, an analysis fairly similar to the analysis of the second case can be done.

Suppose now that G' was obtained from G by an application of Step 2b of the construction. Let r, r', h, p, $\nabla(r, h)$, and ∇' be as in the definition

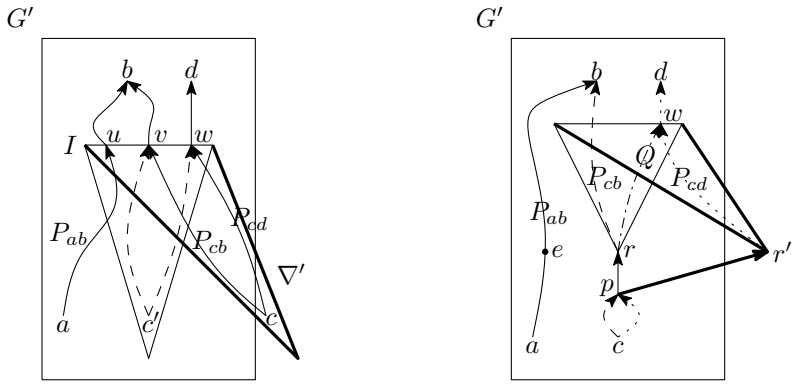


Figure 6.8: Left: Vertex $a \in G$ and $c \in \nabla'$ in analysis of Step 2a Right:

of Step 2b. Take any 4 distinct vertices a, b, c, d such that there are directed paths P_{ab} from a to b, P_{cb} from c to b, and P_{cd} from c to d, each of the same length ℓ. We show that there must be a directed path of length ℓ from d to a. If $level(a) = level(c) \geq level(r)$, then obviously, the same analysis works as for Step 2a. In addition, the vertex level of $level(b) = level(d) > level(p)$ because otherwise all paths are fully inside G and the total rectangularity of G proves the claim.

Furthermore, there must be two (distinct) edges e_1 and e_2 among the edges of P_{ab}, P_{cb} P_{cd}, such that e_1 is an edge in $A = \{\nabla(r, h) \cup (p, r)\}$ and $e_2 \in B = \{\nabla' \cup (p, r')\}$. To see this, remove A from G' to obtain graph G, and observe that F is isomorphic to G. Assume then that there is no edge of P_{ab}, P_{cb} P_{cd} in A. The total rectangularity of F yields the desired path from a to d. Because $\nabla(r, h)$ and ∇' are isolated and p is the only incoming arc of r and r', the only way that at least two of P_{ab}, P_{cb} P_{cd} can be in A and B is if at least two of P_{ab}, P_{cb} P_{cd} enters A and B through p.

If P_{ab} and P_{cd} both cross p, then we obviously have a dipath from a to d. If P_{ab} and P_{cb} meet at p then let P_{ap} be the subpath of P_{ab} from a to b, and P_{cp} the subpath of P_{cb} from c to p. Assume $k = len(P_{ap})(= len(P_{bp}))$, and let Q be the subpath of P_{cd} starting at c and taking the first k arcs. Suppose the endpoint of Q is e. Then by the total rectangularity of G, there is a path of length k from a to e. This path from a to e and the subpath of P_{cd} from e to d provides the desired path from a to d.

If P_{cb} and P_{cd} meet at p, then using the fact that G and F are isomorphic, we can assume WLOG that P_{cb} passes through r and P_{cd} passes through r'. Let $P_{r'w}$ be the subpath of P_{cd} from r' to an endpoint w. Because $P_{r'w}$ is entirely in B, $P_{r'w}$ is a copy of some dipath Q from r to w entirely in A and therefore entirely in G. Let P_{wd} be the subpath of P_{cd} from w to d. Let P_{eb} be the subpath of P_{ab} from e to b, where e is the vertex of P_{ab} such that $level(e) = level(r)$. Let P_{rb} be the subpath of P_{cb} from r to b. Then using the dipaths P_{eb}, P_{rb}, QP_{wd}, we obtain a dipath R from e to d using the total rectangularity of G. Then following P_{ab} up to e and then using R we obtain the desired dipath from a to d. □

The other direction of Theorem 113 follows from Lemma 119, which we prove in the rest of this section.

Lemma 116. *Assume that G is a totally rectangular DAG, r is a vertex of G, and $h \in \mathbb{Z}^+$ such that $A = \nabla(r, h)$ is an arborescence in G. Suppose that G contains a dipath $P = v_0, \ldots, v_h$ of length h such that v_h is an endpoint*

of A, but P and A do not share any other vertices. Then $A' = \nabla(v_0, h)$ is isomorphic to A through an isomorphism φ, such that φ is the identity map on the endpoints of A'. Furthermore, A' and A share only the endpoints.

Proof. We induct on the height h of A. When $h = 1$, the result trivially follows from the assumption that G is totally rectangular. Assume that $h > 1$. Remove the root r of A to obtain sub-arborescences $A_1, \ldots, A_s$, each of height $h-1$, and with roots $r_1, \ldots, r_s$, respectively. Fix an arbitrary endpoint e_i for each A_i. Once the following Claim is established, it is easy to see that the inductive hypothesis yields the result.

Claim: There are dipaths $Q_1, \ldots, Q_s$ of length h from v_0 to e_i, for each $i \in [s]$, such that the only vertex any pair of these paths share is v_0. Furthermore for any $i \in [s]$, Q_i does not contain any inner vertices of A.

If $s = 1$, i.e. r has only one out-neighbor, then the claim obviously follows. So assume that $s \geq 2$, and fix a dipath Q_i from v_0 to e_i for each $i \in [s]$ as follows. Let Q be a path from r to e_i. Since by assumption v_h is an endpoint of A, there is a path Q' from r to v_h. By the total rectangularity of G, there must be a path of length h from v_0 to e_i. Let this path be Q_i.

Assume for the sake of contradiction that there is pair of paths Q_i and Q_j such that they share the vertex x and $x \neq v_0$. Suppose x is as far as possible from v_0 (note that if the distance of x from v_0 to x is different on Q_i and Q_j, then G cannot be layered). Let $Q_{i,x}$ and $Q_{j,x}$ be the subpath of Q_i and Q_j from x to e_i and e_j, respectively. Suppose the $len(Q_{i,x}) = len(Q_{j,x}) = \ell$. Let R be the reverse directed path of length ℓ in S_i starting at e_i and ending at

some vertex t. Then $Q_{i,x}$, $Q_{j,x}$ and R together with the total rectangularity of G guarantee the existence of a path of length ℓ from t to e_j. This provides two distinct dipaths of length h from r to e_j, contradicting the fact that A is an arborescence.

Suppose for contradiction that for some $i \in [s]$, Q_i contains an inner vertex f of A. Let $Q_{i,f}$ be the subpath of Q_i from v_0 to f, and set $k = len(Q_{i,f})$. Let P_{rf} be a dipath from r to f of length k. Then by the rectangularity of G, the existence of the paths $Q_{i,f}$, P_{rf} and $v_0, \ldots, v_k$ guarantees the existence of a path of length k from r to v_k, and therefore v_k is an inner vertex of A, a contradiction. □

Proposition 117. *Let G be a digraph, r a vertex of G, and $h \in \mathbb{Z}^+$. Then $\nabla(r, h)$ is an arborescence if, and only if for every pair of distinct vertices v, u in $\nabla(r, h)$, there is at most one dipath from v to u.*

Proof. One direction is trivial. For the other direction, assume that for each pair of distinct vertices v, u there is at most one directed path from v to u. Then there is at most one directed path from r to any other vertex w of $\nabla(r, h)$. By the definition of $\nabla(r, h)$, there is at least one directed path from r to w. So there is exactly one directed path from r to w, and we are done. □

Lemma 118. *Let G be a totally rectangular DAG, r, a vertex of G, and $h \in \mathbb{Z}^+$. Assume that $\nabla(r, h)$ is isolated in G. Then $\nabla(r, h)$ is either an arborescence, or it can be obtained from an arborescence by repeated applica-*

tions of Step 2b of the construction in Figure 6.6.

Proof. If $\nabla(r, h)$ is an arborescence there is nothing to do. If $\nabla(r, h)$ is not an arborescence, then by Proposition 117, there is at least one pair of vertices u,v such that there are at least two distinct directed paths from u to v. It is sufficient to show that in this scenario, we can produce a subgraph G_s of G in which there are less directed paths from u to v, and an application of Step 2b of the construction yields G from G_s.

Take the largest $1 \leq g < h$ such that $\nabla(r, g)$ is still an arborescence (note that $\nabla(r, 1)$ is always an arborescence). Then we can find an endpoint v of $\nabla(r, g+1)$ end a vertex u in $\nabla(r, g)$ (a subgraph of $\nabla(r, g+1)$) such that there are two disjoint dipaths from u to v. Choose such a pair of vertices u and v such that these two distinct directed paths $P = u, w_1, \ldots, w_m, v$ and $Q = u, w'_1, \ldots, w'_m, v$ are shortest possible.

By the choice of u and v, there is no vertex pair u', v' in $\nabla(w_1, m)$ such that there is more than one directed path from u' to v'. Therefore $\nabla(w_1, m)$ is an arborescence. To see that $\nabla(w_1, m)$ is in fact an isolated arborescence, assume for contradiction that there is an edge (s, t) such that t is an inner vertex of $\nabla(w_1, m)$ and $s \notin \nabla(w_1, m)$. Then t is also an inner vertex of $\nabla(r, h)$, and because $\nabla(r, h)$ is isolated, s must be in $\nabla(r, h)$. This implies that $\nabla(r, h)$ contains some vertex s' such that there are at least two directed paths from s' to t. By Proposition 117, this would contradict the choice of g because $\nabla(r, g)$ is an arborescence. Similarly, $\nabla(w'_1, m)$ is an isolated arborescence in G. By a similar argument, w_1 or w_2 has only the in-neighbor

u.

Using the subpath $w'_1, \ldots, w'_m, v$ of Q and Lemma 116, we obtain that $\nabla(w'_1, m)$ is isomorphic to $\nabla(w_1, m)$. Let G_s be the graph obtained from G be removing $\nabla(w'_1, m)$ and the edge u, w'_1. It is clear now that we can use Step 2b of the construction to obtain G from G_s. □

Lemma 119. *Let G be a totally rectangular DAG. Then each component of G can be built using the construction in Figure 6.6.*

Proof. WLOG, we assume that G has only one component. By Theorem 91, G retracts onto a directed path of some length n. We show that if G is not a directed path, we can always define a totally rectangular DAG (which has one component) such that G' is smaller than G and an application of a step of the construction gives G from G'. Let L_0 be the bottommost vertex level of G. Let L_i be the first level above L_0 such that it contains a vertex w which has at least two in-neighbors. Consider the subgraph of G between L_0 and L_i and call it H.

Assume first that H contains a sink that is not in L_i. Let t be such a sink with the smallest possible level. By the choice of w, t has precisely one in-neighbor. Find a shortest reverse directed path P from t to a vertex c such that c has at least two out-neighbors. To see that such a vertex exists, observe that there is an oriented path Q from t to w because G is connected. Since t is a sink, the first arc of Q is a backward arc. As we follow the vertices of Q, eventually we must meet a forward arc because $level(t) < level(w)$.

Set $\ell = len(P)$. Since c has at least two out-neighbors, there is a dipath R starting at c and distinct from P. We can choose R so that it has length ℓ because t is a sink closest to L_0. Every vertex of R must have in-degree one because of the choice of w. Furthermore, P and R shares only the vertex c because otherwise the choice of w would be violated. Let the endpoint of R be t'. Remove P from G to obtain G'. Then clearly, G' is connected, totally rectangular, and we can obtain G back from G' by applying Step 2c with $\Delta(t', \ell)$.

Assume now that H contains no sinks except in L_i. Suppose that there is a source s in H such that $level(s) = i - k$ and $\nabla(s, k)$ is not an arborescence. Observe that by the choice of L_i, $\nabla(s, k)$ is isolated, and therefore by Lemma 118 we can assume that $\nabla(s, k)$ is an arborescence.

Suppose that there is a connected component of H that is not an (balanced) arborescence. Let C be a such a component. Then C contains more than one source, so let s and s' be sources such that $level(s') \leq level(s)$. Assume that $level(s) = i - k$. Then $\nabla(s, k)$ is an arborescence. In fact, $\nabla(s, k)$ is an isolated arborescence because H contains vertices with in-degree 2 and sinks only in L_i. Now using Lemma 116 and Step 2a, we can assume that each connected component of H is an arborescence.

Repeating this procedure as many times as necessary, we always decrease the present graph, and we end up with a balanced arborescence. This can easily be constructed using Step 2c from a directed path. □

A close reading of Lemma 119 reveals that if G is a totally rectangular

DAG, then each component of G can be built using the construction in Figure 6.6 in a very structured manner. This construction is explicit in Figure 6.7.

We observe the following properties of this restricted construction. In Steps 1(b)i and 1(b)ii, $\nabla(v, m-k)$ is *always isolated*. In Step 1(b)iii $F = \Delta(v, \cdot)$ is always a reverse directed path, and every vertex of F has precisely one in-neighbor.

6.5 Some Applications to the Constraint Satisfaction Problem

By Theorem 91, the core of a Maltsev digraph is either a directed path or a disjoint union of cycles, and for such digraphs the following is not difficult to show. In this section, we return to the notation from Chapter 4.

Corollary 120. *Let* $\mathbf{H}$ *be a Maltsev digraph. Then* co-CSP($\mathbf{H}$) *can be defined in symmetric Datalog of constant width and therefore* CSP($\mathbf{H}$) *is in* L.

Proof. By Theorem 3.1, we can assume that $\mathbf{H}$ is a dipath or a disjoint union of cycles. For such a digraph $\mathbf{H}$, it is not too difficult to write a symmetric program of constant width for co-CSP($\mathbf{H}$). The high level idea is the following. If $\mathbf{H}$ is a dipath, then we just have to check if the input graph contains any oriented path whose height is larger than the height of $\mathbf{H}$. If $\mathbf{H}$ has height ℓ, we have an IDB i for each $i \in \{0, 1, \ldots, \ell+1\}$. The program

"follows" an oriented path in the input and keeps track of the level of the current vertex using the above IDBs. If I_{i+1} is activated, then there is a path of height at least $i+1$ and the program accepts.

For an oriented cycle $\mathbf{C}$ consisting of m edges, $\mathbf{G}$ homomorphically *does not map* to $\mathbf{C}$ iff $\mathbf{C}$ contains an oriented cycle of net length ℓ such that $\ell \not\equiv 0 \mod m$. Again, this is easy to check with a constant width program. The programs for different cycles are easy to combine without blowing up the width, and we just give the basic idea. Assume for example that the target is the disjoint union of two oriented cycles for which we have programs $\mathcal{D}_1$ and $\mathcal{D}_2$. Instead of the goal predicate of $\mathcal{D}_1$ having arity zero, we modify the program so that the goal predicate has arity one and its variable is instantiated to a vertex v of the detected oriented cycle. Now we add a few extra rules that find an arbitrary oriented path starting at v and ending at some vertex u. Finally, we add $\mathcal{D}_2$ but we modify it so that any oriented cycle it detects contains the vertex u. □

The above results also have applications to the list homomorphism problem. In particular, the following result can be viewed as further extending the results in Chapter 4.

Corollary 121. *Let* $\mathbf{H}$ *be a digraph that has a conservative Maltsev polymorphism. Then* co-CSP($\mathbf{H}^L$) *is in symmetric Datalog (and therefore* CSP($\mathbf{H}^L$) *in* L*).*

Proof. Assume that the conservative Maltsev polymorphism is *mal*. Using

Corollary 107, it is easy to obtain a conservative majority polymorphism maj for $\mathbf{H}$. Because because both mal and maj are conservative, they also preserve $\mathbf{H}^L$. If a structure has a majority polymorphism, then the corresponding CSP is in (linear) Datalog [30]. We know from [31] and also from Chapter 3 that if co-CSP($\mathbf{H}^L$) is in Datalog and $\mathbf{H}^L$ is preserved by a Maltsev operation, then co-CSP($\mathbf{H}^L$) is in symmetric Datalog. □

Chapter 7

On CSPs in NL[1]

7.1 Introduction

In this chapter, we modestly contribute to the understanding of CSPs in NL. Based on the observation that any CSP known to be in NL is also known to be definable by a linear Datalog program, Dalmau conjectured that every CSP in NL can be defined by a linear Datalog program [27] (recall the discussion about Conjectures 1 and 2 in Subsection 1.3.3 of Chapter 1, and also note that we do not make any complexity theoretic assumptions in this chapter). Linear Datalog($\mathfrak{suc}$,$\neg$) (linDat($\mathfrak{suc}$,$\neg$)) denotes the extension of linear Datalog in which we allow negation and access to an order over the domain of the input. A linDat($\mathfrak{suc}$,$\neg$) program can be evaluated in NL, and it is also known that any problem in NL can be defined by a linDat($\mathfrak{suc}$,$\neg$)

[1]The contents of this chapter were published as the second part of [35].

program [27, 46, 56]. Therefore, one way to prove the above conjecture would be to show that any CSP that can be defined by a linDat($\mathfrak{suc}$,$\neg$) program can also be defined by a linear Datalog program (as suggested in Section 8 of [28], the journal version of [27]). We consider a restriction of the conjecture because proving it in its full generality would separate NL from P (using [1], and as mentioned in Subsection 1.3.3 of Chapter 1).

Read-once linear Datalog($\mathfrak{suc}$) (1-linDat($\mathfrak{suc}$)) is a subclass of linDat($\mathfrak{suc}$,$\neg$), but a subclass that has interesting computational abilities, and for which we are able to find the chink in the armor. We can easily define some NL-complete problems in 1-linDat($\mathfrak{suc}$), such as the CSP directed *st*-connectivity (*st*-Conn), and also problems that are not homomorphism-closed, such as determining if the input graph is a clique on 2^n vertices, $n \geq 1$, (where each vertex of a clique may or may not have a self-loop). Because any problem that can be defined with a linear Datalog program must be homomorphism-closed, it follows that 1-linDat($\mathfrak{suc}$) can define nontrivial problems which are in NL but which are not definable by any linear Datalog program. However, in this chapter we show that if co-CSP($\mathbf{B}$) can be defined by a 1-linDat($\mathfrak{suc}$) program, then co-CSP($\mathbf{B}$) can also be defined by a linear Datalog program. The crux of our argument applies the general case of the Erdős-Ko-Rado theorem to show that a 1-linDat($\mathfrak{suc}$) program does not have enough "memory" to handle structures of unbounded pathwidth.

Our proof establishing the above result for 1-linDat($\mathfrak{suc}$) programs can be adapted to show a parallel result for a subclass of *nondeterministic branching*

programs, which constitute an important and well-studied class of computational models (see the book [81]). More precisely, we show that if co-CSP($\mathbf{B}$) can be defined by a *poly-size family of read-once*[2] *monotone nondeterministic* branching programs (mnBP1(poly)) then co-CSP($\mathbf{B}$) can also be defined by a linear Datalog program.[3]

Finally, our results can be interpreted as lower-bounds on a wide class of CSPs: if $\mathbf{B}$ does not have bounded pathwidth duality, then co-CSP($\mathbf{B}$) cannot be defined with any 1-linDat($\mathfrak{suc}$) program or with any mnBP1(poly). A specific example of such a CSP would be the P-complete Horn-3Sat problem, and more generally, Larose and Tesson showed that any CSP whose associated *variety admits the unary, affine or semilattice types* does not have bounded pathwidth duality (see [63] for details).

7.2 Definitions

Let τ be a vocabulary. A *successor τ-structure* $\mathbf{S}$ is a relational structure with vocabulary $\tau \cup \{\mathfrak{first}, \mathfrak{last}, \mathfrak{suc}\}$, where $\mathfrak{first}$ and $\mathfrak{last}$ are unary symbols and $\mathfrak{suc}$ is a binary symbol. Without loss of generality, the domain S is defined as $\{1, \ldots, n\}$, $\mathfrak{first}^{\mathbf{S}} = \{1\}$, $\mathfrak{last}^{\mathbf{S}} = \{n\}$, and $\mathfrak{suc}^{\mathbf{S}}$ contains all pairs $(i, i+1)$, $i \in [n-1]$. Because $\mathfrak{first}^{\mathbf{S}}$, $\mathfrak{last}^{\mathbf{S}}$ and $\mathfrak{suc}^{\mathbf{S}}$ depend only on n, they

[2] Our read-once restriction for nondeterministic branching programs is less stringent than the usual restriction because we require the programs to be read-once only on certain inputs.

[3] A 1-linDat($\mathfrak{suc}$) can be converted into an mnBP1(poly), so another way to present our results would be to do the proofs in the context of mnBP1s, and then to conclude the parallel result for 1-linDat($\mathfrak{suc}$).

are called *built-in* relations. When we say that a class of successor structures is homomorphism/isomorphism-closed (isomorphism-closed is defined in the obvious way), all structures under consideration are successor structures, and we understand that homomorphism/isomorphism closure, respectively, is required only for non-built-in relations.

Definition 122 (Split Operation)**.** A *split operation* produces a τ-structure $\mathbf{A}'$ from a τ-structure $\mathbf{A}$ as follows. For an element $a \in A$ let T_a be defined as

$$T_a = \{(\mathbf{t}, R, i) \mid \mathbf{t} = (t_1, \ldots, t_r) \in R^{\mathbf{A}} \text{ where } R \in \tau, \text{ and } t_i = a\}.$$

If $|T_a| \leq 1$, no split operation can be applied. Otherwise we choose a strict nonempty subset T of T_a, and for each triple $(\mathbf{t}, R, i) \in T$, we replace $\mathbf{t} = (t_1, \ldots, t_r)$ in $R^{\mathbf{A}}$ with $(t_1, \ldots, t_{i-1}, a', t_{i+1}, \ldots, t_r)$p to obtain $\mathbf{A}'$, where a' is a new element and $A' = A \cup \{a'\}$.

Definition 123 (Split-Minimal, Critical)**.** Let $\mathcal{C}$ be a class of structures over the same vocabulary. We say that a structure $\mathbf{A} \in \mathcal{C}$ is *split-minimal in* $\mathcal{C}$ if for every possible nonempty sequence of split operations applied to $\mathbf{A}$, the resulting structure is not in $\mathcal{C}$. We say that a structure $\mathbf{A} \in \mathcal{C}$ is *critical in* $\mathcal{C}$ if no proper substructure of $\mathbf{A}$ is in $\mathcal{C}$.

For a class of successor τ-structures, criticality and split-minimality is meant only with respect to non-built-in relations.

Definition 124 (Read-Once Derivation)**.** We say that a derivation $\mathscr{D}$ is

read-once if every $R(\mathbf{t})$ that appears in $\mathscr{D}$ appears exactly once in $\mathscr{D}$, except when R is the special EDB $\mathfrak{suc}$, $\mathfrak{first}$, or $\mathfrak{last}$, defined above.

Definition 125 (Read-Once Datalog)**.** Let $\mathcal{P}$ be a (linear, symmetric) Datalog program that defines a class of structures $\mathcal{C}$. If for every critical and split-minimal element of $\mathcal{C}$ there is a $\mathcal{P}$-derivation that is read-once, then we say that $\mathcal{P}$ is *read-once*.

Definition 126 (Read-Once mnBP1)**.** A *monotone nondeterministic branching program* (mnBP) H with variables $X = \{x_1, \ldots, x_n\}$ computes a Boolean function $f_H : \{0,1\}^n \to \{0,1\}$. H is a directed graph with distinguished nodes s and t and some arcs labeled with variables from X (not all arcs must be labeled). An assignment σ to the variables in X defines in a natural way a subgraph H_σ of H. The function f_H is defined as $f_H(\sigma) = 1$ if and only if H_σ has a directed path from s to t (an *accepting path*). The size of an mnBP is $|V_H|$.

Let $\mathcal{F}$ be a poly-size family of mnBP1s (mnBP1(poly)) that defines a class of structures $\mathcal{C}$ over a vocabulary τ. (The encoding is done in the straightforward manner, i.e. there is a variable for every possible $(R, \mathbf{t})$ where $R \in \tau$ and $\mathbf{t}$ is a tuple.) If for every structure in $\mathcal{C}$ there is an accepting path that queries every variable at most once, then we say that $\mathcal{F}$ is *read-once*. (This read-once condition can be made a bit weaker.)

7.2.1 Examples

We give some examples of problems definable by a 1-linDat($\mathfrak{suc}$) program or by an mnBP1(poly). The program in Section 2.2.4, Figure 2.1 without rule 3 is a read-once linear Datalog($\mathfrak{suc}$) program that defines the problem directed st-Conn. To see that this program $\mathcal{P}_{st-\text{Conn}}$ is read-once, let $\mathbf{G}$ be any input that is accepted (we do not even need $\mathbf{G}$ to be critical and split-minimal). Then we find a directed path in $E^{\mathbf{G}}$ connecting an element of $S^{\mathbf{G}}$ to an element of $T^{\mathbf{G}}$ without repeated edges. We build a $\mathcal{P}_{st-\text{Conn}}$-derivation for this path in the obvious way.

For this section, by a clique we mean an ordinary undirected clique but each vertex may or may not have a self-loop. Let EvenCliques be the class of cliques of even size. The read-once linear Datalog($\mathfrak{suc}$) program $\mathcal{P}_{\text{EC}}$ below defines EvenCliques. The goal predicate of $\mathcal{P}_{\text{EC}}$ is G_2, and E is the symbol for the edge relation of the input. The first part of $\mathcal{P}_{\text{EC}}$ checks if the domain size n of the input is even. The second part goes through all pairs $(x, y) \in [n]^2$, and at the same time, checks if (x, y) is an edge in E. This is achieved by accessing the order on the domain. Program $\mathcal{P}_{\text{EC}}$ goes through every pair of vertices precisely once, so every $\mathcal{P}_{\text{EC}}$-derivation is read-once, and therefore $\mathcal{P}_{\text{EC}}$ is read-once.

In fact, we can easily test much more complicated arithmetic properties than the property of being even (e.g. being a power of k) with a 1-linDat($\mathfrak{suc}$) program. We note that EvenCliques or "cliques with any domain size property" cannot be defined by a linear Datalog program because a non-empty set

$$
\begin{aligned}
I(y) &\leftarrow \mathfrak{first}(x) \wedge \mathfrak{suc}(x,y) \\
I(z) &\leftarrow I(x) \wedge \mathfrak{suc}(x,y) \wedge \mathfrak{suc}(y,z) \\
G_1 &\leftarrow I(x) \wedge \mathfrak{last}(x) \\
J(x,y) &\leftarrow G_1 \wedge \mathfrak{first}(x) \wedge \mathfrak{first}(y) \\
J(x,z) &\leftarrow J(x,y) \wedge \mathfrak{suc}(y,z) \wedge E(x,z) \wedge E(z,x) \\
J(z,w) &\leftarrow J(x,y) \wedge \mathfrak{last}(y) \wedge \mathfrak{suc}(x,z) \wedge \mathfrak{suc}(z,w) \wedge \\
&\qquad E(z,w) \wedge E(w,z) \\
G_2 &\leftarrow J(x,y) \wedge \mathfrak{suc}(x,y) \wedge \mathfrak{last}(y).
\end{aligned}
$$

Figure 7.1: The read-once linear Datalog($\mathfrak{suc}$) program $\mathcal{P}_{\text{EC}}$ for Even-Cliques.

of cliques is never closed under homomorphisms. Since it is not difficult to convert a 1-linDat($\mathfrak{suc}$) program into an mnBP1(poly), the aforementioned problems can also be defined with an mnBP1(poly).

The additional power the successor relation gives to 1-linDat is at least twofold. For example, read-once linear Datalog($\mathfrak{suc}$) can do some arithmetic, as demonstrated above. In addition, let's define the *density* of a graph to be the number of edges divided by the number of vertices. The density of an n-clique is $\binom{n}{2}/n = \theta(n)$. As demonstrated above, access to an order allows read-once linear Datalog($\mathfrak{suc}$) to accept only structures of linear density. On the other hand, any linear Datalog program $\mathcal{P}$ accepts structures of arbitrary low density. For let $\mathbf{S}$ be a structure accepted by $\mathcal{P}$. Then adding sufficiently many new elements to the domain of $\mathbf{S}$ yields a structure $\mathbf{S}'$ whose density is arbitrarily close to 0, and $\mathbf{S}'$ is still accepted by $\mathcal{P}$. One consequence of

Corollary 128 is that if a read-once linear Datalog($\mathfrak{suc}$) defines co-CSP($\mathbf{B}$), then both aforementioned additional abilities are of no use.

7.3 Main Results

We begin with stating the results for 1-linDat($\mathfrak{suc}$) and poly-size families of mnBP1s discussed in the Introduction.

Theorem 127. *Let $\mathcal{C}$ be a homomorphism-closed class of successor τ-structures. If $\mathcal{C}$ can be defined by a 1-linDat($\mathfrak{suc}$) program of width (j, k), then every critical and split-minimal element of $\mathcal{C}$ has a $(j, k + j)$-path decomposition.*

Corollary 128. *If* co-CSP($\mathbf{B}$) *can be defined by a 1-linDat($\mathfrak{suc}$) program of width (j, k), then* co-CSP($\mathbf{B}$) *can also be defined by a linear Datalog program of width $(j, k + j)$.*

Theorem 129. *Let $\mathcal{C}$ be a homomorphism-closed class of successor τ-structures. If $\mathcal{C}$ can be defined by a family of mnBP1s of size $O(n^j)$, then every critical and split-minimal element of $\mathcal{C}$ has a $(j, r + j)$-path decomposition, where r is the maximum arity of the symbols in τ.*

Corollary 130. *If* co-CSP($\mathbf{B}$) *can be defined by a family of mnBP1s of size $O(n^j)$, then* co-CSP($\mathbf{B}$) *can also be defined by a linear Datalog program of width $(j, r + j)$, where r is the maximum arity of the relation symbols in the vocabulary of $\mathbf{B}$.*

As discussed before, a wide class of CSPs–CSPs whose associated *variety admits the unary, affine or semilattice types*–does not have bounded path-width duality [63]. It follows that all these CSPs are not definable by any 1-linDat($\mathfrak{suc}$) program, or with any mnBP1 of poly-size. An example of such a CSP is the P-complete CSP Horn-3Sat.

After some definitions, we give a high-level description of the proof of Theorem 127. Any τ-structure $\mathbf{M}$ with domain size n can be naturally converted into an isomorphic successor structure $\mathbf{M}(\pi)$, where π is a bijective function $\pi : M \to \{1, \ldots, n\}$. We define the domain $M(\pi)$ as $\{1, \ldots, n\}$ (note that this automatically defines $\mathfrak{first}^{M_\pi}$, $\mathfrak{last}^{M_\pi}$ and $\mathfrak{suc}^{M_\pi}$) and for any $R \in \tau$, and $(t_1, \ldots, t_{\mathrm{ar}(R)}) \in R^{\mathbf{M}}$, we place the tuple $(\pi(t_1), \ldots, \pi(t_{\mathrm{ar}(R)}))$ into $R^{\mathbf{M}_\pi}$). When we want to emphasize that a structure under consideration is a successor τ-structure, we use the subscript s, for example $\mathbf{M}_s$. Given a successor τ-structure $\mathbf{M}_s$, $\mathbf{M}$ denotes the structure $\mathbf{M}_s$ but with the relations $\mathfrak{first}^{\mathbf{M}_s}$, $\mathfrak{last}^{\mathbf{M}_s}$ and $\mathfrak{suc}^{\mathbf{M}_s}$ removed.

We make the simple but important observation that we are interested only in isomorphism-closed classes. For example, co-CSP($\mathbf{B}$) is obviously isomorphism-closed. We will crucially use the fact that if $\mathbf{M}_s$ is accepted by a 1-linDat($\mathfrak{suc}$) program $\mathcal{P}$, then $\mathcal{P}$ must also accept $\mathbf{M}(\pi)$ for any bijective function π. We are ready to describe the intuition behind the proof of Theorem 127.

A 1-linDat($\mathfrak{suc}$) program that ensures that the class of successor-structures $\mathcal{C}$ it defines is homomorphism-closed (and therefore isomorphism-closed) does

not have enough "memory"–due to its restricted width–to also ensure that some key structures in $\mathcal{C}$ are "well-connected". If these key structure are not too connected, then we can define co-CSP($\mathbf{B}$) in linear Datalog.

The more detailed proof plan is the following. Assume that co-CSP($\mathbf{B}$), where the input is a successor structure, is defined by a linDat($\mathfrak{suc}$) program $\mathcal{P}$ of width (j, k). We choose a "minimal" structure $\mathbf{M}$ in $\mathcal{C}$ that is accepted, and assume for contradiction that $\mathbf{M}$ does not have width (j, k). Then roughly speaking, for all possible "permutations of the domain elements of $\mathbf{M}$", $\mathbf{M}$ must be accepted; therefore for each of these isomorphic structures, $\mathcal{P}$ must be able to provide a derivation. Because this procedure will provide many enough derivations, we will be able to find some derivations which are of a desired form. The identification of these "good" derivations also crucially uses the generalized Erdős-Ko-Rado theorem. Once these derivations are detected, they can be combined to produce a derivation that "encodes" a structure of bounded pathwidth. The structures of bounded pathwidth produced this way can be used to define co-CSP($\mathbf{B}$) in linear Datalog. We give the formal proofs.

We need the following additional definitions for the proofs. In addition to extracting Ex($\mathscr{D}$) from $\mathscr{D}$, we can also extract a decomposition of Ex($\mathscr{D}$) reminiscent of a path decomposition. For each $\ell \in [q]$, we define a tuple structure $\tilde{\mathbf{B}}_\ell$ by adding $(R, \mathbf{t})$ to $\tilde{\mathbf{B}}_\ell$ if $R(\mathbf{t})$ appears in ρ_ℓ. In such a representation of Ex($\mathscr{D}$), we call $\tilde{\mathbf{B}}_\ell$ the ℓ-th *bag*, and $(\tilde{\mathbf{B}}_1, \ldots, \tilde{\mathbf{B}}_q)$ the *tuple distribution* of Ex($\mathscr{D}$). It will be useful to remove empty bags from the list of bags

$(\tilde{\mathbf{B}}_1, \ldots, \tilde{\mathbf{B}}_q)$ to obtain the sequence $(\tilde{\mathbf{B}}_{i_1}, \ldots, \tilde{\mathbf{B}}_{i_t})$, where $i_\ell < i_{\ell'}$ if $\ell < \ell'$. For simpler notation, we renumber $(\tilde{\mathbf{B}}_{i_1}, \tilde{\mathbf{B}}_{i_2} \ldots, \tilde{\mathbf{B}}_{i_s})$ to $(\tilde{\mathbf{B}}_1, \tilde{\mathbf{B}}_2, \ldots, \tilde{\mathbf{B}}_t)$. We call the sequence $(\tilde{\mathbf{B}}_1, \ldots, \tilde{\mathbf{B}}_t)$ the *pruned tuple distribution* of $\mathscr{D}$. The following is easy to prove.

Proposition 131. *Let $\mathbf{A}'$ be a τ-structure obtained from a τ-structure $\mathbf{A}$ by applying a sequence of split operations. Then $\mathbf{A}' \to \mathbf{A}$.*

We recall the following theorem tailored a bit to our needs.

Theorem 132 (Erdős-Ko-Rado, general case, see e.g. [44])**.** *Suppose that $\mathcal{F}$ is a family of s-subsets of $\{1, \ldots, n\}$, where $n \geq n_0(s, j+1)$. Suppose that for any two sets $S_1, S_2 \in \mathcal{F}$, $|S_1 \cap S_2| \geq j+1$. Then $|\mathcal{F}| \leq \binom{n-(j+1)}{s-(j+1)} = O(n^{s-(j+1)})$.*

Proof of Theorem 127. Let the read-once linear Datalog($\mathfrak{suc}$) program that defines $\mathcal{C}$ be $\mathcal{P}$. Let $\mathbf{M}$ be a structure in $\mathcal{C}$ such that $\mathbf{M}$ is critical and split-minimal, but assume for contradiction that $\mathbf{M}$ has no (j,k)-path decomposition. Suppose that $M = \{m_1, \ldots, m_s\}$. We choose a large enough n divisible by s (for convenience): how large n should be will become clear later. We begin with constructing a class of successor structures from $\mathbf{M}$. Let $\varphi : M \to \{1, \ldots, n\}$ be a function that for all $i \in [s]$, maps m_i to one of the numbers in $\left\{(i-1) \cdot \frac{n}{s} + 1, \ldots, i \cdot \frac{n}{s}\right\}$. We call such a function an *embedder*. Observe that there are $(\frac{n}{s})^s$ possible embedder functions. For each embedder φ, we define a successor structure $\mathbf{M}_\varphi$ as follows. $\mathbf{M}_\varphi$ is obtained

from $\mathbf{M}$ by renaming m_i to $\varphi(m_i)$ for each $i \in [s]$, and adding all numbers inside $\{1, \ldots, n\}$ but not in the range of φ to the domain of the structure.

Obviously for any embedder φ, $\mathbf{M}_\varphi$ contains an isomorphic copy of $\mathbf{M}$, and therefore $\mathbf{M} \to \mathbf{M}_\varphi$. Since $\mathcal{C}$ is closed under homomorphisms (and successor-invariant), it follows that for any embedder φ, $\mathbf{M}_\varphi$ is accepted by $\mathcal{P}$. Our goal now is to show that $\mathcal{P}$ accepts a structure that can be obtained from $\mathbf{M}$ by applying a nonempty sequence of split operations. This would contradict the split-minimality of $\mathbf{M}$ with respect to $\mathcal{C}$.

Let $\varphi_1, \ldots, \varphi_t$ be an enumeration of all $t = (\frac{n}{s})^s$ embedders, and $\mathbf{M}_{\varphi_1}, \ldots, \mathbf{M}_{\varphi_t}$ the corresponding successor structures. Since $\mathcal{P}$ is read-once, we can assume that for each $i \in [t]$, there is a read-once $\mathcal{P}$-derivation for $\mathbf{M}_{\varphi_i}$:

$$\mathscr{D}(\mathbf{M}_{\varphi_i}) = (\rho_1^i, \lambda_1^i), \ldots, (\rho_{q_i}^i, \lambda_{q_i}^i).$$

For each $\mathscr{D}(\mathbf{M}_{\varphi_i})$ we denote its pruned tuple distribution as $(\tilde{\mathbf{B}}_1^i, \ldots, \tilde{\mathbf{B}}_{w_i}^i)$. Let $\psi_i(\tilde{\mathbf{B}}_1^i, \ldots, \tilde{\mathbf{B}}_{w_i}^i)$ denote $(\tilde{\mathbf{M}}_1^i, \ldots, \tilde{\mathbf{M}}_{w_i}^i)$, where $\tilde{\mathbf{M}}_\ell^i$ for each $\ell \in [w_i]$ is obtained as follows. For every $(R, \mathbf{t}) \in \tilde{\mathbf{B}}_\ell^i$, place $(R, \varphi_i^{-1}(\mathbf{t}))$ into $\tilde{\mathbf{M}}_\ell^i$. We call $\psi_i(\tilde{\mathbf{B}}_1^i, \ldots, \tilde{\mathbf{B}}_{w_i}^i)$ the *prototype* of $(\tilde{\mathbf{B}}_1^i, \ldots, \tilde{\mathbf{B}}_{w_i}^i)$. We say that two pruned tuple distributions $(\tilde{\mathbf{B}}_1^i, \ldots, \tilde{\mathbf{B}}_{w_i}^i)$ and $(\tilde{\mathbf{B}}_1^{i'}, \ldots, \tilde{\mathbf{B}}_{w_{i'}}^{i'})$ are *similar* if they have the same prototypes, i.e. $\psi_i(\tilde{\mathbf{B}}_1^i, \ldots, \tilde{\mathbf{B}}_{w_i}^i) = \psi_{i'}(\tilde{\mathbf{B}}_1^{i'}, \ldots, \tilde{\mathbf{B}}_{w_{i'}}^{i'})$.

Note that the codomain of ψ_i, for any i, is a sequence S of bags such that a bag contains elements of $\tilde{\mathbf{M}}$. Because by definition, every bag in S is nonempty, and $\mathscr{D}(\mathbf{M}_{\varphi_i})$ is read-once, we have that $|S| \leq |\tilde{\mathbf{M}}|$. Therefore

the number of possible bag sequences can be upper-bounded by a function of s; let this upper bound be c_s. It follows that there must be at least $t' = \frac{t}{c_s}$ embedders $\varphi_{i_1}, \ldots, \varphi_{i_{t'}}$ such that for any $\ell, \ell' \in \{i_1, i_2, \ldots, i_{t'}\}$, $(\tilde{\mathbf{B}}^{\ell}_1, \ldots, \tilde{\mathbf{B}}^{\ell}_{w_\ell})$ and $(\tilde{\mathbf{B}}^{\ell'}_1, \ldots, \tilde{\mathbf{B}}^{\ell'}_{w_{\ell'}})$ are similar. Let the common prototype of all these similar pruned tuple distributions be $(\tilde{\mathbf{M}}_1, \ldots, \tilde{\mathbf{M}}_w)$ (*i.e.* $\psi_{i_1}(\tilde{\mathbf{B}}^{i_1}_1, \ldots, \tilde{\mathbf{B}}^{i_1}_{w_{i_1}})$). Because $\tilde{\mathbf{M}}$ is critical, it follows that $\tilde{\mathbf{M}} = \tilde{\mathbf{M}}_1 \cup \cdots \cup \tilde{\mathbf{M}}_w$.[4]

To give a heads-up to the reader, our goal now is to construct a derivation $\mathscr{D}'$ using the derivations $\mathscr{D}(\mathbf{M}_{\varphi_{i_1}}), \mathscr{D}(\mathbf{M}_{\varphi_{i_2}}), \ldots, \mathscr{D}(\mathbf{M}_{\varphi_{i_{t'}}})$, such that $\mathrm{Ex}(\mathscr{D}')$ is isomorphic to a structure $\tilde{\mathbf{M}}'$ that can be obtained from $\tilde{\mathbf{M}}$ by a nonempty sequence of split operations. Because $\tilde{\mathbf{M}}$ is split-minimal, this contradiction will complete the proof.

Define $X_g = \tilde{M}_1 \cup \cdots \cup \tilde{M}_g$, and $Y_g = \tilde{M}_g \cup \cdots \cup \tilde{M}_w$ for $g \in [w]$. If there is no $g \in [w-1]$ such that $|X_g \cap Y_{g+1}| > j$, then we construct a $(j, k+j)$-path decomposition $S_1, \ldots, S_w$ for $\mathbf{M}$ as follows. Define $S_1 = \tilde{M}_1$, $S_w = \tilde{M}_w$, and $S_\ell = \tilde{M}_\ell \cup (X_{\ell-1} \cap Y_{\ell+1})$, for $2 \leq \ell \leq w-1$. The first condition of Definition 17 is obviously satisfied. For the second condition, take S_i and $S_{i'}$ and $i < \ell < i'$. If $a \in S_i \cap S_{i'}$ then $a \in \tilde{M}_{i''}$ and $a \in \tilde{M}_{i'''}$ for some $i'' \leq i$ and $i' \leq i'''$, so $a \in S_\ell$. For the first part of the third condition observe that because $\mathcal{P}$ has width (j, k), $|\tilde{M}_\ell| \leq k$. Because we added at most j new elements to $\tilde{M}_\ell$ to obtain S_ℓ, $|S_\ell| \leq k + j$ for any ℓ. For the second part of

[4]Note that because $\tilde{\mathbf{M}}$ is critical and $\mathcal{C}$ is homomorphism closed, $\tilde{\mathbf{M}}$ cannot contain isolated elements except when $\tilde{\mathbf{M}}$ is a structure with a single element and no tuples. In this case the only critical and split-minimal element is $\tilde{\mathbf{M}}$ and the empty set is a $(0, 0)$-path decomposition for $\tilde{\mathbf{M}}$.

the third condition, observe that $S_\ell \subseteq X_\ell$ and $S_{\ell+1} \subseteq Y_{\ell+1}$, so $|S_\ell \cap S_{\ell+1}| \leq j$ for any ℓ.

For the other case, suppose that for some g, $|X_g \cap Y_{g+1}| > j$. Recall that for each $\ell \in \{i_1, i_2, \ldots, i_{t'}\}$, $\tilde{\mathbf{M}}_g$ was constructed from the bag $\tilde{\mathbf{B}}^\ell_g$, and $\tilde{\mathbf{B}}^\ell_g$ was constructed from a rule $\rho^\ell_{g_\ell}$ for some g_ℓ, i.e. the g_ℓ-th rule in the derivation $\mathscr{D}(\mathbf{M}_{\varphi_\ell}) = (\rho^\ell_1, \lambda^\ell_1), \ldots, (\rho^\ell_{q_\ell}, \lambda^\ell_{q_\ell})$. Let ι be the number of IDBs of $\mathcal{P}$ and κ the maximum arity of any IDB of $\mathcal{P}$. Recall that since $\mathcal{P}$ has width (j, k), any IDB contains at most j variables. Assume that the head IDB of $\rho^\ell_{g_\ell}$ is $I^\ell_{g_\ell}(\mathbf{x}^\ell{}_{g_\ell})$. Then there are at most $\iota j^\kappa n^j$ possibilities for the head IDB $I^\ell_{g_\ell}$ together with its variables instantiated to numbers in $[n]$. This means that there is an IDB I and a tuple $\mathbf{t}$ such that for at least $t'' = \frac{t'}{\iota j^\kappa n^j}$ values of $\ell \in \{i_1, i_2, \ldots, i_{t'}\}$, it holds that $I^\ell_{g_\ell} = I$, and $\lambda^\ell_{g_\ell}(\mathbf{x}^\ell{}_{g_\ell}) = \mathbf{t}$. Let these t'' values be $\{\ell_1, \ldots, \ell_{t''}\}$.

We establish later that we can choose values $\ell_a, \ell_b \in \{\ell_1, \ldots, \ell_{t''}\}$ such that the following inequality holds:

$$\left(\tilde{B}^{\ell_a}_1 \cup \cdots \cup \tilde{B}^{\ell_a}_w\right) \cap \left(\tilde{B}^{\ell_b}_1 \cup \cdots \cup \tilde{B}^{\ell_b}_w\right) \leq j. \tag{7.1}$$

Assuming that we have such ℓ_a and ℓ_b, we define $\mathscr{D}'$ as:

$$(\rho^{\ell_a}_1, \lambda^{\ell_a}_1), \ldots, (\rho^{\ell_a}_{g_{\ell_a}}, \lambda^{\ell_a}_{g_{\ell_a}}), (\rho^{\ell_b}_{g_{\ell_b}+1}, \lambda^{\ell_b}_{g_{\ell_b}+1}), \ldots, (\rho^{\ell_b}_{q_{\ell_b}}, \lambda^{\ell_b}_{q_{\ell_b}}).$$

That is, we "cut" the derivations $\mathscr{D}(\mathbf{M}_{\varphi_{\ell_a}})$ at the g_{ℓ_a}-th rule, and cut the derivation $\mathscr{D}(\mathbf{M}_{\varphi_{\ell_b}})$ at the g_{ℓ_b}-th rule, and concatenate the first part of

$\mathscr{D}(\mathbf{M}_{\varphi_{\ell_a}})$ with the second part of $\mathscr{D}(\mathbf{M}_{\varphi_{\ell_b}})$. $\mathscr{D}'$ is a valid derivation because at the point of concatenation, the head IDB of $\rho^{\ell_a}_{g_{\ell_a}}$ is the same as the IDB in the body of $\rho^{\ell_b}_{g_{\ell_b}+1}$, and the variables of this IDB are instantiated to the same values in both rules. Observe that the pruned tuple distribution of $\mathscr{D}'$ is $(\tilde{\mathbf{B}}^{\ell_a}_1, \ldots, \tilde{\mathbf{B}}^{\ell_a}_g, \tilde{\mathbf{B}}^{\ell_b}_{g+1}, \ldots, \tilde{\mathbf{B}}^{\ell_b}_w)$. Set $\tilde{\mathbf{B}} = \tilde{\mathbf{B}}^{\ell_a}_1 \cup \cdots \cup \tilde{\mathbf{B}}^{\ell_a}_g \cup \tilde{\mathbf{B}}^{\ell_b}_{g+1} \cup \cdots \cup \tilde{\mathbf{B}}^{\ell_b}_w$.

Claim. $\tilde{\mathbf{B}}$ *is isomorphic to a structure that can be obtained from* $\tilde{\mathbf{M}}$ *by a nonempty sequence of split operations.*

Proof of Claim. Observe that the substructure $\tilde{\mathbf{M}}_1 \cup \cdots \cup \tilde{\mathbf{M}}_g$ of $\tilde{\mathbf{M}}$ is isomorphic to $\tilde{\mathbf{B}}^{\ell_a}_1 \cup \cdots \cup \tilde{\mathbf{B}}^{\ell_a}_g$ through φ_{ℓ_a}. Similarly, $\tilde{\mathbf{M}}_{g+1} \cup \cdots \cup \tilde{\mathbf{M}}_w$ is isomorphic to $\tilde{\mathbf{B}}^{\ell_b}_{g+1} \cup \cdots \cup \tilde{\mathbf{B}}^{\ell_b}_w$ through φ_{ℓ_b}. Our goal is to understand the difference between $\tilde{\mathbf{M}}$ and $\tilde{\mathbf{B}}$.

Notice that because any embedder maps $m_i \in M$ into the interval

$$\left\{(i-1)\cdot\frac{n}{s}+1, \ldots, i\cdot\frac{n}{s}\right\},$$

and for any $i \neq i'$,

$$\left\{(i-1)\cdot\frac{n}{s}+1, \ldots, i\cdot\frac{n}{s}\right\} \cap \left\{(i'-1)\cdot\frac{n}{s}+1, \ldots, i'\cdot\frac{n}{s}\right\} = \emptyset,$$

if $i \neq i'$, then $\varphi_{\ell_a}(m_i) \neq \varphi_{\ell_b}(m_{i'})$. Therefore φ_{ℓ_a} and φ_{ℓ_b} can return the same value only if they both get the same input. The set $X_g \cap Y_{g+1}$ can be thought of as those elements of $\tilde{\mathbf{M}}$ where $\tilde{\mathbf{M}}_1 \cup \cdots \cup \tilde{\mathbf{M}}_g$ and $\tilde{\mathbf{M}}_{g+1} \cup \cdots \cup \tilde{\mathbf{M}}_w$ are "glued together" to obtain $\tilde{\mathbf{M}}$. Let $U = \tilde{B}^{\ell_a}_1 \cup \cdots \cup \tilde{B}^{\ell_a}_g$ and $V = \tilde{B}^{\ell_b}_{g+1} \cup \cdots \cup \tilde{B}^{\ell_b}_w$.

The set $U \cap V$ can be thought of as those elements of $\tilde{\mathbf{B}}$ where $\tilde{\mathbf{B}}_1^{\ell_a} \cup \cdots \cup \tilde{\mathbf{B}}_g^{\ell_a}$ and $\tilde{\mathbf{B}}_{g+1}^{\ell_b} \cup \cdots \cup \tilde{\mathbf{B}}_w^{\ell_b}$ are "glued together" to obtain $\tilde{\mathbf{B}}$.

If for all elements $m \in X_g \cap Y_{g+1}$, $\varphi_{\ell_a}(m) = \varphi_{\ell_b}(m)$, then $\tilde{\mathbf{B}}$ would be isomorphic to $\tilde{\mathbf{M}}$, i.e. $\tilde{\mathbf{B}}_1^{\ell_a} \cup \cdots \cup \tilde{\mathbf{B}}_g^{\ell_a}$ would be glued to $\tilde{\mathbf{B}}_{g+1}^{\ell_b} \cup \cdots \cup \tilde{\mathbf{B}}_w^{\ell_b}$ to obtain $\tilde{\mathbf{B}}$ the same way as $\tilde{\mathbf{M}}_1 \cup \cdots \cup \tilde{\mathbf{M}}_g$ is glued to $\tilde{\mathbf{M}}_{g+1} \cup \cdots \cup \tilde{\mathbf{M}}_w$ to obtain $\tilde{\mathbf{M}}$. But by Inequality 7.1, $|X_g \cap Y_{g+1}| > |U \cap V|$. In other words, there are some elements $m \in X_g \cap Y_{g+1}$ which have one copy for φ_{ℓ_a}, and another copy for φ_{ℓ_b} in $\tilde{\mathbf{B}}$. Identifying $\varphi_{\ell_a}(m)$ and $\varphi_{\ell_b}(m)$ for all such m would convert $\tilde{\mathbf{B}}$ to a structure isomorphic to $\tilde{\mathbf{M}}$. Now it is easy to see that going backwards, splitting elements of $\tilde{\mathbf{M}}$ would yield a structure isomorphic to $\tilde{\mathbf{B}}$. □

It remains to show why we can choose ℓ_a and ℓ_b to satisfy Inequality 7.1. Note that $t'' = \frac{\binom{n}{s}^s}{c_s \iota j^\kappa n^j} \geq \Omega(n^{s-j})$. Also note that for any $\ell' \in \{\ell_1, \ldots, \ell_{t''}\}$, $\tilde{B}_1^{\ell'} \cup \cdots \cup \tilde{B}_w^{\ell'}$ is an s-subset of $[n]$. So by Theorem 132, if for every pair $\ell_a, \ell_b \in \{\ell_1, \ldots, \ell_{t''}\}$, $\left(\tilde{B}_1^{\ell_a} \cup \cdots \cup \tilde{B}_w^{\ell_a}\right) \cap \left(\tilde{B}_1^{\ell_b} \cup \cdots \cup \tilde{B}_w^{\ell_b}\right) \geq j+1$, then $t'' \leq O(n^{s-j-1})$. But as observed $t'' \geq \Omega(n^{s-j})$, so for a large enough n (as a function of s,j, ι and κ, so n can be chosen in advance) Inequality 7.1 must hold for some $\ell_a, \ell_b \in \{\ell_1, \ldots, \ell_{t''}\}$. □

Proof of Corollary 128. Let $\mathcal{O} = \text{co-CSP}(\mathbf{B})$, i.e. the set of all those successor structures that do not homomorphically map to $\mathbf{B}$. We construct an obstruction set $\mathcal{O}'$ for $\mathbf{B}$ such that every structure in $\mathcal{O}'$ has pathwidth $(j, k+j)$. $\mathcal{O}'$ is the set of all critical and split minimal structures of $\mathcal{O}$. Theorem 127 tells us that every structure in $\mathcal{O}'$ has a $(j, k+j)$-path decomposition.

To see that $\mathcal{O}'$ is an obstruction set for $\mathbf{B}$, take any structure $\mathbf{S} \in$ co-CSP($\mathbf{B}$) $= \mathcal{O}$. Keep on applying split operations to $\mathbf{S}$ and taking substructures of $\mathbf{S}$ (again, these operations are with respect to non-built-in relations only), as long as the resulting structure is still in $\mathcal{O}$. That is, if we apply any split operation to $\mathbf{S}'$, or if we take any substructure of it, then the resulting structure is not in $\mathcal{O}$ any more. Then $\mathbf{S}' \in \mathcal{O}'$ because $\mathbf{S}'$ is critical and split minimal with respect to $\mathcal{O}$. Using Proposition 131, we also see that $\mathbf{S}' \to \mathbf{S}$.

Because $\mathcal{O}'$ is an obstruction set for $\mathbf{B}$ such that every structure in $\mathcal{O}'$ has width $(j, k+j)$, it follows from [27] that co-CSP($\mathbf{B}$) is definable in linear $(j, k+j)$-Datalog. □

These proofs can be easily adapted for mnBP1s to obtain Theorem 129 and Corollary 130.

Chapter 8

Conclusions and Open Problems

This thesis, among other things, establishes the membership of some large classes of CSPs in the complexity class L and also in symmetric Datalog. In Chapter 3, we described two new dualities to show membership of CSPs in symmetric Datalog. As an application, we gave a short new proof that "Maltsev + Datalog $\Rightarrow$ symmetric Datalog", i.e. if co-CSP($\mathbf{B}$) is expressible in Datalog and $\mathbf{B}$ has a Maltsev polymorphism then co-CSP($\mathbf{B}$) is in symmetric Datalog. In Chapter 4, we completely characterized the complexity of the list homomorphism for graphs, and in particular, we identified all list homomorphism problems in L (modulo standard complexity theoretic assumptions) and in symmetric Datalog. In Chapter 6, among other things, we showed that the list homomorphism for Maltsev digraphs is in symmetric

Datalog.

All the above result support the algebraic symmetric Datalog conjecture, i.e. the conjecture that co-CSP($\mathbf{B}$) is in symmetric Datalog if and only if the variety corresponding to $\mathbf{B}$ admits only the Boolean type. One of the next research challenges is to characterize those *digraphs* for which the list homomorphism problem is in L (and symmetric Datalog). A more specific question is: is the class of digraphs that have polymorphisms $f_1, \ldots, f_n$, for some $n \geq 1$, satisfying identities (4.1)–(4.3) (see Chapter 4) the class of digraphs for which the list homomorphism problem is in L (and symmetric Datalog)? ($\star$) Note that the existence of the above polymorphisms is necessary by Lemma 45 because if any of the unary, lattice or semilattice types is present, then co-CSP($\mathbf{B}$) cannot be in symmetric Datalog by results in [63]. The characterization of the $\mathsf{L} - \mathsf{NL}$ dichotomy for the list homomorphism problem for oriented paths, and the result in Chapter 6 that for Maltsev digraphs the list homomorphism problem is in symmetric Datalog can be considered as initial steps toward identifying all digraphs for which the list homomorphism problem is in symmetric Datalog and L.

Remark: Note the subtle issue that in the above question indicated by ($\star$), we are also indirectly asking the following: if the unary, lattice and semilattice types are omitted in the variety corresponding to a digraph with lists, does that imply that the affine type is also omitted? In particular, observe that if $n = 1$, then the identities (4.1)–(4.3) force f_1 to be a Maltsev operation. So the answer to the above question when $n = 1$ is yes, since if

$\mathbf{G}$ has a Maltsev polymorphism, then it also has a majority polymorphism (both in the conservative and non-conservative cases, see [58] and Chapter 6), and therefore Lemma 45 tells us that the affine type is omitted.

Applying a recent breakthrough result [6], the algebraic symmetric Datalog conjecture can be proved by showing that if co-CSP($\mathbf{B}$) is in Datalog and $\mathbf{B}$ has polymorphisms $f_1, \ldots, f_n$, for some $n \geq 1$, satisfying identities (4.1)–(4.3) (for simplicity, we will say that $\mathbf{B}$ is $(n+1)$-permutable), then co-CSP($\mathbf{B}$) is in symmetric Datalog. This seems to be a tough question and one intermediate step could be to show the weaker result that if co-CSP($\mathbf{B}$) is in linear Datalog and $\mathbf{B}$ is $(n+1)$-permutable, then co-CSP($\mathbf{B}$) is in symmetric Datalog. One way to attack this problem would be trying to generalize our simple proof relying on symmetric bounded pathwidth duality that if $\mathbf{B}$ has a Maltsev polymorphism and co-CSP($\mathbf{B}$) is in (linear) Datalog then in fact, co-CSP($\mathbf{B}$) is in symmetric Datalog (Chapter 3).

The algebraic linear Datalog conjecture states that co-CSP($\mathbf{B}$) is linear Datalog if and only if the variety corresponding to $\mathbf{B}$ contains only the Boolean or lattice types. Dalmau and Larose made a first key step in this direction by proving that if $\mathbf{B}$ has a majority polymorphism, then co-CSP($\mathbf{B}$) is in linear Datalog. This result has been generalized recently: if $\mathbf{B}$ has a near-unanimity polymorphism, then co-CSP($\mathbf{B}$) is in linear Datalog [10]. The obvious open problem is to generalize these results to show the algebraic linear Datalog conjecture. A starting point is given in [54], where Theorem 9.11 tells us what it means that a variety corresponding to $\mathbf{B}$ contains only the

Boolean or lattice types in terms of polymorphisms.

Bibliography

[1] F. Afrati and S. S. Cosmadakis. Expressiveness of restricted recursive queries. In *Proceedings of the 42th ACM Symposium on Theory of Computing (STOC)*, pages 113–126, 1989. 5, 125

[2] E. Allender, M. Baul, N. Immerman, H. Schnoor, and H. Vollmer. The complexity of satisfiability problems: refining Schaefer's theorem. In *In Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, pages 71–82. Springer-Verlag, 2005. 10, 47

[3] E. Allender, M. Bauland, N. Immerman, H. Schnoor, and H. Vollmer. The complexity of satisfiability problems: Refining Schaefer's theorem. *Journal of Computer and System Sciences*, 75(4):245–254, 2009. 3

[4] L. Barto. The dichotomy for conservative constraint satisfaction problems revisited. In *LICS*, pages 301–310, 2011. 47

[5] L. Barto and J. Bulin. CSP dichotomy for special polyads. *Submitted*, 2012. 102

[6] L. Barto and M. Kozik. Constraint satisfaction problems of bounded width. In *Proceedings of The 50th Annual Symposium on Foundations of Computer Science (FOCS)*, 2009. 3, 4, 13, 20, 137

[7] L. Barto, M. Kozik, M. Maróti, and T. Niven. CSP dichotomy for special triads. *Proceedings of the AMS*, 137:2921–2934, 2009. 102

[8] L. Barto, M. Kozik, and T. Niven. Graphs, polymorphisms and the complexity of homomorphism problems. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 789–796, 2008. 2, 102

[9] L. Barto, M. Kozik, and T. Niven. The CSP dichotomy holds for digraphs with no sources and no sinks (a positive answer to a conjecture of bang-jensen and hell). *SIAM J. Comput.*, 38(5):1782–1802, 2009. 2, 46, 66

[10] L. Barto, M. Kozik, and R. Willard. Near unanimity constraints have bounded pathwidth duality. In *LICS*, pages 125–134, 2012. 13, 137

[11] M. Bodirsky and J. Nešetřil. Constraint satisfaction with countable homogeneous templates. In *Proceedings of Computer Science Logic and the 8th Kurt Gödel Colloquium*, volume 2803 of *Lecture Notes in Computer Science*, pages 44–57. Springer-Verlag, 2003. 11

[12] R. C. Brewster, T. Feder, P. Hell, J. Huang, and G. MacGillivray. Near-unanimity functions and varieties of reflexive graphs. *SIAM J. Discrete Math.*, 22(3):938–960, 2008. 49, 64, 102

[13] A. Bulatov. A dichotomy theorem for constraints on a three-element set. In *Proceedings 43rd IEEE Symposium on Foundations of Computer Science, FOCS'02*, pages 649–658. IEEE Computer Society, 2002. 2

[14] A. Bulatov and V. Dalmau. A simple algorithm for Mal'tsev constraints. *SIAM Journal on Computing*, 36(1):16–27, 2006. 13, 103

[15] A. Bulatov, P. Jeavons, and A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005. 1, 2, 11, 49

[16] A. A. Bulatov. Tractable conservative constraint satisfaction problems. In *LICS '03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, pages 321–330, Ottawa, Canada, 2003. IEEE Press. 2, 47

[17] A. A. Bulatov, P. Jeavons, and A. Krokhin. Constraint satisfaction problems and finite algebras. In *ICALP '00: Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 272–282, London, UK, 2000. Springer-Verlag. 2

[18] A. A. Bulatov, A. A. Krokhin, and B. Larose. Dualities for constraint satisfaction problems. In *Complexity of Constraints*, pages 93–124, 2008. 4, 20, 53

[19] A. A. Bulatov and M. Valeriote. Recent results on the algebraic approach to the csp. In *Complexity of Constraints*, pages 68–92, 2008. 4

[20] G. Buntrock, U. Hertrampf, C. Meinel, and C. Damm. Structure and importance of logspace-mod-classes. In *Proceedings of the 8th Annual Symposium on Theoretical Aspects of Computer Science*, pages 360–371, 1991. 3

[21] C. Carvalho, V. Dalmau, and A. Krokhin. Caterpillar duality for constraint satisfaction problems. In *LICS '08: Proceedings of the 2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 307–316, Washington, DC, USA, 2008. IEEE Computer Society. 4

[22] C. Carvalho, V. Dalmau, and A. A. Krokhin. Csp duality and trees of bounded pathwidth. *Theor. Comput. Sci.*, 411(34-36):3188–3208, 2010. 4

[23] C. Carvalho, L. Egri, M. Jackson, and T. Niven. On Maltsev digraphs. In *Proceedings of the 6th International Computer Science Symposium in Russia (CSR)*, pages 181–194, 2011. 7, 102

[24] D. Cohen and P. Jeavons. *The complexity of constraint languages*, chapter 8. Elsevier, 2006. 4, 49

[25] D. Cohen, P. Jeavons, P. Jonsson, and M. Koubarakis. Building tractable disjunctive constraints. *J. ACM*, 47(5):826–853, 2000. 1

[26] M. C. Cooper, D. A. Cohen, and P. G. Jeavons. Characterising tractable constraints. *Artificial Intelligence*, 65:347–361, 1994. 10

[27] V. Dalmau. Constraint satisfaction problems in non-deterministic logarithmic space. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP, pages 414–425. Springer-Verlag, 2002. 13, 19, 20, 36, 125, 135

[28] V. Dalmau. Linear Datalog and bounded path duality of relational structures. *Logical Methods in Computer Science*, 1(1), 2005. 3, 4, 5, 10, 94, 125

[29] V. Dalmau. Generalized majority-minority operations are tractable. *Logical Methods in Computer Science*, 2(4), 2006. 2

[30] V. Dalmau and A. Krokhin. Majority constraints have bounded pathwidth duality. *European Journal of Combinatorics*, 29(4):821–837, 2008. 13, 20, 35, 47, 52, 94, 124

[31] V. Dalmau and B. Larose. Maltsev + Datalog → symmetric Datalog. In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 297–306, 2008. 4, 6, 20, 28, 35, 124

[32] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artif. Intell.*, 34:1–38, 1987. 1

[33] M. E. Dyer and D. Richerby. On the complexity of #CSP. In *STOC*, pages 725–734, 2010. 114

[34] L. Egri. The complexity of constraint satisfaction problems and symmetric Datalog. Master's thesis, McGill University, 2007. 5

[35] L. Egri. On constraint satisfaction problems below P. In *CSL*, pages 203–217, 2011. 6, 7, 19, 125

[36] L. Egri, A. Krokhin, B. Larose, and P. Tesson. The complexity of the list homomorphism problem for graphs. In *STACS*, pages 335–346, 2010. 6, 46, 98

[37] L. Egri, A. Krokhin, B. Larose, and P. Tesson. The complexity of the list homomorphism problem for graphs. *Theory of Computing Systems (Special Issue of STACS 2010)*, 2011. In press. 6, 46, 98

[38] L. Egri, B. Larose, and P. Tesson. Symmetric Datalog and constraint satisfaction problems in logspace. In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 193–202, 2007. 3, 13, 19, 35, 53

[39] L. Egri, B. Larose, and P. Tesson. Directed st-connectivity is not expressible in symmetric Datalog. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming (ICALP), Part II*, pages 172–183, 2008. 5

[40] T. Feder. Classification of homomorphisms to oriented cycles and of k-partite satisfiability. *SIAM Journal on Discrete Mathematics*, 14(4):471–480, 2001. 20, 94

[41] T. Feder, P. Hell, and J. Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999. 46, 65, 66

[42] T. Feder, P. Hell, and J. Huang. Bi-arc graphs and the complexity of list homomorphisms. *J. Graph Theory*, 42(1):61–80, 2003. 46, 47, 48, 54, 64, 65, 102

[43] T. Feder and M. Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998. 2, 19, 46, 102

[44] P. Frankl and R. L. Graham. Old and new proofs of the Erdös-Ko-Rado Theorem. *Journal of Sichuan University Natural Science Edition*, 26, 1989. 131

[45] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA., 1979. 11

[46] E. Grädel. Capturing complexity classes by fragments of second-order logic. *Theoretical Computer Science*, 101(1):35–57, 1992. 125

[47] F. Gurski. Characterizations for co-graphs defined by restricted nlc-width or clique-width operations. *Discrete Mathematics*, 306(2):271–277, 2006. 56

[48] J. Hagemann and A. Mitschke. On n-permutable congruences. *Algebra Universalis*, 3:8–12, 1973. 103

[49] R. Häggkvist, P. Hell, D. J. Miller, and V. Neumann-Lara. On multiplicative graphs and the product conjecture. *Combinatorica*, 8:63–74, 1988. 43, 93

[50] P. Hell and J. Nesetřil. *Graphs and homomorphisms*. Oxford lecture series in mathematics and its applications. Oxford University Press, 2004. 46, 64

[51] P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48:92–110, 1990. 102

[52] P. Hell and A. Rafiey. The dichotomy of list homomorphisms for digraphs. In *SODA*, pages 1703–1713, 2011. 103

[53] P. Hell and X. Zhu. Homomorphisms to oriented paths. *Discrete Mathematics*, 132:107–114, 1994. 37, 40

[54] D. Hobby and R. McKenzie. *The Structure of Finite Algebras*, volume 76 of *Contemporary Mathematics*. American Mathematical Society, Providence, R.I., 1988. 4, 20, 49, 50, 51, 66, 137

[55] P. Idziak, P. Markovic, R. McKenzie, M. Valeriote, and R. Willard. Tractability and learnability arising from algebras with few subpowers. In *LICS '07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 213–224, 2007. 2

[56] N. Immerman. *Descriptive complexity*. Graduate Texts in Computer Science. Springer, 1999. 125

[57] P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997. 2

[58] A. Kazda. Maltsev digraphs have a majority polymorphism. *European Journal of Combinatorics*, 32(3):390–397, 2011. 6, 102, 103, 108, 109, 110, 111, 112, 115, 137

[59] L. M. Kirousis. Fast parallel constraint satisfaction. *Artificial Intelligence*, 64(1):147–160, 1993. 10

[60] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992. 1

[61] B. Larose, C. Loten, and C. Tardif. A characterisation of first-order constraint satisfaction problems. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages 201–210, 2006. 47, 52, 54, 89

[62] B. Larose and P. Tesson. Universal algebra and hardness results for constraint satisfaction problems. In *Proc. of ICALP*, pages 267–278, 2007. 47, 52, 53

[63] B. Larose and P. Tesson. Universal algebra and hardness results for constraint satisfaction problems. *Theoretical Computer Science*, 410(18):1629–1647, 2009. 4, 5, 19, 20, 92, 126, 130, 136

[64] B. Larose and L. Zádori. Bounded width problems and algebras. *Algebra Universalis*, 56(3-4):439–466, 2007. 20

[65] D. Lesaint, N. Azarmi, R. Laithwaite, and P. Walker. Engineering dynamic scheduler for work manager. *BT Technology Journal*, 16(3):16–29, 1998. 1

[66] L. Libkin. *Elements of finite model theory*. Springer, 2004. 13

[67] P. Lincoln and J. C. Mitchell. Algorithmic aspects of type inference with subtypes. In *POPL '92: Proceedings of the 19th ACM SIGPLAN-SIGACT symposium on principles of programming languages*, pages 293–304, New York, NY, USA, 1992. ACM Press. 1

[68] M. Maróti and R. McKenzie. Existence theorems for weakly symmetric operations. *Algebra Universalis*, 59(3-4):463–489, 2008. 66

[69] R. M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003. 54

[70] R. McKenzie, G. McNulty, and W. Taylor. *Algebras, Lattices and Varieties*, volume I. Wadsworth and Brooks, California, 1987. 49, 66

[71] P. Meseguer. Constraint satisfaction problem: An overview. *AICOM*, 2:3–16, 1989. 1

[72] J. C. Mitchell. Coercion and type inference. In *POPL '84: Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 175–185, New York, NY, USA, 1984. ACM Press. 1

[73] C. M. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, 1994. 9, 11

[74] A. F. Pixley. Distributivity and permutability of congruence relations in equational classes of algebras. *Proceedings of the American Mathematical Society (AMC)*, 14:105–109, 1963. 35

[75] V. Pratt and J. Tiuryn. Satisfiability of inequalities in a poset. *Fundam. Inf.*, 28(1-2):165–182, 1996. 1

[76] L. Purvis and J. P. Constraint tractability theory and its application to the product development process for a constraint-based scheduler. In *Proceedings 1st International Conference on the Practical Applications of Constraint Technologies and Logic Programming, PACLP'99*, pages 63–79, 1999. 1

[77] O. Reingold. Undirected st-connectivity in log-space. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 376–385, 2005. 19, 98, 100

[78] T. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th ACM Symposium on Theory of Computing (STOC)*, pages 216–226, 1978. 3, 9, 10

[79] M. Valeriote. A subalgebra intersection property for congruence-distributive varieties. *Canadian Journal of Mathematics*, 61(2):451–464, 2009. 50, 51, 65

[80] M. Wand and P. O'Keefe. On the complexity of type inference with coercion. In *FPCA '89: Proceedings of the fourth international conference on functional programming languages and computer architecture*, pages 293–298, New York, NY, USA, 1989. ACM Press. 1

[81] I. Wegener. *Branching programs and binary decision diagrams: theory and applications*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA, 2000. 126