Polymorphisms, and how to use them

Libor Barto[1], Andrei Krokhin[2], and Ross Willard[3]

1 **Department of Algebra, Faculty of Mathematics and Physics**
Charles University, Prague, Czech Republic
libor.barto@gmail.com
2 **School of Engineering and Computing Sciences**
University of Durham, UK
andrei.krokhin@durham.ac.uk
3 **Department of Pure Mathematics**
University of Waterloo, Canada
ross.willard@uwaterloo.ca

Abstract

This article describes the algebraic approach to Constraint Satisfaction Problem that led to many developments in both CSP and universal algebra. No prior knowledge of universal algebra is assumed.

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity: General, F.2.0 Discrete Mathematics: General.

Keywords and phrases constraint satisfaction, complexity, universal algebra, polymorphism

Digital Object Identifier 10.4230/DFU.Vol7.15301.1

1 Introduction

The Constraint Satisfaction Problem (CSP) provides a common framework for expressing a wide range of both theoretical and real-life combinatorial problems [111]. Roughly, these are problems where one is given a collection of constraints on overlapping sets of variables and the goal is to assign values to the variables so as to satisfy the constraints. This computational paradigm is very general, and includes many specific well-known problems from several areas of Computer Science and Mathematics. In the last 20 years, complexity-theoretic aspects of CSPs have attracted a great deal of attention at the top level in Theoretical Computer Science. The main reason for this is that the CSP paradigm strikes a perfect balance between generality and structure: it is general enough to reflect very many important computational phenomena, yet it has enough structure that can be exploited to gain very deep insights into these phenomena. The CSP paradigm is often used to tackle the following fundamental general question: *What kind of mathematical structure in computational problems allows for efficient algorithms?*

The topic of this paper is a very active theoretical subfield which studies the computational complexity and other algorithmic properties of the decision version of CSP over a fixed constraint language on a finite domain. This restricted framework is still broad enough to include many decision problems from the class NP, yet it is narrow enough to potentially allow for complete classifications of all such CSP problems.

One particularly important achievement is the understanding of what makes the problems over a fixed constraint language computationally easy or hard. It is not surprising that hardness comes from a lack of symmetry. However, the usual objects capturing symmetry, automorphisms (or endomorphisms) and their groups (or semigroups), are not sufficient in

The Constraint Satisfaction Problem: Complexity and Approximability. *Dagstuhl Follow-Ups*, Volume 7, ISBN 978-3-95977-003-3.
Editors: Andrei Krokhin and Stanislav Živný; pp. 1–44

Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

this context. It turns out that the complexity of the CSP over a fixed constraint language is determined by more general symmetries of it: polymorphisms and their clones.

Our aim is to introduce the basics of this exciting area in a way that is understandable to readers with a basic knowledge of computational complexity (see [106, 3]). Our particular focus is on explaining, with worked-out examples, how polymorphisms are applied to obtain positive and negative algorithmic results and why this approach is natural for many classification problems about CSPs and constraint languages. We do not assume any knowledge of algebra and minimize the algebraic terminology that we (define and) use, so the deep universal algebra, which is at the technical core of many advanced results in this direction, stays in the background. Many papers in the reference list contain deep algebra, though.

The structure of the survey is as follows. In Section 2 we give the basic definitions and examples of CSPs over a fixed constraint language, and discuss the main goals of the research programme that we survey. In Section 3 we describe various standard reductions between CSPs with different constraint languages, which on the one hand allows one to group together constraint languages with the same computational properties of the corresponding decision CSPs, and on the other hand paves the path to the algebraic approach to our classification problems. In Section 4 we explain how polymorphisms are used as classifiers for constraint languages and how this leads to hardness results and complexity classification conjectures. In Section 5 we explain how polymorphisms are used to guarantee correctness of algorithms that do not use polymorphisms in their execution. In Section 6 we discuss an algorithm that does use polymorphisms in an essential way in its execution.

2 CSP over a fixed constraint language

A constraint – such as $R(x_3, x_1, x_4)$ – restricts the allowed values for a tuple of variables – in this case (x_3, x_1, x_4) – to be an element of a particular relation on the domain – in this case $R \subseteq D^3$.[1] By an n-ary *relation* R on a domain D we mean a subset of the n-th cartesian power D^n. It is sometimes convenient to work with the corresponding *predicate* which is a mapping from D^n to $\{\text{true}, \text{false}\}$ specifying which tuples are in R. We will use both formalisms, so e.g. $(a, b, c) \in R$ and $R(a, b, c)$ both mean that the triple $(a, b, c) \in D^3$ is from the relation R.

An instance of the CSP is a list of constraints, e.g.,

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

where R, S, T are relations of appropriate arity on a common fixed domain D and x, y, z, w are variables. A mapping f assigning values from the domain to the variables is a *solution* if it satisfies all the constraints, that is, in our example,

$$R(f(x)) \text{ and } S(f(y), f(y), f(z)) \text{ and } T(f(y), f(w)) \ .$$

A standard formal definition of an instance of the CSP over a finite domain goes as follows.

▶ **Definition 1.** An *instance of the CSP* is a triple $P = (V, D, \mathcal{C})$ with
- V a finite set of *variables*,
- D a finite *domain*,

[1] There are also other types of constraints considered in the literature, e.g. valued and global constraints [111].

- $\mathcal{C}$ a finite list of constraints, where each constraint is a pair $C = (\mathbf{x}, R)$ with
  - $\mathbf{x}$ a tuple of variables of length n, called the *scope* of C, and
  - R an n-ary relation on D, called the *constraint relation* of C.

An *assignment*, that is, a mapping $f : V \to D$, *satisfies* a constraint $C = (\mathbf{x}, R)$ if $f(\mathbf{x}) \in R$, where f is applied component-wise. An assignment f is a *solution* if it satisfies all constraints.

Three basic computational problems associated with an instance are the following:

- **Decision.** Does the given instance have a solution? (A related problem, the *search problem*, is to find some solution if at least one solution exists.)
- **Optimization.** Even if the instance has no solution, find an optimal assignment, i.e., one that satisfies the maximum possible number of constraints. (*Approximation algorithms* are also extensively studied, where the aim is, for example, to find an assignment that satisfies at least 80% of the number of constraints satisfied by an optimal assignment.)
- **Counting.** How many solutions does the given instance have? (This problem also has an approximation version: *approximate counting*.)

To study the computational complexity of these problems we need to specify a representation of instances. In particular, we will assume that the constraint relation in every constraint is given by a list of all its members. Note, however, that for most of the problems considered in this article any reasonable representation can be taken.

2.1 Constraint languages

Even the easiest of the problems, decision, is computationally hard: It contains many NP-complete problems including, e.g., 3-SAT (see Example 3). However, certain natural restrictions ensure tractability. The main types of restrictions that have been studied are *structural restrictions*, which limit how constraints interact, and *language restrictions*, which limit the choice of constraint relations.

In this paper, we focus just on decision problems with language restrictions. See [92] for optimization problems and a generalization to valued CSPs, [70, 102] for approximation, [81] for counting, [24, 25, 107] for a generalization to infinite domains, and [105] for work on structural restrictions.

▶ **Definition 2.** A *constraint language* $\mathcal{D}$ is a finite set of relations on a common finite domain, D. We use $\mathrm{CSP}(\mathcal{D})$ to denote the restriction of the general CSP decision problem to instances in which the domain is D and all constraint relations are from $\mathcal{D}$.

We remark that constraint languages (on a finite domain) are often defined to also include infinite sets of relations. For such languages, one can define the complexity in terms of finite subsets, or else one has to specify the choice of representation of instances. For simplicity, we focus on finite constraint languages.

2.2 Examples

▶ **Example 3.** An instance of the standard NP-complete problem [106, 3], 3-SAT, is a Boolean formula in conjunctive normal form with exactly three literals per clause. For example, the formula,

$$\varphi \quad = \quad (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_4 \vee x_5 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_4 \vee \neg x_3)$$

is a satisfiable instance of 3-SAT. (Any assignment making x_1 and x_2 false, satisfies φ.) 3-SAT is equivalent to $\mathrm{CSP}(\mathcal{D}_{3\mathrm{SAT}})$, where $D_{3\mathrm{SAT}} = \{0, 1\}$ and

$$\mathcal{D}_{3\mathrm{SAT}} = \{S_{ijk} : i, j, k \in \{0, 1\}\}, \text{ where } S_{ijk} = \{0, 1\}^3 \setminus \{(i, j, k)\} \ .$$

For example, the above formula φ corresponds to the following instance of $\mathrm{CSP}(\mathcal{D}_{\mathrm{3SAT}})$

$$S_{010}(x_1, x_2, x_3),\ S_{101}(x_4, x_5, x_1),\ S_{111}(x_1, x_4, x_3)\ .$$

More generally, for a natural number k, k-SAT denotes a similar problem where each clause is a disjunction of k literals.

Since 3-SAT is NP-complete, it follows that k-SAT is NP-complete for each $k \geq 3$. On the other hand, 2-SAT is solvable in polynomial time, and is in fact complete for the complexity class NL (non-deterministic logarithmic space) under log-space reductions [106, 3] (see also Example 9).

▶ **Example 4.** 1-in-3-SAT is $\mathrm{CSP}(\mathcal{D}_{\mathrm{1in3SAT}})$ where $\mathcal{D}_{\mathrm{1in3SAT}}$ contains the single relation $\{(0,0,1),(0,1,0),(1,0,0)\}$. This problem is well known to be NP-complete [112].

▶ **Example 5.** HORN-3-SAT is a restricted version of 3-SAT, where each clause may have at most one positive literal. This problem is equivalent to $\mathrm{CSP}(\mathcal{D}_{\mathrm{HornSAT}})$ for $\mathcal{D}_{\mathrm{HornSAT}} = \{S_{110}, S_{111}, C_0, C_1\}$ where $C_0 = \{0\}$ and $C_1 = \{1\}$. HORN-3-SAT is solvable in polynomial time, in fact, it is a P-complete problem under log-space reductions [3, 106].

▶ **Example 6.** For a fixed natural number k, the k-COLORING problem is to decide whether it is possible to assign colors $\{0, 1, \ldots, k-1\}$ to the vertices of an input graph in such a way that adjacent vertices receive different colors. This problem is equivalent to $\mathrm{CSP}(\mathcal{D}_{\mathrm{kCOLOR}})$, where $D_k = \{0, 1, 2, \ldots, k-1\}$ and $\mathcal{D}_{\mathrm{kCOLOR}} = \{\neq_k\}$ consists of a single relation – the binary inequality relation $\neq_k = \{(a,b) \in D_k^2 : a \neq b\}$.

Indeed, given an instance of $\mathrm{CSP}(\mathcal{D}_{\mathrm{kCOLOR}})$, we can form a graph whose vertices are the variables and whose edges correspond to the binary constraints (that is, x has an edge to y iff the instance contains the constraint $x \neq_k y$). It is easily seen that the original instance has a solution if and only if the obtained graph is k-colorable. The translation in the other direction is similar.

The k-COLORING problem is NP-complete for $k \geq 3$ [106, 3]. 2-COLORING is equivalent to deciding whether an input graph is bipartite. It is solvable in polynomial time, in fact, it is in the complexity class L (where L stands for logarithmic space) by a celebrated result of Reingold [109], and it is an L-complete problem under first-order reductions.

▶ **Example 7.** Given two digraphs $G = (V(G), E(G))$ and $H = (V(H), E(H))$, a mapping $f : V(G) \to V(H)$ is a *homomorphism* from G to H if f preserves edges, that is, $(u,v) \in E(G)$ implies $(f(u), f(v)) \in E(H)$. The problem whether an input digraph G admits a homomorphism to a fixed digraph H is also known as the H-COLORING problem and has been actively studied in graph theory [72], see also [96]. The k-COLORING problem is a special case of H-COLORING where H is the complete graph on k vertices.

For any digraph H, let $D = V(H)$ and let $\mathcal{D}_H$ be the language that contains just the binary relation $E(H)$. For any digraph H, the problem $\mathrm{CSP}(\mathcal{D}_H)$, corresponds to the H-COLORING problem, where the input digraph G is given by the scopes of the constraints. If we add all nonempty subsets of $V(H)$ as unary relations to $\mathcal{D}_H$, then the resulting CSP is known as LIST H-COLORING [72]. If we add just the singleton subsets of $V(H)$ as unary relations to $\mathcal{D}_H$, then the resulting CSP is known as One-or-All LIST H-COLORING [63, 64].

▶ **Example 8.** Let p be a prime number. An input of 3-LIN(p) is a system of linear equations over the p-element field GF(p), where each equation contains 3 variables, and the question is whether the system has a solution. This problem is equivalent to $\mathrm{CSP}(\mathcal{D}_{\mathrm{3LIN_p}})$, where

$D_{\text{3LIN}_\text{P}} = \text{GF}(p)$ and $\mathcal{D}_{\text{3LIN}_\text{P}}$ consists of all affine subspaces R_{abcd} of $\text{GF}(p)^3$ of dimension 2, where

$$R_{abcd} = \{(x, y, z) \in \text{GF}(p)^3 : ax + by + cz = d\} \ .$$

This problem is solvable in polynomial time, e.g. by Gaussian elimination.[2] It is complete for a somewhat less familiar complexity class Mod_pL [46].

▶ **Example 9.** An instance of the s, t-connectivity problem, STCON, consists of a directed graph and two of its vertices, s and t. The question is whether there exists a directed path from s to t.

A closely related (but not identical) problem is $\text{CSP}(\mathcal{D}_{\text{STCON}})$, where the domain is $D_{\text{STCON}} = \{0, 1\}$ and $\mathcal{D}_{\text{STCON}} = \{C_0, C_1, I\}$, $C_0 = \{0\}$, $C_1 = \{1\}$, $I = \{(0, 0), (0, 1), (1, 1)\}$. Indeed, given an instance of $\text{CSP}(\mathcal{D}_{\text{STCON}})$ we form a directed graph much as we did in Example 6 and label some vertices 0 or 1 according to the unary constraints. Then the original instance has a solution if and only if there is no directed path from a vertex labeled 1 to a vertex labeled 0. Thus $\text{CSP}(\mathcal{D}_{\text{STCON}})$ can be solved by invoking the complement of STCON, the s, t-non-connectivity problem, several times.

Both STCON and $\text{CSP}(\mathcal{D}_{\text{STCON}})$ can clearly be solved in polynomial time. By the Immerman-Szelepcsényi theorem [75, 115] both problems are NL-complete (under log-space reductions).

In the same way, the s, t-connectivity problem for undirected graphs is closely related to $\text{CSP}(\mathcal{D}_{\text{USTCON}})$, where $D_{\text{USTCON}} = \{0, 1\}$ and $\mathcal{D}_{\text{USTCON}} = \{C_0, C_1, =\}$. These problems are L-complete by [109].

2.3 The dichotomy conjecture

The most fundamental problem in the area was formulated in the landmark paper by Feder and Vardi [65].

▶ Conjecture 1 (The dichotomy conjecture). For each finite constraint language $\mathcal{D}$, the problem $\text{CSP}(\mathcal{D})$ is in P or is NP-complete.[3]

Recall that if P $\neq$ NP, then there are problems of intermediate complexity in NP [95]. Feder and Vardi argued that the class of CSPs over fixed constraint languages is a good candidate for the largest natural class of problems which exhibit a P versus NP-complete dichotomy.

At that time the conjecture was supported by two major cases: the dichotomy theorem for all languages over a two-element domain by Schaefer [112] and the dichotomy theorem for languages consisting of a single binary symmetric relation by Hell and Nešetřil [71].

Feder and Vardi identified two sources of polynomial-time solvability and made several important contributions towards understanding them. In particular, they observed that the known polynomial cases were tied to algebraic closure properties and asked whether polynomial solvability for CSP can always be explained in such a way. This was confirmed

[2] The problem of solving general systems of linear equations over $\text{GF}(p)$ without the restriction on number of variables cannot be faithfully phrased as $\text{CSP}(\mathcal{D})$, even if we allow $\mathcal{D}$ to consist of all affine subspaces, since the input representation of the latter problem can be substantially larger. However, a system of linear equations can be easily rewritten to an instance of 3-LIN(p) by introducing new variables.

[3] It is conjectured in [34] that the dichotomy remains true without the finiteness assumption on $\mathcal{D}$ (the domain D still needs to be finite), if constraint relations in inputs are given by full lists of their members. Namely, the local-global conjecture states that $\text{CSP}(\mathcal{D})$ is in P (NP-complete) whenever $\text{CSP}(\mathcal{D}')$ is in P (NP-complete) for every (some) finite $\mathcal{D}' \subseteq \mathcal{D}$.

by Jeavons, Cohen and Gyssens [80, 78], and these and subsequent papers based on this connection to algebra brought the area to another level, which probably could not be accessed with only combinatorial tools (such as those in [112] or [71]).

2.4 Alternative views

Note that if we order (or just name) relations in a constraint language $\mathcal{D}$ with domain D, then $\mathcal{D}$ can be viewed as a *relational structure* $(D; R_1, R_2, \dots)$, or equivalently as a *relational database*, with universe D.

Recall that a *Boolean conjunctive query* over the database $\mathcal{D}$ is an existential sentence whose quantifier-free part is a conjunction of atoms. CSP($\mathcal{D}$) is exactly the problem of deciding whether $\mathcal{D}$ satisfies a given Boolean conjunctive query. For example, the instance

$$R_1(x),\ R_1(w),\ R_3(y,y,z),\ R_7(y,w),\ R_7(x,y) \tag{1}$$

has a solution if and only if the sentence

$$(\exists x, y, z, w \in D)\ R_1(x) \wedge R_1(w) \wedge R_3(y,y,z) \wedge R_7(y,w) \wedge R_7(x,y)$$

is true in $\mathcal{D}$.

From this perspective, it is natural to ask what happens if we allow some other combination of logical connectives from $\{\exists, \forall, \wedge, \vee, \neg, =, \neq\}$. It turns out that out of the 2^7 cases only 3 are interesting (the other cases either reduce to these, or are almost always easy or hard by known results): $\{\exists, \wedge\}$ which is CSP, $\{\exists, \forall, \wedge\}$ which is so-called *quantified CSP*, and $\{\exists, \forall, \wedge, \vee\}$. Determining the complexity of quantified CSP is also an active research area [49] with a possible trichotomy – P, NP-complete or PSPACE-complete. Recently, a tetrachotomy was obtained for the last case [101] – for every $\mathcal{D}$, the corresponding problem is either in P, NP-complete, co-NP-complete, or Pspace-complete.

The CSP over a fixed language can also be formulated as the *homomorphism problem* between relational structures with a fixed target structure [65, 78]. Assume that we have two relational structures $\mathcal{E} = (E; S_1, S_2, \dots)$ and $\mathcal{D} = (D; R_1, R_2, \dots)$ which are similar, i.e. they have the same number of relations and the corresponding relations have the same arity. A *homomorphism* from $\mathcal{E}$ to $\mathcal{D}$ is a mapping $h : E \to D$ such that, for all i, if $\mathbf{a} = (a_1, a_2, \dots) \in S_i$ then $h(\mathbf{a}) = (h(a_1), h(a_2), \dots) \in R_i$. Then CSP($\mathcal{D}$) is equivalent to the problem of deciding whether a given relational structure $\mathcal{E}$ similar to $\mathcal{D}$ has a homomorphism to $\mathcal{D}$. The idea of the translation is shown in Examples 6 and 9. In general, to see the translation from the homomorphism form to the constraint form, view the set E as the set of variables and transform every tuple $\mathbf{x}$ in a relation S_i in $\mathcal{E}$ to a constraint $R_i(\mathbf{x})$. To see the translation back, let E (the domain of $\mathcal{E}$) be the set of all variables appearing in a given CSP instance, and let each relation S_i contain all tuples $\mathbf{x}$ such that this CSP instance contains a constraint $R_i(\mathbf{x})$. For example, if we translate the CSP($\mathcal{D}$) instance appearing above in (1) into a relational structure $\mathcal{E}$, then we have $E = \{x, y, z, w\}$ and $S_1 = \{x, w\}, S_3 = \{(y, y, z)\}, S_7 = \{(y, w), (x, y)\}$, with all the other relations S_i empty.

3 Reductions between constraint languages

This section describes relational constructions that allow one to reduce one CSP with a fixed constraint language to another. These constructions have algebraic counterparts, described in the following section, and this translates many (complexity) classification problems about constraint languages into algebraic classifications.

If a computational problem $\mathcal{A}$ can simulate (in some sense) another problem $\mathcal{B}$, then $\mathcal{A}$ is at least as hard as $\mathcal{B}$. This simple idea is widely used in computational complexity; for instance, NP-completeness is often shown by a gadget reduction of a known NP-complete problem to the given one. A crucial fact for the algebraic theory of the CSP is that a so called primitive positive (pp-, for short) interpretation between constraint languages gives such a reduction between corresponding CSPs (more precisely, if $\mathcal{D}$ pp-interprets $\mathcal{E}$, then $\mathrm{CSP}(\mathcal{E})$ is reducible to $\mathrm{CSP}(\mathcal{D})$). Pp-interpretations have been, indirectly, one of the main subjects of universal algebra for the last 80 years!

We will define three increasingly more general techniques for simulation between CSPs:

$$\text{pp-definition} \subseteq \text{pp-interpretation} \subseteq \text{pp-construction}.$$

In Section 4 we give algebraic characterizations for these techniques, which guide the algebraic approach.

The algebraic theory of CSPs was developed in a number of papers including [80, 78, 34, 97, 20]. The viewpoint taken here is close to [20, 24]. All results in this section come from these sources unless stated otherwise.

To simplify formulations, all structures (relational or algebraic) are assumed to have finite domains, all constraint languages are assumed to contain *finitely many* relations, all of them *nonempty*. By a *reduction* we mean a log-space reduction (although first-order reductions are often possible under additional weak assumptions).

3.1 Primitive positive interpretations (= gadgets)

An important special case of pp-interpretability is pp-definability.

▶ **Definition 10.** Let $\mathcal{D}, \mathcal{E}$ be constraint languages on the same domain $D = E$. We say that $\mathcal{D}$ *pp-defines* $\mathcal{E}$ (or $\mathcal{E}$ is pp-definable from $\mathcal{D}$) if each relation in $\mathcal{E}$ can be defined by a first order formula which only uses relations in $\mathcal{D}$, the equality relation, conjunction and existential quantification.

This terminology comes from model theory, where a first order formula is called *primitive* if it has the form $\exists \bar{y} \bigwedge_{i<n} \alpha_i(\bar{x}, \bar{y})$ where each $\alpha_i(\bar{x}, \bar{y})$ is an atomic or negated atomic formula. A *primitive positive* (or pp-) formula is a negation-free primitive formula.

Rephrasing the above definition without using logic, $\mathcal{D}$ pp-defines $\mathcal{E}$ if $\mathcal{D}$ and $\mathcal{E}$ have the same domain and every relation R_i in $\mathcal{E}$ can be represented by a gadget using relations from $\mathcal{D}$, as follows. There is an instance P_i of $\mathrm{CSP}(\mathcal{D} \cup \{=\})$ and subset X_i of variables in P_i such that the set of all solutions to P_i, when projected down to X_i, gives precisely the relation R_i.

▶ **Example 11.** Recall constraint languages $\mathcal{D}_{\mathrm{3SAT}}, \mathcal{D}_{\mathrm{Horn3SAT}}$, and $\mathcal{D}_{\mathrm{STCON}}$ from Examples 3, 5, and 9. We now show that $\mathcal{D}_{\mathrm{3SAT}}$ pp-defines $\mathcal{D}_{\mathrm{Horn3SAT}}$, which in turn pp-defines $\mathcal{D}_{\mathrm{STCON}}$. To show the first definition, notice that $C_0(x) = S_{111}(x, x, x)$ and $C_1(x) = S_{000}(x, x, x)$. For the second, it is enough to check that $I(x, y)$ holds if and only if $\exists z(C_1(z) \wedge S_{110}(z, x, y))$ holds.

▶ **Theorem 12.** *If $\mathcal{D}$ pp-defines $\mathcal{E}$, then $\mathrm{CSP}(\mathcal{E})$ is reducible to $\mathrm{CSP}(\mathcal{D})$.*

Proof by example. Let R be an arbitrary ternary relation on a domain D. Consider the relations on D defined by

$$S(x, y) \text{ iff } (\exists z) R(x, y, z) \wedge R(y, y, x), \quad T(x, y) \text{ iff } R(x, x, x) \wedge (x = y) \ ,$$

where the existential quantification is understood over D. The relations S and T are defined by pp-formulae, therefore the constraint language $\mathcal{D} = \{R\}$ pp-defines the constraint language $\mathcal{E} = \{S, T\}$.

We sketch the reduction of CSP($\mathcal{E}$) to CSP($\mathcal{D}$) using the instance

$$S(x_3, x_2),\ T(x_1, x_4),\ S(x_2, x_4)\ .$$

We first replace S and T with their pp-definitions by introducing a new variable for each quantified variable:

$$R(x_3, x_2, y_1),\ R(x_2, x_2, x_3),\quad R(x_1, x_1, x_1),\ x_1 = x_4,\quad R(x_2, x_4, y_2),\ R(x_4, x_4, x_2)$$

and then we get rid of the equality constraint $x_1 = x_4$ by identifying these variables. This way we obtain an instance of CSP($\mathcal{D}$):

$$R(x_3, x_2, y_1),\ R(x_2, x_2, x_3),\ R(x_1, x_1, x_1),\ R(x_2, x_1, y_2),\ R(x_1, x_1, x_2)\ .$$

Clearly, the new instance of CSP($\mathcal{D}$) has a solution if and only if the original instance does. ◀

This simple theorem provides a quite powerful tool for comparing CSPs over different languages *on the same domain*. A more powerful tool, which can also be used to compare languages with different domains, is pp-interpretability. Informally, a constraint language $\mathcal{D}$ pp-interprets $\mathcal{E}$, if the domain of $\mathcal{E}$ is a pp-definable relation (from $\mathcal{D}$) modulo a pp-definable equivalence, and the relations of $\mathcal{E}$ (viewed, in a natural way, as relations on D) are also pp-definable from $\mathcal{D}$.[4] Formally:

▶ **Definition 13.** Let $\mathcal{D}, \mathcal{E}$ be constraint languages. We say that $\mathcal{D}$ *pp-interprets* $\mathcal{E}$ if there exist a natural number n, a set $F \subseteq D^n$, and an onto mapping $f : F \to E$ such that $\mathcal{D}$ pp-defines

- the relation F,
- the f-preimage of the equality relation on E, and
- the f-preimage of every relation in $\mathcal{E}$,

where by the f-preimage of a k-ary relation S on E we mean the nk-ary relation $f^{-1}(S)$ on D defined by

$$f^{-1}(S)(x_{11}, \ldots, x_{1k}, x_{21}, \ldots, x_{2k}, \ldots, x_{n1}, \ldots, x_{nk})$$

iff

$$S(f(x_{11}, \ldots, x_{n1}), \ldots, f(x_{1k}, \ldots, x_{nk}))\ .$$

When $F = E = D^n$ and f is the identity mapping, we also say that $\mathcal{E}$ is a *pp-power* of $\mathcal{D}$.

▶ **Theorem 14.** *If $\mathcal{D}$ pp-interprets $\mathcal{E}$, then* CSP($\mathcal{E}$) *is reducible to* CSP($\mathcal{D}$).

Proof sketch. The properties of the mapping f from Definition 13 allow us to rewrite an instance of CSP($\mathcal{E}$) to an instance of the CSP over a constraint language which is pp-definable from $\mathcal{D}$. Then we apply Theorem 12. ◀

[4] This is the classical notion of interpretation from model theory restricted to pp-formulas.

3.2 Homomorphic equivalence, cores and singleton expansions

Let $\mathcal{D}$ and $\mathcal{E}$ be constraint languages with domains D and E, respectively. We say that $\mathcal{D}$ and $\mathcal{E}$ are *homomorphically equivalent* if the relations in them can be ordered so that $\mathcal{D}$ and $\mathcal{E}$ become similar structures and there exist homomorphisms $e : \mathcal{D} \to \mathcal{E}$ and $g : \mathcal{E} \to \mathcal{D}$ (recall definitions from Section 2.4).

▶ **Theorem 15.** *Let $\mathcal{D}$ and $\mathcal{E}$ be homomorphically equivalent constraint languages. Then* $\mathrm{CSP}(\mathcal{D})$ *and* $\mathrm{CSP}(\mathcal{E})$ *are reducible to each other.*

Proof idea. An instance of $\mathrm{CSP}(\mathcal{D})$ has a solution if and only if the corresponding instance of $\mathrm{CSP}(\mathcal{E})$, obtained by replacing each $R_i \in \mathcal{D}$ with $S_i \in \mathcal{E}$, has a solution. Specifically, direct applications of mappings e and g transform solutions of one instance to solutions of another one. ◀

A mapping $f : D \to D$ is called an *endomorphism* of $\mathcal{D}$ if it is a homomorphism from $\mathcal{D}$ to itself, that is, $f(R) := \{(f(a_1), f(a_2), \dots) \mid (a_1, a_2, \dots) \in R\} \subseteq R$ for every $R \in \mathcal{D}$.

A language $\mathcal{D}$ is a *core* if every endomorphism of $\mathcal{D}$ is a bijection. It is not hard to show that if f is an endomorphism of a constraint language $\mathcal{D}$ with minimal range, then $f(\mathcal{D}) = \{f(R) \mid R \in \mathcal{D}\}$ is a core. Moreover, this core is unique up to isomorphism, therefore we speak about *the core* of $\mathcal{D}$.

An important fact is that we can add all singleton unary relations to a core constraint language without increasing the complexity of its CSP. For a constraint language $\mathcal{D}$, its *singleton expansion* is the language $\mathcal{E} = \mathcal{D} \cup \{C_a : a \in D\}$, where C_a denotes the unary relation $C_a = \{a\}$.

▶ **Theorem 16.** *Let $\mathcal{D}$ be a core constraint language and let $\mathcal{E}$ be the singleton expansion of $\mathcal{D}$. Then* $\mathrm{CSP}(\mathcal{E})$ *is reducible to* $\mathrm{CSP}(\mathcal{D})$.

Proof idea. The crucial step is to observe that the set of endomorphisms of $\mathcal{D}$, viewed as a $|D|$-ary relation, is pp-definable from $\mathcal{D}$. More precisely, the relation

$$S = \{(f(a_1), \dots, f(a_n)) : f \text{ is an endomorphism of } \mathcal{D}\} \ ,$$

where $a_1, \dots, a_n$ is a list of all elements of D, is pp-definable from $\mathcal{D}$ (even without existential quantification). Indeed, f is, by definition, an endomorphism of $\mathcal{D}$ if for every $R \in \mathcal{D}$ of arity $\mathrm{ar}(R)$ and every $(b_1, \dots, b_{\mathrm{ar}(R)}) \in R$ we have $(f(b_1), \dots, f(b_{\mathrm{ar}(R)})) \in R$. This directly leads to a pp-definition of S:

$$S(x_{a_1}, \dots, x_{a_n}) \text{ iff } \bigwedge_{R \in \mathcal{D}} \bigwedge_{(b_1, \dots, b_{\mathrm{ar}(R)}) \in R} R(x_{b_1}, \dots, x_{b_{\mathrm{ar}(R)}}) \ .$$

Given an instance of $\mathrm{CSP}(\mathcal{E})$ we introduce new variables $x_{a_1}, \dots, x_{a_n}$, we replace every constraint of the form $C_a(x)$ by $x = x_a$, and we add the constraint $S(x_{a_1}, \dots, x_{a_n})$. In this way we obtain an instance of $\mathrm{CSP}(\mathcal{D} \cup \{=\})$. Clearly, if the original instance has a solution, then the new instance has a solution as well. In the other direction, if g is a solution to the new instance, then its values on $x_{a_1}, \dots, x_{a_n}$ determine an endomorphism f of $\mathcal{D}$. As $\mathcal{D}$ is a core, f is a bijection, thus f^{-1} is an endomorphism as well, and $f^{-1} \circ g$ restricted to the original variables is a solution of the original instance. ◀

We will call constraint languages containing all singleton unary relations *idempotent*. Note that an idempotent constraint language is automatically a core as the only endomorphism is the mapping that sends each element to itself.

An interesting property of an idempotent constraint language $\mathcal{D}$ is that the search problem for CSP($\mathcal{D}$) is solvable in polynomial time whenever CSP($\mathcal{D}$) is. The idea is to use self-reduction: for a satisfiable instance, find good values for variables, in order, by checking satisfiability of the instance enhanced with appropriate unary singleton constraints.

3.3 Example

▶ **Example 17.** We show that 3-SAT is reducible to 3-COLORING via singleton expansion and a pp-interpretation with $n = 1$.

Recall the constraint language $\mathcal{D}_{3\text{COLOR}} = \{\neq_3\}$ of 3-COLORING from Example 6 and the constraint language $\mathcal{D}_{3\text{SAT}} = \{S_{000}, \dots, S_{111}\}$ of 3-SAT from Example 3.

Since $\mathcal{D}_{3\text{COLOR}}$ is a core, CSP($\mathcal{D}'_{3\text{COLOR}}$), where $\mathcal{D}'_{3\text{COLOR}} = \{\neq_3, C_0, C_1, C_2\}$, is reducible to CSP($\mathcal{D}_{3\text{COLOR}}$) by Theorem 16. By Theorem 14, it is now enough to show that $\mathcal{D}'_{3\text{COLOR}}$ pp-interprets $\mathcal{D}_{3\text{SAT}}$. We give a pp-interpretation with $n = 1$, $F = \{0, 1\}$, and f the identity map (see Definition 13). The unary relation $\{0, 1\}$ can be pp-defined by

$$F(x) \text{ iff } (\exists y)\ C_2(y) \wedge x \neq_3 y \quad (\text{iff } x \neq 2)\ .$$

The preimage of the equality relation is the equality relation on $\{0, 1\}$ which is clearly pp-definable. The relation S_{000} can be defined by

$$S_{000}(x_1, x_2, x_3) \text{ iff } (\exists y_1, y_2, y_3, z)\ C_2(z) \wedge y_1 \neq_3 y_2 \wedge y_2 \neq_3 y_3 \wedge y_1 \neq_3 y_3 \wedge \bigwedge_{i=1,2,3} z \neq_3 x_i \wedge T(x_i, y_i)\ ,$$

where T is the binary relation

$$T(x, y) \text{ iff } (\exists u, v)\ C_1(u) \wedge u \neq v \wedge x \neq v \wedge y \neq v$$

The other relations S_{ijk} are defined similarly.

While it is easy to verify that the presented pp-definitions work, it is not so easy to find them without any tools. The proof of Theorem 30 gives an algorithm to produce pp-definitions whenever they exist (although the obtained definitions will usually be very long).

3.4 Pp-constructibility

We now discuss how the reductions from the previous two subsections can be combined.

▶ **Definition 18.** A constraint language $\mathcal{D}$ *pp-constructs* a constraint language $\mathcal{E}$ if there is a sequence of constraint languages $\mathcal{D} = \mathcal{C}_1, \dots, \mathcal{C}_k = \mathcal{E}$ such that, for each $1 \leq i < k$

- $\mathcal{C}_i$ pp-interprets $\mathcal{C}_{i+1}$, or
- $\mathcal{C}_i$ is homomorphically equivalent to $\mathcal{C}_{i+1}$, or
- $\mathcal{C}_i$ is a core and $\mathcal{C}_{i+1}$ its singleton expansion.

The following is a corollary of Theorems 14, 15, and 16.

▶ **Corollary 19.** *If $\mathcal{D}$ pp-constructs $\mathcal{E}$, then* CSP($\mathcal{E}$) *is reducible to* CSP($\mathcal{D}$).

It turns out that any finite sequence of operations in pp-constructibility can be replaced by only two operations. Recall the notion of pp-power from Definition 13.

▶ **Theorem 20.** *A constraint language $\mathcal{D}$ pp-constructs a constraint language $\mathcal{E}$ if and only if $\mathcal{E}$ is homomorphically equivalent to a pp-power of $\mathcal{D}$.*

An example of idempotent constraint languages $\mathcal{D}$ and $\mathcal{E}$ such that D pp-constructs $\mathcal{E}$, but does not pp-interpret $\mathcal{E}$, can be found in [20].

3.5 Tractability conjecture

Pp-constructibility is a reflexive and transitive relation on the class of constraint languages. By identifying equivalent languages, i.e. languages which mutually pp-construct each other, we get a partially ordered set, the *pp-constructibility poset*, in which $\mathcal{D} \leq \mathcal{E}$ iff $\mathcal{D}$ pp-constructs $\mathcal{E}$. Corollary 19 then says that the "higher" we are in the poset the "easier" the CSP we are dealing with. 3-SAT is terribly hard – we will see later (see Example 31) that its constraint language is the least element of this poset. Strikingly, all known NP-complete CSPs have this property. Bulatov, Jeavons and Krokhin [34] conjectured that this is not a coincidence.

▶ Conjecture 2 (Tractability conjecture). If a constraint language $\mathcal{D}$ does not pp-construct the language of 3-SAT, then $\mathrm{CSP}(\mathcal{D})$ is solvable in polynomial time.

This conjecture (together with the matching hardness result) is also known as the *algebraic dichotomy conjecture* because many equivalent formulations, including the original one, are stated in terms of algebraic operations; see subsection 4.4.

Actually, the original conjecture in [34] was stated (in an equivalent algebraic form) for the case when $\mathcal{D}$ is an idempotent language and instead of pp-construction it used what was essentially pp-interpretation with $n = 1$, but this is equivalent to the conjecture stated above (see [33, 20]).

Remarkably, the seminal paper of Feder and Vardi included a conjecture very similar to the Tractability conjecture; see [65, Conjecture 2]. In essence, their conjecture was that $\mathrm{CSP}(\mathcal{D})$ should be solvable in polynomial time provided the core of $\mathcal{D}$ does not pp-define a constraint language whose core is isomorphic to the language of 1-in-3-SAT (see Example 4). This conjecture as stated is false. Indeed, the language $\mathcal{D}_{3\mathrm{COLOR}} = \{\neq_3\}$ of 3-COLORING is a core, it is invariant under under all permutations of $\{0, 1, 2\}$, and it is easy to see that all relations pp-definable in $\mathcal{D}_{3\mathrm{COLOR}}$ also have this invariance property. Therefore, $\mathcal{D}_{3\mathrm{COLOR}}$ cannot pp-define any relation whose core has a two-element domain, and yet $\mathrm{CSP}(\mathcal{D}_{3\mathrm{COLOR}})$ is obviously NP-complete. Note that we showed in Example 17 that the singleton expansion of $\mathcal{D}_{3\mathrm{COLOR}}$ can pp-interpret (with $n = 1$) the language of 3-SAT. In fact, it follows (see Example 31) that $\mathcal{D}_{3\mathrm{COLOR}}$ pp-constructs all constraint languages.

Similar hardness results and conjectures have been formulated for other computational/-descriptive complexity classes. See subsection 4.4.

3.6 Other reductions

Recall that if two constraint languages pp-construct each other, then their corresponding CSP problems are equivalent up to logspace reductions. Thus to understand the complexity of CSPs (for example, to resolve the Tractability conjecture), it suffices to consider just one constraint language from each equivalence class in the pp-constructibility poset.

By combining Theorems 15 and 16, we obtain the "reduction to the idempotent case."

▶ **Theorem 21.** *For every constraint language $\mathcal{D}$ there is an idempotent constraint language $\mathcal{D}'$ such that $\mathcal{D}$ and $\mathcal{D}'$ pp-construct each other.*

The following theorem has appeared in various closely related forms in the literature. It is useful because it allows one to work only with binary constraints, which often simplifies the design and analysis of algorithms for CSPs.

▶ **Theorem 22.** *For any constraint language $\mathcal{D}$, there is a constraint language $\mathcal{D}'$ such that*

- *all relations in $\mathcal{D}'$ are at most binary, and*
- *$\mathcal{D}$ and $\mathcal{D}'$ pp-construct each other.*

Proof sketch. Let ℓ be the maximum arity of a relation in $\mathcal{D}$. Define a constraint language $\mathcal{D}'$ as follows. Let $D' = D^\ell$. For each relation R (say of arity k) in $\mathcal{D}$, $\mathcal{D}'$ contains a unary relation R' such that $(a_1, \ldots, a_\ell) \in R'$ if and only if $(a_1, \ldots, a_k) \in R$. In addition, for all $1 \leq k \leq \ell$, $\mathcal{D}'$ contains a binary relation E_k defined as follows: $((a_1, \ldots, a_\ell), (b_1, \ldots, b_\ell)) \in E_k$ if and only if $a_1 = b_k$. It can be seen directly from definitions that $\mathcal{D}'$ is a pp-power of $\mathcal{D}$, and so $\mathcal{D}$ pp-constructs $\mathcal{D}'$.

In the opposite direction, for each unary relation $R' \in \mathcal{D}'$, consider the following relation on D': $R''(x_1, \ldots, x_k) = \exists x_R : R'(x_R) \wedge E_1(x_1, x_R) \wedge \ldots \wedge E_k(x_k, x_R)$. Let $\mathcal{D}'' = \{R'' \mid R \in \mathcal{D}\}$, so $\mathcal{D}'$ pp-defines $\mathcal{D}''$. Now, it is straightforward to check that $\mathcal{D}$ and $\mathcal{D}''$ are homomorphically equivalent, with mappings $e : D \to D^\ell$ and $g : D^\ell \to D$ defined as $e(x) = (x, \ldots, x)$ and $g((x_1, \ldots, x_\ell)) = x_1$. ◀

By using Theorem 21, Theorem 22 can be strengthened to make $\mathcal{D}'$ idempotent.

At the expense of forgoing pp-constructible equivalence, we can replace any constraint language with a language consisting of a single binary relation. The following result is essentially from [65]; see also the improvement in [45].

▶ **Theorem 23.** *For every constraint language $\mathcal{D}$ there is a digraph $H = (V, E)$ such that*

1. *$\{E\}$ pp-constructs $\mathcal{D}$, and*
2. *$\mathrm{CSP}(\{E\})$ is logspace-reducible to (and hence equivalent to) $\mathrm{CSP}(\mathcal{D})$.*

It is known [83] that the previous theorem cannot be improved so that $\mathcal{D}$ and $\{E\}$ each pp-construct the other.

It follows from the previous theorem that every $\mathrm{CSP}(\mathcal{D})$ is logspace-equivalent to an H-coloring problem. In a similar vein, Feder and Vardi proved [65] that every $\mathrm{CSP}(\mathcal{D})$ is logspace-equivalent to some $\mathrm{CSP}(\mathcal{E})$ where $\mathcal{E}$ is the singleton expansion of a partial ordering, and also to $\mathrm{CSP}(\mathcal{E}')$ where $\mathcal{E}'$ is the singleton expansion of the (symmetric irreflexive) edge relation of a bipartite graph. These results, while undoubtedly interesting, might be taken to suggest that the CSP classification problem needs to be tackled through a careful analysis of combinatorial objects such as digraphs, posets, or bipartite graphs. However, another interpretation is that these objects are complex enough to encode all CSPs. The algebraic approach, that we are about to describe, gives a better, more fruitful, alternative for complexity analysis of CSPs.

4 Polymorphisms as classifiers of constraint languages

4.1 Definitions and examples

The link between relations and operations is provided by a natural notion of compatibility. An n-ary operation f on a finite set D (that is, a mapping $f : D^n \to D$) is *compatible* with a k-ary relation $R \subseteq D^k$ if f applied component-wise to any n-tuple of elements of R gives an element of R. In more detail, whenever (a_{ij}) is an $n \times k$ matrix such that every row is in R, then f applied to the columns gives a k-tuple which is in R as well. If f is compatible with R then one also says that f is a *polymorphism* of R, and that R is *invariant* under f.

▶ **Example 24.** Consider the ternary *majority* operation f on $\{0, 1\}$, which always returns the (unique) repeated value among its arguments. It is a very easy exercise to check that this operation is compatible with any binary relation on $\{0, 1\}$: indeed, applying f to the columns of any 3×2 matrix with $0/1$ entries always gives one of the rows of this matrix.

We say that an operation f on D is a *polymorphism* of a constraint language $\mathcal{D}$ if f is compatible with every relation in $\mathcal{D}$. Note that a unary polymorphism is the same as an endomorphism. Endomorphisms can be thought of as symmetries, so polymorphisms can be viewed as symmetries of higher arities.

▶ **Example 25.** Every polymorphism f of an idempotent constraint language is *algebraically idempotent*, that is, it satisfies $f(a, a, \dots, a) = a$ for each a in the domain.

▶ **Example 26.** It is very easy to check that the binary operation $f(x, y) = \min(x, y)$ on $\{0, 1\}$ is a polymorphism of $\mathcal{D}_{\mathrm{HornSAT}}$ from Example 5. For example, to see that f is compatible with the relation $S_{110} = \{0, 1\}^3 \setminus \{(1, 1, 0)\}$, notice that, for any 2×3 matrix with 0/1 entries, if f applied to the columns of the matrix gives tuple $(1, 1, 0)$ then one of the rows of the matrix must be this same tuple.

▶ **Example 27.** The ternary operation $f(x, y, z) = x - y + z \mod p$ on $\mathrm{GF}(p)$ is a polymorphism of $\mathcal{D}_{\mathrm{3LIN_p}}$ from Example 8. Indeed, each relation in this structure is an affine subspace of $\mathrm{GF}(p)^3$ and f applied to the columns of a 3×3 matrix gives a triple, which is an affine combination of its rows (with coefficients 1, -1, 1).

In fact, it is easy to check that f is compatible with $R \subseteq \mathrm{GF}(p)$ if and only if it is an affine subspace of $\mathrm{GF}(p)$. It follows that if f is a polymorphism of $\mathcal{D}$, then an instance of $\mathrm{CSP}(\mathcal{D})$ can be written as a system of linear equations over $\mathrm{GF}(p)$ and therefore $\mathrm{CSP}(\mathcal{D})$ is solvable in polynomial time, for example, by Gaussian elimination.

Observe that for $p = 2$, f is the ternary *minority* function $f(x, y, z) = x + y + z \mod 2$.

▶ **Example 28.** Consider the following generalisation of the operation f from Example 24. For any finite set D, the *dual discriminator* operation on D is the ternary operation d such that $d(x, y, z) = x$ if x, y, z are all different and, otherwise, $d(x, y, z)$ is the repeated value among x, y, z. It is a useful easy exercise to check that d is compatible with a binary relation R on D if and only if the relation has one of the following forms:

($\vee$) $x = a \vee y = b$ for $a, b \in D$,

(π) $x = \pi(y)$ where π is a permutation on D,

($\times$) $A \times B$ where A and B are subsets of D,

($\cap$) intersection of a relation of type ($\vee$) or (π) with a relation of type ($\times$).

The key observation is that, for any binary relation R compatible with d, and for any $a, a', b, b', c \in D$ with $b \neq b'$, we have $d((a', c)(a, b), (a, b')) = (a, c)$. We remark that, generally, it is rare to have such an explicit description of relations having a given polymorphism.

Another useful way of viewing polymorphisms is that they provide an algebraic structure on solution sets of instances. In other words, they provide a uniform way to combine solutions to instances to form a new solution, which we illustrate with the following example.

▶ **Example 29.** Recall the majority operation f on $\{0, 1\}$ from Example 24. Consider any 2-SAT instance, for example, this one: $(x \vee \overline{y}) \wedge (y \vee \overline{z}) \wedge (y \vee \overline{u}) \wedge (x \vee u)$. Take any three solutions to this instance: for example, $\mathbf{a}, \mathbf{b}, \mathbf{c}$ described in the diagram below. It is easy to check that they are indeed solutions: simply check that each of them satisfies each constraint in the instance. If we apply f to these solutions coordinate-wise, as described in the diagram, we obtain a new assignment $f(\mathbf{a}, \mathbf{b}, \mathbf{c})$. It is also a solution to this instance, and there is no need to go through the constraints in the instances to check that each constraint is satisfied, since this is directly guaranteed by the fact that f is compatible with all binary relations on

$\{0,1\}$.

$$\begin{array}{rcccccc} & & x & y & z & u & & \\ \mathbf{a} = & (& 1 & 1 & 1 & 0 &) & \text{sat} \\ \mathbf{b} = & (& 1 & 1 & 0 & 1 &) & \text{sat} \\ \mathbf{c} = & (& 1 & 0 & 0 & 0 &) & \text{sat} \\ & & f\downarrow & f\downarrow & f\downarrow & f\downarrow & & \\ f(\mathbf{a},\mathbf{b},\mathbf{c}) = & (& 1 & 1 & 0 & 0 &) & \text{sat} \end{array}$$

Notice that f cannot be used to combine solutions to HORN-3-SAT or 3-SAT instances. Indeed, the relation $S_{111} = \{0,1\}^3 \setminus \{(1,1,1)\}$ is not compatible with f; it is easy to see that applying f to tuples $(0,1,1),(1,0,1),(1,1,0) \in S_{111}$ gives $(1,1,1)$. We discuss polymorphisms of $\mathcal{D}_{3\mathrm{SAT}}$ again in Example 31.

We remark that many algorithms that we discuss in the subsequent sections use polymorphisms, in design or in analysis, to combine solutions of subinstances of a given CSP instance in order to maintain or improve useful problem-specific properties of such solutions.

4.2 Polymorphisms as an algebraic counterpart of pp-definability

The set of all polymorphisms of $\mathcal{D}$ will be denoted by $\mathbf{D}$. This algebraic object has the following two properties.

- $\mathbf{D}$ *contains all projections*[5], that is, all operations of the form
$$\pi_i^n(a_1,\ldots,a_n) = a_i.$$
- $\mathbf{D}$ *is closed under composition*, that is, any operation built from operations in $\mathbf{D}$ by composition also belongs to $\mathbf{D}$.

For example, if $\mathbf{D}$ contains a unary operation h, a binary operation g and a ternary operation f then the operation $f'(x,y,z) = f(f(x,x,h(y)),g(y,z),g(z,x))$ is also in $\mathbf{D}$.

Sets of operations with these properties are called *concrete clones* (or *function clones*, or simply *clones*); therefore we refer to $\mathbf{D}$ as the *clone of polymorphisms* of $\mathcal{D}$ [6]. It is known that every concrete clone is the clone of polymorphisms of some (possibly infinite) constraint language [67, 26].

The notions of polymorphism and invariance form the basis of a well-known Galois correspondence between sets of relations and operations on a finite set [67, 26], which implies that the clone of polymorphisms controls pp-definability in the following sense.

▶ **Theorem 30.** *Let $\mathcal{D}$, $\mathcal{E}$ be constraint languages with $D = E$. Then $\mathcal{D}$ pp-defines $\mathcal{E}$ if and only if $\mathbf{D} \subseteq \mathbf{E}$.*

Proof sketch. The implication "⇒" follows directly from definitions. For the other implication it is enough to prove that if R is a relation compatible with every polymorphism of $\mathcal{D}$, then R is pp-definable from $\mathcal{D}$. A crucial step is a more general version of the observation made in the proof of Theorem 16: For any k, the set of k-ary polymorphisms of $\mathcal{D}$ can be viewed as a $|D|^k$-ary relation S on D, and this relation is pp-definable from $\mathcal{D}$. Now R can be defined from such a relation S (where k is the number of tuples in R) by existential quantification over suitable coordinates. This proof is illustrated in Example 32 below. ◀

[5] In some research communities such operations are called dictators.
[6] Similar fonts will be used to denote other languages and their corresponding clones, e.g. $\mathcal{E}$ and $\mathbf{E}$.

In view of this result, Theorem 12 says that the complexity of CSP($\mathcal{D}$) only depends on the clone $\mathbf{D}$. More precisely, if $\mathbf{D} \subseteq \mathbf{E}$, then CSP($\mathcal{E}$) is reducible to CSP($\mathcal{D}$). Moreover, the proof of Theorem 30 gives a generic pp-definition of $\mathcal{E}$ from $\mathcal{D}$, which gives us a generic reduction of CSP($\mathcal{E}$) to CSP($\mathcal{D}$).

▶ **Example 31.** It is a nice exercise to show that the language $\mathcal{D}_{3\mathrm{SAT}}$ of 3-SAT has no polymorphisms except for the projections, and the same holds for the language of 1-in-3-SAT. This means (by Theorem 30) that $\mathcal{D}_{3\mathrm{SAT}}$ pp-defines every constraint language with domain $\{0,1\}$. It follows (see also Theorem 36) that $\mathcal{D}_{3\mathrm{SAT}}$ pp-constructs (in fact, even pp-interprets) every constraint language, so it is the least element of the pp-constructibility poset, as claimed earlier. Moreover, it follows from Theorem 30 that $\mathcal{D}_{3\mathrm{SAT}}$ and $\mathcal{D}_{1\mathrm{in}3\mathrm{SAT}}$ pp-define each other, hence they are in the same (i.e. the least) element of the pp-constructibility poset.

▶ **Example 32.** Another nice exercise for the reader is to show that the language $\mathcal{D}'_{3\mathrm{COLOR}} = \{\neq_3, C_0, C_1, C_2\}$ on the domain $\{0,1,2\}$ (see Example 17) also does not have any polymorphisms except for projections. It follows that every relation on $\{0,1,2\}$ is pp-definable from $\mathcal{D}'_{3\mathrm{COLOR}}$. We show how the proof of Theorem 30 produces a pp-definition of some relation, say, the binary relation

$$R = \{(0,1),(0,2),(1,1),(2,2)\} \ .$$

Since R contains 4 pairs, we pp-define the 3^4-ary relation

$$S = \{(f(0,0,0,0), f(0,0,0,1), \dots, f(2,2,2,2)) : f \text{ is a 4-ary polymorphism of } \mathcal{D}'_{3\mathrm{COLOR}}\}.$$

which corresponds to the set of all 4-ary polymorphisms of $\mathcal{D}'_{3\mathrm{COLOR}}$:

$$S(x_{0000}, \dots, x_{2222}) \text{ iff } \bigwedge_i x_{iiii} = i \wedge \bigwedge_{i_1 \neq i_2, j_1 \neq j_2, k_1 \neq k_2, l_1 \neq l_2} x_{i_1 j_1 k_1 l_1} \neq_3 x_{i_2 j_2 k_2 l_2} \ .$$

Let's see that the above formula actually defines S: it is clear that indices show how to interpret each 3^4-tuple in S as a 4-ary operation on $\{0,1,2\}$. The first $\bigwedge$-part in the above formula states that the interpreted operation is compatible with C_0, C_1, C_2, while the second one states that it is compatible with $\neq_3$. Now we existentially quantify over all variables in S but x_{0012} and x_{1212} – the exceptions are those variables whose indices correspond to the first and the second (resp.) coordinates of pairs in R. The obtained binary relation $R'(x_{0012}, x_{1212})$ contains R since S contains the four tuples corresponding to the cases when f is a projection $\pi^4_i : \{0,1,2\}^4 \to \{0,1,2\}$, and R' is contained in R since R is compatible with every polymorphism of $\mathcal{D}'_{3\mathrm{COLOR}}$.

Note that the definition of S_{000} from Example 17 obtained in this way contains 3^7 variables.

For other examples similar to Example 32, but worked out in more detail, see [77].

The proof of Theorem 30 (see also the above example) gives an algorithm for constructing a pp-definition of a relation R from a given structure $\mathcal{D}$, whenever there is one. This pp-definition can be terribly long. However, the algorithm is optimal in the sense that the problem of deciding of whether such a pp-definition exists is co-NEXPTIME-hard [118].

4.3 Height-1 identities and pp-constructibility

We explained above that the complexity of CSP($\mathcal{D}$) depends only on the polymorphisms of $\mathcal{D}$. In this section, we refine this statement by specifying the properties of polymorphisms

that determine the complexity: these are *identities* satisfied by polymorphisms, i.e. equations that hold for all choices of values for the variables, and more specifically *height-1* identities, i.e. those in which each side has exactly one occurrence of an operation symbol.

We will explain identities by example.

▶ **Example 33.** A binary operation f on D is called a *semilattice* operation if it satisfies the following three *identities*

$$f(f(x,y),z) = f(x,f(y,z)),\ \ f(x,y) = f(y,x),\ \text{and}\ f(x,x) = x,$$

which means that the above equalities hold for all choices of values in D for the variables. The first identity above (known as associativity) is not height-1, as both sides have two occurrences of f. The second identity (commutativity) is height-1. The third identity (idempotence) is not height-1 as its right side has no occurrence of an operation symbol.

In general, identities can involve more than one operation, e.g. $f_3(x,x,y) = f_4(x,x,x,y)$. Note that height-1 identities involving operations in a clone can be expressed "within" the clone. For example, the identity $f_3(x,x,y) = f_4(x,x,x,y)$ is satisfied iff the statement $f_3(\pi^2_1,\pi^2_1,\pi^2_2) = f_4(\pi^2_1,\pi^2_1,\pi^2_1,\pi^2_2)$ is true in the clone.

▶ **Definition 34.** A mapping H from a clone $\mathbf{D}$ to a clone $\mathbf{E}$ is called an *h1 clone homomorphism* if

- it preserves the arities of operations,
- it preserves height-1 identities; that is,

$$H(g(\pi^k_{i_1},\dots,\pi^k_{i_n})) = H(g)(\pi^k_{i_1},\dots,\pi^k_{i_n})$$

where $g \in \mathbf{D}$ is n-ary.

Note that while an h1 clone homomorphism preserves height-1 identities, it is not required to preserve non-height-1 identities.

▶ **Example 35.** Assume that $\mathbf{D}$ contains a semilattice operation f and H is an h1 clone homomorphism from $\mathbf{D}$ to $\mathbf{E}$. Then the operation $H(f)$ on E is commutative, but not necessarily idempotent or associative. However, if $g = f(f(x,y),z)$ then $H(g)$ satisfies identities such as $H(g)(x,y,z) = H(g)(y,z,x) = H(g)(y,x,z)$ and $H(g)(x,x,y) = H(g)(x,y,y)$ because they are height-1 and g satisfies them.

The following is proved in [20].

▶ **Theorem 36.** *Let $\mathcal{D}$, $\mathcal{E}$ be constraint languages. Then $\mathcal{D}$ pp-constructs $\mathcal{E}$ if and only if there exists an h1 clone homomorphism from $\mathbf{D}$ to $\mathbf{E}$.*

Thus if $\mathcal{D} \leq \mathcal{E}$ in the pp-constructibility ordering (i.e. $\mathcal{D}$ pp-constructs $\mathcal{E}$), then $\mathbf{E}$ satisfies all properties of the form "there exist operations $f_1,\dots,f_n$ satisfying height-1 identities $\Lambda_1,\dots,\Lambda_k$" which are satisfied by $\mathbf{D}$.[7] A compactness argument shows that the converse is also true: if $\mathcal{D} \not\leq \mathcal{E}$, then there is a finite system of height-1 identities which is satisfied by some operations from $\mathbf{D}$ but is not satisfied by any operations in $\mathbf{E}$. Since the position of $\mathcal{D}$ in the pp-constructibility ordering determines the complexity of CSP($\mathcal{D}$) up to logspace reductions, we have that the following holds *unconditionally*:

[7] Algebraists call such properties *strong (height-1) Mal'tsev conditions*.

The complexity of $\mathrm{CSP}(\mathcal{D})$, up to logspace reductions, depends only on the finite systems of height-1 identities satisfied by the operations in the clone of polymorphisms of $\mathcal{D}$.

More is true. Suppose C is a set of constraint languages with the property that if $\mathcal{D} \in C$ and $\mathrm{CSP}(\mathcal{E})$ is logspace reducible to $\mathrm{CSP}(\mathcal{D})$, then $\mathcal{E} \in C$. (For example, C could be the class of $\mathcal{D}$ for which $\mathrm{CSP}(\mathcal{D})$ is in P.) Then C is an upward-closed subset of the pp-constructibility partial order. Assume that C is not the set of all constraint languages. It follows that

1. There exist $\mathcal{E}_1, \mathcal{E}_2, \ldots$ such that $C = \{\mathcal{D} : \mathcal{D} \not\leq \mathcal{E}_i \text{ for all } i\}$, and hence
2. C is the set of constraint languages $\mathcal{D}$ such that for every i there exists a finite system of height-1 identities which is not satisfied by $\mathbf{E}_i$ but is satisfied by $\mathbf{D}$.

In other words, C can be characterized by a set of "forbidden" constraint languages (with respect to pp-constructibility), and also by a (possibly infinite) disjunction of (possibly infinite) systems of height-1 identities on polymorphisms. One advantage of this perspective is that it replaces a negative characterization of a class of interest (forbidden constraint languages) with a positive one (existence of polymorphisms satisfying height-1 identities). For this observation to be useful, however, the height-1 identities must be manageable, and the discovery of manageable height-1 identities characterizing classes of interest is one of the achievements of the algebraic method. We will illustrate this in the following subsection.

4.4 Classifications and conjectures

In this section we give examples in which classes of constraint languages are characterized both by forbidden constraint languages and by height-1 identities of polymorphisms.

For the sake of readability, our terminology will stray from longstanding traditions in the literature. Specifically, our definitions of *Taylor, weak NU, cyclic*, and *Siggers* operations do *not* require that the operation is idempotent. We will write *idempotent Taylor, idempotent weak NU*, etc. for what is ordinarily called *Taylor, weak NU,* etc.[8] (Note that we cannot bring ourselves to apply this convention to Mal'tsev, majority and NU operations.) It's easy to see that a language $\mathcal{D}$ has one of the polymorphisms mentioned in this section (e.g. Taylor) if and only if the core of $\mathcal{D}$ has an idempotent version of that polymorphism. Hence, even though we state all conjectures in this section without assuming idempotence, it is enough to prove them in the idempotent case.

▶ **Definition 37.** A *Taylor* operation is a k-ary ($k \geq 2$) operation f such that, for each $1 \leq i \leq k$, f satisfies an identity of the form

$$f(z_{i,1}, \ldots, z_{i,i-1}, \underset{i}{x}, z_{i,i+1}, \ldots, z_{i,k}) = f(z'_{i,1}, \ldots, z'_{i,i-1}, \underset{i}{y}, z'_{i,i+1}, \ldots, z'_{i,k}) \tag{2}$$

where all $z_{i,j}, z'_{i,j}$ are in $\{x, y\}$.

A useful way to view this definition is that the identities (2) prevent f from being the ith projection (for each i) whenever the domain has more than one element. In fact, it is easy to see that Taylor identities are the weakest height-1 identities involving a single operation that prevent this operation from being a projection.

The following theorem can be derived from [116] (see also [34]).

▶ **Theorem 38.** *For any constraint language $\mathcal{D}$, the following are equivalent:*

[8] [50] uses *quasi-Taylor, quasi-WNU* etc. for the not-necessarily-idempotent versions.

1. *$\mathcal{D}$ does not pp-construct the language of* 3-SAT.
2. *$\mathcal{D}$ has a Taylor polymorphism of some arity.*

The class of constraint languages which do not pp-construct the language of 3-SAT is precisely the class of $\mathcal{D}$ for which the Tractability conjecture asserts that CSP($\mathcal{D}$) is in P. Hence constraint languages with Taylor polymorphisms are of particular interest. Although the defining condition of a Taylor operation looks rather cumbersome, it has been shown that, rather surprisingly, for any constraint language $\mathcal{D}$, the property of having a Taylor polymorphism is equivalent to a number of simpler conditions, as described below.

- A *weak near-unanimity (WNU)* operation is a k-ary ($k \geq 2$) operation f satisfying the identities

$$f(y, x, x, \ldots, x, x) = f(x, y, x, \ldots, x, x) = \ldots = f(x, x, x, \ldots, x, y); \tag{3}$$

- A *cyclic* operation is a k-ary ($k \geq 2$) operation f satisfying the identity

$$f(x_1, x_2, \ldots, x_k) = f(x_2, \ldots, x_k, x_1); \tag{4}$$

- A *Siggers* operation is a 4-ary operation f satisfying the identity [9]

$$f(y, x, y, z) = f(x, y, z, x). \tag{5}$$

▶ **Theorem 39.** *For every constraint language $\mathcal{D}$, the following are equivalent:*

1. *$\mathcal{D}$ has a Taylor polymorphism;*
2. *$\mathcal{D}$ has a WNU polymorphism [104];*
3. *$\mathcal{D}$ has a cyclic polymorphism [13];*
4. *$\mathcal{D}$ has a Siggers polymorphism [85, 113].*

For other conditions equivalent to the presence of an (idempotent) Taylor polymorphism, see [18, 33, 34, 73, 94].

▶ **Corollary 40.** *If a constraint language $\mathcal{D}$ has no Taylor (equivalently, no WNU, cyclic, or Siggers) polymorphism then* CSP($\mathcal{D}$) *is NP-complete.*

Using the above results, the Tractability conjecture (Conjecture 2), combined with Corollary 40, can be re-stated as follows.

▶ Conjecture 3 (Algebraic dichotomy conjecture). If a constraint language $\mathcal{D}$ has a Taylor (equivalently, a WNU, cyclic, or Siggers) polymorphism, then CSP($\mathcal{D}$) is solvable in polynomial time; otherwise, it is NP-complete.

More informally, the (open part of the) Algebraic Dichotomy Conjecture says:

If $\mathcal{D}$ has a nontrivial higher-dimensional symmetry (hence providing a nontrivial way to combine solutions), then CSP($\mathcal{D}$) **should be tractable.**

We remark that Conjecture 3 is often misquoted as "CSP($\mathcal{D}$) is NP-complete if all polymorphisms of $\mathcal{D}$ are projections, and CSP($\mathcal{D}$) is tractable otherwise." This form of the conjecture is false, as the following example shows.

[9] Using different variables, $f(r, a, r, e) = f(a, r, e, a)$ – mnemonic due to Ryan O'Donnell.

▶ **Example 41.** Consider the map $f : \{0,1,2\} \to \{0,1\}$ such that $f(0) = 0$ and $f(1) = f(2) = 1$ and consider the constraint language $\mathcal{D} = \{f^{-1}(S_{ijk}) \mid S_{ijk} \in \mathcal{D}_{3\mathrm{SAT}}\}$ on $\{0,1,2\}$. Note that any relation in $\mathcal{D}$ cannot distinguish elements 1 and 2, in the sense that any 1 anywhere in it can be replaced by 2 and vice versa. Hence, if we take any projection π_i^n on $\{0,1,2\}$ and any operation $f : \{1,2\}^n \to \{1,2\}$ and define f' on $\{0,1,2\}$ so that

$$f'(\mathbf{t}) = \begin{cases} f(\mathbf{t}), & \text{if } \mathbf{t} \in \{1,2\}^n \\ \pi_i^n(\mathbf{t}), & \text{otherwise} \end{cases}$$

then f' is a polymorphism of $\mathcal{D}$. It is clear that CSP($\mathcal{D}$) is NP-complete and has many polymorphisms other than projections. One can even take the singleton expansion of $\mathcal{D}$ - it would be a core, and each operation f' built above would remain a polymorphism provided f is idempotent. However, all height-1 identities satisfied by such operations must also be satisfied by projections.

▶ **Example 42.** We show how to apply cyclic operations to prove the dichotomy theorem for undirected graphs [71].

Let R be a symmetric binary relation viewed as an undirected graph and let $\mathcal{D} = \{R\}$. If R contains a loop then CSP($\mathcal{D}$) is trivially tractable. If R is bipartite, then the core of $\mathcal{D}$ is an edge and CSP($\mathcal{D}$) is essentially 2-COLORING, which is tractable.

Let $\mathcal{D}' = \{R', ...\}$ be the singleton expansion of the core of $\mathcal{D}$. If R is not bipartite and does not contain a loop, then R' does not contain a loop, but does contain a closed walk $a_1, a_2, \ldots, a_p, a_1$ for some prime $p > |D'|$. It was shown in [13] that if $\mathbf{D}'$ contains some idempotent cyclic operation then it contains such an operation of every prime arity greater than $|D'|$, so we can assume that $\mathbf{D}'$ contains a cyclic operation t of arity p. Since t is a polymorphism, the pair

$$t((a_1,a_2),\ldots,(a_{p-1},a_p),(a_p,a_1)) = (t(a_1,\ldots,a_p), t(a_2,\ldots,a_p,a_1))$$

is in R', but it is a loop since t is cyclic. This contradiction shows that $\mathbf{D}'$ does not contain an idempotent cyclic operation of arity p, and hence it does not have an idempotent cyclic operation of any arity. Because $\mathcal{D}'$ is idempotent, $\mathbf{D}'$ has no cyclic operation. $\mathcal{D}$ pp-constructs $\mathcal{D}'$; hence $\mathcal{D}$ has no cyclic operation (as the height-1 identity characterizing a cyclic operation of $\mathbf{D}$ would also be satisfied by some operation of $\mathbf{D}'$). Hence $\mathcal{D}$ does not have a cyclic polymorphism, so CSP($\mathcal{D}$) is NP-complete.

▶ **Example 43.** A digraph is called *smooth* if it contains neither sources nor sinks, i.e. all its vertices have positive in- and out-degrees. It was shown in [17] that if a smooth digraph has a WNU polymorphism then its core must be a disjoint union of directed cycles. If a smooth digraph H has such a form, it is an easy exercise to show that the corresponding H-COLORING problem CSP($\mathcal{D}_H$) is solvable in polynomial time. If H does not have such a form, then CSP($\mathcal{D}_H$) is NP-complete by Corollary 40.

Consider the digraph H having only 3 vertices x, y, z and 4 edges yx, xy, yz, zx. It is a smooth core digraph, so it follows from the above that the corresponding problem CSP($\mathcal{D}_H$) is NP-complete. It can be easily checked that it is the smallest digraph with this property. Moreover, this digraph was used in [85] to obtain identity (5) above, which can be viewed as follows: if a digraph with a Siggers polymorphism contains H as a subdigraph, then it also contains a loop.

Next, we consider the class of constraint languages $\mathcal{D}$ for which CSP($\mathcal{D}$) has *bounded width* (meaning that all unsatisfiable instances of CSP($\mathcal{D}$) can be refuted via local propagation).

The "obvious" obstructions to bounded width, besides 3-SAT, are 3-LIN(p) for primes p. Let C be the set of constraint languages which do not pp-construct the language of 3-LIN(p) for any prime p. A notable success of the algebraic method was the proof that C is precisely the set of constraint relations having bounded width [14, 31]. The original proof of Barto and Kozik [12] hinged on the equivalence of the first and second items in the following theorem.

▶ **Theorem 44.** *For any constraint language $\mathcal{D}$, the following are equivalent:*

1. *$\mathcal{D}$ does not pp-construct the language of* 3-LIN(p)*, for any p;*
2. *$\mathcal{D}$ has WNU polymorphisms of all but finitely many arities [99, 104];*
3. *$\mathcal{D}$ has a k-ary WNU polymorphism for each $k \geq 3$ (see [91]);*
4. *$\mathcal{D}$ has a ternary WNU polymorphism f_3 and a 4-ary WNU polymorphism f_4 such that $f_3(x,x,y) = f_4(x,x,x,y)$ [91];*
5. *for some $k \geq 3$, $\mathcal{D}$ has a k-ary polymorphism f satisfying, for each $1 \leq i \leq k$, an identity of the form*

$$f(z_{i,1},\ldots,z_{i,i-1},\underset{i}{x},z_{i,i+1},\ldots,z_{i,k}) = f(z_{i,1},\ldots,z_{i,i-1},\underset{i}{y},z'_{i,i+1},\ldots,z'_{i,k}) \tag{6}$$

where all $z_{i,j}, z'_{i,j}$ are in $\{x,y\}$ [91].

For other conditions equivalent to those from Theorem 44, see [18, 73, 82, 94].

We will discuss CSPs of bounded width in detail in Section 5.

As a third example, consider the class of constraint languages $\mathcal{D}$ for which CSP($\mathcal{D}$) has *bounded linear width*, meaning that all unsatisfiable instances of CSP($\mathcal{D}$) can be refuted via local propagation in a linear fashion (think refuting unsatisfiable 2-SAT instances by following paths). This refutation property can be formalised via Linear Datalog, among other equivalent ways, see [44]. Another way to express this property is that every instance CSP($\mathcal{D}$) can solved by forming, in logspace, a certain directed graph (of local inferences) and solving STCON on this digraph, see [44, 62]. This shows that problems CSP($\mathcal{D}$) of bounded linear width are in complexity class NL.

The obvious obstructions to bounded linear width are 3-LIN(p) for primes p and HORN-3-SAT. Let C' be the class of constraint languages which do not pp-construct the language of 3-LIN(p) for any prime p, nor the language of HORN-3-SAT. Thus C' contains the class of constraint languages whose CSP has bounded linear width (and possibly more). The following theorem gives a manageable algebraic characterization of C'.

▶ **Theorem 45.** *For any constraint language $\mathcal{D}$, the following are equivalent:*

1. *$\mathcal{D}$ pp-constructs neither the language of* 3-LIN(p)*, for any p, nor that of* HORN-3-SAT*;*
2. *for some $n \geq 2$, $\mathcal{D}$ has ternary polymorphisms $d_0,\ldots,d_n$ satisfying the following identities* [73]*:*

$$d_0(x,y,z) = d_0(x,x,x), \tag{7}$$
$$d_n(x,y,z) = d_n(z,z,z), \tag{8}$$
$$d_i(x,y,y) = d_{i+1}(x,y,y) \quad \text{and} \quad d_i(x,y,x) = d_{i+1}(x,y,x) \text{ if } i \text{ is even, } i < n, \tag{9}$$
$$d_i(x,x,y) = d_{i+1}(x,x,y) \text{ if } i \text{ is odd, } i < n. \tag{10}$$

3. *for some $k \geq 3$, $\mathcal{D}$ has a k-ary polymorphism f satisfying, for each $1 \leq i \leq k$, an identity of the form*

$$f(x,\ldots,x,\underset{i}{x},z_{i,i+1},\ldots,z_{i,k}) = f(x,\ldots,x,\underset{i}{y},z'_{i,i+1},\ldots,z'_{i,k}) \tag{11}$$

where all $z_{i,j}, z'_{i,j}$ are in $\{x,y\}$ [66].

It is known [91] that the above theorem cannot have an equivalent condition involving only a bounded number of functions of bounded arity (such as Siggers polymorphism or condition (4) from Theorem 44), so at least one of the number and the arity of operations involved in any characterization must be unbounded.

This theorem has an interesting complexity-theoretic consequence. HORN-3-SAT is P-complete (and thus unlikely to be in NL). The relationship of problems 3-LIN(p), and hence of classes $\mathrm{Mod}_p\mathrm{L}$, with the class NL is unknown (though there is evidence that NL is contained in $\mathrm{Mod}_p\mathrm{L}$ for every p [2, 110]). As mentioned above, bounded linear width guarantees membership in NL [52], and moreover, all problems CSP($\mathcal{D}$) known to be in NL have bounded linear width.

▶ **Corollary 46.** *Let $\mathcal{D}$ be a constraint language having polymorphisms satisfying one of the equivalent conditions in Theorem 44. If $\mathcal{D}$ has no ternary polymorphisms satisfying conditions (7)–(10), or equivalently, no polymorphism satisfying condition (11), of Theorem 45, then* CSP($\mathcal{D}$) *is P-complete and cannot have bounded linear width.*

As mentioned earlier, the class of constraint languages having polymorphisms witnessing the equivalent conditions of Theorem 45 contains the class of constraint languages whose CSP has bounded linear width. It was suggested in [97] that the two classes actually coincide.

▶ Conjecture 4 (Bounded Linear Width Conjecture). If a constraint language $\mathcal{D}$ has polymorphisms as described in one of the equivalent conditions in Theorem 45 then CSP($\mathcal{D}$) has bounded linear width and hence belongs to NL.

We discuss progress towards resolving Conjecture 4 in Section 5.7.

If for some reason you are interested in the class of constraint languages which do not pp-construct the language of HORN-3-SAT, but with no restriction on pp-constructing the language of any 3-LIN(p), then you are in luck; algebraic descriptions of this class are also known. For example:

▶ **Theorem 47.** [73] *For any constraint language $\mathcal{D}$, the following are equivalent:*

- *$\mathcal{D}$ does not pp-construct the language of* HORN-3-SAT.
- *for some $k \geq 3$, $\mathcal{D}$ has a k-ary polymorphism f satisfying, for each $1 \leq i \leq k$, an identity of the form*

$$f(x, \ldots, x, \underset{i}{x}, z_{i,i+1}, \ldots, z_{i,k}) = f(z'_{i,1}, \ldots, z'_{i,i-1}, \underset{i}{y}, z'_{i,i+1}, \ldots, z'_{i,k}) \tag{12}$$

where all $z_{i,j}, z'_{i,j}$ are in $\{x, y\}$.

Example 11 shows that the structure $\mathcal{D}_{\mathrm{HornSAT}}$ from Example 5 pp-defines the structure $\mathcal{D}_{\mathrm{STCON}}$ from Example 9. Thus, by forbidding $\mathcal{D}_{\mathrm{STCON}}$ instead of $\mathcal{D}_{\mathrm{HornSAT}}$ in Theorem 47, we further restrict the class of constraint languages, but again obtain a class with a known algebraic characterization.

▶ **Theorem 48.** [68] *For any constraint language $\mathcal{D}$, the following are equivalent:*

- *$\mathcal{D}$ does not pp-construct $\mathcal{D}_{STCON}$;*
- *for some $t \geq 2$, $\mathcal{D}$ has ternary polymorphisms $p_0, \ldots, p_t$ satisfying the following identities:*

$$p_0(x, y, z) = p_0(x, x, x), \tag{13}$$
$$p_t(x, y, z) = p_t(z, z, z), \tag{14}$$
$$p_i(x, x, y) = p_{i+1}(x, y, y) \quad \text{for all } i < t. \tag{15}$$

Here is one final example characterizing forbidden languages by height-1 identities.

▶ **Theorem 49.** [73] *For any constraint language $\mathcal{D}$, the following are equivalent:*

- *$\mathcal{D}$ pp-constructs neither the language of* 3-LIN(p)*, for any p, nor that of* STCON*;*
- *for some $n \geq 0$, $\mathcal{D}$ has 4-ary polymorphisms $f_0, \ldots, f_n$ satisfying the following identities:*

$$f_0(x,y,y,z) = f_0(x,x,x,x) \tag{16}$$
$$f_n(x,x,y,z) = f_n(z,z,z,z) \tag{17}$$
$$f_i(x,x,y,x) = f_{i+1}(x,x,y,x) \quad \text{and} \quad f_i(x,x,y,y) = f_{i+1}(x,x,y,y), \quad \text{for } i < n. \tag{18}$$

Some problems CSP($\mathcal{D}$) can be solved by forming a certain undirected graph (of local inferences) on a given instance and then solving STCON on this graph, see [62]. This property is called bounded symmetric width, and the corresponding CSPs belong to the complexity class L.

It was shown in [97] that any language not satisfying the conditions of Theorem 49 cannot have bounded symmetric width and that the corresponding CSP is hard for at least one of complexity classes NL and $\mathrm{Mod}_p\mathrm{L}$ (for some p).

▶ Conjecture 5 (Bounded Symmetric Width Conjecture). [97] If a constraint language $\mathcal{D}$ has polymorphisms as described in one of the equivalent conditions in Theorem 49 then CSP($\mathcal{D}$) has bounded symmetric width and hence belongs to L.

We discuss progress towards resolving Conjecture 5 in Section 5.7.

It is interesting to note that the complexity classifications obtained or conjectured through these algebraic results are obviously conditional on complexity-theoretic assumptions, while the classifications related to (various notions of) width are unconditional.

4.5 Taxonomy of systems of linear identities

Although the reader might find the identities in Theorems 38, 39, 44, 45, 47, 48 and 49 to be somewhat random, in fact it is something of a minor miracle that the classes of constraint languages considered in those theorems have manageable algebraic descriptions. The proofs of these theorems are highly nontrivial and use sophisticated tools from universal algebra. What made their discovery possible is the serendipitous fact that universal algebraists have been studying the connection between identities and "good algebraic structure" for almost 50 years.

In particular, the systems of identities appearing in Theorems 38, 39, 47, 48 and 49, as well as some equivalent characterizations of the classes described in Theorems 44 and 45, when strengthened by adding idempotency, have been known to be of fundamental importance since the 1980s in the context of "tame congruence theory" [73]. Height-1 identities and idempotency identities $f(x,x,\ldots,x) = x$ are both examples of *linear* identities; these are identities in which each side has *at most* one occurrence of an operation symbol.

In addition to these well-studied systems of linear identities arising in tame congruence theory, there is a robust taxonomy of "classical" systems of linear identities, all stronger than the Taylor identities. This allows one to approach classification problems, and especially their positive parts, for language-based CSP (such as Conjecture 3) as follows. First prove the result for stronger systems of identities, and then move to weaker identities, gradually approaching the identities which determine the (conjectured) boundary.

We now introduce some (more) of the more important linear identities that have played a role in the algebraic theory of CSP. Most of these identities have been studied in universal algebra before the link with the CSP was discovered.

- A *near-unanimity (NU)* operation is an n-ary ($n \geq 3$) operation f which satisfies

$$f(y,x,x,\dots,x,x) = f(x,y,x,\dots,x,x) = \dots = f(x,x,x,\dots,x,y) = x; \tag{19}$$

  The last equality in (19) is the difference between NU and WNU operations. A ternary NU operation is usually called a *majority* operation. The dual discriminator operation from Example 28 is a majority operation which was often used as a starting point in many CSP-related classifications.
  For example, the operation $f(x_1,\dots,x_n) = \bigvee_{i<j}(x_i \wedge x_j)$ on $\{0,1\}$ is an NU operation.
- A *symmetric* operation is an n-ary operation satisfying all identities of the form

$$f(x_1,x_2,\dots,x_n) = f(x_{\pi(1)},x_{\pi(2)},\dots x_{\pi(n)}), \tag{20}$$

  where π is a permutation of the set $\{1,2,\dots,n\}$.
- A *totally symmetric (TS)* operation is an n-ary operation satisfying all identities of the form

$$f(x_1,x_2,\dots,x_n) = f(y_1,y_2,\dots y_n), \tag{21}$$

  where $\{y_1,y_2,\dots,y_n\} = \{x_1,x_2,\dots,x_n\}$. This includes all identities of the form (20) as well as identities such as $f(x,x,y) = f(x,y,y)$.
  If $f_2(x_1,x_2)$ is a semilattice operation (Example 33) then, for all $n \geq 2$, $f_n(x_1,\dots,x_n) = f_2(x_1,f_2(x_2,(\dots f_2(x_{n-1},x_n))))$ is a TS operation.
- A *Mal'tsev* [10] operation is a ternary operation f satisfying the identities

$$f(x,x,y) = f(y,x,x) = y. \tag{22}$$

  If in addition it satisfies $f(x,y,x) = y$ then it is called a *minority* operation.
  A typical Mal'tsev operation is $f(x,y,z) = x - y + z$ where $(D,+)$ is an Abelian group (see Example 27).
- Ternary operations $d_0,d_1,\dots,d_n$ are *Jónsson* operations if they satisfy the identities of Theorem 45, as well as the identities

$$d_i(x,y,x) = x \quad \text{for all } i \leq n. \tag{23}$$

- Ternary operations $d_0,d_1,\dots,d_n,p$ are *Gumm* operations if $d_0,\dots,d_n$ satisfy all the identities of Jónsson operations *except* equation (8) from Theorem 45, as well as the identities

$$d_n(x,y,y) = p(x,y,y) \quad \text{and} \quad p(x,x,y) = y. \tag{24}$$

- For $k \geq 2$, an *k-edge* operation is a $(k+1)$-ary operation f satisfying identities

$$\begin{aligned} &f(x,x,y,y,y,\dots,y,y) = y \\ &f(x,y,x,y,y,\dots,y,y) = y \\ &f(y,y,y,x,y,\dots,y,y) = y \\ &f(y,y,y,y,x,\dots,y,y) = y \\ &\qquad\qquad\vdots \\ &f(y,y,y,y,y,\dots,y,x) = y \end{aligned} \tag{25}$$

[10] Sometimes spelled as Maltsev/Mal'cev/Malcev - correctly pronounced with the soft 'l' followed by 'ts'.

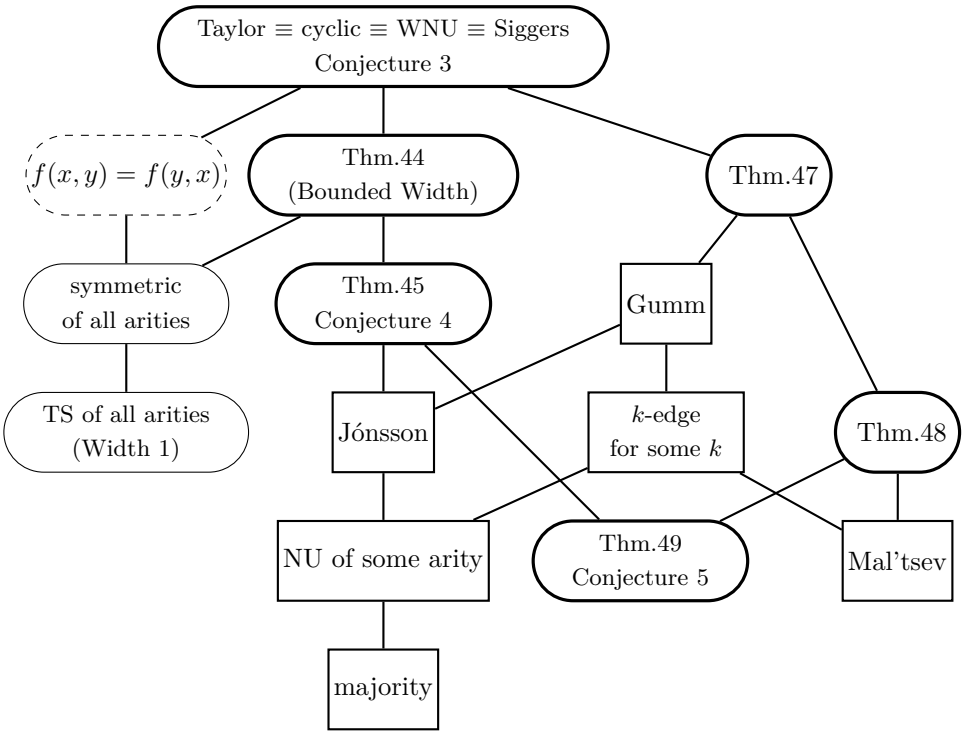

Figure 1 Taxonomy of important systems of linear identities. Upward paths in the diagram correspond to weakening conditions, i.e. increasing corresponding classes of constraint languages.

An easy way to parse these identities is this: these are Mal'tsev identities glued with the NU identities as follows. If f depends on the first three variables only, then these identities are the same as (22), but with the first two coordinates swapped (in particular, 2-edge is essentially permuted Mal'tsev). If f depends on all but the first two variables, then these identities are the same as (19), but with x and y swapped.

Figure 1 shows the relative strength of the mentioned linear identities on finite domains. The higher items correspond to weaker conditions, and hence to larger classes of constraint languages. More precisely, if a constraint language has polymorphisms witnessing one set of identities in the diagram, then these operations can be composed with themselves and projections to obtain operations satisfying the identities "higher" in the diagram. Such a composition is possible in any function clone on any (finite or infinite) domain. Each edge shown in the diagram is strict in the foregoing sense; however, Barto has shown that two of the edges collapse in the following weaker sense: if a constraint language (on a finite domain, with finitely many relations) has Jónsson polymorphisms then it must also have an NU polymorphism [8]. A similar statement holds for Gumm and edge polymorphisms [9].

By the convention we have followed in this paper, the systems of identities enclosed in rectangles are idempotent by definition, while the systems of identities circled in ovals are height-1 and hence are not assumed to be idempotent. The systems circled in thick ovals, when restricted to the idempotent case, correspond to robust tame congruence theoretic conditions.

We note that [76] contains a useful list (and a diagram) of many universal-algebraic conditions relevant for the CSP, though their list is oriented more (than ours) towards prominence of conditions in universal algebra.

Not all natural-looking linear identities make a good choice for attacks on the open conjectures. For example, commutativity (in the dashed oval in Figure 1) is one of the simplest of the Taylor-type identities, but approaching Conjecture 3 by looking at commutative (or commutative idempotent) polymorphisms is not (known to be) a great idea. On the other hand, finding a CSP algorithm for constraint languages satisfying the identities of Theorem 47 would likely be viewed as a more "natural" (and more feasible) step.

The problems of deciding whether a given constraint language has a given type of polymorphism(s) are sometimes called "meta-problems." See [50] for a survey and new results on the complexity of meta-problems.

5 Polymorphisms in algorithms I: Proving correctness

The most natural idea to decide an instance of the CSP is to first derive some "obvious" consequences of the constraints. If, for example, an instance of HORN-3-SAT (see Example 5) contains the constraints $C_1(x)$, $C_1(y)$, and $S_{110}(x,y,z)$, then we can derive a new constraint $C_1(z)$. This information can be then used to derive, e.g., $C_1(w)$ from $S_{110}(z,x,w)$, etc. In fact, for HORN-3-SAT, if we cannot derive any new unary constraints in this way and we do not get contradictory unary constraints like $C_0(x)$ and $C_1(x)$, then the instance has a solution: simply obey the unary constraints and set the values of the remaining variables to 0. This is essentially the standard unit propagation (polynomial) algorithm for HORN-3-SAT.

More generally, given an instance of CSP($\mathcal{D}$) we can try to derive the strongest "obvious" unary constraints.[11] For some constraint languages all unsatisfiable instances can be refuted in this way. Such languages are said to have *width 1* and they are discussed in Subsection 5.1.

We illustrate a stronger constraint propagation (or local consistency) procedure by considering the following instance of CSP($\mathcal{D}_{\mathrm{2COLOR}}$) (see Example 6).

$$x_1 \neq x_2,\ x_2 \neq x_3,\ x_3 \neq x_4,\ x_4 \neq x_5,\ x_5 \neq x_1\ .$$

From $x_1 \neq x_2$ and $x_2 \neq x_3$ we can derive a new binary constraint $x_1 = x_3$. Using this and $x_3 \neq x_4$, we can derive $x_1 \neq x_4$. Finally, from $x_1 \neq x_4$, $x_4 \neq x_5$, we conclude that $x_1 = x_5$, which contradicts the constraint $x_1 \neq x_5$, and we refute the instance. In this particular example, it was enough to derive new binary constraints by considering at most three variables at a time. In fact, every unsatisfiable instance of 2-COLORING can be refuted in this way; we say that $\mathcal{D}_{\mathrm{2COLOR}}$ has *width (2,3)*. In Subsection 5.3, we give a more general result that will also apply to e.g. 2-SAT.

An even stronger consistency algorithm could derive all k-ary constraints that can be derived from the instance by considering at most l variables at a time. As long as k and l are fixed, we get a polynomial time algorithm that, for some constraint languages, correctly decides every instance of the CSP. The limit of such approaches is now fully understood, see Subsection 5.5.

We warn the reader that there are many related, but somewhat different notions capturing the required level of local consistency. In order to add to the confusion, we use the term "width (k,l)" for what is more often called "relational width (k,l)". However, the notion of "solvability by a local consistency algorithm" (meaning "having width (k,l) for some k,l") is the same for all common choices of definitions.

[11] The relations of the derived constraints do not need to belong to $\mathcal{D}$.

5.1 1–minimality and TS polymorphisms

The idea of the strongest obvious unary constraints can be formalized as follows.

▶ **Definition 50.** An instance of the CSP is 1–*minimal* (or *arc consistent*) if it contains a unique unary constraint $P_x(x)$ for each variable x and, for any constraint $R(x_1, \dots, x_k)$ and any $i = 1, \dots k$, the projection[12] of R onto the i-th coordinate is equal to P_{x_i}.

Every instance of the CSP can be converted in polynomial time to a 1-minimal instance with the same set of solutions. A straightforward (although not optimal) way to achieve this is as follows.

for every variable x **do** $P_x := D$; add the constraint $D(x)$
repeat
  for every constraint $R(x_1, \dots, x_k)$ **do**
    let $R' := R \cap \prod_i P_{x_i}$
    for $i = 1$ **to** k **do** $P_{x_i} := P_{x_i} \cap \mathrm{proj}_i R'$
    replace $R(x_1, \dots, x_k)$ with $R'(x_1, \dots, x_k)$
  end for
until none of the P_x's changed

We allow ourselves to call this algorithm *the* 1-*minimality algorithm*, since different implementations will derive the same unary constraints.

If any of the P_x's are empty after running the algorithm, then the original instance has no solution and we say that the 1-minimality algorithm refutes the instance. For some constraint languages, every non-refuted instance has a solution and thus we obtain a polynomial time algorithm for the corresponding CSP.

▶ **Definition 51.** We say that a constraint language $\mathcal{D}$, or CSP($\mathcal{D}$), has *width* 1 if the 1-minimality algorithm refutes every unsatisfiable instance of CSP($\mathcal{D}$) (and thus CSP($\mathcal{D}$) is solvable in polynomial time).

The following theorem [65, 60] characterizes width 1 languages in terms of polymorphisms. The proof shows a simple application of polymorphisms, namely TS polymorphisms, to prove correctness of an algorithm. Recall that the value of a TS operation t depends only on the set of its arguments, and we will write $t(\{d_1, \dots, d_n\})$ instead of $t(d_1, \dots, d_n)$.

▶ **Theorem 52.** *A constraint language $\mathcal{D}$ admits TS polymorphisms of all arities if and only if $\mathcal{D}$ has width* 1*.*

Sketch of proof. Assume that $\mathcal{D}$ admits TS polymorphisms of all arities, take an instance of CSP($\mathcal{D}$), run the 1-minimality algorithm, and assume that the instance is not refuted, that is, P_x is nonempty for every variable x. We need to show that the resulting instance has a solution.

Note that the constraint relations of the resulting 1-minimal instance may not belong to $\mathcal{D}$. However, each new relation is defined in a primitive positive way from the original constraints; therefore all polymorphisms of $\mathcal{D}$ are still compatible with the relations in the new instance. Take a TS polymorphism t of a sufficiently large arity n and define $f(x) := t(d_1, \dots, d_n)$ where $P_x = \{d_1, \dots, d_n\}$.

We claim that f is a solution. So, let us consider a constraint $R(x_1, \dots, x_k)$ and show that $(f(x_1), \dots, f(x_k)) \in R$. Take an $n \times k$ matrix so that its rows are in R and the set of

[12] Not to be confused with projection operations π_i^n.

elements in the i-th column is equal to P_{x_i}. This is easily achieved from 1-minimality as soon as $n \geq |D| \cdot k$. Now the identities that define a TS operation guarantee that t applied to the columns gives $(f(x_1), \ldots, f(x_k))$ and this tuple is in R, as t is compatible with R.

For the other implication, it is possible to create an instance of CSP($\mathcal{D}$) which is not refuted by 1-minimality and which essentially says that $\mathcal{D}$ has TS polymorphisms of all arities. We refer the reader to [65, 60] for details. ◀

Note that we do not need to (explicitly) know any TS polymorphism to run the 1-minimality algorithm, but these polymorphisms guarantee that the algorithm correctly decides all instances.

5.2 Linear Programming and symmetric polymorphisms

Another line of research discovered a connection between consistency notions and certain convex programming relaxations of the CSP, namely the *canonical Linear Programming (LP) relaxation* and the *canonical Semidefinite Programming (SDP) relaxation*. We now outline the LP relaxation here and describe the SDP relaxation in Section 5.6.

For simplicity, we will restrict to instances that contain variables $x_1, \ldots, x_n$ and exactly one binary constraint $P_{i,j}(x_i, x_j)$ for every $1 \leq i < j \leq n$ (and no other constraints). In particular, there are $m = n(n-1)/2$ constraints. The task to find an assignment satisfying the maximum number of constraints can be phrased as a 0–1 integer program as follows. The variables are $\lambda_{i,a} \in \{0,1\}$ for each $1 \leq i \leq n$, $a \in D$ (where D is the domain) and $\sigma_{i,j,a,b} \in \{0,1\}$ for each $1 \leq i < j \leq n$, $a, b \in D$. The intended meaning is that $\lambda_{i,a} = 1$ iff x_i is assigned the value a and $\sigma_{i,j,a,b} = 1$ iff x_i is assigned a and x_j is assigned b. The task is to maximize

$$\mathrm{Opt} = \frac{1}{m} \sum_{1 \leq i < j \leq n} \sum_{(a,b) \in P_{i,j}} \sigma_{i,j,a,b}$$

subject to the constraints

$$\sum_{a \in D} \lambda_{i,a} = 1 \quad \text{for each } 1 \leq i \leq n$$

$$\sum_{b \in D} \sigma_{i,j,a,b} = \lambda_{i,a} \quad \text{for each } 1 \leq i < j \leq n,\ a \in D$$

$$\sum_{a \in D} \sigma_{i,j,a,b} = \lambda_{j,b} \quad \text{for each } 1 \leq i < j \leq n,\ b \in D$$

Since we are only concerned with deciding whether a solution exists, we only care whether all the constraints are satisfied, that is, whether $\mathrm{Opt} = 1$.

The *canonical (or basic) LP relaxation* relaxes the 0–1 constraints to $\lambda_{i,a} \in [0,1]$ and $\sigma_{i,j,a,b} \in [0,1]$. The optimization problem thus becomes solvable in polynomial time, but the new optimum $\mathrm{Opt}_{\mathrm{LP}}$ may be greater than Opt. We say that this relaxation decides the CSP if an instance has a solution whenever $\mathrm{Opt}_{\mathrm{LP}} = 1$. The canonical LP relaxation is at least as strong as 1-minimality. Indeed, it is not hard to see that if $\mathrm{Opt}_{\mathrm{LP}} = 1$, then $P_x = \{a \in D : \lambda_{i,a} > 0\}$, together with appropriately restricted binary constraints, is 1-minimal. In fact, the relaxation is somewhat stronger [93]:

▶ **Theorem 53.** CSP($\mathcal{D}$) *is decided by the canonical LP relaxation if and only if* $\mathcal{D}$ *has symmetric polymorphisms of all arities.*

Sketch of proof. Assume that $\mathcal{D}$ admits symmetric polymorphisms of all arities. Take an instance of CSP($\mathcal{D}$) and solve its canonical LP relaxation. If $\mathrm{Opt}_{\mathrm{LP}} < 1$ then clearly the

instance is not satisfiable. Assume that $\mathrm{Opt}_{\mathrm{LP}} = 1$ and show that in this case the instance is satisfiable. In fact, we show that a symmetric polymorphism of appropriate arity can be used to round an optimal LP solution to a satisfying assignment.

Take an optimal LP solution and denote the values taken by variables by $\sigma^*_{i,j,a,b}$ and $\lambda^*_{i,a}$. We can assume that all these values are rational. Let N be an integer such that $N \cdot \sigma^*_{i,j,a,b}$ is integer for all i, j, a, b (and hence, from LP constraints, all numbers $N \cdot \lambda^*_{i,a}$ are also integers). Let s be a symmetric polymorphism of arity N. For each variable x_i in the original CSP instance, let $f(x_i) = s(d_1, \dots, d_N)$ where each value a appears among the d_i's exactly $N \cdot \lambda^*_{i,a}$ times. This definition is correct because $\sum_{a \in D} \lambda^*_{i,a} = 1$ (from LP constraints) and because s is symmetric (so the order of the d_i's is irrelevant). We will show that f is a solution.

Note that, since $\mathrm{Opt}_{\mathrm{LP}} = 1$, we have that, for each i, j, $\sum_{(a,b) \in P_{i,j}} \sigma_{i,j,a,b} = 1$. In particular, if $\sigma_{i,j,a,b} > 0$ for some a, b then $(a, b) \in P_{ij}$. Fix indices i, j. Take any $N \times 2$ matrix such that each pair $(a, b) \in P_{ij}$ appears as a row in the matrix exactly $N \cdot \sigma^*_{i,j,a,b}$ times. The LP constraints and the fact that s is symmetric directly imply that, when applying s to the columns of this matrix, one gets $(f(x_i), f(x_j))$.

For the other implication, if $\mathcal{D}$ has no symmetric polymorphism of some arity m, then it is possible to construct an unsatisfiable instance of $\mathrm{CSP}(\mathcal{D})$ with $\mathrm{Opt}_{\mathrm{LP}} = 1$. We refer the reader to [93] for details. ◀

It was claimed in [93] that any structure $\mathcal{D}$ has symmetric polymorphisms of all arities if and only if it has TS polymorphisms of all arities, but this claim turned out to be false – see Example 99 in [94] for a counter-example.

5.3 (2,3)-minimality and majority polymorphisms

Next we introduce a stronger consistency notion than 1-minimality. For clarity, we define this concept only for *binary instances*, that is, instances containing only unary and binary constraints. A general definition will be given in Subsection 5.5.

▶ **Definition 54.** A binary instance of the CSP is $(2, 3)$*–minimal* if it contains a unique unary constraint $P_x(x)$ for each variable x and a unique binary constraint $P_{x,y}(x, y)$ for any pair of distinct variables x, y such that

- for every pairwise distinct variables x, y, z and every $(a, b) \in P_{x,y}$, there exists $c \in P_z$ such that $(a, c) \in P_{x,z}$ and $(b, c) \in P_{y,z}$;
- for any pairwise distinct variables x, y, $P_{x,y} = P^{-1}_{y,x}$ and the projection of $P_{x,y}$ onto the first (second, respectively) coordinate is equal to P_x (P_y, resp.).

It is helpful to visualize a binary $(2, 3)$-minimal instance as a multipartite graph: it has one partite set for each variable x, whose set of vertices is equal to (a disjoint copy of) P_x, and vertices $a \in P_x$, $b \in P_y$ are adjacent if $(a, b) \in P_{x,y}$. The first condition in Definition 54 then means that by restricting to any three partite sets we obtain a graph in which every edge belongs to a triangle. Note that a solution of the instance corresponds to a clique that contains exactly one vertex from each partite set.

Similarly to 1-minimality, every (binary) instance of the CSP can be converted in polynomial time to a $(2, 3)$-minimal instance with the same set of solutions. We say that the original instance is refuted by this algorithm if some P_x is empty.

▶ **Definition 55.** We say that a constraint language $\mathcal{D}$ has *width* $(2, 3)$ if the $(2, 3)$-minimality algorithm refutes every unsatisfiable instance of $\mathrm{CSP}(\mathcal{D})$.

The following theorem [65, 79] says that each constraint language with a majority polymorphism, such as any set of unary and binary relations on $\{0,1\}$ (see Example 24), has width $(2,3)$. In particular, we get that 2-COLORING and 2-SAT are solvable in polynomial time.

▶ **Theorem 56.** *If a constraint language $\mathcal{D}$ has a majority polymorphism, then $\mathcal{D}$ has width $(2,3)$.*

Sketch of proof. We will only consider binary instances.

Take a majority polymorphism m of $\mathcal{D}$ and consider the $(2,3)$-minimal instance associated to a given instance of CSP($\mathcal{D}$). As in the proof of Theorem 52, it can be argued that m is compatible with P_x and $P_{x,y}$ for any x, y. We show that the instance has a solution provided every P_x is nonempty.

Recall from the remarks after Definition 54 that "every edge extends to a triangle". We will show that "every triangle extends to a 4–clique". Take any variables x, y, z, w and any $a \in P_x$, $b \in P_y$, $c \in P_z$ that form a triangle, that is, $(a,b) \in P_{x,y}$, $(a,c) \in P_{x,z}$, and $(b,c) \in P_{y,z}$. Using the edge–to–triangle property three times we get $d_1, d_2, d_3 \in P_w$ such that bcd_1, acd_2, and abd_3 are triangles. We claim that $d = m(d_1, d_2, d_3)$ together with a, b, c form a 4–clique. By the second property from Definition 54, there exists $a' \in P_x$ such that $(a', d_1) \in P_{x,w}$. Applying m to the columns of the 3×2 matrix with rows (a', d_1), (a, d_2), (a, d_3) (all in $P_{x,w}$) gives $(m(a', a, a), m(d_1, d_2, d_3)) \in P_{x,w}$, which is equal to (a, d) - here we use one of the identities defining a majority operation. Similarly, $(b, d) \in P_{y,w}$ and $(c, d) \in P_{z,w}$, which proves the claim.

In a similar way, we can show that every 4–clique extends to a 5–clique, and so on. Summarizing, every edge extends to a $|V|$–clique – a solution to the instance. ◀

The proof shows that for a constraint language $\mathcal{D}$ with a majority polymorphism, a greedy algorithm can be used to find a solution to any instance of CSP($\mathcal{D}$): after running the $(2,3)$–minimality algorithm, we pick, one by one, assignments to variables so that they stay consistent with the constraints. Such languages are said to have *strict width* $(2,3)$ and they are in fact characterized by the existence of a majority polymorphism [65].

The arguments can be easily generalized to NU polymorphisms if an appropriate consistency level is enforced. Namely, a constraint language has an NU polymorphism of arity n if and only if it has strict width $(n-1, n)$ [65].

5.4 Interlude: Boolean CSPs

Before we move on to discuss the general concept of bounded width, we show how to use polymorphisms to prove Schaefer's dichotomy theorem [112] for CSPs over a two–element domain. The only additional fact we need is the following lemma.

▶ **Lemma 57.** *Every idempotent (recall Example 25) clone on $D = \{0,1\}$ that contains a non–projection contains one of the following operations: the binary* max*, the binary* min*, the ternary majority, or the ternary minority.*

Sketch of proof. An old result by Post [108] completely describes all clones on $\{0,1\}$ and we can thus simply use his classification. However, proving this lemma is significantly easier than the full classification and we sketch one possible approach.

The only binary idempotent operations are the two projections, max, and min. Therefore, let us assume that the only binary operations in our idempotent clone are projections. Then, for each ternary operation f, the operations $h(x,y) = f(x,x,y)$, $h'(x,y) = f(x,y,x)$, and

$h''(x,y) = f(y,x,x)$ must all be projections. This reduces the number of idempotent ternary operations to be considered to 8. Now 3 of them are projections, then there are the majority and the minority. The remaining 3 operations differ only in the order of arguments, so we are left with 1. This is the so called Pixley operation and both majority and minority can be composed from it. It remains to show that each idempotent operation f is a projection provided the clone does not contain any non–projection binary or ternary operation. This can be done e.g. by case analysis using only that each $g(x,y,z) = f(x/y/z,\dots,x/y/z)$ is a projection. ◀

We are ready to show a simple proof of the dichotomy theorem for Boolean CSPs.

▶ **Theorem 58.** *Let $\mathcal{D}$ be a constraint language with domain $D = \{0,1\}$. Then*
- *either its polymorphism clone contains a constant unary operation or one of the four operations in Lemma 57 and then* CSP($\mathcal{D}$) *is solvable in polynomial time, or*
- CSP($\mathcal{D}$) *is NP–complete.*

Proof. If the polymorphism clone $\mathbf{D}$ of $\mathcal{D}$ contains a constant unary operation with value a, then all relations contain a constant tuple $(a,a,\dots,a)$ and then all instances of CSP($\mathcal{D}$) have a solution. Otherwise, $\mathcal{D}$ is a core. Then either $\mathbf{D}$ contains only essentially unary operations, or not. In the first case, the singleton expansion $\mathcal{E}$ has only trivial polymorphisms (=projections), $\mathcal{E}$ pp-defines (by Theorem 30) all structures on $\{0,1\}$ including $\mathcal{D}_{3SAT}$, and then CSP($\mathcal{E}$) as well as CSP($\mathcal{D}$) are NP–complete by Theorem 12 and Theorem 16. In the second case, the polymorphism clone also contains an idempotent non–projection. By the previous lemma, $\mathbf{D}$ contains one of the four operations and then CSP($\mathcal{D}$) is solvable in polynomial time by Theorem 52, Theorem 56, or Example 27. ◀

As a non–trivial exercise, the reader may verify a finer description of the polynomial cases: if $\mathcal{D}$ has min as a polymorphism then $\mathcal{D}$ is pp-definable from $\mathcal{D}_{\mathrm{HornSAT}}$ (and dually for max), if $\mathcal{D}$ has the majority polymorphism then $\mathcal{D}$ is pp-definable from $\mathcal{D}_{\mathrm{2SAT}}$, and if $\mathcal{D}$ has the minority polymorphism then $\mathcal{D}$ is pp-definable from $\mathcal{D}_{\mathrm{3LIN2}}$.

We remark that classifications of Boolean CSPs with respect to other complexity classes and with respect to width notions mentioned in previous section can be found in [1, 97].

5.5 Characterization of bounded width

An elegant way to formalize "polynomial solvability by constraint propagation" in general is by means of (k,l)-minimality.

▶ **Definition 59.** Let $1 \le k \le l$ be integers. An instance of the CSP is (k,l)-minimal if
- no scope of a constraint contains repeated variables,
- every l-element set of variables is within the scope of some constraint, and
- for any at most k-element set of variables W and any two constraints whose scope contains W, the projections of these constraints onto W coincide.

The reader may notice a formal difference between this definition and its special case in Definition 54. Indeed, a $(2,3)$-minimal instance in the latter does not contain any ternary constraints, while the former one requires that any triple of variables is covered by a constraint. However, the difference is only cosmetic. The sole purpose of the second condition in Definition 59 is to ensure an analogue of the first condition in Definition 54.

For fixed k,l, there is a straightforward polynomial algorithm, the *(k,l)–minimality algorithm*, to transform any CSP instance into a (k,l)-minimal instance with the same set of solutions. As before, instances with an empty constraint relation are refuted.

▶ **Definition 60.** We say that a constraint language $\mathcal{D}$ has width (k, l) if the (k, l)–minimality algorithm refutes every unsatisfiable instance of CSP($\mathcal{D}$).

We say that $\mathcal{D}$ has *width* k if it has width (k, l) for some l and it has *bounded width* if it has width k for some k. (Recall again that bounded width CSPs are solvable in polynomial time.)

The notion of bounded width comes in various versions and equivalent forms. Bounded width is equivalent to solvability by a Datalog program [65], to the existence of a winning strategy in a certain pebble game [65], to having bounded treewidth duality [44], and to definability in an infinitary finite-variable logic [4, 87].

As mentioned in Subsection 4.4, a typical problem which cannot be efficiently solved by local propagation algorithms is solving systems of linear equations [65].

▶ **Theorem 61.** *For any prime p, the constraint language $\mathcal{D}_{3\mathrm{LIN}_p}$ does not have bounded width.*

There is an analogue of Corollary 19 for bounded width [100]: If $\mathcal{D}$ pp-constructs $\mathcal{E}$ and $\mathcal{D}$ has bounded width, then so does $\mathcal{E}$. Therefore, pp-constructing $\mathcal{D}_{3\mathrm{LIN}_p}$ is the "obvious" obstruction to having bounded width. Is it the only obstruction? A positive answer was conjectured in several equivalent forms in [65, 100, 36] (see also [98]).

The process of resolving this, so called *bounded width conjecture*, nicely illustrates the role of universal algebra in identifying meaningful intermediate classes.

- Extending the positive result for semilattice polymorphisms, Bulatov [37] confirmed the conjecture for constraint languages with so called 2–semilattice polymorphisms.
- Very natural candidates for extending the positive result for near unanimity polymorphisms are constraint languages with Jónsson polymorphisms (see Subsection 4.5). Indeed, clones with Jónsson operations are among the most studied objects in universal algebra. Partial results were obtained in [86, 48] and a full solution for Jónsson polymorphisms given in [11].

Helped greatly by these partial results, the bounded width conjecture was confirmed in [14] and independently in an unpublished manuscript by Bulatov [31] (see also [42]).

▶ **Theorem 62.** *A constraint language $\mathcal{D}$ has bounded width if and only if $\mathcal{D}$ does not pp-construct $\mathcal{D}_{3\mathrm{LIN}_p}$ for any prime p.*

The proof from [14] is, to some extent, explained in another survey [16] in this volume. Here we only mention two differences from the arguments in Theorems 52 and 56. First, polymorphisms characterizing the necessary condition for bounded width, such as those in Theorem 44, are not used directly. They are first iteratively composed to get polymorphisms of large arities with properties helpful for the proof. Second, the solution is not directly obtained from a sufficiently consistent instance. Instead, polymorphisms serve to "condense" the constraints to smaller and smaller subsets of the domain, while preserving a sufficient degree of consistency.

5.6 Sufficient levels of consistency

Bounded width CSPs are those for which enforcing a certain level of local consistency guarantees a solution. What degree of consistency is actually needed?

The proof of Theorem 61 from [14] uses a consistency notion which is, for binary instances, stronger than 1-minimality and weaker than $(2, 3)$–minimality. A small refinement from [10]

shows that an appropriate notion of consistency, still weaker than $(2,3)$–minimality, is enough in general. In particular, any bounded width CSP has width $(2,3)$.

A substantial strengthening of Theorem 61 by Kozik [90] weakens the consistency requirement even further. In particular, his result implies that the following consistency procedure, so called *Singleton Arc Consistency*, or *SAC*, is sufficient: ensure that, for each variable x and any $a \in P_x$, the 1-minimality algorithm does not refute the instance even with the added unary constraint $C_a(x)$.

As we mentioned before, there is a connection between consistency notions and certain convex programming relaxations of the CSP, namely the *canonical Linear Programming (LP) relaxation* (see Section 5.2) and the *canonical Semidefinite Programming (SDP) relaxation.*

Again for simplicity, we will restrict to instances that contain variables $x_1, \ldots, x_n$ and exactly one binary constraint $P_{i,j}(x_i, x_j)$ for every $1 \leq i < j \leq n$ (and no other constraints). In particular, there are $m = n(n-1)/2$ constraints. To describe the *canonical SDP relaxation*, we will re-use notation from the Section 5.2, but the variables $\lambda_{i,a}$ now become vectors in Euclidean space (of dimension $O(n)$), $\sigma_{i,j,a,b}$ is required to be equal to the dot product $\lambda_{i,a} \cdot \lambda_{j,b}$ (which is required to be non-negative) and the first constraint is replaced by the requirement that $\lambda_{i,a}$, $a \in D$ are pairwise orthogonal and sum up to a fixed unit vector (the remaining two constraints are then redundant). As before, we say that this relaxation decides a CSP if $\mathrm{Opt}_{\mathrm{SDP}} = 1$ only if the instance has a solution. It was proved in [15] that the canonical SDP relaxation not only decides every bounded width CSP, but it can also be used to give a polynomial time *robust algorithm* for such CSPs, that is, provide "almost solutions" even to "almost satisfiable" instances. More formally, a robust algorithm is an approximation algorithm which, on every instance where $(1-\epsilon)$-fraction of constraints can be satisfied, returns an assignment that satisfies at least $(1-g(\epsilon))$-fraction of constraints, where g is such that $g(\epsilon) \to 0$ as $\epsilon \to 0$. By combining this result from [15] with [69, 57], we get that efficient robust solvability is equivalent to bounded width.

▶ **Theorem 63.** *The following are equivalent for any constraint language $\mathcal{D}$.*

- *$\mathcal{D}$ has bounded width.*
- *$\mathcal{D}$ has width $(2,3)$.*
- *$\mathrm{CSP}(\mathcal{D})$ is solvable by SAC.*
- *$\mathrm{CSP}(\mathcal{D})$ is decided by the canonical SDP relaxation.*
- *$\mathrm{CSP}(\mathcal{D})$ has a robust polynomial algorithm (this item is only equivalent to the rest if $P \neq NP$).*

5.7 Results about linear and symmetric width

We briefly discussed the notions of linear width and symmetric width in Section 4.4. These notions, introduced in [52] and [62], respectively, attract attention for several reasons. They have many natural equivalent descriptions in terms of logic and in combinatorial terms (see, e.g. [44]). They have natural necessary conditions in terms of very simple forbidden constraint languages and well-known algebraic conditions (see Theorems 45 and 49). These necessary conditions are conjectured to be sufficient, see Conjectures 4 and 5. Moreover, having bounded linear (resp. symmetric) width is conjectured to be the single reason for $\mathrm{CSP}(\mathcal{D})$ to be in NL and L, respectively.

Progress towards Conjecture 4 has been made by using the taxonomy from Section 4.5. The conjecture has been confirmed for increasingly weaker assumptions: that $\mathcal{D}$ has the dual discriminator polymorphism [52], a majority polymorphism [56], an NU polymorphism [19] (and hence Jónsson polymorphisms [8]). In [47], the conjecture was also confirmed for a

class of constraint languages consisting of (possibly all) languages of width 1 that satisfy the conditions of Theorem 45; this class contains languages without NU polymorphisms. It seems that that the current results approach the full conjecture very closely, and the next step will probably be the full resolution of the conjecture.

To have bounded symmetric width for a language $\mathcal{D}$, it is necessary that $\mathcal{D}$ has bounded linear width and satisfies the conditions of Theorem 48 (see Fig. 1). It was shown in [59] that any constraint language of bounded linear width that has a Mal'tsev polymorphism has bounded symmetric width. Other partial results towards Conjecture 5 (specifically related to H-COLORING) can be found in [54]. Recently Kazda proved [84] that every structure $\mathcal{D}$ having bounded linear width and satisfying the conditions of Theorem 48 in fact has bounded symmetric width. Thus, a characterization of CSPs of bounded linear width would also give a characterization of CSPs of bounded symmetric width. In particular, Conjecture 5 reduces to Conjecture 4.

6 Polymorphisms in algorithms II: Cogs in the works

Gaussian elimination not only solves 3-LIN(p), it also describes all the solutions in the sense that the algorithm can output a small (polynomial in n, the number of variables) set of points in $\mathrm{GF}(p)^n$ so that the affine hull of these points is equal to the solution set of the original instance. A sequence of papers [65, 35, 32, 53] culminating in [74, 23] pushed this idea, in a way, to its limit.

6.1 Few subpowers

We need some terminology to state the result. Let $\mathcal{D}$ be a constraint language and $\mathbf{D}$ its clone of polymorphisms. Let us call a relation on D a *subpower* of $\mathbf{D}$ if it is pp-definable from $\mathcal{D}$, or equivalently by Theorem 30, if it is invariant under all the polymorphisms of $\mathcal{D}$. Note that the set of solutions of any instance of CSP($\mathcal{D}$) can be viewed as a subpower of $\mathbf{D}$. If R is a subpower of $\mathbf{D}$ and $X \subseteq R$, then we say that X is an *algebraic generating set* of R if R is the smallest subpower of $\mathbf{D}$ containing X. In this case R is precisely the set of values of polymorphisms of $\mathcal{D}$ applied coordinate-wise to tuples from X. Now $\mathbf{D}$ has *few subpowers* if it satisfies any of the equivalent conditions in the following theorem.

▶ **Theorem 64.** *[23] For any clone $\mathbf{D}$, the following are equivalent.*

1. *There is a polynomial p such that $|\{R \subseteq D^n \mid R \text{ is a subpower of } \mathbf{D}\}| \leq 2^{p(n)}$*
2. *There is a polynomial q such that each subpower $R \subseteq D^n$ of $\mathbf{D}$ has an algebraic generating set with at most $q(n)$ elements.*
3. *For some $k \geq 2$, $\mathbf{D}$ contains a k-edge operation.*

The name "few subpowers" comes from condition (1) of the above theorem, but we will use conditions (2) and (3). See Section 4.5 for the definition of a k-edge operation.

To describe the examples to follow, we need a few more definitions.

▶ **Definition 65.** Let $X \subseteq D^n$.

1. If $i_1, i_2, \ldots, i_k$ is a sequence of indices from $\{1, \ldots, n\}$, then the *projection of X onto coordinates $i_1, \ldots, i_k$*, denoted $\mathrm{proj}_{i_1,\ldots,i_k} X$, is the set $\{(a_{i_1}, \ldots, a_{i_k}) : (a_1, \ldots, a_n) \in R\}$.
2. A *fork* (of arity n) is a triple (i, a, b) with $1 \leq i \leq n$, $a, b \in D$, and $a \neq b$.
3. A *realization* of a fork (i, a, b) is a pair $\mathbf{a}, \mathbf{b} \in D^n$ satisfying $a_j = b_j$ for all $j < i$ and $(a_i, b_i) = (a, b)$. If $\mathbf{a}, \mathbf{b} \in X$ then we say that (i, a, b) is *realized in X*.

▶ **Example 66.** Suppose $\mathcal{D}$ has a Mal'tsev polymorphism (see Subsection 4.5) and $\mathbf{D}$ is its clone of polymorphisms. Then $\mathbf{D}$ has few subpowers [32]. To prove this, let $R \subseteq D^n$ be a subpower of $\mathbf{D}$. We will show that R has a generating set of size at most $q(n) := cn$ where $c = |D|^2$.

We say that a subset $X \subseteq R$ *witnesses single projections and forks* if (i) $\mathrm{proj}_i R = \mathrm{proj}_i X$ for all $i = 1, \dots, n$, and (ii), R and X realize the same forks. It is easy to see that if X is a minimal subset of R which witnesses single projections and forks, then $|X| \leq q(n)$.

It turns out that if $X \subseteq R$ and X witnesses single projections and forks, then X generates R. To see this, let R' be the set of tuples obtained by applying polymorphisms of $\mathcal{D}$ coordinate-wise to tuples from X. Clearly, R' is a subpower of $\mathbf{D}$ and $X \subseteq R' \subseteq R$. To prove that $R' = R$, let i be maximum index so that $\mathrm{proj}_{1,2,\dots,i} R = \mathrm{proj}_{1,2,\dots,i} R'$, and suppose for the sake of contradiction that $i < n$. Choose $\mathbf{a} \in R$ satisfying $\mathrm{proj}_{1,2,\dots,i+1}(\mathbf{a}) \notin \mathrm{proj}_{1,2,\dots,i+1} R'$ (such $\mathbf{a}$ must exist by our choice of i and assumption that $i \neq n$). Also by our choice of i, there is a tuple $\mathbf{b} \in R'$ satisfying $\mathrm{proj}_{1,2,\dots,i}(\mathbf{b}) = \mathrm{proj}_{1,2,\dots,i}(\mathbf{a})$. Since $\mathbf{a}, \mathbf{b} \in R$, they witness that R realizes the fork $(i+1, a_{i+1}, b_{i+1})$. X must also realize this fork, say by $\mathbf{c}, \mathbf{d}$. Let $\mathbf{e} = f(\mathbf{b}, \mathbf{d}, \mathbf{c})$ where f is the Mal'tsev polymorphism of $\mathcal{D}$ and the application of f to $\mathbf{b}, \mathbf{d}, \mathbf{c}$ is done coordinate-wise. The first $i+1$ coordinates of $\mathbf{b}, \mathbf{d}, \mathbf{c}$ have the form

$$\begin{aligned}
\mathbf{b} &= (a_1, \dots, a_i, b_{i+1}, ___) \\
\mathbf{d} &= (c_1, \dots, c_i,\ b_{i+1}, ___) \\
\mathbf{c} &= (c_1, \dots, c_i,\ a_{i+1}, ___).
\end{aligned}$$

Because f is a Mal'tsev polymorphism, it satisfies $f(y, x, x) = f(x, x, y) = y$ for all $x, y \in D$. In particular, $e_j = f(a_j, c_j, c_j) = a_j$ for $j = 1, \dots, i$, and $e_{i+1} = f(b_{i+1}, b_{i+1}, a_{i+1}) = a_{i+1}$. That is, $\mathbf{a}$ and $\mathbf{e}$ agree on the first $i+1$ coordinates. Also note that $\mathbf{e} \in R'$ since $\mathbf{b}, \mathbf{c}, \mathbf{d} \in R'$ and R' is invariant under f. But this is contrary to the choice of $\mathbf{a}$.

The above proof that $R' = R$ used only the following properties of R': (i) $X \subseteq R' \subseteq R$, and (ii) R' is invariant under the Mal'tsev polymorphism f. This proves that R is the *closure* of X under f, meaning that R is the output Y of the following "closure" algorithm.

input X, f
let $Y := X$
repeat
  for every $\mathbf{a}, \mathbf{b}, \mathbf{c} \in Y$ **do**
    let $Y := Y \cup \{f(\mathbf{a}, \mathbf{b}, \mathbf{c})\}$
  end for
until Y doesn't change

This observation is used crucially in the few subpowers algorithm.

▶ **Example 67.** Suppose $\mathcal{D}$ has a majority polymorphism (see Subsection 4.5). Then its clone of polymorphisms $\mathbf{D}$ has few subpowers [6]. Indeed, if $R \subseteq D^n$ is a subpower of $\mathbf{D}$, $X \subseteq R$, and X and R have the same "double projections," that is, $\mathrm{proj}_{i,j} X = \mathrm{proj}_{i,j} R$ for all $1 \leq i, j \leq n$, then X generates R. Indeed, let R' be the subpower generated by X, so $X \subseteq R' \subseteq R$. The idea of the proof that $R' = R$ is very similar to the proof of Theorem 56; show that R' and R have the same projection onto any triple of coordinates, then any 4-tuple of coordinates, etc. Since a suitable set $X \subseteq R$ having the same double projections as R can be found satisfying $|X| \leq cn^2$, R has a small generating set. As in the previous example, the proof actually shows that R is the closure of X under the majority polymorphism.

More generally, if $\mathcal{D}$ has a k-ary NU polymorphism, then a subpower $R \subseteq D^n$ is generated by any set $X \subseteq R$ which has the same projection as R onto each set of coordinates of size

$k-1$. Such an X can always be found satisfying $|X| \leq dn^{k-1}$ (for a suitable constant d). Again R is the closure of X under the NU polymorphism.

Note that if $\mathcal{D}, R$, and X are as in either Example 66 or 67, then one cannot expect to efficiently run the closure algorithm to construct R from X, since the size of R could be exponential in the size of X. However, one can efficiently calculate the projection of R onto any small set s of coordinates (say of size at most 5), and even find a (small) subset of R containing X whose projection onto s agrees with R. This is because R is the closure of X under a single fixed operation f, so we can simply run the closure algorithm for X and f, but adding new elements to Y only when their projection onto s is new. With this restriction, there is a constant upper bound ($|D|^{|s|}$) to the number of times the main loop of the closure algorithm will be repeated; hence this restricted algorithm runs in polynomial time. This observation is one of the fundamental tools of the few subpowers algorithm.

6.2 The Few Subpowers Algorithm

▶ **Theorem 68.** *[74] Let $\mathcal{D}$ be an idempotent constraint language. If the clone of polymorphisms $\mathbf{D}$ has few subpowers, then* CSP($\mathcal{D}$) *can be solved in polynomial time (moreover, the algorithm can output a generating set for the set of all solutions).*

Suppose $\mathbf{D}$ has few subpowers. The main idea of the few subpowers algorithm, which can be traced back to [65], is the following. Given a CSP($\mathcal{D}$) instance $P = (V, D, \mathcal{C})$ with, say, all constraint relations at most binary, we can enumerate $V = \{x_1, \dots, x_n\}$ and $\mathcal{C} = (C_1, \dots, C_m)$ and consider the decreasing sequence

$$D^n = R_0 \supseteq R_1 \supseteq \cdots \supseteq R_i \supseteq \cdots \supseteq R_m \tag{26}$$

of subpowers of $\mathbf{D}$, where R_i is the set of solutions to the first i constraints. The aim of the algorithm is to construct a small generating set X_i for each R_i. Then X_m will be a small generating set for the set of solutions to P, so P has a solution if and only if X_m is not empty. It should be easy to construct the generating set X_0 for D^n. Thus the chief task of the algorithm is that of finding the "next" X_{i+1} given X_i and C_{i+1}. We sketch how this is done in the two easiest cases: $\mathcal{D}$ has a majority or Mal'tsev polymorphism.

▶ Case 1. $\mathcal{D}$ has a majority polymorphism f.

We assume that X is a small subset of the subpower R which has the same double projections as R, and C is a (say) binary constraint $((x_k, x_\ell), S)$. Let R' be the "next" subpower determined by X and C; that is, $R' = \{\mathbf{a} \in R : S(a_k, a_\ell)\}$. The goal is to find a small subset X' of R' which has the same double projections as R'. This is easy. As shown in Example 67, R is the closure under f of X. Thus for each pair (i, j) of coordinates, we can (using the restricted closure algorithm for f) find a small subset $X_{i,j}$ of R satisfying $\mathrm{proj}_{i,j,k,\ell} X_{i,j} = \mathrm{proj}_{i,j,k,\ell} R$. If we let $X_1 = \bigcup_{i<j} X_{i,j}$, then the set $X' = \{\mathbf{a} \in X_1 : S(a_k, a_\ell)\}$ is a small subset of R' and has the same double projections as R', as required. Note that the few subpowers algorithm in this case consists of repeated applications of the restricted closure algorithm using the majority polymorphism.

The case when $\mathcal{D}$ has an NU polymorphism is handled similarly.

▶ Case 2. $\mathcal{D}$ has a Mal'tsev polymorphism f.

We assume that X is a small subset of the subpower R which witnesses single projections and forks, and C is an at-most binary constraint. Again let R' be the "next" subpower determined by X and C. The goal this time is to find a small subset X' of R' witnessing single projections and forks (of R'). Witnessing single projections is done analogously to the

argument in the majority case. Witnessing forks requires some ingenuity.[13] We first show how to do this in a very special case: C is a singleton unary constraint $(x_k, \{c\})$ (recall that $\mathcal{D}$ is idempotent) *and* the projections of R onto each coordinate $j < k$ have size 1. In this situation, necessary conditions for a fork (i, a, b) to be realized in R' are

1. $i > k$

2. (i, a, b) is realized in X, say by $\mathbf{a}, \mathbf{b}$.

3. R has a tuple $\mathbf{c}$ whose projection on coordinates k, i satisfies $(c_k, c_i) = (c, a)$.

(Note that we can efficiently find $\mathbf{a}, \mathbf{b}, \mathbf{c}$ when they exist, or determine that they do not exist.) In fact, these conditions are also sufficient, because the tuples $\mathbf{c}$ and $\mathbf{d} := f(\mathbf{c}, \mathbf{a}, \mathbf{b})$ belong to R' and can be shown to realize the fork (i, a, b) by reasoning similar to that in Example 66, using the Mal'tsev operation identities.

Now we consider the general case where C is a (say) binary constraint $((x_k, x_\ell), S)$. Let (i, a, b) be a fork. A necessary condition for (i, a, b) to be realized in R' is that R contain a tuple $\mathbf{c}$ whose projection onto coordinates i, k, ℓ is in $\{a\} \times S$. Because R is the closure under f of X, we can find such $\mathbf{c}$ when it exists, by the restricted closure algorithm. Suppose such $\mathbf{c}$ is found. Note that $\mathbf{c} \in R'$ and $c_i = a$. Now a key property of R' is that if the fork (i, a, b) is realized in R', then $\mathbf{c}$ is one half of such a realization. For suppose $\mathbf{u}, \mathbf{v} \in R'$ realize (i, a, b). Then $\mathbf{d} := f(\mathbf{c}, \mathbf{u}, \mathbf{v})$ is in R' and, arguing as above, $\mathbf{c}, \mathbf{d}$ realize (i, a, b).

So we just need to search for a tuple $\mathbf{d}$ in R which agrees with $\mathbf{c}$ on its first $i-1$ coordinates, equals b at coordinate i, and which satisfies $(d_k, d_\ell) \in S$. This search is accomplished by cutting down R to the subrelation $R_1 = \{\mathbf{x} \in R : x_1 = c_1 \;\&\; x_2 = c_2 \;\&\; \cdots \;\&\; x_{i-1} = c_{i-1}\}$. Using the previous "special case" argument $i - 1$ times, we can find a small subset X_1 of R_1 which witnesses projections and forks for R_1, and then use X_1 and the restricted closure algorithm to search for an element $\mathbf{d} \in R_1$ (if it exists) whose projection onto coordinates i, k, ℓ is in $\{b\} \times S$. These searches can be done in polynomial time, using the restricted closure algorithm.

In summary, the implementation of the few subpowers algorithm in the Mal'tsev case is a carefully orchestrated sequence of applications of the restricted closure algorithm for the Mal'tsev polymorphism, with some additional computations using this polymorphism.

In general, the few subpowers algorithm [74] works more or less as the union of the NU and Mal'tsev cases, following Dalmau [53] who was the first to combine these two cases. By Theorem 64, if the polymorphism clone of $\mathcal{D}$ has few subpowers, then $\mathcal{D}$ has a k-edge polymorphism for some $k \geq 2$. A notion of "nice" small generating sets (analogous to witnessing all $(k-1)$-ary projections and forks) for subpowers is worked out in [23], and the above argument in the Mal'tsev case is more or less repeated verbatim. The point we wish to make here is that the edge polymorphism is used essentially and repeatedly by the algorithm. Without an edge polymorphism, the above implementation of the few subpowers algorithm cannot be executed.

6.3 Limits of the few subpowers algorithm

Given an operation f on a finite set D, let $\mathcal{D}_f$ be the (infinite) set of all relations invariant under f. $\mathrm{CSP}(\mathcal{D}_f)$ is called a "global" problem encompassing all of the "local" problems $\mathrm{CSP}(\mathcal{D})$ where $\mathcal{D}$ ranges over finite subsets of $\mathcal{D}_f$.

If the polymorphism clone $\mathbf{D}$ of $\mathcal{D}_f$ contains a k-edge operation, then the few subpowers algorithm as formulated in [74] actually solves the global problem $\mathrm{CSP}(\mathcal{D}_f)$ in polynomial

[13] This argument is due to Dalmau [32, 53].

time.[14] Conversely, suppose there exists a polynomial-time algorithm which solves the global problem CSP($\mathcal{D}_f$) *and* broadly follows the "small generating sets" algorithm-idea outlined at the beginning of Subsection 6.2. An essential requirement of this algorithm-idea is that

> There is a polynomial $p(n)$ such that if $R \subseteq D^n$ is a subpower of $\mathbf{D}$ *and* R occurs as R_i in equation (26), for some instance of CSP($\mathcal{D}_f$), then R has a generating set of size at most $p(n)$.

Because of how $\mathcal{D}_f$ is defined, *every* subpower $R \subseteq D^n$ of $\mathbf{D}$ can arise as R_i in equation (26) for some instance of CSP($\mathcal{D}_f$) (for the simple reason that every subpower R belongs to $\mathcal{D}_f$). Hence the displayed requirement and Theorem 64 imply $\mathbf{D}$ must contain a k-edge operation. It is in this sense that polymorphism clones having a k-edge operation form the natural "limit" of the "small generating sets" algorithm-idea outlined at the beginning of Subsection 6.2.

However if one is primarily interested (as we are) in CSP over finite constraint languages, rather than in global problems, then the limit of the "small generating sets" algorithm-idea is not settled. If $\mathcal{D}$ is finite, then the subpowers of the polymorphism clone which arise as R_i in equation (26) are precisely the relations which are pp-definable from $\mathcal{D}$ without using $\exists$. It is not known if there exists a finite constraint language $\mathcal{D}$ which does not have a k-edge polymorphism for any k, and yet which has the property that every $\exists$-free pp-defined relation R has a small generating set. If such a constraint language $\mathcal{D}$ exists, it might be a candidate for a "small generating sets" algorithm more general than the few subpowers algorithm.

6.4 Combining algorithms

Suppose $\mathcal{D}$ is idempotent, has a Taylor polymorphism, but does not have polymorphisms as described in Theorem 44 implying bounded width (see Theorem 62), nor does $\mathcal{D}$ have an edge polymorphism implying few subpowers. What tools are available to solve CSP($\mathcal{D}$)?

Starting from an instance $P = (V, D, \mathcal{C})$ of CSP($\mathcal{D}$), one strategy is to "shrink" P to a 1-minimal CSP instance Q satisfying the following properties:

1. You can prove that Q has a solution if and only if P does.
2. The constraint relations of Q are pp-definable from $\mathcal{D}$.
3. For each $x \in V$, if $Q_x(x)$ is the unique unary constraint of Q on x, then Q_x is a proper subset of D.

Suppose now that there exists a polymorphism f of $\mathcal{D}$ such that for each $x \in V$, the restriction of f to Q_x is an edge operation. Then the few subpowers algorithm, suitably adapted, can determine whether Q (and hence P) has a solution. Similarly, if $\mathcal{D}$ has polymorphisms which, when restricted to each Q_x, satisfy identities implying bounded width, then the (2,3)-minimality algorithm will correctly tell whether Q (and hence P) has a solution.

Thus a possible strategy is to shrink P to the point where some polymorphism(s) of $\mathcal{D}$ become "nice" when restricted to the new domain of each variable. This is essentially the strategy followed (along with much deeper tricks) to prove the results described in this subsection. The first result of this kind is the following result of Bulatov [38].

▶ **Theorem 69.** *Suppose $\mathcal{D}$ has a Taylor polymorphism and $|D| = 3$. Then* CSP($\mathcal{D}$) *is solvable in polynomial time. Hence the Dichotomy Conjecture holds for constraint languages with domains of size 3.*

[14] For this result it is important to maintain the convention that each constraint relation is given by a list of its members – not, for example, by a small generating set.

Reading [38] is not for the faint at heart – the core argument is over 40 pages of dense mathematical argument involving consideration of many cases.

The second result that we want to mention is the solution [39, 7, 41], due originally to Bulatov, of the Dichotomy Conjecture in the "conservative" case.

▶ **Theorem 70.** *Suppose $\mathcal{D}$ includes every nonempty subset of D as a unary relation. If $\mathcal{D}$ has a Taylor polymorphism, then* CSP($\mathcal{D}$) *is solvable in polynomial time.*

An operation f is called *conservative* if $f(a_1, \ldots, a_n) \in \{a_1, \ldots, a_n\}$ for all $a_i \in D$. The above theorem can be equivalently re-stated as "if a constraint language $\mathcal{D}$ has a conservative Taylor polymorphism then CSP($\mathcal{D}$) is tractable." The first proof from [39] was quite long and involved, but the more recent papers [7, 41] give two different much shorter proofs.

Technically, Bulatov's approach in [39, 41] is via a careful analysis of a colored graph associated with a constraint language (see [36, 42]). The domain of this graph is the same as the domain of the languages, and the colored edges reflect the local structure of polymorphisms. The approach in [7] is based on absorption theory, see survey [16].

See also [43] for a brief overview of other attempts, as well as a new approach, to combine known algorithms for solving CSP($\mathcal{D}$).

7 Conclusion

We have seen that the complexity of the decision problem for CSP over a fixed constraint language on a finite domain depends on "higher arity symmetries" – polymorphisms of the language, and more specifically, on the identities satisfied by these polymorphisms. Significant progress has been achieved using this insight, but the main problem, the dichotomy conjecture, is still open. The view shared by many experts at the Dagstuhl seminar (to which this volume is a follow-up) is that the main obstacle towards further progress is the insufficient understanding of the convoluted ways in which system of linear equations can appear in CSP instances.

The authors are often being asked to give a specific example of a (preferably small) constraint language $\mathcal{D}$ that has a Taylor polymorphism, but such that CSP($\mathcal{D}$) is not yet known to be tractable. A computer-assisted analysis of small digraphs and their polymorphisms (with respect to the taxonomy from Fig.1) can be found in [22]. There is an example of a 6-element digraph $\mathcal{D}$ there, whose singleton expansion has a Taylor polymorphism, but such that neither the bounded width algorithm nor the few subpowers algorithm alone can solve CSP($\mathcal{D}$). However, it was shown [103] that an ad hoc combination of the two algorithms solves it. It is the authors' belief that such explicit examples are not easy to find, but they tend not to provide useful insights into how to overcome the difficulties in resolving the dichotomy conjecture.

Polymorphisms have been successfully applied to other variants of CSP over a fixed constraint language. For the (exact) counting problem, the dichotomy has been proved in [40] and then substantially simplified in [61], with polymorphisms (and universal algebra in general) playing a considerable role. This is discussed in survey [81] in this volume, along with many developments in complexity classification of approximate counting problems and for computing partition functions, where there are still many open problems.

A generalization of the algebraic theory for the exact optimization problem (where the goal is find an optimal solution), and for the more general valued CSPs, was given in [51]. The so-called fractional polymorphisms, which are special probability distributions on polymorphisms, play a key role there. Very strong classification results are known in this

direction [88, 89, 117], see survey [92] in this volume. In essence, complexity classifications here are complete, but there are many open questions regarding approximate optimization.

The two different views on optimization – maximize satisfaction or minimize dissatisfaction – are obviously equivalent when the goal is to find an optimal solution. However, they exhibit very different behavior with respect to approximation. For the emerging applicability of polymorphisms for the analysis of approximability of the maximization version, see [29, 30]. For the minimization version, there is a series of results about the so-called robust approximation algorithms for CSPs where polymorphisms (and their taxonomy) play a significant role. The full characterization of constraint languages admitting a robust algorithm is known [15] (see Theorem 44), but the refined classification (taking the quality of approximation into account) is far from complete, see [57, 58, 55].

The so-called weak polymorphisms have recently been applied to the so-called promise CSPs [5, 27, 28], with motivation coming from the study of inapproximability. An interesting feature of weak polymorphisms is that they cannot be composed, and yet they can be useful for complexity analysis.

We have only discussed languages with finite domains. The algebraic theory extends to interesting subclasses of the infinite domain CSP; see surveys [24, 25, 107] and recent results [20, 21]. This area contains very many open problems.

Is the polymorphism approach only applicable to CSPs over fixed languages? Or are we merely seeing a piece of a bigger theory?

Polymorphisms have recently been applied, for example, in a non-CSP context of social choice theory [114]. Are there other applications of polymorphisms beyond algebra and CSP?

Acknowledgment

The authors thank Eric Allender for explaining state-of-the-art knowledge about relationships between complexity classes NL and $\mathrm{Mod}_p\mathrm{L}$. The authors also thank anonymous referees for providing many useful comments. L. Barto gratefully acknowledges the support of the Grant Agency of the Czech Republic, grant GAČR 13-01832S. R. Willard gratefully acknowledges the support of the Natural Sciences and Engineering Research Council of Canada.

References

1 Eric Allender, Michael Bauland, Neil Immerman, Henning Schnoor, and Heribert Vollmer. The complexity of satisfiability problems: Refining Schaefer's theorem. *Journal of Computer and System Sciences*, 75(4):245 – 254, 2009.

2 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting uniform and nonuniform upper bounds. *J. Comput. Syst. Sci.*, 59(2):164–181, 1999.

3 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

4 Albert Atserias, Andrei A. Bulatov, and Anuj Dawar. Affine systems of equations and counting infinitary logic. *Theor. Comput. Sci.*, 410(18):1666–1683, 2009.

5 Per Austrin, Johan Håstad, and Venkatesan Guruswami. $(2 + \epsilon)$-Sat is NP-hard. In *Proceedings, Foundations of Computer Science (FOCS)*, pages 1–10, 2014.

6 Kirby A. Baker and Alden F. Pixley. Polynomial interpolation and the Chinese remainder theorem for algebraic systems. *Mathematische Zeitschrift*, 143:165–174, 1975.

7 Libor Barto. The dichotomy for conservative constraint satisfaction problems revisited. In *Proceedings, Logic in Computer Science (LICS)*, pages 301–310, 2011.

8 Libor Barto. Finitely related algebras in congruence distributive varieties have near unanimity terms. *Canad. J. Math.*, 65(1):3–21, 2013.

9 Libor Barto. Finitely related algebras in congruence modular varieties have few subpowers. to appear in *J. European Math. Soc.*, 2015.

10 Libor Barto. The collapse of the bounded width hierarchy. *J. Log. Comput.*, 26(3):923–943, 2016.

11 Libor Barto and Marcin Kozik. Congruence distributivity implies bounded width. *SIAM Journal on Computing*, 39(4):1531–1542, 2009.

12 Libor Barto and Marcin Kozik. Constraint satisfaction problems of bounded width. In *Proceedings, Foundations of Computer Science (FOCS)*, pages 595–603, 2009.

13 Libor Barto and Marcin Kozik. Absorbing subalgebras, cyclic terms, and the constraint satisfaction problem. *Logical Methods in Computer Science*, 8(1), 2012.

14 Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61(1):3:1–3:19, January 2014.

15 Libor Barto and Marcin Kozik. Robustly solvable constraint satisfaction problems. *SIAM Journal on Computing*, 45(4):1646–1669, 2016.

16 Libor Barto and Marcin Kozik. Absorption in universal algebra and CSP. In *The Constraint Satisfaction Problem: Complexity and Approximability*, pages 45–78. 2017.

17 Libor Barto, Marcin Kozik, and Todd 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, 2008/09.

18 Libor Barto, Marcin Kozik, and David Stanovský. Mal'tsev conditions, lack of absorption, and solvability. *Algebra universalis*, 74(1):185–206, 2015.

19 Libor Barto, Marcin Kozik, and Ross Willard. Near unanimity constraints have bounded pathwidth duality. In *Proceedings, Logic in Computer Science (LICS)*, pages 125–134, 2012.

20 Libor Barto, Jakub Opršal, and Michael Pinsker. The wonderland of reflections. to appear in *Israel J. Math.*, 2016.

21 Libor Barto and Michael Pinsker. The algebraic dichotomy conjecture for infinite domain constraint satisfaction problems. In *Proceedings, Logic in Computer Science (LICS)*, pages 615–622, 2016.

22 Libor Barto and David Stanovský. Polymorphisms of small digraphs. *Novi Sad J. Math.*, 40(2):95–109, 2010.

23 Joel Berman, Pawel Idziak, Petar Marković, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Varieties with few subalgebras of powers. *Transactions of The American Mathematical Society*, 362:1445–1473, 2010.

24 Manuel Bodirsky. Constraint satisfaction problems with infinite templates. In Nadia Creignou, Phokion G. Kolaitis, and Heribert Vollmer, editors, *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 196–228. Springer, 2008.

25 Manuel Bodirsky and Marcello Mamino. Constraint satisfaction problems over numeric domains. In *The Constraint Satisfaction Problem: Complexity and Approximability*, pages 79–111. 2017.

26 V. G. Bodnarčuk, L. A. Kalužnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. I, II. *Kibernetika (Kiev)*, (3):1–10; ibid. 1969, no. 5, 1–9, 1969.

27 Joshua Brakensiek and Venkatesan Guruswami. New hardness results for graph and hypergraph colorings. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:29, 2016.

28 Joshua Brakensiek and Venkatesan Guruswami. Promise constraint satisfaction: Algebraic structure and a symmetric boolean dichotomy. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:183, 2016.

29 Jonah Brown-Cohen and Prasad Raghavendra. Combinatorial optimization algorithms via polymorphisms. *CoRR*, abs/1501.01598, 2015.

30 Jonah Brown-Cohen and Prasad Raghavendra. Correlation decay and tractability of CSPs. In *Proceedings, Automata, Languages, and Programming (ICALP)*, pages 79:1–79:13, 2016.

31 Andrei Bulatov. Bounded relational width. manuscript, 2009.

32 Andrei Bulatov and Víctor Dalmau. A simple algorithm for Mal′tsev constraints. *SIAM J. Comput.*, 36(1):16–27 (electronic), 2006.

33 Andrei Bulatov and Peter Jeavons. Algebraic structures in combinatorial problems. Technical Report MATH-AL-4-2001, Technische Universität Dresden, 2001.

34 Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34:720–742, March 2005.

35 Andrei A. Bulatov. Mal'tsev constraints are tractable. *Electronic Colloquium on Computational Complexity (ECCC)*, (034), 2002.

36 Andrei A. Bulatov. A graph of a relational structure and constraint satisfaction problems. In *Proceedings, Logic in Computer Science (LICS)*, pages 448–457, 2004.

37 Andrei A. Bulatov. Combinatorial problems raised from 2-semilattices. *J. Algebra*, 298(2):321–339, 2006.

38 Andrei A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. ACM*, 53(1):66–120 (electronic), 2006.

39 Andrei A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Logic*, 12(4):24:1–24:66, July 2011.

40 Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *J. ACM*, 60(5):34:1–34:41, October 2013.

41 Andrei A. Bulatov. Conservative constraint satisfaction re-revisited. *J. Comput. System Sci.*, 82(2):347–356, 2016.

42 Andrei A. Bulatov. Graphs of relational structures: restricted types. In *Proceedings, Logic in Computer Science (LICS)*, pages 642–651, 2016.

43 Andrei A. Bulatov. Constraint Satisfaction Problems over semilattice block Mal'tsev algebras. *CoRR*, arXiv:1701.02623, 2017.

44 Andrei A. Bulatov, Andrei Krokhin, and Benoit Larose. Complexity of constraints. chapter Dualities for Constraint Satisfaction Problems, pages 93–124. Springer-Verlag, Berlin, Heidelberg, 2008.

45 Jakub Bulin, Dejan Delic, Marcel Jackson, and Todd Niven. A finer reduction of constraint problems to digraphs. *Logical Methods in Computer Science*, 11(4), 2015.

46 Gerhard Buntrock, Carsten Damm, Ulrich Hertrampf, and Christoph Meinel. Structure and importance of logspace-mod class. *Mathematical systems theory*, 25(3):223–237, 1992.

47 Catarina Carvalho, Víctor Dalmau, and Andrei Krokhin. CSP duality and trees of bounded pathwidth. *Theoretical Computer Science*, 411(34-36):3188–3208, 2010.

48 Catarina Carvalho, Víctor Dalmau, Petar Marković, and Miklós Maróti. CD(4) has bounded width. *Algebra Universalis*, 60(3):293–307, 2009.

49 Hubie Chen. Meditations on quantified constraint satisfaction. In *Logic and Program Semantics*, pages 35–49, 2012.

50 Hubie Chen and Benoit Larose. Asking the metaquestions in constraint tractability. arXiv:1604.00932, April 2016.

51 David A. Cohen, Martin C. Cooper, Páidí Creed, Peter Jeavons, and Stanislav Živný. An algebraic theory of complexity for discrete optimisation. *SIAM Journal on Computing*, 42(5):1915–1939, 2013.

52 Víctor Dalmau. Linear Datalog and bounded path duality for relational structures. *Logical Methods in Computer Science*, 1(1), 2005.

53 Víctor Dalmau. Generalized majority-minority operations are tractable. *Log. Methods Comput. Sci.*, 2(4):4:1, 14, 2006.

54 Víctor Dalmau, László Egri, Pavol Hell, Benoit Larose, and Arash Rafiey. Descriptive complexity of list h-coloring problems in logspace: A refined dichotomy. In *Proceedings, Logic in Computer Science (LICS)*, pages 487–498, 2015.

55 Víctor Dalmau, Marcin Kozik, Andrei Krokhin, Konstantin Makarychev, Yury Makarychev, and Jakub Opršal. Robust algorithms with polynomial loss for near-unanimity CSPs. In *Proceedings, Discrete Algorithms (SODA)*, pages 340–357, 2017.

56 Víctor Dalmau and Andrei Krokhin. Majority constraints have bounded pathwidth duality. *European Journal of Combinatorics*, 29(4):821–837, 2008.

57 Víctor Dalmau and Andrei Krokhin. Robust satisfiability for CSPs: Hardness and algorithmic results. *ACM Trans. Comput. Theory*, 5(4):15:1–15:25, November 2013.

58 Víctor Dalmau, Andrei Krokhin, and Rajsekar Manokaran. Towards a characterization of constant-factor approximable Min CSPs. In *Proceedings, Discrete Algorithms (SODA)*, pages 847–857, 2015.

59 Víctor Dalmau and Benoit Larose. Maltsev + datalog –> symmetric datalog. In *Proceedings, Logic in Computer Science (LICS)*, pages 297–306, 2008.

60 Víctor Dalmau and Justin Pearson. Closure functions and width 1 problems. In *Proceedings, Principles and Practice of Constraint Programming (CP)*, pages 159–173, 1999.

61 Martin E. Dyer and David Richerby. An effective dichotomy for the counting constraint satisfaction problem. *SIAM J. Comput.*, 42(3):1245–1274, 2013.

62 László Egri, Benoit Larose, and Pascal Tesson. Symmetric datalog and constraint satisfaction problems in logspace. In *Proceedings, Logic in Computer Science (LICS)*, pages 193–202, 2007.

63 Tomas Feder and Pavol Hell. List homomorphisms to reflexive graphs. *J. Combin. Theory Ser. B*, 72(2):236–250, 1998.

64 Tomas Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Combinatorica*, 19(4):487–505, 1999.

65 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, 1998.

66 Ralph Freese and Ralph McKenzie. Maltsev families of varieties closed under join or Maltsev product. *Algebra Universalis*, in press.

67 David Geiger. Closed systems of functions and predicates. *Pacific J. Math.*, 27:95–100, 1968.

68 J. Hagemann and A. Mitschke. On n-permutable congruences. *Algebra Universalis*, 3(1):8–12, 1973.

69 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48:798–859, July 2001.

70 Johan Håstad. On the efficient approximability of constraint satisfaction problems. In *Surveys in combinatorics 2007*, volume 346 of *London Math. Soc. Lecture Note Ser.*, pages 201–221. Cambridge Univ. Press, Cambridge, 2007.

71 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Combin. Theory Ser. B*, 48(1):92–110, 1990.

72 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*, volume 28 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2004.

73 David Hobby and Ralph McKenzie. *The structure of finite algebras*, volume 76 of *Contemporary Mathematics*. American Mathematical Society, Providence, RI, 1988.

74 Pawel M. Idziak, Petar Markovic, Ralph McKenzie, Matthew Valeriote, and Ross Willard. Tractability and learnability arising from algebras with few subpowers. *SIAM J. Comput.*, 39(7):3023–3037, 2010.

75 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, October 1988.

76 Marcel Jackson, Tomasz Kowalski, and Todd Niven. Complexity and polymorphisms for digraph constraint problems under some basic constructions. *International Journal of Algebra and Computation*, 26(07):1395–1433, 2016.

77 Peter Jeavons. Constructing constraints. In *Proceedings, Principles and Practice of Constraint Programming (CP)*, pages 2–16, 1998.

78 Peter Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200(1–2):185 – 204, 1998.

79 Peter Jeavons, David Cohen, and Martin Cooper. Constraints, consistency and closure. *Artificial Intelligence*, 101(1-2):251–265, 1998.

80 Peter Jeavons, David Cohen, and Marc Gyssens. Closure properties of constraints. *J. ACM*, 44(4):527–548, 1997.

81 Mark Jerrum. Counting constraint satisfaction problems. In *The Constraint Satisfaction Problem: Complexity and Approximability*, pages 201–228. 2017.

82 Jelena Jovanović, Petar Marković, Ralph McKenzie, and Matthew Moore. Optimal strong Mal'cev conditions for congruence meet-semidistributivity in locally finite varieties. *Algebra universalis*, 76(3):305–325, 2016.

83 Alexandr Kazda. Maltsev digraphs have a majority polymorphism. *European J. Combin.*, 32(3):390–397, 2011.

84 Alexandr Kazda. n-permutability and linear datalog implies symmetric datalog. *CoRR*, abs/1508.05766, 2015.

85 Keith Kearnes, Petar Marković, and Ralph McKenzie. Optimal strong Mal'cev conditions for omitting type 1 in locally finite varieties. *Algebra Universalis*, 72(1):91–100, 2014.

86 Emil Kiss and Matthew Valeriote. On tractability and congruence distributivity. *Log. Methods Comput. Sci.*, 3(2):2:6, 20 pp. (electronic), 2007.

87 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000.

88 Vladimir Kolmogorov, Andrei Krokhin, and Michal Rolinek. The complexity of general-valued CSPs. In *Proceedings, Foundations of Computer Science (FOCS)*, pages 1246–1258, 2015.

89 Vladimir Kolmogorov, Johan Thapper, and Stanislav Živný. The power of linear programming for general-valued CSPs. *SIAM J. Comput.*, 44(1):1–36, 2015.

90 Marcin Kozik. Weak consistency notions for all the CSPs of bounded width. In *Proceedings, Logic in Computer Science (LICS)*, pages 633–641, 2016.

91 Marcin Kozik, Andrei Krokhin, Matt Valeriote, and Ross Willard. Characterizations of several Maltsev conditions. *Algebra universalis*, 73(3):205–224, 2015.

92 Andrei Krokhin and Stanislav Živný. The complexity of valued CSPs. In *The Constraint Satisfaction Problem: Complexity and Approximability*, pages 229–261, 2017.

93 Gabor Kun, Ryan O'Donnell, Suguru Tamaki, Yuichi Yoshida, and Yuan Zhou. Linear programming, width-1 CSPs, and robust satisfaction. In *Proceedings, Innovations in Theoretical Computer Science (ITCS)*, pages 484–495, 2012.

94 Gábor Kun and Mario Szegedy. A new line of attack on the dichotomy conjecture. *European Journal of Combinatorics*, 52, Part B:338 – 367, 2016. Special Issue: Recent Advances in Graphs and Analysis.

95 Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22:155–171, 1975.

96 Benoît Larose. Algebra and the complexity of digraph CSPs: a survey. In *The Constraint Satisfaction Problem: Complexity and Approximability*, pages 263–282. 2016.

97 Benoît Larose and Pascal Tesson. Universal algebra and hardness results for constraint satisfaction problems. *Theor. Comput. Sci.*, 410:1629–1647, April 2009.

98 Benoit Larose, Matt Valeriote, and László Zádori. Omitting types, bounded width and the ability to count. *Internat. J. Algebra Comput.*, 19(5):647–668, 2009.

99 Benoit Larose and László Zádori. Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras. *Internat. J. Algebra Comput.*, 16(3):563–581, 2006.

100 Benoit Larose and László Zádori. Bounded width problems and algebras. *Algebra Universalis*, 56(3-4):439–466, 2007.

101 Florent Madelaine and Barnaby Martin. A tetrachotomy for positive first-order logic without equality. In *Proceedings, Logic in Computer Science (LICS)*, pages 311–320, 2011.

102 Konstantin Makarychev and Yuri Makarychev. Approximation algorithms for CSPs. In *The Constraint Satisfaction Problem: Complexity and Approximability*, pages 283–320. 2017.

103 Petar Marković. Private communication, 2011.

104 Miklós Maróti and Ralph McKenzie. Existence theorems for weakly symmetric operations. *Algebra Universalis*, 59(3-4):463–489, 2008.

105 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. ACM*, 60(6):42:1–42:51, November 2013.

106 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Reading, MA, 1994.

107 Michael Pinsker. Algebraic and model theoretic methods in constraint satisfaction. *CoRR*, abs/1507.00931, 2015.

108 E.L. Post. *The Two-Valued Iterative Systems of Mathematical Logic. (AM-5)*. Annals of Mathematics Studies. Princeton University Press, 1941.

109 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, September 2008.

110 Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000.

111 Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

112 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings, Theory of Computation (STOC)*, pages 216–226, 1978.

113 Mark H. Siggers. A strong Mal'cev condition for locally finite varieties omitting the unary type. *Algebra universalis*, 64(1-2):15–20, 2010.

114 Mario Szegedy and Yixin Xu. Impossibility theorems and the universal algebraic toolkit. *CoRR*, abs/1506.01315, 2015.

115 Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Inf.*, 26(3):279–284, November 1988.

116 Walter Taylor. Varieties obeying homotopy laws. *Canad. J. Math.*, 29(3):498–527, 1977.

117 Johan Thapper and Stanislav Živný. The complexity of finite-valued CSPs. *J. ACM*, 63(4):37:1–37:33, 2016.

118 Ross Willard. Testing expressibility is hard. In *Proceedings, Principles and Practice of Constraint Programming (CP)*, pages 9–23, 2010.