CONSTRAINT SATISFACTION PROBLEM: WHAT MAKES THE PROBLEM EASY

DMITRIY ZHUK

ABSTRACT

The Constraint Satisfaction Problem is the problem of deciding whether there is an assignment to a set of variables subject to some specified constraints. Systems of linear equations, graph coloring, and many other combinatorial problems can be expressed as Constraint Satisfaction Problems for some constraint language. In 1993 it was conjectured that for any constraint language the problem is either solvable in polynomial time, or NP-complete, and for many years this conjecture was the main open question in the area. After this conjecture was resolved in 2017, we finally can say what makes the problem hard and what makes the problem easy. In the first part of the paper, we give an elementary introduction to the area, explaining how the full classification appeared and why it is formulated in terms of polymorphisms. We discuss what makes the problem NP-hard, what makes the problem solvable by local consistency checking, and explain briefly the main idea of one of the two proofs of the conjecture. The second part of the paper is devoted to the extension of the CSP, called Quantified CSP, where we allow using both universal and existential quantifiers. Finally, we discuss briefly other variants of the CSP, as well as some open questions related to them.

MATHEMATICS SUBJECT CLASSIFICATION 2020

Primary 3B70; Secondary 08A70, 68Q17, 03D15, 68T27, 03B10

KEYWORDS

Constraint satisfaction problem, computational complexity, CSP dichotomy, quantified constraints

1. INTRODUCTION

Probably the main question in theoretical computer science is to understand why some computational problems are easy (solvable in polynomial time) while others are difficult (NP-hard, PSpace-hard, and so on). What is the difference between P and NP? Why a system of linear equations can be solved in polynomial time by the Gaussian elimination but we cannot check whether a graph is 3-colorable in polynomial time (if we believe that P $\neq$ NP). What is the principal difference between these two problems? To work on this question, first we would like to classify the problems by whether they are solvable in polynomial time (*tractable*) or NP-complete. Even for very simple decision problems, sometimes we do not know the answer.

For example, a system of linear equations in $\mathbb{Z}_2$ can be solved by Gaussian elimination, but if we are allowed to add one linear equation with usual sum for integers then the problem becomes NP-complete [26]. Surprisingly, the complexity is not known if we can add one equation modulo 24 to a system of linear equations in $\mathbb{Z}_2$ (variables are still from $\{0, 1\}$) [17]. In the paper we give a formal definition to such problems and discuss why some of them can be solved in polynomial time, while others are NP-hard.

2. CONSTRAINT SATISFACTION PROBLEM

The above problems are known as the Constraint Satisfaction Problem (CSP), which is the problem of deciding whether there is an assignment to a set of variables subject to some specified constraints. Formally, the *Constraint Satisfaction Problem* is defined as a triple $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$, where

- $\mathbf{X} = \{x_1, \dots, x_n\}$ is a set of variables,
- $\mathbf{D} = \{D_1, \dots, D_n\}$ is a set of the respective domains,
- $\mathbf{C} = \{C_1, \dots, C_m\}$ is a set of constraints,

where each variable x_i can take on values in the nonempty domain D_i, every *constraint* $C_j \in \mathbf{C}$ is a pair (t_j, R_j) where t_j is a tuple of variables of length m_j, called the *constraint scope*, and R_j is an m_j-ary relation on the corresponding domains, called the *constraint relation*.

The question is whether there exists *a solution* to $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$, that is, a mapping that assigns a value from D_i to every variable x_i such that for each constraint C_j the image of the constraint scope is a member of the constraint relation.

To simplify the presentation, we assume that the domain of every variable is a finite set A. We also assume that all the relations are from a set Γ, which we call *the constraint language*. Then the Constraint Satisfaction Problem over a constraint language Γ, denoted CSP(Γ), is the following decision problem: given a conjunctive formula

$$R_1(v_{1,1}, \dots, v_{1,n_1}) \wedge \dots \wedge R_s(v_{s,1}, \dots, v_{s,n_s}),$$

where $R_1, \dots, R_s \in \Gamma$, and $v_{i,j} \in \{x_1, \dots, x_n\}$ for every i, j, decide whether this formula is satisfiable. Note that in the paper we do not distinguish between relations and predicates, and in the previous formula we write relations meaning predicates.

2.1. Examples

It is well known that many combinatorial problems can be expressed as CSP(Γ) for some constraint language Γ. Moreover, for some Γ the corresponding decision problem can be solved in polynomial time; while for others it is NP-complete. It was conjectured that CSP(Γ) is either in P or NP-complete [29]. Let us consider several examples.

System of linear equations. Let $A = \{0, 1\}$ and

$$\Gamma = \{a_1x_1 + a_2x_2 + \dots + a_kx_k = a_0 \mid a_0, a_1, \dots, a_k \in \mathbb{Z}_2\},$$

i.e., Γ consists of all linear equations in the field $\mathbb{Z}_2$. Then CSP(Γ) is equivalent to the problem of solving a system of linear equations, which is solvable by the Gaussian elimination in polynomial time, thus, CSP(Γ) is in P.

Graph 2-coloring. To color a graph using two colors, we just need to choose a color of every vertex so that adjacent vertices have different colors. We assign a variable to each vertex, and encode the two colors with 0 and 1. For an edge between the ith and jth vertices, we add the constraint $x_i \neq x_j$. For instance, the 5-cycle is equivalent to the CSP instance

$$(x_1 \neq x_2) \wedge (x_2 \neq x_3) \wedge (x_3 \neq x_4) \wedge (x_4 \neq x_5) \wedge (x_5 \neq x_1).$$

Hence, the problem of graph 2-coloring is equivalent to CSP(Γ) for $A = \{0, 1\}$ and $\Gamma = \{\neq\}$. This problem can be solved locally. We choose a color of some vertex, then we color their neighbors with a different color, and so on. Either we will color all the vertices, or we will find an odd cycle, which means that the graph is not colorable using two colors. Thus, this problem is solvable in polynomial time.

Graph 3-coloring. Similarly, the problem of coloring a graph using 3 colors is equivalent to CSP(Γ) for $A = \{0, 1, 2\}$ and $\Gamma = \{\neq\}$. Unlike the graph 2-coloring, this problem is known to be NP-complete [1].

NAE-satisfability and 1IN3-satisfability. Suppose $A = \{0, 1\}$. NAE is the ternary not-all-equal relation, that is, NAE $= \{0, 1\}^3 \setminus \{(0, 0, 0), (1, 1, 1)\}$. 1IN3 is the ternary 1-in-3 relation, that is, 1IN3 $= \{(0, 0, 1), (0, 1, 0), (1, 0, 0)\}$. As it is known [40], both CSP($\{$NAE$\}$) and CSP($\{$1IN3$\}$) are NP-complete.

The main goal of this paper is to explain why the first two examples are in P, while the others are NP-hard.

2.2. Reduction from one language to another

To prove the hardness result, we usually reduce a problem to a known NP-hard problem. Let us show how we can go from one constraint language to another. CSP(Γ) can be viewed as the problem of evaluating a sentence

$$\exists x_1 \dots \exists x_n \big(R_1(v_{1,1}, \dots, v_{1,n_1}) \wedge \dots \wedge R_s(v_{s,1}, \dots, v_{s,n_s})\big), \tag{2.1}$$

where all variables are existentially quantified. Hence, if we could express one language using conjunctions and existential quantifiers from another language, then we get a reduction from one CSP to another. Let us explain how it works on a concrete example.

Let $\mathrm{NA1} = \{0,1\}^3 \setminus \{(1,1,1)\}$, that is, a ternary relation that holds whenever not all elements are 1. Let $A = \{0,1\}$, $\Gamma_1 = \{\mathrm{NA1}, \neq\}$, and $\Gamma_2 = \{\mathrm{1IN3}\}$. Let us show that $\mathrm{CSP}(\Gamma_1)$ and $\mathrm{CSP}(\Gamma_2)$ are (polynomially) equivalent. We may check that

$$(x \neq y) = \exists u \exists v \ \mathrm{1IN3}(x, y, u) \wedge \mathrm{1IN3}(u, u, v). \tag{2.2}$$

If fact, from $\mathrm{1IN3}(u, u, v)$ we derive that $u = 0$, hence $x \neq y$. Similarly, we have

$$\begin{aligned} \mathrm{NA1}(x, y, z) = {} & \exists x' \exists y' \exists z' \exists x'' \exists y'' \exists z'' \ \mathrm{1IN3}(x', y', z') \\ & \wedge \mathrm{1IN3}(x, x', x'') \wedge \mathrm{1IN3}(y, y', y'') \wedge \mathrm{1IN3}(z, z', z''). \end{aligned} \tag{2.3}$$

If $x = y = z = 1$, then $x' = y' = z' = 0$, which contradicts $\mathrm{1IN3}(x', y', z')$. In all other cases, we can find an appropriate assignment.

Any instance of $\mathrm{CSP}(\Gamma_1)$ can be reduced to an instance of $\mathrm{CSP}(\Gamma_2)$ in the following way. We replace each constraint $(x_i \neq x_j)$ by the right-hand side of (2.2) introducing two new variables. Also, we replace each constraint $\mathrm{NA1}(x_i, x_j, x_k)$ by the right-hand side of (2.3) introducing six new variables. This reduction is obviously polynomial (and even log-space). Similarly, we have

$$\begin{aligned} \mathrm{1IN3}(x, y, z) = {} & \exists x' \exists y' \exists z' \big(\mathrm{NA1}(x, y, y) \wedge \mathrm{NA1}(y, z, z) \wedge \mathrm{NA1}(z, x, x) \\ & \wedge (x \neq x') \wedge (y \neq y') \wedge (z \neq z') \wedge \mathrm{NA1}(x', y', z')\big), \end{aligned}$$

which implies a polynomial reduction from $\mathrm{CSP}(\Gamma_2)$ to $\mathrm{CSP}(\Gamma_1)$.

Let us give a formal definition for the above reduction. A formula of the form $\exists y_1 \dots \exists y_n \Phi$, where Φ is a conjunction of relations from Γ is called *a positive primitive formula (pp-formula) over* Γ. If $R(x_1, \dots, x_n) = \exists y_1 \dots \exists y_n \Phi$, then we say that R is *pp-defined* by this formula, and $\exists y_1 \dots \exists y_n \Phi$ is called its *pp-definition*.

Theorem 2.1 ([35]). *Suppose* Γ_1 *and* Γ_2 *are finite constraint languages such that each relation from* Γ_1 *is pp-definable over* Γ_2. *Then* $\mathrm{CSP}(\Gamma_1)$ *is polynomial time reducible to* $\mathrm{CSP}(\Gamma_2)$.

2.3. Polymorphisms as invariants

If we can pp-define a relation R from a constraint language Γ and $\mathrm{CSP}(\{R\})$ is NP-hard, then $\mathrm{CSP}(\Gamma)$ is also NP-hard. How to show that such a relation cannot be pp-defined? To prove that something cannot be done, we usually find some fundamental property (invariant) that is satisfied by anything we can obtain. For the relations, the operations play the role of invariants.

We say that an operation $f : A^n \to A$ *preserves* a relation R of arity m if for any tuples $(a_{1,1}, \dots, a_{1,m}), \dots, (a_{n,1}, \dots, a_{n,m}) \in R$ the tuple

$$\big(f(a_{1,1}, \dots, a_{n,1}), \dots, f(a_{1,m}, \dots, a_{n,m})\big)$$

is in R. In this case we also say that f is a *polymorphism* of R, and R is an *invariant* of f. We say that an operation *preserves a set of relations* Γ if it preserves every relation in Γ. In this case we also write f is *a polymorphism of* Γ or $f \in \mathrm{Pol}(\Gamma)$. It can be easily checked that if f preserves Γ, then f preserves any relation R pp-definable from Γ. Moreover, we can show [15, 31] that $\mathrm{Pol}(\Gamma_1) \subseteq \mathrm{Pol}(\Gamma_2)$ if and only if Γ_2 is pp-definable over Γ_1, which means that the complexity of $\mathrm{CSP}(\Gamma)$ depends only on $\mathrm{Pol}(\Gamma)$.

Example 1. Let R be the linear order relation on $\{0, 1, 2\}$, i.e.,

$$R = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 1 & 2 & 1 & 2 & 2 \end{pmatrix},$$

where columns are tuples from the relation. Then "an n-ary operation f preserves R" means that for all

$$\begin{pmatrix} a_1 \\ b_1 \end{pmatrix}, \dots, \begin{pmatrix} a_n \\ b_n \end{pmatrix} \in R,$$

that is, $a_i \leq b_i$, we have

$$f\begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix} := \begin{pmatrix} f(a_1, \dots, a_n) \\ f(b_1, \dots, b_n) \end{pmatrix} \in R,$$

that is, $f(a_1, \dots, a_n) \leq f(b_1, \dots, b_n)$. In other words, f is monotonic. For instance, the operations max and min are monotonic. By the above observation, we know that any relation pp-definable from R is also preserved by min and max.

Example 2. Let $A = \{0, 1\}$. Let us show that 1IN3 cannot be pp-defined from NA1 and $x \leq y$. We can check that the conjunction $x \wedge y$ (an operation on $\{0, 1\}$) preserves both NA1 and $x \leq y$. However, $x \wedge y$ does not preserve 1IN3 as we have

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \wedge \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \notin 1\mathrm{IN}3\,.$$

For more information on polymorphisms and how they can be used to study the complexity of the CSP, see [6].

2.4. Local consistency

The first step of almost any algorithm solving a CSP instance is checking local consistency. For instance, if a constraint forces a variable to be equal to 0, then we could substitute 0 and remove this variable.

Suppose we have a CSP instance

$$R_1(v_{1,1}, \dots, v_{1,n_1}) \wedge \dots \wedge R_s(v_{s,1}, \dots, v_{s,n_s}). \tag{2.4}$$

This instance is called *1-consistent (also known as arc-consistent)*, if for any variable x any two constraints $R_i(v_{i,1},\dots,v_{i,n_i})$ and $R_j(v_{j,1},\dots,v_{j,n_j})$ having this variable in the scope have the same projection onto this variable. This means that for every variable x there exists $D_x \subseteq A$, called the *domain of* x, such that the projection of any constraint on x is D_x.

Sometimes we need a stronger consistency (similar to singleton-arc-consistency in [36]). We say that $z_1 - C_1 - z_2 - \dots - C_{l-1} - z_l$ is *a path* in a CSP instance $\mathcal{I}$ if z_i, z_{i+1} are in the scope of the constraint C_i for every $i \in \{1,2,\dots,l-1\}$. We say that *a path* $z_1 - C_1 - z_2 - \dots - C_{l-1} - z_l$ *connects* b *and* c if there exist $a_1, a_2, \dots, a_l \in A$ such that $a_1 = b$, $a_l = c$, and the projection of each C_i onto z_i, z_{i+1} contains the tuple (a_i, a_{i+1}). A CSP instance $\mathcal{I}$ is called *cycle-consistent* if it is 1-consistent and for every variable z and $a \in D_z$ any path starting and ending with z in $\mathcal{I}$ connects a and a.

It is not hard to find a polynomial procedure making the instance 1-consistent or cycle-consistent. For 1-consistency, the idea is to find a variable where the consistency is violated, then reduce the domain D_x of this variable and reduce the corresponding relations. We repeat this while some constraints violate consistency. Finally, we either get a 1-consistent instance, or we get a contradiction (derive that $D_x = \varnothing$). For cycle-consistency, we should go deeper. For every variable x and every value $a \in D_x$, we reduce the domain of x to $\{a\}$ and check whether the remaining instance can be made 1-consistent. If not, then x cannot be equal to a, and a can be excluded from the domain D_x.

Later we will show that in some cases 1-consistency and cycle-consistency are enough to solve a CSP instance, that is, any consistent instance has a solution. See [5,36] for more information about local consistency conditions.

2.5. CSP over a 2-element domain

The complexity of CSP(Γ) for each constraint language Γ on $\{0,1\}$ was described in 1978 [40]. This classification can be formulated nicely using polymorphisms.

Theorem 2.2 ([34,40]). *Suppose $A = \{0,1\}$, Γ is a constraint language on A. Then* CSP(Γ) *is solvable in polynomial time if*

(1) 0 *preserves* Γ, *or*

(2) 1 *preserves* Γ, *or*

(3) $x \vee y$ *preserves* Γ, *or*

(4) $x \wedge y$ *preserves* Γ, *or*

(5) $xy \vee yz \vee xz$ *preserves* Γ, *or*

(6) $x + y + z$ *preserves* Γ.

CSP(Γ) *is NP-complete otherwise.*

Let us consider each case and explain how the polymorphisms make the problem easy. Note that the cases (1) and (2), (3) and (4) are dual to each other, that is why we consider only one in each pair in detail.

0 preserves Γ. This case is almost trivial. "The constant 0 preserves a relation $R \in \Gamma$" means that $R(0, 0, \dots, 0)$ holds. If 0 preserves all relations from Γ, then $(0, 0, \dots, 0)$ is always a solution of a CSP instance, which makes the problem CSP(Γ) trivial.

$x \vee y$ preserves Γ. Let us show how to solve an instance of CSP(Γ) if $x \vee y \in \mathrm{Pol}(\Gamma)$. First, we make our instance 1-consistent. Then, unless we get a contradiction, every variable x has its domain D_x which is either $\{0\}$, or $\{1\}$, or $\{0, 1\}$. We claim that if we send the variables with domain $\{0\}$ to 0, and the variables with the domain $\{1\}$ and $\{0, 1\}$ to 1, then we get a solution. In fact, if we apply $x \vee y$ to all the tuples of some constraint, we obtain a tuple consistent with the solution. Thus, 1-consistency guarantees the existence of a solution in this case.

$xy \vee yz \vee xz$ preserves Γ. The operation $xy \vee yz \vee xz$ returns the most popular value and is known as a majority operation. It is not hard to check [2] that any relation preserved by a majority operation can be represented as a conjunction of binary relations, and we may assume that Γ consists of only binary relations. As it is shown in Section 2.8, for a 2-element domain this gives a polynomial algorithm for CSP(Γ). Additionally, we can show [36, 47] that any cycle-consistent instance of CSP(Γ) has a solution. Hence to solve an instance, it is sufficient to make it cycle-consistent, and unless we obtain an empty domain (contradiction) the instance has a solution.

$x + y + z$ preserves Γ. It is known (see Lemma 2.8) that $x + y + z$ preserves a relation R if and only if the relation R can be represented as a conjunction of linear equations. Thus, CSP(Γ) is equivalent to the problem of solving of a system of linear equations in the field $\mathbb{Z}_2$, which is tractable.

2.6. CSP solvable by local consistency checking

As we see in the previous section all tractable CSPs on a 2-element domain can be solved by two algorithms. The first algorithm just checks some local consistency (1-consistency, cycle-consistency) and, if a sufficient level of consistency achieved, we know that the instance has a solution. The second algorithm is the Gaussian elimination applied to a system of linear equations. In this section we discuss when the first algorithm is sufficient and why some instances can be solved by a local consistency checking, while others require something else.

To simplify the presentation in this section, we assume that all constant relations $x = a$ are in the constraint language. In this case any polymorphism f of Γ is *idempotent*, that is, $f(x, x, \dots, x) = x$. This restriction does not affect the generality of the results because we can always consider the core of the constraint language and then add all constant relations (see [34]). Consider the following system of linear equations in $\mathbb{Z}_p$:

$$\begin{cases} x_1 + x_2 = x_3 + 0, \\ x_3 + 0 = x_4 + x_5, \\ x_4 + 0 = x_1 + x_6, \\ x_5 + x_6 = x_2 + 1. \end{cases} \tag{2.5}$$

If we calculate the sum of all equations, we will get $0 = 1$, which means that the system does not have a solution. Nevertheless, we may check that the system is cycle-consistent, which means that the cycle-consistency does not guarantee the existence of a solution for linear equations. In fact, we can show that there does not exist a local consistency condition that guarantees the existence of a solution of a system of linear equations (see [5]).

As it was shown in [5,47] if CSP(Γ) cannot be solved by cycle-consistency checking then we can express a linear equation modulo p using Γ. Since our constraint language is on a domain A, we could not expect to pp-define the relation $x_1 + x_2 = x_3 + x_4 \pmod p$. Instead, we claim that there exist $S \subseteq A$ and a surjective mapping $\varphi : A \to \mathbb{Z}_p^s$ such that the relation

$$\{(a_1, a_2, a_3, a_4\} \mid a_1, a_2, a_3, a_4 \in S, \varphi(a_1) + \varphi(a_2) = \varphi(a_3) + \varphi(a_4)\} \tag{2.6}$$

is pp-definable. This means that the linear equation is defined on some S modulo some equivalence relation defined by φ. To avoid such a transformation, we could introduce the notion of pp-constructability and say that $x_1 + x_2 = x_3 + x_4 \pmod p$ is pp-constructable from Γ. To keep everything simple, we do not define pp-constructability and use it informally hoping that the idea of this notion is clear from our example. For more details about pp-constructability, see [7].

If such a linear equation cannot be pp-defined (pp-constructed) then there should be some operation that preserves Γ but not the linear equation modulo p. An operation f is called *a Weak Near Unanimity Operation (WNU)* if it satisfies the following identity:

$$f(y, x, x, \dots, x) = f(x, y, x, \dots, x) = \dots = f(x, x, \dots, x, y).$$

It is not hard to check that an idempotent WNU of arity p does not preserve a nontrivial linear equation modulo p (see Lemma 4.9 in [47]). Thus, the existence of an idempotent p-ary WNU polymorphism of Γ guarantees that a linear equation modulo p cannot be pp-defined (pp-constructed). That is why a relation satisfying (2.6) is called *p-WNU-blocker*. Hence, if Γ has WNU polymorphisms of all arities then no linear equations can appear. The following theorem confirms that nothing but linear equations could be an obstacle for the local consistency checking.

Theorem 2.3 ([47]). *Suppose Γ is a constraint language containing all constant relations. The following conditions are equivalent:*

(1) *every cycle-consistent instance of* CSP(Γ) *has a solution;*

(2) Γ *has a WNU polymorphisms of all arities $n \geq 3$;*

(3) *there does not exist a p-WNU-blocker pp-definable from* Γ.

Thus, the fact that we cannot express (pp-define, pp-construct) a nontrivial linear equation makes the problem solvable by the cycle-consistency checking.

2.7. CSP Dichotomy Conjecture

In this subsection, we formulate a criterion for CSP(Γ) to be solvable in polynomial time. This criterion is known as the CSP Dichotomy Conjecture, it was formulated almost 30 years ago [28,29] but was an open question until 2017 [19,20,42,44].

Theorem 2.4 ([19,20,42,44]). *Suppose Γ is a constraint language on a finite set A. Then*

(1) CSP(Γ) *is solvable in polynomial time if Γ is preserved by a WNU;*

(2) CSP(Γ) *is NP-complete otherwise.*

The reason why the existence of a WNU polymorphism makes the problem easy is the fact that we cannot pp-define a strong relation giving us NP-hardness. A relation $R = (B_0 \cup B_1)^3 \setminus (B_0^3 \cup B_1^3)$, where $B_0, B_1 \subseteq A$, $B_0 \neq \varnothing$, $B_1 \neq \varnothing$, and $B_0 \cap B_1 = \varnothing$, is called *a WNU-blocker*. Such relations are similar to the not-all-equal (NAE) relation on $\{0, 1\}$, where B_0 means 0 and B_1 means 1. Instead of the existence of a pp-definable WNU-blocker, we could say that the relation NAE is pp-constructable from Γ. Note that CSP($\{\text{NAE}\}$) and CSP($\{R\}$) for a WNU-blocker R are NP-complete problems.

We can check (see Lemma 4.8 in [47]) that a WNU operation does not preserve a WNU-blocker. Moreover, we have the following criterion.

Lemma 2.5 ([47]). *A constraint language Γ containing all constant relations is preserved by a WNU if and only if there is no WNU-blocker pp-definable from Γ.*

Thus, CSP(Γ) is solvable in polynomial time if and only if a WNU-blocker cannot be pp-defined. Hence, the fact that we cannot pp-construct the not-all-equal relation makes the problem easy, and a WNU is an operation that guarantees that this relation cannot be pp-constructed.

2.8. How to solve CSP if pp-definable relations are simple

Below we discuss how the fact that only simple relations can be pp-defined from Γ can help to solve CSP(Γ) in polynomial time. In this case we can calculate the sentence explicitly eliminating existential quantifiers one by one. I believe that a similar idea should work for any Γ preserved by a WNU, which will give us a simple algorithm for CSP(Γ).

CSP(Γ) can be viewed as the following problem. Given a sentence

$$\exists x_1 \dots \exists x_n \big(R_1(v_{1,1}, \dots, v_{1,n_1}) \wedge \dots \wedge R_s(v_{s,1}, \dots, v_{s,n_s})\big),$$

we need to check whether it holds. To do this, let us remove the quantifiers one by one. Let

$$\Delta_{n-1}(x_1, \dots, x_{n-1}) = \exists x_n \big(R_1(v_{1,1}, \dots, v_{1,n_1}) \wedge \dots \wedge R_s(v_{s,1}, \dots, v_{s,n_s})\big).$$

In general, Δ_{n-1} could be any relation of arity $n-1$, and even to write this relation we need $|A|^{n-1}$ bits. Nevertheless, we believe that if CSP(Γ) is tractable then the relation Δ_{n-1} (or the important part of it) has a compact representation and can be efficiently computed. Then we calculate $\Delta_{n-2}, \Delta_{n-3}, \dots, \Delta_0$, where $\Delta_{i-1}(x_1, \dots, x_{i-1}) = \exists x_i \Delta_i(x_1, \dots, x_i)$, and the value of Δ_0 is the answer we need.

We may check that on a 2-element domain we have

$$\begin{aligned}&\exists x_n \bigl(R_1(v_{1,1}, \dots, v_{1,n_1}) \wedge \dots \wedge R_s(v_{s,1}, \dots, v_{s,n_s})\bigr)\\ &\quad = \bigwedge_{i,j \in \{1,2,\dots,s\}} \bigl(\exists x_n R_i(v_{i,1}, \dots, v_{i,n_i}) \wedge R_j(v_{j,1}, \dots, v_{j,n_j})\bigr).\end{aligned} \tag{2.7}$$

The implication $\Rightarrow$ is obvious. To prove $\Leftarrow$ assume that the left-hand side does not hold. Then the conjunctive part does not hold on both $(x_1, \dots, x_{n-1}, 0)$ and $(x_1, \dots, x_{n-1}, 1)$. Hence, there exist i and j such that $R_i(v_{i,1}, \dots, v_{i,n_i})$ does not hold on $(x_1, x_2, \dots, x_{n-1}, 0)$ and $R_j(v_{j,1}, \dots, v_{j,n_j})$ does not hold on $(x_1, x_2, \dots, x_{n-1}, 1)$. Hence, the (i, j)-part of the right-hand side does not hold.

There are two problems if we use (2.7) to solve the CSP. First, as we mentioned above, the relation $R_{i,j}(\dots) = \exists x_n R_i(v_{i,1}, \dots, v_{i,n_i}) \wedge R_j(v_{j,1}, \dots, v_{j,n_j})$ probably does not have a compact representation. Second, if we remove the quantifiers $\exists x_n, \exists x_{n-1}, \dots, \exists x_1$ one by one, potentially we could get an exponential number of relations in the formula. Let us show how these problem are solved for concrete examples on a 2-element domain.

2.9. System of linear equations in $\mathbb{Z}_2$

Let $A = \{0, 1\}$ and let Γ consist of linear equations in $\mathbb{Z}_2$. Suppose that for every i we have

$$R_i(v_{i,1}, \dots, v_{i,n_i}) = (a_1^i x_n + a_2^i x_2 + \dots + a_n^i x_n = a_0^i).$$

For $a_n^i = a_n^j = 1$, we have

$$\begin{aligned}&R_{i,j}(\dots)\\ &\quad := \exists x_n \bigl(R_i(v_{i,1}, \dots, v_{i,n_i}) \wedge R_j(v_{j,1}, \dots, v_{j,n_j})\bigr)\\ &\quad = (a_1^i x_1 + a_2^i x_2 + \dots + a_{n-1}^i x_{n-1} + a_0^i = a_1^j x_1 + a_2^j x_2 + \dots + a_{n-1}^j x_{n-1} + a_0^j).\end{aligned}$$

If $a_n^i = 0$ then the constraint $R_i(v_{i,1}, \dots, v_{i,n_i})$ does not depend on x_n, so we keep it as it is when remove the quantifier. Hence, in every case we have a compact representation of Δ_{n-1}. To avoid the exponential growth of the number of the constraints, we use the idea from the Gaussian elimination. Choose k such that $a_n^k = 1$, then calculate only $R_{k,1}, \dots, R_{k,s}$ and ignore all the other relations. Thus, in this case we have

$$\begin{aligned}\Delta_{n-1}(x_1, \dots, x_{n-1}) &= \exists x_n \bigl(R_1(v_{1,1}, \dots, v_{1,n_1}) \wedge \dots \wedge R_s(v_{s,1}, \dots, v_{s,n_s})\bigr)\\ &= \bigwedge_{j \in \{1,2,\dots,s\}} \bigl(\exists x_n R_k(v_{k,1}, \dots, v_{k,n_k}) \wedge R_j(v_{j,1}, \dots, v_{j,n_j})\bigr).\end{aligned} \tag{2.8}$$

Proceeding this way, we calculate $\Delta_{n-2}, \Delta_{n-3}, \dots, \Delta_0$. Note that (2.8) holds not only for linear equations but whenever a variable x_n is uniquely determined by the other variables in $R_k(v_{k,1}, \dots, v_{k,n_1})$.

2.10. 2-satisfability

Let $A = \{0, 1\}$ and let Γ consist of all binary relations. In this case $R_{i,j}$ is also binary, which means that we do not have a problem with a compact representation. Also, every time we eliminate a quantifier and caclulate Δ_i, we remove the repetitive constraints. Therefore, in each Δ_i we cannot have more than $i \cdot i \cdot 2^{2^2}$ constraints because we have i different variables and 2^{2^2} different binary relations on $\{0, 1\}$.

As we see, the main question in both examples is the existence of a compact representation. In the first example we represent any relation as a conjunction of linear equations, in the second we represent as a conjunction of binary relations. We could ask when such a compact representation exists. Let $s_\Gamma(n)$ be the number of pp-definable from Γ relations of arity n. If $\log_2 s_\Gamma(n)$ grows exponentially then we need exponential space to encode relations of arity n and we cannot expect a compact representation. We say that Γ has few subpowers if $\log_2 s_\Gamma(n) < p(n)$ for a polynomial $p(n)$. It turns out that there is a simple criterion for the constraint language to have few subpowers. An operation t is called *an edge operation* if it satisfies the following identities:

$$\begin{aligned}
t(x, x, y, y, y, \dots, y, y) &= y,\\
t(x, y, x, y, y, \dots, y, y) &= y,\\
t(y, y, y, x, y, \dots, y, y) &= y,\\
t(y, y, y, y, x, \dots, y, y) &= y,\\
&\dots\\
t(y, y, y, y, y, \dots, x, y) &= y,\\
t(y, y, y, y, y, \dots, y, x) &= y.
\end{aligned}$$

Theorem 2.6 ([9]). *A constraint language Γ containing all constant relations has few subpowers if and only if it has an edge polymorphism.*

We can show that if Γ has few subpowers then the pp-definable relations have a natural compact representation, which gives a polynomial algorithm for CSP(Γ) [33]. Note that two examples of an edge operation were given earlier in this paper. The first example is a majority operation satisfying $m(y, y, x) = m(y, x, y) = m(x, y, y) = y$. By adding 3 dummy variables in the beginning, we get the required properties of an edge operation. Another example is $x + y + z$ on $\{0, 1\}$. By adding dummy variables at the end, we can easily satisfy all the identities. Very roughly speaking, any few subpowers case is just a combination (probably very complicated) of the majority case and the linear case.

2.11. Strong subuniverses and a proof of the CSP Dichotomy Conjecture

In this subsection, we consider another simple idea that can solve the CSP in polynomial time. This idea is one of the two main ingredients of the proof of the CSP Dichotomy Conjecture in [42,44].

Assume that for every variable x whose domain is D_x, $|D_x| > 1$, we can choose a subset $B_x \subsetneq D_x$ such that if the instance has a solution, then it has a solution with $x \in B_x$.

In this case we can reduce the domains iteratively until the moment when each domain has exactly one element, which usually gives us a solution.

As we saw in Section 2.5, if Γ is preserved by $x \vee y$ and the instance is 1-consistent then we can safely reduce the domain of a variable to $\{1\}$. Similarly, if Γ is preserved by the majority operation $xy \vee yz \vee xz$ and the instance is cycle-consistent, then we can safely reduce the domain $\{0, 1\}$ to $\{0\}$ and $\{1\}$ [47]. It turns out that this idea can be generalized for any constraint language preserved by a WNU operation.

A unary relation $B \subseteq A$ is called *a subuniverse* if B is pp-definable over Γ. It can be easily checked that all the domains D_x that appear while checking consistency (see Section 2.4) are subuniverses. Let us define three types of strong subuniverses:

Binary absorbing subuniverse. We say that B' is *a binary absorbing subuniverse of* B if there exists a binary operation $f \in \mathrm{Pol}(\Gamma)$ such that $f(B', B) \subseteq B'$ and $f(B, B') \subseteq B'$. For example, if the operation $x \vee y$ preserves Γ then $\{1\}$ is a binary absorbing subuniverse of $\{0, 1\}$ and $x \vee y$ is a binary absorbing operation.

Ternary absorbing subuniverse. We say that B' is *a ternary absorbing subuniverse of* B if there exists a ternary operation $f \in \mathrm{Pol}(\Gamma)$ such that $f(B', B', B) \subseteq B'$, $f(B', B, B') \subseteq B'$, and $f(B, B', B') \subseteq B'$. For example, if the majority operation $xy \vee yz \vee xz$ preserves Γ, then both $\{0\}$ and $\{1\}$ are ternary absorbing subuniverses of $\{0, 1\}$. Since we can always add a dummy variable to a binary absorbing operation, any binary absorbing subuniverse is also a ternary absorbing subuniverse.

To define the last type of strong subalgebras we need some understanding of the Universal Algebra. We do not think a concrete definition is important here, that is why if a reader thinks the definition is too complicated, we recommend to skip it and think about the last type as something similar to the first two.

PC subuniverse. A set F of operations is called *Polynomially Complete* (PC) if any operation can be derived from F and constants using composition. We say that B' is *a PC subuniverse* of B if there exists a pp-definable equivalence relation $\sigma \subseteq B \times B$ such that $\mathrm{Pol}(\Gamma)/\sigma$ is PC.

A subset B' of B is called *a strong subuniverse* if B' is a ternary absorbing subuniverse or a PC subuniverse.

Theorem 2.7 ([47]). *Suppose Γ contains all constant relations and is preserved by a WNU operation, $B \subseteq A$ is a subuniverse. Then*

(1) *there exists a strong subuniverse $B' \subsetneq B$, or*

(2) *there exists a pp-definable nontrivial equivalence relation σ on B and $f \in \mathrm{Pol}(\Gamma)$ such that $(B; f)/\sigma \cong (\mathbb{Z}_p^k; x - y + z)$.*

As it follows from the next lemma, the second condition implies that any pp-definable relation (modulo σ) can be viewed as a system of linear equations in a field.

Lemma 2.8 ([32]). *Suppose $R \subseteq \mathbb{Z}_p^n$ preserved by $x - y + z$. Then R can be represented as a conjunction of relations of the form $a_1x_1 + \cdots + a_nx_n = a_0 \pmod{p}$.*

For CSPs solvable by the local consistency checking, strong subuniverses have the following property.

Theorem 2.9 ([47]). *Suppose*

(1) *Γ is a constraint language containing all constant relations;*

(2) *Γ is preserved by a WNU of each arity $n \geq 3$;*

(3) *$\mathcal{I}$ is a cycle-consistent instance of* $\mathrm{CSP}(\Gamma)$*;*

(4) *D_x is the domain of a variable x;*

(5) *B is a strong subalgebra of D_x.*

Then $\mathcal{I}$ has a solution with $x \in B$.

Thus, strong subuniverses have the required property that we cannot loose all the solutions when we restrict a variable to it. As it was proved in [44], a similar theorem holds for any constraint language preserved by a WNU operation (with additional consistency conditions on the instance). We skip this result because it would require too many additional definitions.

As we see from Theorem 2.7, for every domain D_x either we have a strong subuniverse and can reduce the domain of some variable, or, modulo some equivalence relation, we have a system of linear equations in a field. If Γ has a WNU polymorphism of each arity $n \geq 3$, then we always have the first case; hence, we can iteratively reduce the domains until the moment when all the domains have just one element, which gives us a solution. That is why any cycle-consistent instance in this situation has a solution. If we always have the second case then this situation is similar to a system of linear equations, but different linear equations can be mixed which makes it impossible to apply usual Gaussian elimination. Nevertheless, the few subpowers algorithm solves the problem [33].

For many years the main obstacle was that these two situations can be mixed and at the moment we do not know an elegant way how to split them. Nevertheless, the general algorithm for tractable CSP presented in [44] is just a smart combination of these two ideas:

- if there exists a strong subalgebra, reduce
- if there exists a system of linear equations, solve it.

For more information about this approach as well as its connection with the second general algorithm see [3].

2.12. Conclusions

Even though we still do not have a simple algorithm that solves all tractable Constraint Satisfaction Problems, we understand what makes the problem hard, and what makes the problem easy. First, we know that in all the hard cases we can pp-construct (pp-define) the not-all-equal relation, which means that all the NP-hard cases have the same nature. Second, if the CSP is not solvable locally then we can pp-construct (pp-define) a linear equation in a field. Moreover, any domain of a tractable CSP either has a strong subalgebra and we can (almost) safely reduce the domain, or there exists a system of linear equations on this domain. This implies that any tractable CSP can be solved by a smart combination of the Gaussian elimination and local consistency checking, and emphasizes the exclusive role of the linear case in Universal Algebra and Computational Complexity.

Note that both CSP algorithms in [20, 44] depend exponentially on the size of the domain, and we could ask whether there exists a universal polynomial algorithm that works for any constraint language Γ admitting a WNU polymorphism.

Problem 1. Does there exist a polynomial algorithm for the following decision problem: given a conjunctive formula $R_1(v_{1,1},\dots,v_{1,n_1}) \wedge \dots \wedge R_s(v_{s,1},\dots,v_{s,n_s})$, where all relations $R_1,\dots,R_s$ are preserved by a WNU, decide whether this formula is satisfiable.

If the domain is fixed then the above problem can be solved by the algorithms from [19,42]. In fact, we know from [4, THEOREM 4.2] that from a WNU on a domain of size k we can always derive a WNU (and also a cyclic operation) of any prime arity greater than k. Thus, we can find finitely many WNU operations on a domain of size k such that any constraint language preserved by a WNU is preserved by one of them. It remains to apply the algorithm for each WNU and return a solution if one of them gave a solution.

3. QUANTIFIED CSP

A natural generalization of the CSP is the *Quantified Constraint Satisfaction Problem* (QCSP), where we allow to use both existential and universal quantifiers. Formally, for a constraint language Γ, QCSP(Γ) is the problem to evaluate a sentence of the form

$$\forall x_1 \exists y_1 \dots \forall x_n \exists y_n \; R_1(v_{1,1},\dots,v_{1,n_1}) \wedge \dots \wedge R_s(v_{s,1},\dots,v_{s,n_s}),$$

where $R_1,\dots,R_s \in \Gamma$, and $v_{i,j} \in \{x_1,\dots,x_n,y_1,\dots,y_n\}$ for every i,j (see [16,23,24,37]). Unlike the CSP, the problem QCSP(Γ) can be PSpace-hard if the constraint language Γ is powerful enough. For example, QCSP($\{\text{NAE}\}$) and QCSP($\{\text{1IN3}\}$) on the domain $A=\{0,1\}$ are PSpace-hard [25,27], and QCSP($\{\neq\}$) for $|A|>2$ is also PSpace-hard [16]. Nevertheless, if Γ consists of linear equations modulo p then QCSP(Γ) is tractable [16]. It was conjectured by Hubie Chen [22,24] that for any constraint language Γ the problem QCSP(Γ) is either solvable in polynomial time, or NP-complete, or PSpace-complete. Recently, this conjecture was disproved in [48], where the authors found constraint languages Γ such that QCSP(Γ) is coNP-complete (on a 3-element domain), DP-complete (on a 4-element domain), Θ_2^P-complete (on a 10-element domain). Despite the whole zoo of the complexity classes, we still hope to obtain a full classification of the complexity for each constraint language Γ.

Below we consider the main idea that makes the problem easier than PSpace.

3.1. PGP reduction for Π_2 restrictions

For simplicity let us consider the Π_2-restriction of QCSP(Γ), denoted QCSP$^2(\Gamma)$, in which the input is of the form

$$\forall x_1 \dots \forall x_n \exists y_1 \dots \exists y_m R_1(\dots) \wedge \dots \wedge R_s(\dots). \tag{3.1}$$

Such an instance holds whenever the conjunctive formula $R_1(\dots) \wedge \dots \wedge R_s(\dots)$ is solvable for any evaluation of $x_1, \dots, x_n$, which gives us a reduction of the instance to $|A|^n$ instances of CSP(Γ^*), where by Γ^* we denote $\Gamma \cup \{(x = a) \mid a \in A\}$. If we need to check $|A|^n$ tuples, which is exponentially many, this does not make the problem easier. Nevertheless, sometimes it is sufficient to check only polynomially many tuples. Let us consider a concrete example.

System of linear equations. Suppose $A = \{0, 1\}$ and Γ consists of linear equations in $\mathbb{Z}_2$. Let us check that the instance (3.1) holds for $(x_1, \dots, x_n) = (0, \dots, 0)$, and $(x_1, \dots, x_n) = (0, \dots, 0, 1, 0, \dots, 0)$ for any position of 1. To do this, we solve the CSP instance $R_1(\dots) \wedge \dots \wedge R_s(\dots) \wedge \bigwedge_{i=1}^{n}(x_i = 0)$, and for every $j \in \{1, 2, \dots, n\}$ we solve the instance $R_1(\dots) \wedge \dots \wedge R_s(\dots) \wedge (x_j = 1) \wedge \bigwedge_{i \neq j}(x_i = 0)$. Each instance is a system of linear equations and can be solved in polynomial time. If at least one of the instances does not have a solution, then the instance (3.1) does not hold. Assume that all of them are satisfiable, then consider the relation Δ defined by the following pp-formula over Γ:

$$\Delta(x_1, \dots, x_n) = \exists y_1 \dots \exists y_m R_1(\dots) \wedge \dots \wedge R_s(\dots).$$

Since Γ is preserved by $x + y + z$, Δ is also preserved by $x + y + z$. Applying this operation to the tuples $(0, 0, \dots, 0), (1, 0, \dots, 0), (0, 1, 0, \dots, 0), \dots, (0, 0, \dots, 0, 1) \in \Delta$ coordinatewise, we derive that $\Delta = \{0, 1\}^n$, that is, Δ contains all tuples and (3.1) holds. Thus, we showed that QCSP$^2(\Gamma)$ is solvable in polynomial time.

This idea can be generalized as follows. We say that a set of operations F (or an algebra $(A; F)$) has the *polynomially generated powers (PGP)* property if there exists a polynomial $p(n)$ such that A^n can be generated from $p(n)$ tuples using operations of F. Another behavior that might arise is that there is an exponential function f so that the smallest generating sets for A^n require size at least $f(n)$. We describe this as the *exponentially generated powers (EGP)* property. As it was proved in [43] these are the only two situations we could have on a finite domain. Moreover, it was shown that the generating set in the PGP case can be chosen to be very simple and efficiently computable. As a generating set of polynomial size, we can take the set of all tuples with at most k switches, where a switch is a position in $(a_1, \dots, a_n)$ such that $a_i \neq a_{i+1}$. This gives a polynomial reduction of QCSP$^2(\Gamma)$ to CSP(Γ^*) if Pol(Γ) has the PGP property.

3.2. A general PGP reduction

Let us show that the same idea can be applied to the general form of QCSP(Γ). First, we show how to move universal quantifiers left and convert an instance into the Π_2-form. Notice that the sentence $\exists y_1 \exists y_2 \dots \exists y_s \forall x \Phi$ is equivalent to

$$\forall x^1 \forall x^2 \dots \forall x^{|A|} \exists y_1 \exists y_2 \dots \exists y_s \Phi_1 \wedge \Phi_2 \wedge \dots \wedge \Phi_{|A|},$$

where each Φ_i is obtained from Φ by renaming x by x^i. In this way we can convert any instance $\exists y_1 \forall x_1 \dots \exists y_t \forall x_t \Phi$ of QCSP(Γ) into the Π_2-restriction by moving all universal quantifiers left:

$$\begin{aligned}&\forall x_1^1 \dots \forall x_1^{|A|} \forall x_2^1 \dots \forall x_2^{|A|^2} \dots \forall x_t^1 \dots \forall x_t^{|A|^t}\\ &\exists y_1 \exists y_2^1 \dots \exists y_2^{|A|} \dots \exists y_t^1 \dots \exists y_t^{|A|^{t-1}} \Phi_1 \wedge \Phi_2 \wedge \dots \wedge \Phi_q,\end{aligned} \tag{3.2}$$

where each Φ_i is obtained from Φ by renaming the variables. The only problem with this reduction is that the number of variables and constraints could be exponential. Nevertheless, we can apply the PGP idea to this sentence. If Pol(Γ) has the PGP property then there exists a constant k such that it is sufficient to check (3.2) only on the tuples with at most k switches. Those k switches appear in at most k original x-variables and all the remaining variables can be fixed with constants. This allows reducing QCSP(Γ) to a sentence with a constant number of universal quantifiers or even remove all of them.

Theorem 3.1 ([45]). *Suppose* Pol(Γ) *has the PGP property. Then* QCSP(Γ) *is polynomially equivalent to the modification of* QCSP^2(Γ) *where sentences have at most* $|A|$ *universally quantified variables.*

Theorem 3.2 ([45]). *Suppose* Pol(Γ) *has the PGP property. Then* QCSP(Γ) *is polynomially reduced to* CSP(Γ^*).

This idea gives us a complete classification of the complexity of QCSP(Γ) for a two-element domain.

Theorem 3.3 ([25, 27]). *Suppose* Γ *is a constraint language on* $\{0, 1\}$. *Then* QCSP(Γ) *is solvable in polynomial time if* Γ *is preserved by an idempotent WNU;* QCSP(Γ) *is PSpace-complete otherwise.*

It is known [39] that if Γ admits an idempotent WNU, then it is preserved by $x + y + z$, $x \vee y$, $x \wedge y$, or $xy \vee yz \vee xz$. Hence, to prove the above theorem, it is sufficient to check that these operations guarantee the PGP property, which by Theorem 3.2 gives a polynomial reduction to a tractable CSP. To show the PGP property, we verify that the tuples $(0, 0, \dots, 0)$, $(1, 1, \dots, 1)$, $(1, 0, \dots, 0)$, $(0, 1, 0, \dots, 0)$, $\dots$, $(0, \dots, 0, 1)$ generate $\{0, 1\}^n$ using any of the above operations.

3.3. Does EGP mean hard?

Thus, if Pol(Γ) has the PGP property then we have a nice reduction to CSP, and QCSP(Γ) belongs to NP. What can we say about the complexity of QCSP(Γ) if Pol(Γ) has the EGP property? Hubie Chen conjectured in [24] that QCSP(Γ) is PSpace-complete whenever Pol(Γ) has the EGP property.

For constraint languages Γ containing all constant relations, a characterization of Pol(Γ) that have the EGP property is given in [43], where it is shown that Γ must allow the pp-definition of relations τ_n with the following special form.

Definition 3.4. Let $\alpha \cup \beta = A$, yet neither α nor β equals D. Let $S = \alpha^3 \cup \beta^3$ and τ_n be the $3n$-ary relation given by $\bigvee_{i=1}^{n} S(x_i, y_i, z_i)$.

The complement to S represents the not-all-equal relation and the relations τ_n allow for the encoding of the complement of *Not-All-Equal 3-Satisfiability* (where $\alpha \setminus \beta$ is 0 and $\beta \setminus \alpha$ is 1). Thus, if one has polynomially computable (in n) pp-definitions of τ_n, then it is clear that QCSP(Γ) is co-NP-hard [22]. In light of this observation, it seemed that only a small step remained to proving the actual Chen Conjecture, at least with coNP-hard in place of PSpace-complete.

3.4. Surprising constraint language and the QCSP on a 3-element domain

As we saw in Section 2.7, the CSP is NP-hard if and only the we can pp-define (pp-construct) the not-all-equal relation. In the previous subsection, we mentioned that in the EGP case we can always pp-construct the complement to Not-All-Equal 3-Satisfability, which almost guarantees coNP-hardness. Surprisingly, two constraint languages Γ on $A = \{0, 1, 2\}$ were discovered in [48] for which any pp-definition of τ_n is of exponential size, which makes it impossible to use this reduction.

Theorem 3.5 ([48]). *There exists a constraint language Γ on $\{0, 1, 2\}$ such that*

(1) Pol(Γ) *has the EGP property,*

(2) τ_n *is pp-definable over* Γ

(3) *any pp-definition of τ_n for $\alpha = \{0, 1\}$ and $\beta = \{0, 2\}$ has at least 2^n variables, and*

(4) QCSP(Γ) *is solvable in polynomial time.*

The algorithm in (4) consists of the following three steps. First, it reduces an instance to a Π_2-form $\forall x_1 \dots \forall x_n \exists y_1 \dots \exists y_m \Phi$. Then, by solving polynomially many CSPs, it calculates polynomially many evaluations to $(x_1, \dots, x_n)$ we need to check. Finally, it checks that Φ has a solution for each of these evaluations. It is proved in [48] that this test guarantees that the instance holds.

This result was shocking because of several reasons. Not only it disproved the widely believed Chen Conjecture but showed that we need to worry about the existence of an efficient pp-definition. Before, if we could pp-define a strong relation (such as τ_n) then the problem was hard. Another surprising thing is that we have to calculate the evaluations of $(x_1, \dots, x_n)$ we need to check, In fact, if we do not look inside Φ then we have to check all the tuples from $\{0, 1\}^n$.

Despite the fact that we are far from having a full classification of the complexity of the QCSP, we know the complexity for any constraint language on a 3-element domain containing all constant relations. This classification is given in terms of polymorphisms.

Theorem 3.6 ([48]). *Suppose* Γ *is a finite constraint language on* $\{0, 1, 2\}$ *containing all constant relations. Then* QCSP(Γ) *is either solvable in polynomial time, NP-complete, coNP-complete, or PSpace-complete.*

3.5. Conclusions

Unlike the CSP where the complexity is known for any constraint language Γ here the complexity is wide open.

Problem 2. What is the complexity of QCSP(Γ)?

Moreover, we do not even have a conjecture describing the complexity. We know that for some constraint languages Γ the problem QCSP(Γ) is DP-complete and Θ_2^P-complete, but we do not know whether there are some other complexity classes and whether we have finitely many of them.

Problem 3. What complexity classes (up to polynomial equivalence) can be expressed as QCSP(Γ) for some constraint language Γ?

Now it is hard to believe that there will be a simple classification, that is why it is interesting to start with a 3-element domain (without constant relations) and 4-element domain. Probably, a more important problem is to describe all tractable cases assuming $\mathrm{P} \neq \mathrm{NP}$.

Problem 4. Describe all constraint languages Γ such that QCSP(Γ) is solvable in polynomial time.

4. OTHER VARIANTS OF CSP

The Quantified CSP is only one of many other variations and generalizations of the CSP whose complexity is still unknown. Here we list some of them.

4.1. CSP over an infinite domain

If we consider CSP(Γ) for a constraint language on an infinite domain, the situation changes significantly. As was shown in [11], every computational problem is equivalent (under polynomial-time Turing reductions) to a problem of the form CSP(Γ). In [14] the authors gave a nice example of a constraint language Γ such that CSP(Γ) is undecidable. Let Γ consist of three relations (predicates) $x + y = z$, $x \cdot y = z$ and $x = 1$ over the set of all integers $\mathbb{Z}$. Then the Hilbert's 10th problem can be expressed as CSP(Γ), which proves undecidability of CSP(Γ). Nevertheless, there are additional assumptions that send the CSP back to the class NP and make complexity classifications possible [8,12]. For more information about the infinite domain CSP and the algebraic approach, see [10,14].

4.2. Surjective Constraint Satisfaction Problem

A natural modification of the CSP is *the Surjective Constraint Satisfaction Problem*, where we want to find a surjective solution. Formally, for a constraint language Γ over a domain A, SCSP(Γ) is the following decision problem: given a formula

$$R_1(\dots) \wedge \dots \wedge R_s(\dots),$$

where all relations $R_1, \dots, R_s$ are from Γ, decide whether there exists a surjective solution, that is, a solution with $\{x_1, \dots, x_n\} = A$. Probably, the most natural examples of the Surjective CSP are defined as the surjective graph homomorphism problem, which is equivalent to SCSP(Γ) where Γ consists of one binary relation that is viewed as a graph. An interesting fact about the complexity of the Surjective CSP is that its complexity remained unknown for many years even for very simple graphs and constraint languages. Three most popular examples of such long-standing problems are the complexity for the reflexive 4-cycle (undirected having a loop at each vertex) [38], the complexity for the nonreflexive 6-cycle (undirected without loops) [41], and the complexity of the No-Rainbow-Problem (SCSP($\{N\}$) where $A = \{0, 1, 2\}$ and $N = \{(a, b, c) \mid \{a, b, c\} \neq A\}$) [46]. Even though these three problems turned out to be NP-complete, the complexity seems to be unknown even for graphs of size 5 and cycles.

Problem 5. What is the complexity of SCSP(Γ)?

It was shown in [46] that the complexity of SCSP(Γ) cannot be described in terms of polymorphisms, which disproved the only conjecture about the complexity of SCSP(Γ) we know. This conjecture, formulated by Hubie Chen, says that SCSP(Γ) and CSP(Γ^*) have the same complexity. Nevertheless, this conjecture still can hold for graphs.

Problem 6. Is it true that SCSP($\{R\}$) and CSP($\{R\}^*$) have the same complexity for any binary relation R?

For more results on the complexity of the SCSP, see the survey [13].

4.3. Promise CSP

A natural generalization of the CSP is *the Promise Constraint Satisfaction Problem*, where a promise about the input is given (see [18,21]). Let $\Gamma = \{(R_1^A, R_1^B), \dots, (R_t^A, R_t^B)\}$, where R_i^A and R_i^B are relations of the same arity over the domains A and B, respectively. Then PCSP(Γ) is the following decision problem: given two conjunctive formulas

$$R_{i_1}^A(v_{1,1}, \dots, v_{1,n_1}) \wedge \dots \wedge R_{i_s}^A(v_{s,1}, \dots, v_{s,n_s}),$$
$$R_{i_1}^B(v_{1,1}, \dots, v_{1,n_1}) \wedge \dots \wedge R_{i_s}^B(v_{s,1}, \dots, v_{s,n_s}),$$

where $(R_{i_j}^A, R_{i_j}^B)$ are from Γ for every j and $v_{i,j} \in \{x_1, \dots, x_n\}$ for every i, j, distinguish between the case when both of them are satisfiable, and when both of them are not satisfiable. Thus, we are given two CSP instances and a promise that if one has a solution then another has a solution. Usually, it is also assumed that there exists a mapping (homomorphism) $h : A \to B$ such that $h(R_i^A) \subseteq R_i^B$ for every i. In this case, the satisfiability of the first formula implies the satisfiability of the second one. To make sure that the promise can actually make an NP-hard problem tractable, see Example 2.8 in [21].

The most popular example of the Promise CSP is graph (k, l)-colorability, where we need to distinguish between k-colorable graphs and not even l-colorable, where $k \leq l$. This problem can be written as follows.

Problem 7. Let $|A| = k$, $|B| = l$, $\Gamma = \{(\neq_A, \neq_B)\}$. What is the complexity of PCSP(Γ)?

Recently, it was proved [21] that (k, l)-colorability is NP-hard for $l = 2k - 1$ and $k \geq 3$ but even the complexity of $(3, 6)$-colorability is still not known.

Even for a 2-element domain the problem is wide open, but recently a dichotomy for symmetric Boolean PCSP was proved [30].

Problem 8. Let $A = B = \{0, 1\}$. What is the complexity of PCSP(Γ)?

REFERENCES

[1] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[2] K. A. Baker and F. Pixley, Polynomial interpolation and the Chinese Remainder Theorem for algebraic systems. *Math. Z.* **143** (1975), 165–174.

[3] L. Barto, Z. Brady, A. Bulatov, M. Kozik, and D. Zhuk, Minimal Taylor algebras as a common framework for the three algebraic approaches to the CSP In *2021 36th annual ACM/IEEE symposium on logic in computer science (LICS)*, pp. 1–13, IEEE, 2021.

[4] L. Barto and M. Kozik, Absorbing subalgebras, cyclic terms, and the Constraint Satisfaction Problem. *Log. Methods Comput. Sci.* **8** (2012), no. 1.

[5] L. Barto and M. Kozik, Constraint satisfaction problems solvable by local consistency methods. *J. ACM* **61** (2014), no. 1, 1–19.

[6] L. Barto, A. Krokhin, and R. Willard, *Polymorphisms, and how to use them*. Dagstuhl Follow-Ups 7, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[7] L. Barto, J. Opršal, and M. Pinsker, The wonderland of reflections. *Israel J. Math.* **223** (2018), no. 1, 363–398.

[8] L. Barto and M. Pinsker, The algebraic dichotomy conjecture for infinite domain constraint satisfaction problems. In *2016 31st annual ACM/IEEE symposium on logic in computer science (LICS)*, pp. 1–8, IEEE, 2016.

[9] J. Berman, P. Idziak, P. Marković, R. McKenzie, M. Valeriote, and R. Willard, Varieties with few subalgebras of powers. *Trans. Amer. Math. Soc.* **362** (2010), no. 3, 1445–1473.

[10] M. Bodirsky, Complexity classification in infinite-domain constraint satisfaction. 2012, arXiv:1201.0856.

[11] M. Bodirsky and M. Grohe, Non-dichotomies in constraint satisfaction complexity. In *International colloquium on automata, languages, and programming*, pp. 184–196, Springer, 2008.

[12] M. Bodirsky and J. Kára, The complexity of temporal constraint satisfaction problems. *J. ACM* **57** (2010), no. 2, 9.

[13] M. Bodirsky, J. Kára, and B. Martin, The complexity of surjective homomorphism problems – a survey. *Discrete Appl. Math.* **160** (2012), no. 12, 1680–1690.

[14] M. Bodirsky and M. Mamino, *Constraint satisfaction problems over numeric domains*. Dagstuhl Follow-Ups 7, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[15] V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov, Galois theory for post algebras parts I and II. *Cybernetics* **5** (1969), 243–252, 531–539. 9

[16] F. Börner, A. A. Bulatov, H. Chen, P. Jeavons, and A. A. Krokhin, The complexity of constraint satisfaction games and QCSP. *Inform. and Comput.* **207** (2009), no. 9, 923–944.

[17] J. Brakensiek, S. Gopi, and V. Guruswami, CSPs with global modular constraints: algorithms and hardness via polynomial representations. In *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*, pp. 590–601, Association for Computing Machinery, 2019.

[18] J. Brakensiek and V. Guruswami, Promise constraint satisfaction: structure theory and a symmetric boolean dichotomy. In *Proceedings of the twenty-ninth annual ACM–SIAM symposium on discrete algorithms*, pp. 1782–1801, SIAM, 2018.

[19] A. A. Bulatov, A dichotomy theorem for nonuniform CSPs. In *2017 IEEE 58th annual symposium on foundations of computer science (FOCS)*, pp. 319–330, IEEE, 2017.

[20] A. A. Bulatov, A dichotomy theorem for nonuniform CSPs. 2017, arXiv:1703.03021.

[21] J. Bulín, A. Krokhin, and J. Opršal, Algebraic approach to promise constraint satisfaction. In *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*, pp. 602–613, ACM, 2019.

[22] C. Carvalho, B. Martin, and D. Zhuk, The complexity of quantified constraints using the algebraic formulation. In *42nd international symposium on mathematical foundations of computer science, MFCS 2017, August 21–25, 2017 – Aalborg, Denmark*, pp. 27:1–27:14, Leibniz Center for Informatics, 2017.

[23] H. Chen, The complexity of quantified constraint satisfaction: Collapsibility, sink algebras, and the three-element case. *SIAM J. Comput.* **37** (2008), no. 5, 1674–1701.

[24] H. Chen, Meditations on quantified constraint satisfaction. In *Logic and program semantics – essays dedicated to Dexter Kozen on the occasion of his 60th birthday*, pp. 35–49, Springer, 2012.

[25] N. Creignou, S. Khanna, and M. Sudan, *Complexity classifications of boolean constraint satisfaction problems*. SIAM, 2001.

[26] N. Creignou, H. Schnoor, and I. Schnoor, Non-uniform boolean constraint satisfaction problems with cardinality constraint. In *International workshop on computer science logic*, pp. 109–123, Springer, 2008.

[27] V. Dalmau, Some dichotomy theorems on constant-free quantified boolean formulas *Tech. rep. LSI97-43-R. Llenguatges i Sistemes Informátics—Universitat Politécnica de Catalunya, Barcelona, Spain*, 1997.

[28] T. Feder and M. Y. Vardi, Monotone monadic SNP and constraint satisfaction. In *Proceedings of the twenty-fifth annual ACM symposium on theory of computing*, pp. 612–622, Association for Computing Machinery, 1993.

[29] T. Feder and M. Y. Vardi, The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. *SIAM J. Comput.* **28** (1999), no. 1, 57–104.

[30] M. Ficak, M. Kozik, M. Olsak, and S. Stankiewicz, Dichotomy for symmetric boolean PCSPs. 2019, arXiv:1904.12424.

[31] D. Geiger, Closed systems of functions and predicates. *Pacific J. Math.* **27** (1968), no. 1, 95–100.

[32] W. H. Greub, *Linear algebra*. Grad. Texts in Math. 23, Springer, 2012.

[33] P. Idziak, P. Marković, R. McKenzie, M. Valeriote, and R. Willard, Tractability and learnability arising from algebras with few subpowers. *SIAM J. Comput.* **39** (2010), no. 7, 3023–3037.

[34] P. Jeavons, On the algebraic structure of combinatorial problems. *Theoret. Comput. Sci.* **200** (1998), no. 1–2, 185–204.

[35] P. Jeavons, D. Cohen, and M. C. Cooper, Constraints, consistency and closure. *Artif. Intell.* **101** (1998), no. 1–2, 251–265.

[36] M. Kozik, Weak consistency notions for all the CSPs of bounded width. In *2016 31st annual ACM/IEEE symposium on logic in computer science (LICS)*, pp. 1–9, IEEE, 2016.

[37] B. Martin, Quantified Constraints in Twenty Seventeen. In *The constraint satisfaction problem: complexity and approximability*, edited by A. Krokhin and S. Zivny, pp. 327–346, Dagstuhl Follow-Ups 7, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017.

[38] B. Martin and D. Paulusma, The computational complexity of disconnected cut and 2_{K2}-partition. *J. Combin. Theory Ser. B* **111** (2015), 17–37.

[39] E. L. Post, *The two-valued iterative systems of mathematical logic*. Ann. of Math. Stud. 5, Princeton University Press, Princeton, NJ, 1941.

[40] T. J. Schaefer, The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on theory of computing, STOC'78*, pp. 216–226, ACM, New York, NY, USA, 1978.

[41] N. Vikas, Computational complexity of graph partition under vertex-compaction to an irreflexive hexagon. In *42nd international symposium on mathematical foundations of computer science, MFCS 2017*, pp. 69:1–69:14, LIPIcs 83, Leibniz Center for Informatics, 2017.

[42] D. Zhuk, A proof of CSP dichotomy conjecture. In *2017 IEEE 58th annual symposium on foundations of computer science (FOCS)*, pp. 331–342, IEEE, 2017.

[43] D. Zhuk, The size of generating sets of powers. *J. Combin. Theory Ser. A* **167** (2019), 91–103.

[44] D. Zhuk, A proof of the CSP dichotomy conjecture. *J. ACM* **67** (2020), no. 5, 1–78.

[45] D. Zhuk, The complexity of the Quantified CSP having the polynomially generated powers property. 2021, arXiv:2110.09504.

[46] D. Zhuk, No-rainbow problem and the surjective constraint satisfaction problem. In *2021 36th annual ACM/IEEE symposium on logic in computer science (LICS)*, pp. 1–7, IEEE, 2021.

[47] D. Zhuk, Strong subalgebras and the constraint satisfaction problem. *J. Mult.-Valued Logic Soft Comput.* **36** (2021).

[48] D. Zhuk and B. Martin, QCSP monsters and the demise of the Chen conjecture. In *Proceedings of the 52nd annual ACM SIGACT symposium on theory of computing*, pp. 91–104, 2020.

DMITRIY ZHUK

Department of Mechanics and Mathematics, Lomonosov Moscow State University, Vorobjovy Gory, 119899 Moscow, Russia, zhuk@intsys.msu.ru