Logical Methods in Computer Science
Vol. 14(2:3)2018, pp. 1–24
https://lmcs.episciences.org/

Submitted Sep. 14, 2016
Published Apr. 25, 2018

n-PERMUTABILITY AND LINEAR DATALOG IMPLIES SYMMETRIC DATALOG

ALEXANDR KAZDA

Department of Algebra Charles University Prague, Czech Republic
e-mail address: alex.kazda@gmail.com

Abstract. We show that if $\mathbb{A}$ is a core relational structure such that CSP($\mathbb{A}$) can be solved by a linear Datalog program, and $\mathbb{A}$ is n-permutable for some n, then CSP($\mathbb{A}$) can be solved by a symmetric Datalog program (and thus CSP($\mathbb{A}$) lies in deterministic logspace). At the moment, it is not known for which structures $\mathbb{A}$ will CSP($\mathbb{A}$) be solvable by a linear Datalog program. However, once somebody obtains a characterization of linear Datalog, our result immediately gives a characterization of symmetric Datalog.

1. Introduction

In the last decade, algebraic methods have led to much progress in classifying the complexity of the non-uniform Constraint Satisfaction Problem (CSP). The programming language Datalog, whose origins lie in logic programming and database theory, has been playing an important role in describing the complexity of CSP since at least the classic paper of T. Feder and M. Vardi [FV99], where Feder and Vardi used Datalog to define CSPs of bounded width. In an effort to describe the finer hierarchy of CSP complexity, V. Dalmau [Dal05] asked which CSPs can be solved using the weaker language of linear Datalog, and later L. Egri, B. Larose and P. Tesson [ELT07] introduced the even weaker symmetric Datalog.

We want to show that if CSP($\mathbb{A}$) can be solved by a linear Datalog program (alternatively, has bounded pathwidth duality) and $\mathbf{A}$ generates an n-permutable variety for some n, then CSP($\mathbb{A}$) can be solved by a *symmetric* Datalog program (and so lies in L). While this yields an "if and only if" description of symmetric Datalog, it is not a perfect characterization – describing the structures $\mathbb{A}$ such that CSP($\mathbb{A}$) is solvable by linear Datalog is an open problem. However, once CSPs for which linear Datalog works are classified, we will immediately get an equally good classification of symmetric Datalog CSPs.

In particular, should it turn out that admitting only the lattice and/or Boolean tame congruence types implies bounded pathwidth duality, we would have a neat characterization of problems solvable by symmetric Datalog: It would be the class of problems whose algebras omit all tame congruence theory types except for the Boolean type (we go into greater detail about tame congruence theory in Preliminaries and Conclusions).

Our result is similar to, but incomparable with what V. Dalmau and B. Larose have shown [DL08]: Their proof shows that 2-permutability plus being solvable by Datalog implies solvability by symmetric Datalog. We require both less (n-permutability for some n as

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.23638/LMCS-14(2:3)2018

opposed to 2-permutability) and more (linear Datalog solves CSP($\mathbb{A}$) as opposed to Datalog solves CSP($\mathbb{A}$)).

Our proof strategy is this: First we show in Section 3 how we can use symmetric Datalog to derive new instances from the given instance. Basically, we show that we can run a smaller symmetric Datalog program from inside another. This will later help us to reduce "bad" CSP instances to a form that is easy to deal with. Then, in Section 4 we introduce path CSP instances and show how n-permutability restricts the kind of path instances we can encounter. We use this knowledge in Section 6 to show that for any variety n-permutable $\mathbb{A}$, there is a symmetric Datalog program that decides path instances of CSP($\mathbb{A}$). Finally, in Section 7 we use linear Datalog to go from solving path instances to solving general CSP instances and finish our proof.

When writing this paper, we were mainly interested in ease of exposition, not in obtaining the fastest possible algorithm. We should therefore warn any readers hoping to implement our method in practice that the size of our symmetric Datalog program grows quite quickly with the size of $\mathbb{A}$ and the number of Hagemann-Mitschke terms involved. The main culprit is Lemma 4.5 that depends on Ramsey theory.

2. Preliminaries

All numbers in this paper are integers (most of them positive). If n is a positive integer and a, b are integers, we will use the notation $[n] = \{1, 2, \dots, n\}$ and the notation $[a, b] = \{i \in \mathbb{Z}\colon a \leq i \leq b\}$ (and variants such as $[a, b) = \{i \in \mathbb{Z}\colon a \leq i < b\}$).

We will be talking quite a bit about tuples – either tuples of elements of A or tuples of variables. We will treat both cases similarly: An n-tuple on Y is a mapping $\sigma\colon [n] \to Y$. We will denote the length of the tuple σ by $|\sigma|$, while $\operatorname{Im}\sigma$ will be the set of elements used in σ. Note that if e.g. $\sigma = (x, x, y)$, we can have $|\sigma| > |\operatorname{Im}\sigma|$.

A *relation* on A is any $R \subseteq A^X$ where X is some (finite) set. The *arity* of R is the cardinality of X. Most of the time, we will use $X = [n]$ for some $n \in \mathbb{N}$ and write simply $R \subseteq A^n$.

When $R \subseteq A^n$ is an n-ary relation and $\sigma = (a_1, \dots, a_n)$ is an n-tuple, we will often write $R(\sigma)$ instead of $(a_1, \dots, a_n) \in R$. Given a mapping $f : A \to B$ and an n-tuple $\sigma \in A^n$, we will denote by $f(\sigma)$ the n-tuple $(f(\sigma(1)), \dots, f(\sigma(n))) \in B^n$.

2.1. Algebras and relational structures.

Algebras and relational structures. We will be touching some concepts from universal algebra that would deserve a more detailed treatment than what we will provide here. See [Ber11] for an introduction to universal algebra.

A *relational structure* $\mathbb{A}$ consists of a set A together with a family $\mathcal{R}$ of relations on A, which we call *basic relations* of $\mathbb{A}$. In this paper, we will only consider finite relational structures with finitely many basic relations. We will not allow nullary relations or relations of infinite arity.

An *n-ary operation* on A is any mapping $t\colon A^n \to A$. We say that an n-ary operation t *preserves* the relation R if for all $r_1, \dots, r_n \in R$ we have $t(r_1, r_2, \dots, r_n) \in R$ (where $t(r_1, \dots, r_n)$ is the tuple we obtain by applying t componentwise to $r_1, \dots, r_n$). Given a relational structure $\mathbb{A}$, an n-ary operation t on A is a *polymorphism* of $\mathbb{A}$ if t preserves all basic relations of $\mathbb{A}$.

An *algebra* $\mathbf{A}$ consists of a base set A on which acts a set of *basic operations* of $\mathbf{A}$. We can compose basic operations of $\mathbf{A}$ to get more operations. A *term* in $\mathbf{A}$ is a correctly formed string that consists of variables and basic operation symbols of $\mathbf{A}$ (as well as parentheses and commas) and describes a meaningful composition of operations. For example, if $\mathbf{A}$ has the ternary basic operation r, then "$r(x_3, r(x_1, x_1, x_2), x_4)$" is a term in $\mathbf{A}$ that describes the 4-ary operation (with variables $x_1, \dots, x_4$) we can get by composing r with itself in a particular way. An algebra is *idempotent* if for any term operation t in $\mathbf{A}$ and any $a \in A$ we have $t(a, \dots, a) = a$.

The *algebra of polymorphisms of* $\mathbb{A}$ is the algebra with the universe A whose set of operations consists of all polymorphisms of $\mathbb{A}$. We will use the shorthand $\mathbf{A}$ for this algebra.

A *congruence* α on an algebra $\mathbf{A}$ is any binary equivalence relation that is preserved by all operations of $\mathbf{A}$.

The *relational clone* of $\mathbb{A}$ is the set of all relations on A that can be defined from the basic relations of $\mathbb{A}$ by primitive positive definitions – formulas that only use conjunction, existential quantification and symbols for variables. We will sometimes call members of the relational clone of $\mathbb{A}$ *admissible relations of* $\mathbb{A}$. The importance of the relational clone comes from the fact that $\mathbf{A}$ preserves precisely all relations on A that belong in the relational clone of $\mathbb{A}$ [BKKR69, Gei68].

A *variety* is a class of algebras sharing the same signature (the same basic operation symbols and arities of basic operations) that is closed under taking subalgebras, products and homomorphic images. If $\mathbf{A}$ is an algebra, then the variety generated by $\mathbf{A}$ is the smallest variety that contains $\mathbf{A}$, or equivalently the class of all homomorphic images of subalgebras of powers of $\mathbf{A}$.

Since all algebras in a given variety have the same signature, it makes sense to talk about term operations of a variety. We will be using one particular set of such operations, called Hagemann-Mitschke terms, in our proofs.

Let us fix a positive integer n. We say that a variety V is (congruence) n-permutable if for any algebra $\mathbf{A}$ in V and any pair of congruences α, β of $\mathbf{A}$ it is true that

$$\alpha \vee \beta = \alpha \circ \beta \circ \alpha \circ \dots$$

with $n-1$ composition symbols on the right side (in particular, 2-permutable is the same thing as congruence permutable).

A standard free algebra argument gives us that V is n-permutable if and only if we can find idempotent terms $p_0, p_1, \dots, p_n$ in V such that

$$\begin{aligned} x &= p_0(x, y, z), \\ p_i(x, x, y) &= p_{i+1}(x, y, y) \quad \text{for all } i = 1, 2, \dots, n-1, \\ p_n(x, y, z) &= z. \end{aligned}$$

The above terms are called Hagemann-Mitschke terms and were first obtained in [HM73].

If the algebra of polymorphisms of a relational structure $\mathbb{A}$ generates an n-permutable variety, i.e. if there are Hagemann-Mitschke operations $p_0, p_1, \dots, p_n$ in $\mathbf{A}$, we say simply that $\mathbb{A}$ is variety n-permutable (the "variety" prefix is here to emphasize that the whole variety of $\mathbf{A}$, not just $\mathbf{A}$ itself, needs to have n-permutable congruences). There are several other conditions that connect the behavior of congruences in a variety with the variety having certain term operations. We mention (without going into details) congruence distributivity and congruence semidistributivity in the next section.

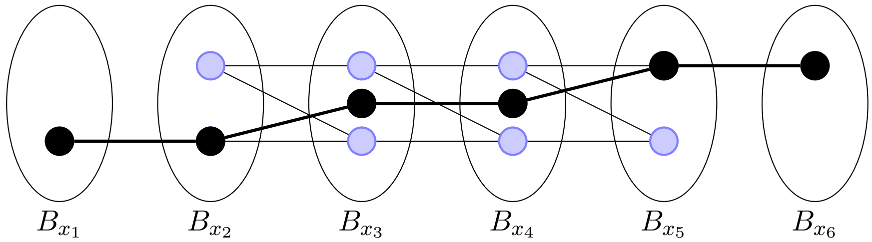

Figure 1: An example of microstructure with six variables $x_1, x_2, \dots, x_6$ and five binary relations (instance solution in bold).

2.2. **Constraint Satisfaction Problem.** Let us fix a relational structure $\mathbb{A} = (A, \mathcal{R})$ and define the non-uniform *Constraint Satisfaction Problem* with the right side $\mathbb{A}$, or CSP($\mathbb{A}$) for short. This problem can be stated in several mostly equivalent ways (in particular, many people prefer to think of CSP($\mathbb{A}$) as a question about homomorphisms between relational structures). We define CSP($\mathbb{A}$) in the language of logical formulas.

Definition 2.1. An instance $I = (V, \mathcal{C})$ of CSP($\mathbb{A}$) consists of a set of variables V and a set of constraints $\mathcal{C}$. Each constraint is a pair (σ, R) where $\sigma \in V^n$ is the *scope* of the constraint and $R \in \mathcal{R}$ is the *constraint relation*. A *solution* of I is a mapping $f\colon V \to A$ such that for all constraints $(\sigma, R) \in \mathcal{C}$ we have $f(\sigma) \in R$.

If I is an instance, we will say that I is *satisfiable* if there exists a solution of I and *unsatisfiable* otherwise. The Constraint Satisfaction Problem with target structure $\mathbb{A}$ has as its input an instance I of CSP($\mathbb{A}$) (encoded in a straightforward way as a list of constraints), and the output is the answer to the question "Is I satisfiable?"

If $I = (V, \mathcal{C})$ is an instance of CSP($\mathbb{A}$), then any CSP($\mathbb{A}$) instance $J = (U, \mathcal{D})$ with $U \subseteq V$ and $\mathcal{D} \subseteq \mathcal{C}$ is called a *subinstance* of I. It easy to see that if I has an unsatisfiable subinstance then I itself is unsatisfiable. If $U \subseteq V$, the subinstance of $I = (V, \mathcal{C})$ induced by U is the instance $I_{\restriction U} = (U, \mathcal{D})$ where $(\sigma, R) \in \mathcal{D}$ if and only if $\operatorname{Im} \sigma \subseteq U$.

We can draw CSP instances whose constraints' arities are at most two as *microstructures* (also known as potato diagrams among universal algebraists): For each variable x we draw the set $B_x \subseteq A$ equal to the intersection of all unary constraints on x. For each binary constraint we draw lines joining the pairs of elements in corresponding sets. A solution of the instance corresponds to the selection of one element b_x in each set B_x in such a way that whenever $\mathcal{C} = ((x, y), R)$ is a constraint, we have $(b_x, b_y) \in R$ (see Figure 1 for an example).

Obviously, CSP($\mathbb{A}$) is always in the class NP, since we can check in polynomial time whether a mapping $f\colon V \to A$ is a solution. Had we let the structure $\mathbb{A}$ be a part of the input, the constraint satisfaction problem would be NP-complete (it is easy to encode, say, 3-colorability of a graph as a CSP instance). However, when one fixes the structure $\mathbb{A}$, CSP($\mathbb{A}$) can become easier.

A relational structure $\mathbb{A}$ is a *core* if any unary polymorphism $f\colon \mathbb{A} \to \mathbb{A}$ is an automorphism (i.e. we cannot retract $\mathbb{A}$ to a smaller relational structure). To classify the complexity of CSP($\mathbb{A}$) for $\mathbb{A}$ finite, it is enough to classify cores, see [HN04, p. 142].

The value of the algebraic approach to CSP is that it connects the complexity of CSP($\mathbb{A}$) to the algebra of polymorphisms $\mathbf{A}$. Let us highlight one such important connection here: The tame congruence theory is a tool that arose from the study of finite algebras in the 1980s. In a nutshell (see [HM96] for more), the theory aims to connect properties of congruences in a variety with the existence of certain term operations in that variety and with the set of so-called *types* that the variety admits (where "admits" means that a certain technical construction can, when run on a suitable algebra from the variety, produce a given type). It turns out [LT09] that we can get lower bounds on the complexity of $\mathrm{CSP}(\mathbb{A})$ from the set of types that the variety generated by $\mathbf{A}$ admits. Let us rephrase four out of the five parts of Theorem 4.1 from [LT09] to include concrete problems that each type brings about:

Theorem 2.2 (B. Larose, P. Tesson; part 1 is due to A. Bulatov, P. Jeavons, and A. Krokhin [BKJ00])**.** *Let $\mathbb{A}$ be a core relational structure (with finitely many basic relations) and let V be the variety generated by $\mathbf{A}$. Then:*

(1) *If V admits the unary type (type 1), then 3-SAT reduces to* $\mathrm{CSP}(\mathbb{A})$ *and hence* $\mathrm{CSP}(\mathbb{A})$ *is* ***NP****-hard,*
(2) *if V admits the semilattice type (type 5), then HORN-3-SAT reduces to* $\mathrm{CSP}(\mathbb{A})$ *and hence* $\mathrm{CSP}(\mathbb{A})$ *is* ***P****-hard,*
(3) *if V admits the affine type (type 2), then there is a prime p such that the complement of the p-GAP problem (see* [LT09] *for definition) reduces to* $\mathrm{CSP}(\mathbb{A})$ *and hence* $\mathrm{CSP}(\mathbb{A})$ *is* $\mathbf{Mod}_p\mathbf{L}$*-hard for some prime p,*
(4) *if V admits the lattice type (type 4), then the directed unreachability problem (the complement of directed reachability) reduces to* $\mathrm{CSP}(\mathbb{A})$ *and hence* $\mathrm{CSP}(\mathbb{A})$ *is* ***NL****-hard.*

All the reductions above are first order reductions (see [Imm99] *for definition) and are very natural from the point of view of universal algebra.*

Notice that one type is missing from the above theorem: The Boolean type (type 3) only gives us very weak lower bounds. When $\mathbf{A}$ generates a variety that admits the Boolean type only, the problem $\mathrm{CSP}(\mathbb{A})$ is believed to be in L (see Conjecture 8.2).

2.3. **Datalog.** The Datalog language offers a way to check the local consistency of CSP instances. A Datalog program P for solving $\mathrm{CSP}(\mathbb{A})$ consists of a list of rules of the form

$$R(\rho) \leftarrow S_1(\sigma_1),\, S_2(\sigma_2),\, \ldots,\, S_\ell(\sigma_\ell),$$

where $R, S_1, \ldots, S_l$ are predicates and $\rho, \sigma_1, \ldots, \sigma_\ell$ are sequences of variables (we will denote the set of all variables used in the program by X). Some predicates of P are designated as *goal predicates* (more on those later).

In general, the predicates can be symbols without any meaning, but in the programs we are about to construct each predicate will correspond to a relation on A, i.e. a predicate $S(x_1, x_2)$ would correspond to some $S \subseteq A^2$. This will often get us in a situation where, say, the symbol R stands at the same time for a relation on A, a predicate of a Datalog program, and a relation on the set V of variables (see below). For the most part, we will depend on context to tell these meanings of R apart, but if there is a risk of confusion we will employ the notation R^A for $R^A \subseteq A^n$, R^P for predicates of P, and R^V for $R^V \subseteq V^n$.

Given a Datalog program P that contains predicates for all basic relations of a relational structure $\mathbb{A}$, we can run P on an instance $I = (V, \mathcal{C})$ of $\mathrm{CSP}(\mathbb{A})$ as follows: For each n-ary predicate R^P of P, we keep in memory an n-ary relation $R^V \subseteq V^n$. Initially, all such relations are empty. To load I into the program, we go through $\mathcal{C}$ and for every $(\sigma, R^A) \in \mathcal{C}$, we add σ to R^V (when designing P, we will always make sure that there is a predicate R^P for each basic relation R^A of $\mathbb{A}$).

After this initialization, P keeps adding tuples of V into relations R^V as per the rules of P: If we can assign values to variables so that the right hand side of some rule holds, then we put the corresponding tuple into the left hand side relation R.

More formally, we say that $P(I)$ derives $R^V(\rho)$ for $\rho \in V^n$, writing $P(I) \vdash R^V(\rho)$, if one of the following happens: We have $(\rho, R^A) \in \mathcal{C}$, or P contains a rule of the form

$$R^P(\tau) \leftarrow S_1^P(\sigma_1),\, S_2^P(\sigma_2),\, \dots,\, S_\ell^P(\sigma_\ell),$$

where $\tau, \sigma_1, \dots, \sigma_\ell$ are tuples of variables from the set of variables X, and there exists a mapping (evaluation) $\omega\colon X \to V$ such that $\omega(\tau) = \rho$ and for each $i = 1, \dots, \ell$ we have $P(I) \vdash S_i^V(\omega(\sigma_i))$.

If a Datalog program P ever uses a rule with a goal predicate on its left side, then the program outputs "Yes," and halts. We will use the symbol G to stand for any of the goal predicates, writing for example $P(I) \vdash G$ as a shorthand for "P run on I derives a relation that is designated as a goal predicate." Another way to implement goal predicates, used e.g. in [FV99], is to introduce a special nullary relation G that is the goal. We do not want to deal with nullary relations, but the distinction is purely a formal one: should the reader want a program with a nullary G, all that is needed is to simply introduce rules of the form $G \leftarrow R(x_1, x_2, \dots)$ where R ranges over the list of goal predicates.

If a goal predicate is not reached, the program $P(I)$ runs until it can not derive any new statements, at which point it outputs "No," and halts. Thanks to the monotonous character of Datalog rules (we only add tuples to predicates, never remove them), any given Datalog program can be evaluated in time polynomial in the size of its input instance I.

Given a Datalog program P and a relational structure $\mathbb{A}$ we say that P *decides* CSP($\mathbb{A}$) if P run on a CSP($\mathbb{A}$) instance I reaches a goal predicate if and only if I is unsatisfiable. We say that CSP($\mathbb{A}$) can be solved by Datalog if there is a Datalog program P that decides CSP($\mathbb{A}$). (Strictly speaking, we should say that P decides $\neg\,$CSP($\mathbb{A}$) in this situation, but that is cumbersome.)

For $R_1, \dots, R_k$ relations and $\sigma_1, \dots, \sigma_k$ tuples of variables, we define the *conjunction* $R_1(\sigma_1) \wedge \dots \wedge R_k(\sigma_k)$ as a relation (resp. predicate) on $\bigcup_i \operatorname{Im} \sigma_i$. For example, $R_1(x_3, x_2, x_2) \wedge R_2(x_3, x_4)$ is a relation of arity 3 on the three variables x_2, x_3, x_4.

To slim down our notation, we will for the most part not distinguish the abstract statement of a Datalog rule (with variables from X) and the concrete realization of the rule (with the evaluation $\omega\colon X \to V$). For example, if P contained this rule α:

$$R(x, z) \leftarrow S(x, y),\, T(y, z)$$

and it happened that $P(I) \vdash S(1, 2)$ and $P(I) \vdash T(2, 2)$, then instead of saying that we are applying the rule α with the evaluation $\omega(x) = 1$, $\omega(y) = \omega(z) = 2$ to add $(1, 2)$ into R, we would simply state that we are using the rule

$$R(1, 2) \leftarrow S(1, 2),\, T(2, 2),$$

even though that means silently identifying y and z in the original rule α.

The power of Datalog for CSP is exactly the same as that of local consistency methods. L. Barto and M. Kozik have given several different natural characterizations of structures $\mathbb{A}$ such that Datalog solves CSP($\mathbb{A}$) [BK14]. However, this is not the end of the story, for there are natural fragments of Datalog which have lower expressive power, but also lower computational complexity.

Predicates that can appear on the left hand side of some rule (and therefore can have new tuples added into them) are called *intensional database symbols* (IDB). Having IDBs on the right hand side of rules enables recursion. Therefore, limiting the occasions when IDBs appear on the right hand side of rules results in fragments of Datalog that can be evaluated faster.

An extreme case of such restriction happens when there is never an IDB on the right hand side of any rule. It is easy to see that such Datalog programs can solve CSP($\mathbb{A}$) if and only if $\mathbb{A}$ has a finite (also called "finitary") duality, i.e. there exists a finite list $\mathcal{Q}$ of unsatisfiable CSP($\mathbb{A}$) instances such that an instance I of CSP($\mathbb{A}$) is unsatisfiable if and only if there exists $J \in \mathcal{Q}$ such that one can rename (and possibly also glue together) variables of J to get a subinstance of I. This property is equivalent to CSP($\mathbb{A}$) being definable in first order logic by [Ats08] (see also the survey [BKL08]). Structures of finite duality are both well understood and rare, so let us look at more permissive restrictions.

A Datalog program is *linear* if there is at most one IDB on the right hand side of any rule. When evaluating Linear Datalog programs, we need to only consider chains of rules that do not branch: It is straightforward to show by induction that if P is a linear Datalog program and I is an instance of the corresponding CSP, then $P(I) \vdash R(\rho)$ if and only if (ρ, R) is a constraint of I or there is a sequence of statements

$$U_1(\varphi_1), U_2(\varphi_2), \ldots, U_m(\varphi_m) = R(\rho)$$

such that for each $i = 2, \ldots, m$ the program P has a rule of the form

$$U_i(\varphi_i) \leftarrow U_{i-1}(\varphi_{i-1}),\, T_1^i(\tau_1^i),\, \ldots,\, T_{\ell_i}^i(\tau_{\ell_i}^i),$$

where U_{i-1} is the IDB in the rule and (τ_j^i, T_j^i) are constraints of I for all $j = 1, \ldots, \ell_i$. The first statement, $U_1(\sigma_1)$, is a special case as P must derive it without using IDBs, i.e. there is a rule of P of the form

$$U_1(\varphi_1) \leftarrow T_1^1(\tau_1^1),\, \ldots,\, T_{\ell_1}^1(\tau_{\ell_1}^1)$$

where all $(\tau_1^1, T_1^1), \ldots, (\tau_{\ell_1}^1, T_{\ell_1}^1)$ are constraints of I.

(Note that this is the first time we are using the "concrete realization of the abstract rule" shorthand.) We will call such a sequence $U_1(\varphi_1), \ldots, U_m(\varphi_m)$ a *derivation* of $R(\rho)$.

Another way to view the computation of a linear Datalog program is to use the digraph $\mathcal{G}(P, I)$: The set of vertices of $\mathcal{G}(P, I)$ will consist of all pairs (ρ, R) where R is an n-ary IDB predicate of P and $\rho \in V^n$. The graph $\mathcal{G}(P, I)$ contains the edge from (ρ, R) to (σ, S) if P contains a rule of the form

$$R(\rho) \leftarrow S(\sigma),\, T_1(\tau_1),\, \ldots,\, T_k(\tau_k),$$

where all (τ_i, T_i) are constraints of I.

It is easy to see that $P(I) \vdash G$ if and only if there is a tuple ρ and an IDB R such that $P(I) \vdash R(\rho)$ in one step, without the use of intermediate IDBs, and there is a directed path from (ρ, R) to a goal predicate in $\mathcal{G}(P, I)$. It is straightforward to verify that deciding the existence of such a path is in NL. (In fact, deciding directed connectivity is NL-complete [AB09, Theorem 4.18, p. 89] and since there is a linear Datalog program that decides directed connectivity, it follows that evaluating linear Datalog programs is NL-complete.)

The exact characterization of structures $\mathbb{A}$ such that there is a linear Datalog program deciding CSP($\mathbb{A}$) is open. A popular conjecture is that CSP($\mathbb{A}$) can be solved by linear Datalog if and only if the variety of $\mathbf{A}$ admits no tame congruence types except for the lattice and Boolean type (or equivalently [HM96, Theorem 9.11] that $\mathbf{A}$ generates a congruence semidistributive variety).

As Larose and Tesson have shown [LT09, Theorem 4.2], admitting no types other than lattice and Boolean is necessary for core relational structures to yield CSPs solvable by linear Datalog. On the other hand Barto, Kozik and Willard proved that if $\mathbb{A}$ admits an NU polymorphism then $\mathrm{CSP}(\mathbb{A})$ can be solved by linear Datalog [BKW12]. This is almost, but not quite, what the necessary condition demands: A finite relational structure $\mathbb{A}$ with finitely many relations has an NU polymorphism if and only if the variety generated by $\mathbf{A}$ is congruence distributive [Bar13] if and only if the variety only admits the lattice and/or Boolean types in a particularly nice way [HM96, Theorem 8.6].

For our purposes, it will be useful to notice that $\mathrm{CSP}(\mathbb{A})$ can be solved by a linear Datalog program if and only if $\mathbb{A}$ has bounded pathwidth duality.

Definition 2.3. $\mathrm{CSP}(\mathbb{A})$ instance $I = (V, \mathcal{C})$ has *pathwidth* at most k if we can cover V by a family of sets $U_1, \dots, U_m$ such that

- $|U_i| \leq k+1$ for each i,
- if $i < j$ and $v \in V$ lies in U_i and U_j, then v also lies in each of $U_{i+1}, \dots, U_{j-1}$, and
- for each constraint $C \in \mathcal{C}$ there is an i such that the image of the scope of C lies entirely in U_i.

The name pathwidth comes from the fact that if we arrange the variables in the order they appear in $U_1, \dots, U_m$ and look at the instance from far away, the "bubbles" $U_1, \dots, U_m$ form a path. The length of the path is allowed to be arbitrary, but the "width" (size of the bubbles and their overlaps) is bounded.

We say that $\mathbb{A}$ has bounded pathwidth duality if there exists a constant k such that for every unsatisfiable instance I of $\mathrm{CSP}(\mathbb{A})$ there exists an unsatisfiable instance J of $\mathrm{CSP}(\mathbb{A})$ of pathwidth at most k such that we can identify some variables of J to obtain a subinstance of I. (This is a translation of the usual definition of duality, which talks about homomorphisms of relational structures, to CSP instances.)

Proposition 2.4 ([Dal05])**.** *Assume that $\mathbb{A}$ is a relational structure. Then $\mathbb{A}$ has bounded pathwidth duality if and only if there exists a linear Datalog program deciding* $\mathrm{CSP}(\mathbb{A})$.

Symmetric Datalog is a more restricted version of linear Datalog, where we only allow *symmetric* linear rules: Any rule with no IDBs on the right hand side is automatically symmetric, so the interesting case is when a rule α has the form

$$R(\rho) \leftarrow S(\sigma) \wedge T_1(\tau_1) \wedge T_2(\tau_2) \wedge \dots,$$

where R, S are (the only) IDBs. If a symmetric program P contains the rule α, then P must also contain the rule α' obtained from α by switching $R(\rho)$ and $S(\sigma)$ (we will call this rule the mirror image of α):

$$S(\sigma) \leftarrow R(\rho) \wedge T_1(\tau_1) \wedge T_2(\tau_2) \wedge \dots$$

Observe that if P is a symmetric Datalog program, then $\mathcal{G}(P, I)$ is always a symmetric graph. Therefore, deciding if $P(I) \vdash G$ is equivalent to an undirected reachability problem. Evaluating symmetric Datalog programs is thus in L thanks to Reingold's celebrated result that undirected reachability is in L [Rei05]. (In fact, undirected reachability is L-complete under first order reductions, as is evaluating symmetric Datalog programs: Consider the symmetric Turing machines introduced in [LP82]. When equipped with logarithmic amount of memory, these machines define the complexity class SL. When one applies the construction in the proof of Theorem 3.16 in [Imm99] to symmetric logspace machines, one gets that undirected reachability is SL-hard modulo first order reductions. Since $\mathsf{L} \subseteq \mathsf{SL}$ [LP82, Theorem 1], undirected reachability is L-hard.)

We will often use Datalog programs whose predicates correspond to relations on A. However, in doing so we will *not* restrict ourselves to just the relations from the relational clone of $\mathbb{A}$. If the predicates $R^P, S_1^P, \dots, S_\ell^P$ correspond to relations $R^A, S_1^A, \dots, S_\ell^A$ on A in some agreed upon way, then we say that the rule

$$R^P(\rho) \leftarrow S_1^P(\sigma_1),\, S_2^P(\sigma_2),\, \dots,\, S_\ell^P(\sigma_\ell),$$

is *consistent* with A if the corresponding implication holds for all tuples of A, i.e. the sentence

$$\forall f : X \to A,\ R^A(f(\rho)) \Leftarrow \left(S_1^A(f(\sigma_1)) \wedge S_2^A(f(\sigma_2)) \wedge \dots \wedge S_\ell^A(f(\sigma_\ell))\right),$$

holds in A (recall that X is the list of all variables used in the rules of P). In other words, a consistent rule records an implication that is true in A.

For $r \in \mathbb{N}$, we construct the *r-ary maximal symmetric Datalog program consistent with* $\mathbb{A}$, denoted by $\mathcal{P}_{\mathbb{A}}^r$, as follows: The program has as predicates all relations of arity at most r on A (these will be IDBs), plus a new symbol for each basic relation of $\mathbb{A}$ of arity at most r (these symbols will correspond to the relations used in constraints and they will never be IDBs; thus we have two symbols for each basic relation of $\mathbb{A}$, only one of which can be on the left hand side of any rule).

The set of rules of $\mathcal{P}_{\mathbb{A}}^r$ will contain all rules α that

(1) are valid linear Datalog rules (i.e. an IDB on the left side, at most one IDB on the right),
(2) use only tuples of variables from $X = \{x_1, \dots, x_r\}$ (i.e. at most r variables at once),
(3) do not have any repetition on the right hand side, i.e. each statement $R(\sigma)$ appears in α at most once (however, the predicate R can be used several times with different tuples of variables),
(4) are consistent with $\mathbb{A}$, and
(5) if α contains an IDB on the right hand side, then the mirror image α' of α is also consistent with $\mathbb{A}$.

We will designate all empty relations of arity at most r as goal predicates. We note that our $\mathcal{P}_{\mathbb{A}}^r$ is a variation of the notion of a canonical symmetric Datalog program (used e.g. in [DL08]).

It is an easy exercise to show that $\mathcal{P}_{\mathbb{A}}^r(I) \vdash S(\sigma)$ if and only if $\mathcal{G}(\mathcal{P}_{\mathbb{A}}^r, I)$ contains a path from (ρ, A) to (σ, S) where A is the unary full relation on A and ρ is arbitrary. Starting with the full relation will help us simplify proofs by induction later.

The set of rules of $\mathcal{P}_{\mathbb{A}}^r$ is large but finite because there are only so many ways to choose a sequence of at most r-ary predicates on r variables without repetition. Since $\mathbb{A}$ and r are not part of the input of CSP($\mathbb{A}$), we do not mind that $\mathcal{P}_{\mathbb{A}}^r$ contains numerous redundant or useless rules.

When we run $\mathcal{P}_{\mathbb{A}}^r$ on a CSP($\mathbb{A}$) instance I, it attempts to narrow down the set of images of r-tuples of variables using consistency:

Observation 2.5. Let $\mathbb{A}$ be a relational structure, $r \in \mathbb{N}$, and $I = (V, \mathcal{C})$ an instance of CSP($\mathbb{A}$). Then:

(1) if $R^A \subseteq A^n$, $\rho \in V^n$ are such that $\mathcal{P}_{\mathbb{A}}^r(I) \vdash R^V(\rho)$, then any solution f of I must satisfy $f(\rho) \in R^A$.

(2) if $\mathcal{P}^r_{\mathbb{A}}(I) \vdash G$, then I is not satisfiable.

Proof. To prove the first claim, consider a path in $\mathcal{G}(P, I)$ that witnesses $P(I) \vdash R(\rho)$:

$$(\rho_1, S_1), (\rho_2, S_2), \ldots, (\rho_m, S_m) = (\rho, R).$$

with $S_1^A = A$.

We claim that if f is a solution of I, then for each $i = 1, \ldots, m$ we must have $f(\rho_i) \in S_i$. We proceed by induction. For $i = 1$, this is trivial.

Assume now that $f(\rho_i) \in S_i$ and that $\mathcal{P}^r_{\mathbb{A}}$ contains a rule α of the form

$$S_{i+1}(\rho_{i+1}) \leftarrow S_i(\rho_i),\, T_1(\tau_1),\, \ldots,\, T_k(\tau_k),$$

where $(\tau_j, T_j) \in \mathcal{C}$ for $j = 1, \ldots, k$. Since $T_j(\tau_j)$ are constraints of I, we have $f(\tau_j) \in T_j$ for each j. From the fact that α is a rule consistent with $\mathbb{A}$, it follows that $f(\rho_{i+1}) \in S_{i+1}$.

The second statement of the Lemma is a consequence of the first, since reaching a goal predicate means that $\mathcal{P}^r_{\mathbb{A}}(I) \vdash \emptyset(\rho)$ for some ρ tuple of variables in V. Using (1), we get that each solution of I must satisfy the impossible condition $f(\rho) \in \emptyset$ and so there cannot be any solution f. $\square$

By Observation 2.5, the only way $\mathcal{P}^r_{\mathbb{A}}$ can fail to decide CSP($\mathbb{A}$) is if there is an unsatisfiable instance I of CSP($\mathbb{A}$) for which $\mathcal{P}^r_{\mathbb{A}}$ does not derive G. Our goal in the rest of the paper is to show that for r large enough and $\mathbb{A}$ nice enough such a situation will not happen.

Let us close this section by talking about necessary conditions for CSP($\mathbb{A}$) to be solvable by symmetric Datalog. An obvious condition is that, since symmetric Datalog is a subset of linear Datalog, CSP($\mathbb{A}$) must be solvable by linear Datalog.

It turns out that the lower bounds from the tame congruence theory are compatible with Datalog. If $\mathbb{A}$ is a core, then for CSP($\mathbb{A}$) to be solvable by symmetric Datalog, $\mathbf{A}$ must omit all tame congruence theory types except for the Boolean type [LT09, Theorem 4.2], from which it follows [HM96, Theorem 9.14] that $\mathbb{A}$ must be variety n-permutable for some n.

Proposition 2.6. *If $\mathbb{A}$ is a core relational structure such that* CSP($\mathbb{A}$) *is solvable by symmetric Datalog, then $\mathbb{A}$ is variety n-permutable for some n and* CSP($\mathbb{A}$) *is solvable by linear Datalog.*

Our goal in this paper is to prove that the conditions of Proposition 2.6 are also sufficient:

Theorem 2.7. *Let $\mathbb{A}$ be a relational structure such that there is a linear Datalog program that decides* CSP($\mathbb{A}$) *and $\mathbb{A}$ admits a chain of n Hagemann-Mitschke terms as polymorphisms. Then there exists an $r \in \mathbb{N}$ such that $\mathcal{P}^r_{\mathbb{A}}$ decides* CSP($\mathbb{A}$).

3. Stacking symmetric Datalog programs

In this section we describe two tricks that allow us essentially to run one Datalog program from inside another. The price we pay for this is that the new program can use fewer variables than the old one.

The first lemma of this section is basically [DL08, Lemma 11] rewritten in our formalism:

Lemma 3.1 (V. Dalmau, B. Larose)**.** *Let $\mathbb{A}$ be a relational structure, $I = (V, \mathcal{C})$ an instance of* CSP($\mathbb{A}$)*, let $S \subseteq A^s$, $R \subseteq A^r$ be two relations, and let $\sigma \in V^s$ and $\rho \in V^r$. Assume that $\mathcal{P}^s_{\mathbb{A}}(I) \vdash S(\sigma)$.*

Then for any $k \geq r + s$ we have

$$\mathcal{P}^k_{\mathbb{A}}(I) \vdash R(\rho) \Leftrightarrow \mathcal{P}^k_{\mathbb{A}}(I) \vdash R(\rho) \wedge S(\sigma).$$

Proof. Let

$$U_1(\varphi_1), U_2(\varphi_2), \dots, U_m(\varphi_m) = S(\sigma)$$

be a path in $\mathcal{G}(I, \mathcal{P}^s_{\mathbb{A}})$ witnessing $\mathcal{P}^s_{\mathbb{A}}(I) \vdash S(\sigma)$. Then it is easy to verify that

$$R(\rho), R(\rho) \wedge U_1(\varphi_1), R(\rho) \wedge U_2(\varphi_2), \dots, R(\rho) \wedge U_m(\varphi_m) = R(\rho) \wedge S(\sigma)$$

is a path in the graph $\mathcal{G}(I, \mathcal{P}^k_{\mathbb{A}})$. Therefore, $\mathcal{P}^k_{\mathbb{A}}(I)$ derives $R(\rho)$ if an only if it derives $R(\rho) \wedge S(\sigma)$. ☐

Repeated use of Lemma 3.1 gets us the following:

Corollary 3.2. *Let $\mathbb{A}$ be a relational structure, I a* CSP($\mathbb{A}$) *instance. Let $S_1, \dots, S_j$ and R be relations on A and $\sigma_1, \dots, \sigma_p, \rho$ be tuples of variables from I.*

If $\mathcal{P}^s_{\mathbb{A}}(I) \vdash S_j(\sigma_j)$ for $j = 1, \dots, p$ and both $|\rho|$ and $|\operatorname{Im} \rho \cup \bigcup_{i=1}^{p} \operatorname{Im} \sigma_i|$ are at most r, then we have:

$$\begin{aligned}
\mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash R(\rho) &\Leftrightarrow \mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash R(\rho) \wedge S_1(\sigma_1) \\
&\Leftrightarrow \mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash R(\rho) \wedge S_1(\sigma_1) \wedge S_2(\sigma_2) \\
&\ \ \vdots \\
&\Leftrightarrow \mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash R(\rho) \wedge S_1(\sigma_1) \wedge \dots \wedge S_p(\sigma_p)
\end{aligned}$$

Definition 3.3. Given an instance $I = (V, \mathcal{C})$ of CSP($\mathbb{A}$), we say that $\mathcal{P}^r_{\mathbb{A}}$ derives the instance $J = (W, \mathcal{D})$ from I, writing $\mathcal{P}^r_{\mathbb{A}}(I) \vdash J$, if $W \subseteq V$ and for each $(\sigma, R) \in \mathcal{D}$ we have $\mathcal{P}^r_{\mathbb{A}}(I) \vdash R(\sigma)$.

Obviously, if $\mathcal{P}^r_{\mathbb{A}}$ derives an unsatisfiable instance from I, then I itself is unsatisfiable. Moreover, a maximal symmetric Datalog program run on I can simulate the run of a smaller maximal symmetric Datalog program on J:

Lemma 3.4. *Let $\mathbb{A} = (A, R_1, \dots, R_n)$ and $\mathbb{B} = (A, S_1, \dots, S_m)$ be two relational structures and let $I = (V, \mathcal{C})$ be an instance of* CSP($\mathbb{A}$)*. Assume that r, s are positive integers and $J = (W, \mathcal{D})$ is an instance of* CSP($\mathbb{B}$) *such that $\mathcal{P}^s_{\mathbb{A}}(I) \vdash J$ and $\mathcal{P}^r_{\mathbb{B}}(J) \vdash G$. Then $\mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash G$.*

Proof. The derivation of $\mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash G$ will follow the derivation $\mathcal{P}^r_{\mathbb{B}}(J) \vdash G$, generating the constraints of J on the fly using $\mathcal{P}^s_{\mathbb{A}}(I)$. Note that since $\mathbb{A}$ and $\mathbb{B}$ share the same base set, the predicates of $\mathcal{P}^r_{\mathbb{B}}$ are also predicates of $\mathcal{P}^{r+s}_{\mathbb{A}}$.

Let $U_1(\varphi_1), U_2(\varphi_2), \dots, U_q(\varphi_q)$ be a derivation of G by $\mathcal{P}^r_{\mathbb{B}}(J)$ such that $U_1 = A$.

We proceed by induction on i from 1 to q and show that $\mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash U_i(\rho_i)$ for all i. Since all goal predicates of $\mathcal{P}^r_{\mathbb{A}}$ are also goal predicates of $\mathcal{P}^{r+s}_{\mathbb{A}}$, this will show that $\mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash G$. The base case is easy: Since U_1 is full, $\mathcal{P}^{r+s}_{\mathbb{A}}$ has the rule "$U_1(\varphi_1) \leftarrow$ ", giving us $\mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash U_1(\varphi_1)$.

Assume that $\mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash U_i(\varphi_i)$. Since $\mathcal{P}^r_{\mathbb{B}}(J)$ derives $U_{i+1}(\varphi_{i+1})$ from $U_i(\varphi_i)$, there have to be numbers $j_1, \dots, j_p$ and tuples $\sigma_1, \dots, \sigma_p$ such that each (σ_k, S_{j_k}) is a constraint of J, and

$$U_{i+1}(\varphi_{i+1}) \leftarrow U_i(\varphi_i),\ S_{j_1}(\sigma_1),\ S_{j_2}(\sigma_2),\ \dots,\ S_{j_p}(\sigma_p)$$

is a rule of $\mathcal{P}^r_{\mathbb{B}}$. From this, it is easy to verify that the following rule, which we will call α, is a rule of $\mathcal{P}^{r+s}_{\mathbb{A}}$:

$$(U_{i+1}(\varphi_{i+1}) \wedge S_{j_1}(\sigma_1) \wedge \cdots \wedge S_{j_p}(\sigma_p)) \leftarrow \\ (U_i(\varphi_i) \wedge S_{j_1}(\sigma_1) \wedge \cdots \wedge S_{j_p}(\sigma_p)),$$

Since $\mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash U_i(\varphi_i)$, Corollary 3.2 yields $\mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash U_i(\varphi_i) \wedge \bigwedge_{k=1}^p S_{j_k}(\sigma_k)$. We then use the rule α to obtain $\mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash U_{i+1}(\varphi_{i+1}) \wedge \bigwedge_{k=1}^p S_{j_k}(\sigma_k)$ and finally use the other implication from Corollary 3.2 to get $\mathcal{P}^{r+s}_{\mathbb{A}}(I) \vdash U_{i+1}(\varphi_{i+1})$, concluding the proof. □

At one point, we will need to look at powers of $\mathbb{A}$. For this, we introduce the following notation: If

$$\sigma = ((s_{1,1}, \ldots, s_{k,1}), \ldots, (s_{\ell,1}, \ldots, s_{\ell,k})) \in (A^k)^\ell$$

is an ℓ-tuple of elements of A^k then by $\overline{\sigma}$ we will mean the $k\ell$-tuple we get by "unpacking" σ into $A^{k\ell}$:

$$\overline{\sigma} = (s_{1,1}, \ldots, s_{k,1}, \ldots, s_{\ell,1}, \ldots, s_{\ell,k}).$$

If $U \subseteq (A^k)^\ell$ is a relation on A^k, we will denote by $\overline{U}$ the relation $\overline{U} = \{\overline{\sigma} \colon \sigma \in U\} \subseteq A^{k\ell}$.

The following lemma generalizes Lemma 3.4 to powers of A. The proof is similar to that of Lemma 3.4 and we omit it for brevity.

Lemma 3.5. *Let $k \in \mathbb{N}$ and assume we have relational structures $\mathbb{A}$ and $\mathbb{B}$ on the sets A and A^k respectively. Assume moreover that $I = (V, \mathcal{C})$ is an instance of* CSP($\mathbb{A}$)*, $S_1, \ldots, S_m$ are basic relations of $\mathbb{B}$, $\sigma_1, \ldots, \sigma_m$ are tuples of elements of V^k, and r, s are positive integers such that:*
(1) *$\mathcal{P}^r_{\mathbb{A}}(I) \vdash \overline{S}_i(\overline{\sigma_i})$ for each $i = 1, \ldots, m$,*
(2) *$\mathcal{P}^s_{\mathbb{B}}(J) \vdash G$, where J is the instance $J = (V^k, \{(\sigma_i, S_i) \mid i = 1, \ldots, m\})$ of* CSP($\mathbb{B}$).
Then $\mathcal{P}^{r+ks}_{\mathbb{A}}(I) \vdash G$.

4. Variety n-permutability on path instances

We begin our construction by showing how variety n-permutability limits the kind of CSP instances a symmetric Datalog program can encounter.

Definition 4.1. An instance $I = (V, \mathcal{C})$ of CSP is a *path instance* of length ℓ if:
(1) V is a linearly ordered set (we use $V = [\ell]$ ordered by size whenever practicable, such as in the rest of this definition),
(2) for each $i \in V$, I contains exactly one unary constraint with scope i; we will denote its constraint relation by $B_i \subseteq A$,
(3) for each $i = 1, 2, \ldots, \ell - 1$, I contains exactly one binary constraint with scope $(i, i+1)$; we denote its constraint relation $B_{i,i+1}$.
(4) I contains no other constraints than the ones named above.

Note that $B_{i,i+1}$ can contain tuples from outside of $B_i \times B_{i+1}$. We allow that to happen to simplify our later arguments.

If I is a path instance of length ℓ and $a \leq b$ are integers, we define the instance I restricted to $[a, b]$ as the subinstance of I induced by all variables of I from the a-th to the b-th (inclusive). We will denote I restricted to $[a, b]$ by $I_{[a,b]}$.

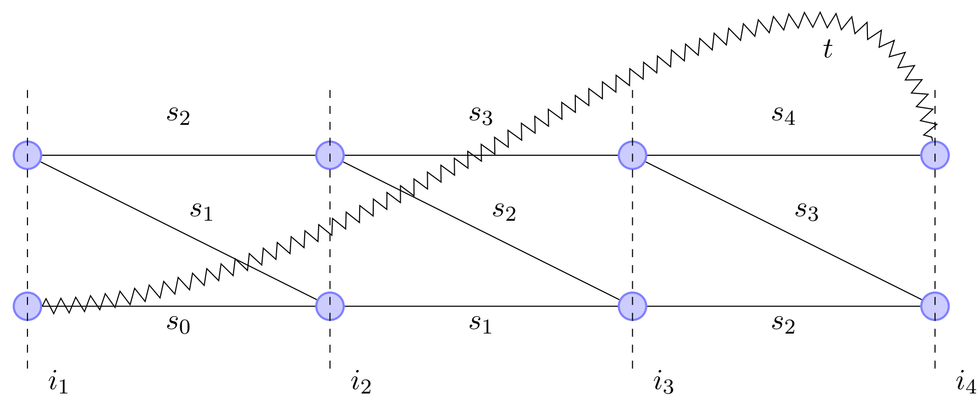

Figure 2: A sketch of a 4-braid. The solution t from Observation 4.3 pictured as a zigzag.

Definition 4.2. Let I be a path CSP instance on $[\ell]$ and $n \geq 2$ be an integer. An n-*braid* (see Figure 2) in I is a collection of $n+1$ solutions $s_0, s_1, s_2, \dots, s_n$ of I together with indices $1 \leq i_1 < \dots < i_n \leq \ell$ such that for all $k = 1, 2, \dots, n-1$ we have

(1) $s_k(i_k) = s_{k+1}(i_k)$, and
(2) $s_{k-1}(i_{k+1}) = s_k(i_{k+1})$.

When we want to explicitly describe a braid, we will often give the $(2n+1)$-tuple

$$(s_0, s_1, \dots, s_n; i_1, \dots, i_n).$$

We care about braids because it is easy to apply Hagemann-Mitschke terms to them to get new solutions of I. This observation is not new; one can find it formulated in a different language in [VF09, Theorem 8.4]:

Observation 4.3 (R. Freese, M. Valeriote)**.** Let $n \in \mathbb{N}$ and let $\mathbf{A}$ be a variety n-permutable algebra, I be a path instance of $\mathrm{CSP}(\mathbf{A})$, and let $(s_0, \dots, s_n;\ i_1, \dots, i_n)$ be an n-braid in I. Then there exists a solution t of I such that $t(i_1) = s_0(i_1)$ and $t(i_n) = s_n(i_n)$.

Proof. Since $\mathbf{A}$ is variety n-permutable, we have a chain of Hagemann-Mitschke terms $p_0, p_1, \dots, p_n$ compatible with constraints of I. All we need to do is apply these terms on $s_0, s_1, \dots, s_n$.

Denote by r_k the mapping $r_k(i) = p_k(s_{k-1}(i), s_k(i), s_{k+1}(i))$ where k goes from 1 to $n-1$; we let $r_0 = s_0$ and $r_n = s_n$. Since p_k is a polymorphism, each r_k is a solution of I. Moreover, one can verify using the Hagemann-Mitschke equations together with the equalities from the definition of an n-braid that for each $k = 1, \dots, n$ we have $r_{k-1}(i_k) = r_k(i_k)$.

Since I is a path instance, we can glue the solutions $r_0, \dots, r_n$ together: The mapping t defined as $t(i) = r_k(i)$ whenever $i_k < i \leq i_{k+1}$ (where we put $i_{-1} = 0$ and $i_{n+1} = \ell$ for convenience) is a solution of I. To finish the proof, it remains to observe that $t(i_1) = s_0(i_1)$ and $t(i_n) = s_n(i_n)$. □

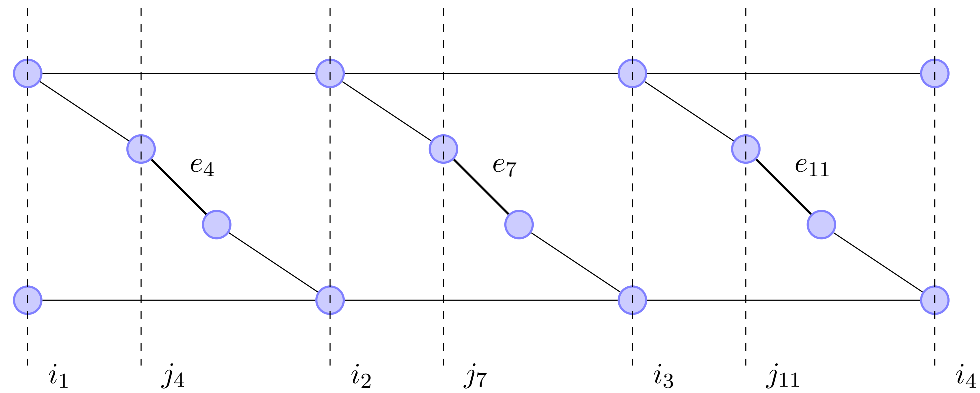

Figure 3: The conclusion of Lemma 4.5 for $n = 3$. Important edges e_i drawn in bold.

Let I be a path instance of CSP. We will say that a binary constraint $B_{i,i+1}$ of I is *subdirect* if $B_i, B_{i+1} \neq \emptyset$, $B_i \subseteq \pi_1(B_{i,i+1} \cap B_i \times B_{i+1})$, and $B_{i+1} \subseteq \pi_2(B_{i,i+1} \cap B_i \times B_{i+1})$. (We have modified the standard definition of subdirectness a bit to account for the fact that $B_{i,i+1}$ can contain tuples outside of $B_i \times B_{i+1}$.) An instance is subdirect if all its constraints are subdirect. Observe that every a subdirect path instance is satisfiable.

Observation 4.4. If I is a subdirect path instance and $e \in (B_i \times B_{i+1}) \cap B_{i,i+1}$, then by walking from e backwards and forwards along the edges defined by the binary constraints of I we get a solution s of I that contains the edge e, that is $(s(i), s(i+1)) = e$.

The following lemma tells us that if a path instance I is subdirect and we mark enough edges in I, we can find an n-braid that goes through many edges of our choosing. It is a Ramsey-like result and we prove it using the Ramsey theorem (see e.g. [vLW01, Theorem 3.3]).

Lemma 4.5. *For every n and N there exists an m with the following property: Let I be a subdirect path CSP instance of length $\ell > m$ such that $|B_i| \leq N$ for each $i \in [\ell]$. Then for any choice of indices $1 \leq j_1 < j_2 < \cdots < j_m < \ell$ and edges $e_k \in B_{j_k,j_k+1} \cap (B_{j_k} \times B_{j_k+1})$ for $k = 1, \ldots, m$, there exists an n-braid $(s_0, \ldots, s_n; i_1, \ldots, i_n)$ in I such that for every $k = 1, 2, \ldots, n-1$ there is a q so that $i_k \leq j_q < i_{k+1}$ and $(s_k(j_q), s_k(j_q + 1)) = e_q$ (that is, between every pair of "crossings" is an edge e_q; see Figure 3).*

Proof. Without loss of generality we can assume that $B_i \subseteq [N]$ for each i. For each $k = 1, \ldots, m$, we choose and fix a solution σ_k of I that contains the edge e_k (which we get from subdirectness of I; see above).

Consider now the complete graph G with vertex set $[m]$ whose edges are colored as follows: For every $u < v$ we color the edge $\{u, v\} \in \binom{m}{2}$ by the pair of numbers $(\sigma_u(j_v), \sigma_v(j_u)) \in [N]^2$. By the Ramsey theorem, if m is large enough then there exists a monochromatic induced subgraph of G on $2n+1$ vertices. To make our notation simpler, we will assume that these vertices are $1, 2, \ldots, 2n+1$.

Thanks to edges of G being monochromatic on $[2n+1]$, we have that σ_u and $\sigma_{u'}$ agree on j_v as long as $u, u', v \in [2n+1]$ and either $u, u' < v$, or $u, u' > v$. Using this, we can easily verify that $(\sigma_1, \sigma_3, \ldots, \sigma_{2n+1}; j_2, j_4, \ldots, j_{2n})$ is an n-braid. For each $k = 1, 2, \ldots, n-1$ we get:

$$\sigma_{2k+1}(j_{2k}) = \sigma_{2k+3}(j_{2k})$$
$$\sigma_{2k+1}(j_{2k+2}) = \sigma_{2k-1}(j_{2k+2})$$

To finish the proof, observe that for every $k \in [n-1]$ we have $j_{2k} < j_{2k+1} < j_{2k+2}$ and the solution σ_{2k+1} was chosen so that it passes through e_{2k+1}, so we can let $q = 2k+1$ and satisfy the conclusion of the lemma. □

Given a path instance I, we will define the sets $C_i \subseteq B_i$ by $C_1 = B_1$ and

$$C_{i+1} = \{b \in B_{i+1} \colon \exists c \in C_i, (c,b) \in B_{i,i+1}\}.$$

The sets C_i correspond to the endpoints of solutions of $I_{[1,i]}$, so I is satisfiable if and only if $C_\ell \neq \emptyset$. We will call an edge $(d,c) \in B_{i,i+1}$ such that $d \in B_i \setminus C_i$ and $c \in C_{i+1}$ a *backward edge*.

Our goal in Section 6 will be to show how to use symmetric Datalog to identify unsatisfiable path CSP($\mathbb{A}$) instances for $\mathbb{A}$ fixed and variety n-permutable. We will see that in the absence of backward edges a simple symmetric Datalog program can identify all unsatisfiable path CSP instances. This is why we want to know what happens when there are many backward edges. It turns out that an variety n-permutable instance that has too many backward edges is never subdirect. In Section 6, this will enable us to reduce the size of the instance.

Lemma 4.6. *For every n and N there exists an m such that if I is a path instance of length $\ell > m$ and $1 < a < b < \ell$ are such that*

(1) *each set B_i has cardinality at most N, and*
(2) *all sets B_i and all relations $B_{i,i+1}$ are invariant under a chain of n Hagemann-Mitschke terms, and*
(3) *there are at least m distinct indices j in $[a,b)$ such that $B_{j,j+1}$ contains a backward edge,*

then the instance $I_{[a,b]}$ is not subdirect.

Proof. We pick m large enough to be able to use Lemma 4.5 for sets B_i of maximum size N and $(n+1)$-braids. Taking this m, we look at what would happen were $I_{[a,b]}$ subdirect.

Let $a \leq j_1 < \cdots < j_m < b$ be a list of indices where backward edges occur in $[a,b)$. For each $k = 1,\ldots,m$, we choose a backward edge $e_{j_k} \in B_{j_k,j_k+1}$ and apply Lemma 4.5 to $I_{[a,b]}$. We obtain an $(n+1)$-braid in $I_{[a,b]}$ that uses $n+1$ of our backward edges; denote the solutions and indices that make up this braid by $s_0,\ldots,s_{n+1}$ and $i_1, i_2, \ldots, i_{n+1}$, respectively. Moreover, since s_1 passes through a backward edge e_j for some $j \in [i_1, i_2)$, we get $s_1(i_2) \in C_{i_2}$. Since the only condition on s_0 is $s_0(i_2) = s_1(i_2)$, we can modify s_0 to ensure $s_0(i_1) \in C_{i_1}$ without breaking the braid. The situation is sketched in Figure 4.

Observation 4.3 then gives us that $I_{[a,b]}$ has a solution t such that $t(i_1) = s_0(i_1) \in C_{i_1}$ and $t(i_n) = s_n(i_n) = s_{n+1}(i_n)$ (shown by a dashed line in Figure 4).

Now it remains to see that since $t(i_n) = s_{n+1}(i_n)$, there is a path from $t(i_n)$ to some backward edge e_j, $j \geq i_{n+1}$. Therefore, $t(i_n) \in B_{i_n} \setminus C_{i_n}$ and solution t witnesses that there is a path from $t(i_1) \in C_{i_1}$ to $t(i_n) \notin C_{i_n}$, a contradiction with the way we have defined the sets C_i. □

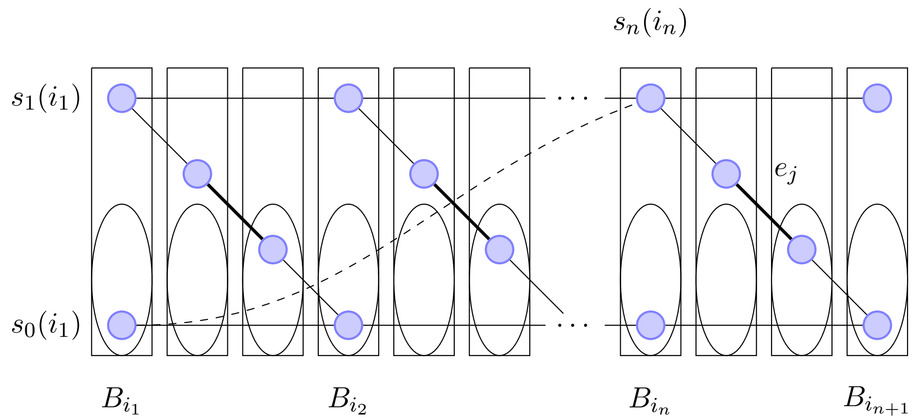

Figure 4: A schematic view of the instance $I_{[a,b]}$ (the ellipses are the sets C_i, backward edges e_j are thick).

5. Undirected reachability on path instances

Given a path CSP instance I, we define the digraph $\mathrm{Conn}(I)$ of I as the directed graph with vertex set equal to the disjoint union of all unary constraints $B_1, \dots, B_n$ and edge set equal to the disjoint union of all binary constraints of I (restricted to the sets B_i). The orientation of $\mathrm{Conn}(I)$ establishes levels on the graph (B_1 is on the first level, B_2 on the second level and so on).

Given a path CSP instance I and numbers $i \leq j$, the relation $\lambda_{I,i,j}$ consists of all pairs $a \in B_i, b \in B_j$ that lie in the same component of weak connectivity of $\mathrm{Conn}(I_{[i,j]})$ (i.e. there is an oriented, but not necessarily directed, path from a to b in $\mathrm{Conn}(I_{[i,j]})$).

Lemma 5.1. *If I is a path CSP instance of $\mathrm{CSP}(\mathbb{A})$ and $i \leq j$, then $\lambda_{I,i,j}$ lies in the relational clone of $\mathbb{A}$.*

Proof. It is easy to see that for $a \in B_i$ and $b \in B_j$ we have $(a, b) \in \lambda_{I,i,j}$ if and only if there is a digraph homomorphism $h\colon P \to \mathrm{Conn}(I_{[i,j]})$ where P is an oriented path which starts at level 0, ends at level $j - i$, has no vertex of level less than 0 or more than $j - i$, and h maps the starting point of P to a and ending point of P to b.

Let now the path P witness $(a, b) \in \lambda_{I,i,j}$ and the path Q witness $(c, d) \in \lambda_{I,i,j}$. By [HN04, Lemma 2.36], $P \times Q$ then contains an oriented path R that goes from level 0 to level $j - i$. By considering projections of $P \times Q$, we obtain that R homomorphically maps to both P and Q and from this it is easy to verify that R witnesses both $(a, b), (c, d) \in \lambda_{I,i,j}$. Since there are only finitely many pairs in $\lambda_{I,i,j}$, we can repeat this procedure to find a path S that witnesses the whole $\lambda_{I,i,j}$. It is then straightforward to translate homomorphisms from S to $\mathrm{Conn}(I_{[i,j]})$ into a primitive positive definition of $\lambda_{I,i,j}$ in $\mathbb{A}$. $\square$

Lemma 5.2. *For every relational structure $\mathbb{A}$, every path instance I of $\mathrm{CSP}(\mathbb{A})$, and every $i \leq j$, we have $\mathcal{P}^3_{\mathbb{A}}(I) \vdash \lambda_{I,i,j}(i, j)$.*

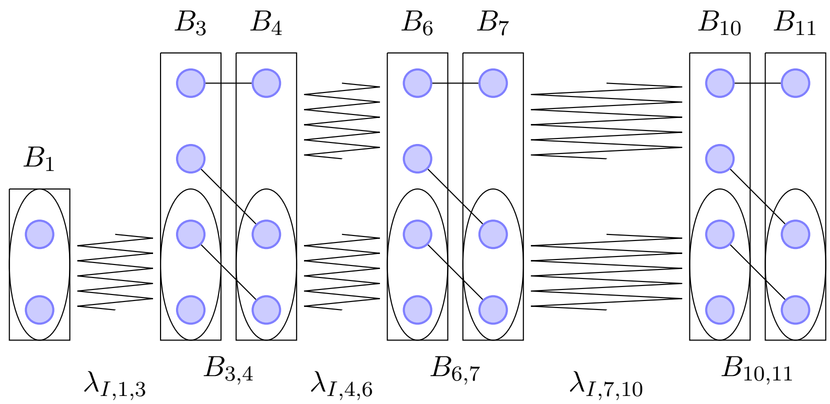

Figure 5: The instance I_λ with $i_1 = 3, i_2 = 6, i_3 = 10$ (ellipses mark the sets $C_i = D_i$).

Proof. Let us fix i and j. For $k \in \{i, i+1, \dots, j\}$, consider the relation

$$\rho_k = \{(a,b) \in B_i \times B_k \colon a,\, b \text{ lie in the same component of weak connectivity of } \mathrm{Conn}(I_{[i,j]})\}.$$

We show by induction on k that $\mathcal{P}^3_{\mathbb{A}}(I) \vdash \rho_k(i,k)$ for every $k = i, \dots, j$. This will be enough, since $\rho_j = \lambda_{I,i,j}$.

The base case $k = i$ is easy: Since $\rho_i \supseteq \{(b,b) \colon b \in B_i\}$, the program $\mathcal{P}^3_{\mathbb{A}}$ contains the rule $\rho_i(x,x) \leftarrow B_i(x)$, so we get $\mathcal{P}^3_{\mathbb{A}}(I) \vdash \rho_i(i,i)$.

The induction step: Assume we have $\mathcal{P}^3_{\mathbb{A}} \vdash \rho_k(i,k)$. Given the definition of ρ_k and ρ_{k+1}, it is straightforward to verify that the pair of rules

$$\rho_{k+1}(x,z) \leftarrow \rho_k(x,y) \wedge B_{k,k+1}(y,z)$$
$$\rho_k(x,y) \leftarrow \rho_{k+1}(x,z) \wedge B_{k,k+1}(y,z)$$

is consistent with $\mathbb{A}$ and therefore present in $\mathcal{P}^3_{\mathbb{A}}$. Applying the first of those rules (with $x = i$, $y = k$, and $z = k+1$) then gives us $\mathcal{P}^3_{\mathbb{A}}(I) \vdash \rho_{k+1}(i,k+1)$, completing the proof. □

Let I be a path instance of $\mathrm{CSP}(\mathbb{A})$ of length ℓ. In the following, we will again be using the sets C_i from Section 4.

Let $1 < i_1 < i_2 < \cdots < i_k < \ell$ be the complete list of all indices i with a backward edge in $B_{i,i+1}$ (i.e. all i such that that $B_{i,i+1} \cap ((B_i \setminus C_i) \times C_{i+1}) \neq \emptyset$). For convenience, we let $i_0 = 0$ and $i_{k+1} = \ell$.

Now consider the new path instance I_λ (see Figure 5) with variable set

$$U = \{1, i_1, i_1+1, i_2, \dots, i_k, i_k+1, \ell\}.$$

We get I_λ from $I_{\restriction U}$ by filling out the gaps by relations $\lambda_{I,i_j+1,i_{j+1}}$: For all j such that $i_j + 1 < i_{j+1}$ (i.e. $I_{\restriction U}$ has no binary constraint between i_j+1 and i_{j+1}), we add the binary constraint $((i_j+1, i_{j+1}), \lambda_{I,i_j+1,i_{j+1}})$ to I_λ. See Figure 5.

By Lemma 5.1, the constraints of I_λ belong to the relational clone of $\mathbb{A}$. Let for each $v \in U$ the set $D_v \subseteq B_v$ consist of all values of $s(v)$ where s is a solution of $(I_\lambda)_{[1,v]}$. It is easy to show by induction on v that $D_v = C_v$ for all $v \in U$. In particular, we have that I_λ is satisfiable if and only if I is satisfiable. Moreover, I_λ has a backward edge in roughly every other binary constraint. Finally, $\mathcal{P}^3_{\mathbb{A}}$ derives I_λ from I by Lemma 5.2.

We can summarize the findings of this section as follows:

Lemma 5.3. *Let $\mathbb{A}$ be a relational structure and let I be an unsatisfiable path instance of* $\mathrm{CSP}(\mathbb{A})$*. Then $\mathcal{P}^3_{\mathbb{A}}$ derives from I the unsatisfiable path instance I_λ with the following property: For all $m \geq 1$, any interval of variables of I_λ of length at least $2m+2$ contains at least m indices with backward edges and the constraints of I_λ are invariant under all polymorphisms of $\mathbb{A}$.*

6. Symmetric Datalog solves all variety n-permutable path instances

In this section, we put together the results from the previous two sections to show that for every variety n-permutable $\mathbb{A}$ there is an M such that $\mathcal{P}^M_{\mathbb{A}}(I) \vdash G$ for every unsatisfiable path instance I of $\mathrm{CSP}(\mathbb{A})$:

Theorem 6.1. *For each N and n there exists $f(n,N) \in \mathbb{N}$ so that whenever $\mathbb{A}$ is a variety n-permutable relational structure and I an unsatisfiable path instance of* $\mathrm{CSP}(\mathbb{A})$ *such that $|B_i| \leq N$ for all i, then $\mathcal{P}^{f(n,N)}_{\mathbb{A}}(I) \vdash G$.*

Proof. We prove the theorem first in the case when $\mathbb{A}$ contains symbols for all binary and unary relations compatible with $\mathbf{A}$, and then show how the general case follows.

We fix n and proceed by induction on N. For $N = 1$, a path instance is unsatisfiable if and only if at least one of $B_{i,i+1}$ does not intersect $B_i \times B_{i+1}$, which $\mathcal{P}^2_{\mathbb{A}}$ easily detects, so $f(n,1) = 2$ works.

Assume that the theorem is true for all structures and all instances with sets B_i smaller than N. Let m be the number from Lemma 4.6 for our n and N. We let $f(n,N) = f(n,N-1) + 2m + 6$ and claim that $\mathcal{P}^{f(n,N)}_{\mathbb{A}}(I) \vdash G$ for any $I \in \mathrm{CSP}(\mathbb{A})$ whose unary constraints B_i have at most N elements. For brevity, let us denote $2m+2$ by L, so we have $f(n,N) = f(n,N-1) + L + 4$.

Our starting point is the instance I_λ from Section 5. By the first part of Lemma 5.3, $\mathcal{P}^3_{\mathbb{A}}(I) \vdash I_\lambda$ and I_λ is an unsatisfiable path CSP instance of $\mathrm{CSP}(\mathbb{A})$. Consider now what $\mathcal{P}^{L+1}_{\mathbb{A}}$ does on I_λ. First of all, if the length of I_λ is at most L, then $\mathcal{P}^{L+1}_{\mathbb{A}}$ can easily check feasibility of I_λ by looking at the whole instance at once. So if I_λ is short, we get $\mathcal{P}^{L+1}_{\mathbb{A}}(I_\lambda) \vdash G$ and we are done (by Lemma 3.4, we have $\mathcal{P}^{L+4}_{\mathbb{A}}(I) \vdash G$). This is why in the rest of the proof we will assume that I_λ is longer than L. We show that $\mathcal{P}^{L+1}_{\mathbb{A}}(I_\lambda)$ derives another unsatisfiable instance K that falls within the scope of the induction hypothesis.

It turns out that I_λ contains many backward edges: By Lemma 5.3, each interval of I_λ of length $2m+2$ contains at least m backward edges. We can thus use Lemma 4.6 to show that any interval of I_λ of length L contains at least one binary constraint that is not subdirect. These constraints will enable us to shrink the unary constraints on I_λ.

Let ℓ be the length of I_λ. For $1 \leq a \leq i \leq b \leq \ell$ we will introduce the following two relations:

$$S_{I_\lambda,[a,b],i} = \{s(i) \colon s \text{ is a solution of } (I_\lambda)_{[a,b]}\},$$
$$S_{I_\lambda,[a,b]} = \{(s(a), s(b)) \colon s \text{ is a solution of } (I_\lambda)_{[a,b]}\}.$$

It is easy to see that these relations lie in the relational clone of $\mathbb{A}$. From the definitions above, it easily follows that $\mathcal{P}^{L+1}_{\mathbb{A}}(I_\lambda) \vdash S_{I_\lambda,[a,b]}(a,b)$ and $\mathcal{P}^{L+1}_{\mathbb{A}}(I_\lambda) \vdash S_{I_\lambda,[a,b],i}(i)$ whenever

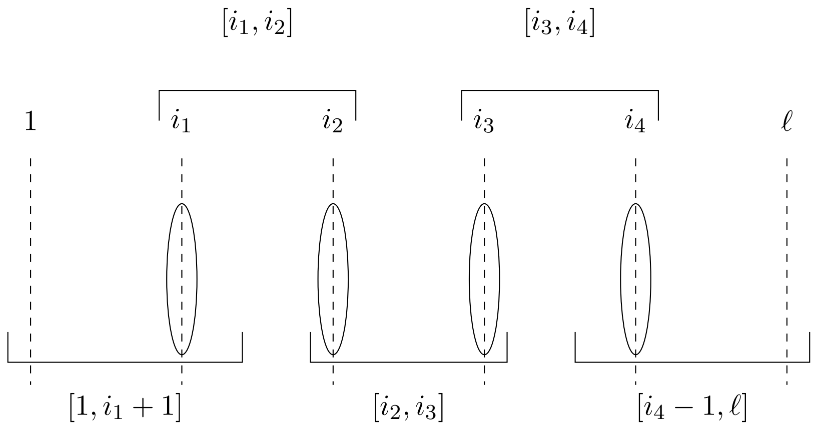

Figure 6: Constructing the instance K by looking at solutions of intervals of I_λ. Unary constraints of K shown as ellipses.

$b - a \leq L$ (this can be done in one step as the program is big enough to simply look at the whole of $(I_\lambda)_{[a,b]}$ at once).

We are now ready to show that $\mathcal{P}_{\mathbb{A}}^{L+1}(I_\lambda) \vdash K$, where K is an unsatisfiable path instance of CSP($\mathbb{A}$) whose unary constraints all have at most $N - 1$ elements.

We construct K as follows: Denote by B'_i the unary constraint on the i-th variable of I_λ. Since subdirectness fails somewhere in $[1, L]$, there is an index $i_1 \in [1, L]$ such that $S_{I_\lambda,[1,i_1+1],i_1}$ is strictly smaller than B'_{i_1}. Looking at $[i_1+1, i_1+L]$, we find an index i_2 where subdirectness fails again, so $S_{I_\lambda,[i_2-1,i_2+1],i_2} \subsetneq B'_{i_2}$. After that, we find $i_3 \in [i_2+1, i_2+L]$ such that $S_{I_\lambda,[i_3-1,i_3+1],i_3} \subsetneq B'_{i_3}$ and so on. Continuing in this manner, we get an increasing sequence of indices $1 \leq i_1 < i_2 < \cdots < i_k \leq \ell$ such that for each $j = 2, \ldots, k-1$ we have $|S_{I_\lambda,[i_j-1,i_j+1],i_j}| \leq |B'_{i_j}| - 1$ and $i_{j+1} - i_j \leq L$. We end this process when $i_k + L > \ell$. To properly analyze what goes on at the end of the chain, we need to consider two cases: When $i_k < \ell$ and when $i_k = \ell$. It is straightforward to verify that in both cases we have $\ell - i_k + 1 \leq L$ and $S_{I_\lambda,[i_k-1,\ell],i_k} \leq |B'_{i_k}| - 1$.

We take these indices i_j and observe that we have the following derivations (see Figure 6 for reference; note that L is at least 2):

$$
\begin{aligned}
&\mathcal{P}_{\mathbb{A}}^{L+1}(I_\lambda) \vdash S_{I_\lambda,[i_j-1,i_j+1],i_j}(i_j) \quad \text{for all } j = 2, \ldots, k-1,\\
&\mathcal{P}_{\mathbb{A}}^{L+1}(I_\lambda) \vdash S_{I_\lambda,[1,i_1+1],i_1}(i_1),\\
&\mathcal{P}_{\mathbb{A}}^{L+1}(I_\lambda) \vdash S_{I_\lambda,[i_k-1,\ell],i_k}(i_k),\\
&\mathcal{P}_{\mathbb{A}}^{L+1}(I_\lambda) \vdash S_{I_\lambda,[i_1,i_2]}(i_1, i_2),\\
&\mathcal{P}_{\mathbb{A}}^{L+1}(I_\lambda) \vdash S_{I_\lambda,[i_2,i_3]}(i_2, i_3),\\
&\qquad\qquad \vdots\\
&\mathcal{P}_{\mathbb{A}}^{L+1}(I_\lambda) \vdash S_{I_\lambda,[i_{k-1},i_k]}(i_{k-1}, i_k).
\end{aligned}
$$

We take these relations and use them to build up our instance K of $\mathrm{CSP}(\mathbb{A})$: The instance K has variables $i_1, i_2, \dots, i_k$. The constraints of K are as follows: K has unary constraints $(i_1, S_{I_\lambda,[1,i_1+1],i_1})$ (for the first variable), $(i_j, S_{I_\lambda,[i_j-1,i_j+1],i_j})$ for $j = 2, \dots, k-1$, and $(i_k, S_{I_\lambda,[i_k-1,\ell],i_k})$ for the last variable. The binary constraints of K are $((i_j, i_{j+1}), S_{I_\lambda,[i_j,i_{j+1}]})$ where $j = 1, \dots, k-1$.

Since the relations $S_{I_\lambda,\dots}$ incorporate all constraints of I_λ, it is straightforward to see that any solution of K would give us a solution of I_λ, so K is unsatisfiable. Moreover, all unary constraints of K have at most $N-1$ members and all constraint relations of K belong to the relational clone of $\mathbb{A}$. By the induction hypothesis, we then have $\mathcal{P}_{\mathbb{A}}^{f(n,N-1)}(K) \vdash G$. It now remains to use Lemma 3.4 twice: We get first $\mathcal{P}_{\mathbb{A}}^{f(n,N-1)+L+1}(I_\lambda) \vdash G$, followed by $\mathcal{P}_{\mathbb{A}}^{f(n,N-1)+L+4}(I) \vdash G$. Since we chose $f(n,N)$ to be $f(n,N-1)+L+4$, we have the desired result $\mathcal{P}_{\mathbb{A}}^{f(n,N)}(I) \vdash G$.

It remains to talk about the case when $\mathbb{A}$ does not contain symbols for all unary and binary compatible relations. Denote by $\mathbb{B}$ the relational structure we get from $\mathbb{A}$ by adding those missing relational symbols. Let I again be an instance of $\mathrm{CSP}(\mathbb{A})$ with each B_i of size at most N. By the above argument, we get $\mathcal{P}_{\mathbb{B}}^{f(n,N)}(I) \vdash G$, so there is a derivation of G in $\mathcal{P}_{\mathbb{B}}^{f(n,N)}$ from the relations of I. Observe now that the instance I only contains relations from $\mathbb{A}$ and that if we take $\mathcal{P}_{\mathbb{B}}^{f(n,N)}$ and delete rules that contain non-IDB predicates (the name used in the literature for non-IDB predicates is extensional database symbols) that are not basic relations of $\mathbb{A}$, we get $\mathcal{P}_{\mathbb{A}}^{f(n,N)}$. Therefore, the derivation of $\mathcal{P}_{\mathbb{B}}^{f(n,N)}(I) \vdash G$ also witnesses that $\mathcal{P}_{\mathbb{A}}^{f(n,N)}(I) \vdash G$ and we are done. □

By taking $M = f(n, |A|)$, we obtain the following corollary:

Corollary 6.2. *For each variety n-permutable relational structure $\mathbb{A}$ there exists $M \in \mathbb{N}$ so that whenever I is an unsatisfiable path instance of* $\mathrm{CSP}(\mathbb{A})$*, then $\mathcal{P}_{\mathbb{A}}^{M}(I) \vdash G$.*

7. From linear to symmetric Datalog

It remains to explain how to move from solving path CSP instances to solving general CSP instances. This is where we will need linear Datalog, or equivalently bounded pathwidth duality.

Given a relational structure $\mathbb{A}$, we use the idea from [BK12, Proposition 13] and define the k-th *bubble power* of $\mathbb{A}$ as the structure $\mathbb{A}^{(k)}$ with the universe A^k and the following basic relations:

(1) All unary relations $S \subseteq A^k$ that can be defined by taking a conjunction of basic relations of $\mathbb{A}$ (we are also allowed to identify variables and introduce dummy variables, but not to do existential quantification), and
(2) all binary relations of the form
$$E_{\mathcal{I}} = \left\{ ((a_1, \dots, a_k), (b_1, \dots, b_k)) \in \left(A^k\right)^2 : \forall (i,j) \in \mathcal{I},\, a_i = b_j \right\}$$
where $\mathcal{I} \subseteq [k]^2$.

In this section, we show that if $\mathbb{A}$ has pathwidth duality at most $k-1$, then all we need to worry about are path CSP instances of $\mathrm{CSP}(\mathbb{A}^{(k)})$. Our method is straightforward, but we need to get a bit technical to take care of all details.

Lemma 7.1. *Let $\mathbb{A}$ be a (finite) relational structure, $k \in \mathbb{N}$. Assume that $\mathbb{A}$ has pathwidth duality $k-1$ and let $s \in \mathbb{N}$ be such that $\mathcal{P}^s_{\mathbb{A}^{(k)}}(I) \vdash G$ for each unsatisfiable path instance I of* $\mathrm{CSP}(\mathbb{A}^{(k)})$. *Then $\mathcal{P}^{k(s+2)}_{\mathbb{A}}$ decides* $\mathrm{CSP}(\mathbb{A})$.

Proof. We need to show that $\mathcal{P}^{k(s+2)}_{\mathbb{A}}(I) \vdash G$ for every unsatisfiable instance I. Since $\mathbb{A}$ has pathwidth duality $k-1$, it is enough to show that $\mathcal{P}^{k(s+2)}_{\mathbb{A}}(J) \vdash G$ whenever $J = (V, \mathcal{C})$ is an unsatisfiable $\mathrm{CSP}(\mathbb{A})$ instance of pathwidth at most $k-1$.

Let $X_1, \dots, X_\ell$ be the partition of V witnessing that J has pathwidth at most $k-1$. If $X_i \subseteq X_{i+1}$ resp. $X_{i+1} \subseteq X_i$ for some i, then we can delete the smaller of the two sets and still have a partition that satisfies Definition 2.3. Therefore, we can assume that all neighboring sets are incomparable. From this, it follows that all sets X_i are pairwise different, because $X_i = X_j$ for $i < j$ implies $X_i \subseteq X_{i+1}$.

We fix a linear order $\prec$ on V. For each i, we will represent X_i by the k-tuple $\chi_i \in X_i^k$ that we get by listing the elements of X_i from $\prec$-minimal to $\prec$-maximal, repeating the $\prec$-maximal element if X_i has less than k elements. Since the sets X_i are pairwise different, we get pairwise different tuples. Recall that $J_{\restriction X_i}$ denotes the subinstance of J induced by X_i.

We now construct an unsatisfiable path instance K of $\mathrm{CSP}(A^{(k)})$. The variable set of K is $\{\chi_1, \dots, \chi_\ell\}$. The constraints are as follows:

(1) For each i, the i-th unary constraint relation B_i lists all solutions of $J_{\restriction X_i}$. More formally, we let
$$B_i = \{\rho \circ \chi_i \colon \rho \in A^{X_i}, \text{ is a solution of } J_{\restriction X_i}\} \subseteq A^k.$$
It is straightforward to verify that B_i is a basic relation of $\mathbb{A}^{(k)}$.

(2) For each $i = 1, 2, \dots, \ell-1$, we encode the intersection of X_i and X_{i+1} by adding the constraint $B_{i,i+1} = E_{\mathcal{I}}$ where $\mathcal{I} = \{(a,b) \colon \chi_i(a) = \chi_{i+1}(b)\}$.

If r is a solution of K, we can construct a solution t of J as follows: For each $v \in V$, find an $i \in [\ell]$ and $j \in [k]$ such that $\chi_i(j) = v$ and let $t(v)$ be the j-th coordinate of $r(\chi_i)$. It is an easy exercise to verify that the t we obtain would be a solution of J. Since J is unsatisfiable, so is K.

Since K is a path instance, we get $\mathcal{P}^s_{\mathbb{A}^{(k)}}(K) \vdash G$. Now extend the set of variables of K to the whole V^k without adding any new constraints. While this new instance K' is no longer a path instance, it is still true that $\mathcal{P}^s_{\mathbb{A}^{(k)}}(K') \vdash G$ (the derivation of G can just ignore the new variables).

We can now use Lemma 3.5: The structure $\mathbb{B}$ in the Lemma will be $\mathbb{A}^{(k)}$ and the relations $S_1, \dots, S_m$ will be $B_1, B_2, \dots, B_\ell$ and $B_{1,2}, B_{2,3}, \dots, B_{\ell-1,\ell}$. It is straightforward to show that $\mathcal{P}^{2k}(J)$ derives the instance K: Each of the statements $\mathcal{P}^{2k}(J) \vdash \overline{B_i}(\overline{\chi_i})$ and $\mathcal{P}^{2k}(J) \vdash \overline{B_{i,i+1}}(\overline{\chi_i}, \overline{\chi_i})$ (where i ranges over $[\ell]$ and $[\ell-1]$, respectively) has a derivation of length one. Lemma 3.5 then gives us that $\mathcal{P}^{ks+2k}_{\mathbb{A}}(J) \vdash G$, concluding the proof. $\square$

We are now ready to prove our main result:

Theorem (Theorem 2.7 restated)**.** *Let $\mathbb{A}$ be a relational structure such that there is a linear Datalog program that decides* CSP($\mathbb{A}$) *and $\mathbb{A}$ admits a chain of n Hagemann-Mitschke terms as polymorphisms. Then there exists a number M so that $\mathcal{P}_{\mathbb{A}}^{M}$ decides* CSP($\mathbb{A}$)*.*

Proof. Since there is a linear Datalog program that decides CSP($\mathbb{A}$), there is a $k \in \mathbb{N}$ so that $\mathbb{A}$ has pathwidth duality at most k.

It is straightforward to verify that the basic relations of the bubble power $\mathbb{A}^{(k)}$ are compatible with the Hagemann-Mitschke terms of $\mathbb{A}$ applied componentwise (recall that the universe of $\mathbb{A}^{(k)}$ is the k-th power of A), so $\mathbb{A}^{(k)}$ is variety n-permutable. Corollary 6.2 then gives us that there is an integer M' such that the program $\mathcal{P}_{\mathbb{A}^{(k)}}^{M'}$ derives the goal predicate on any unsatisfiable path instance of CSP($\mathbb{A}^{(k)}$). Therefore, Lemma 7.1 gives us that $\mathcal{P}_{\mathbb{A}}^{(k+2)M'}$ decides CSP($\mathbb{A}$). $\square$

8. Conclusions

In Theorem 2.7, we gave a characterization of the class of CSPs solvable by symmetric Datalog programs. Unfortunately, our result depends on understanding the power of linear Datalog; the characterization of CSPs solvable by linear Datalog is an open problem at the moment.

However, once somebody obtains a characterization of linear Datalog, our result immediately gives a characterization of symmetric Datalog. To see how that could come about, let us reexamine some conjectures about the CSPs solvable by fragments of Datalog [LT09] that would give us a characterization of symmetric Datalog:

Conjecture 8.1 (B. Larose, P. Tesson)**.** Let $\mathbb{A}$ be a finite relational structure such that the algebra of polymorphisms of $\mathbb{A}$ generates a variety that only admits the lattice and/or Boolean tame congruence theory types (equivalently, the variety is congruence semidistributive). Then there is a linear Datalog program that decides CSP($\mathbb{A}$).

An alternative way to settle the complexity of CSPs solvable by symmetric Datalog would be to replace “linear Datalog” in Theorem 2.7 by just “Datalog”. In particular, if the following were true, we would get a characterization of symmetric Datalog, too:

Conjecture 8.2 (B. Larose, P. Tesson)**.** Let $\mathbb{A}$ be a relational structure such that the algebra of polymorphisms $\mathbf{A}$ of $\mathbb{A}$ is idempotent and generates a variety that only admits the Boolean tame congruence theory type. Then CSP($\mathbb{A}$) is solvable by linear Datalog.

If Conjecture 8.1 or 8.2 is true, then the following are equivalent for any core relational structure $\mathbb{A}$:

(1) $\mathbb{A}$ is variety n-permutable for some n and CSP($\mathbb{A}$) is solvable by Datalog.
(2) The idempotent reduct of $\mathbf{A}$ generates a variety that admits only the tame congruence theory type 3.
(3) There exists a symmetric Datalog program that decides CSP($\mathbb{A}$).

Here the implication (1) $\Rightarrow$ (3) (or (2) $\Rightarrow$ (3)) is the unknown one. Implication (3) $\Rightarrow$ (2) follows from [LT09, Theorem 4.2], while [HM96, Theorem 9.15] together with the characterization of problems solvable by Datalog [BK14] gives us (1) $\Leftrightarrow$ (2).

We end with another citation of [LT09] whose consequences we find tantalizing: Assume that $\mathsf{L} \neq \mathsf{NL}$ and $\mathsf{L} \neq \mathsf{Mod}_p\mathsf{L}$ for any p prime. Then we can add a fourth statement to the above list:

(d) CSP($\mathbb{A}$) is in L modulo first order reductions.

From one side, symmetric Datalog programs can be evaluated in logarithmic space. For the other implication, we cite Theorem 2.2 to see that unless $\mathbf{A}$ only admits the Boolean type, there is a first order reduction to CSP($\mathbb{A}$) from a problem that is NL-hard or $\mathsf{Mod}_p\mathsf{L}$-hard for some p.

Acknowledgments

I would like to thank Ralph McKenzie for support and mentoring. I'm grateful to Pascal Tesson for helping me understand first order reductions better and to Benoit Larose for pointing out an error in the definition of subdirect constraints in the preprint of this paper. Finally, I thank the referees for their perceptive and valuable comments.

This work was begun at Vanderbilt University, Nashville, TN, USA, and was finished at IST Austria and Charles University. Funded by European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no. 616160 and PRIMUS/SCI/12 project of the Charles University.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: A modern approach*. Cambridge University Press, New York, 2009.

[Ats08] Albert Atserias. On digraph coloring problems and treewidth duality. *European Journal of Combinatorics*, 29(4):796–820, May 2008.

[Bar13] Libor Barto. Finitely related algebras in congruence distributive varieties have near unanimity terms. *Canadian Journal of Mathematics*, 65:3–21, 2013.

[Ber11] Clifford Bergman. *Universal Algebra: Fundamentals and Selected Topics*. Chapman & Hall/CRC Press, Boca Raton and New York and Abingdon, 1 edition, 2011.

[BK12] Libor Barto and Marcin Kozik. Robust satisfiability of constraint satisfaction problems. *Proceedings of the 44th symposium on Theory of Computing*, pages 931–940, 2012.

[BK14] Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *Journal of the ACM*, 61(1):3:1–3:19, January 2014.

[BKJ00] Andrei A. Bulatov, Andrei A. Krokhin, and Peter Jeavons. Constraint satisfaction problems and finite algebras. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, ICALP '00, pages 272–282, London, UK, 2000. Springer-Verlag.

[BKKR69] V.G. Bodnarchuk, L.A. Kaluzhnin, V.N. Kotov, and B.A. Romov. Galois theory for Post algebras. I. *Cybernetics*, 5(3):243–252, 1969.

[BKL08] Andrei Bulatov, Andrei Krokhin, and Benoit Larose. Dualities for constraint satisfaction problems. In Nadia Creignou, Phokion G. Kolaitis, and Heribert Vollmer, editors, *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 93–124. Springer Berlin Heidelberg, 2008.

[BKW12] Libor Barto, Marcin Kozik, and Ross Willard. Near unanimity constraints have bounded pathwidth duality. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 125–134, 2012.

[Dal05] Victor Dalmau. Linear Datalog and bounded path duality of relational structures. *Logical Methods in Computer Science*, 1(1:5):1–32, 2005.

[DL08] Victor Dalmau and Benoit Larose. Maltsev + Datalog $\Rightarrow$ Symmetric Datalog. In *LICS '08: Proceedings of the 2008 23rd Annual IEEE Symposium on Logic in Computer Science*, pages 297–306, Washington, 2008. IEEE Computer Society.

[ELT07] László Egri, Benoit Larose, and Pascal Tesson. Symmetric Datalog and constraint satisfaction problems in logspace. In *LICS '07: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science*, pages 193–202, Washington, 2007. IEEE Computer Society.

[FV99] Tomás Feder and Moshe 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, 1999.

[Gei68] David Geiger. Closed systems of functions and predicates. *Pacific Journal of Mathematics.*, 27(1):95–100, 1968.

[HM73] Joachim Hagemann and Aleit Mitschke. On n-permutable congruences. *Algebra universalis*, 3(1):8–12, 1973.

[HM96] David Hobby and Ralph McKenzie. *The Structure of Finite Algebras*, volume 76 of *Contemporary Mathematics*. American Mathematical Society, Providence, 1996.

[HN04] Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, New York, 2004.

[Imm99] Neil Immerman. *Descriptive complexity*. Springer, New York, 1999.

[LP82] Harry R. Lewis and Christos H. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19(2):161–187, 1982.

[LT09] Benoit Larose and Pascal Tesson. Universal algebra and hardness results for constraint satisfaction problems. *Theoretical Computer Science*, 410(18):1629–1647, 2009.

[Rei05] Omer 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, New York, NY, USA, 2005. ACM.

[VF09] Matthew A. Valeriote and Ralph Freese. On the complexity of some Maltsev conditions. *International Journal of Algebra and Computation*, 19(2):451–464, 2009.

[vLW01] J. H. van Lint and R. M. Wilson. *A Course in Combinatorics*. Cambridge University Press, 2nd edition, 2001.