The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory

Tomás Feder Moshe Y. Vardi
IBM Almaden Research Center
650 Harry Road
San Jose, California 95120-6099

Abstract

This paper starts with the project of finding a large subclass of NP which exhibits a dichotomy. The approach is to find this subclass via syntactic prescriptions. While, the paper does not achieve this goal, it does isolate a class (of problems specified by) "Monotone Monadic SNP without inequality" which may exhibit this dichotomy. We justify the placing of all these restrictions by showing that classes obtained by using only two of the above three restrictions do not show this dichotomy, essentially using Ladner's Theorem. We then explore the structure of this class. We show all problems in this class reduce to the seemingly simpler class CSP. We divide CSP into subclasses and try to unify the collection of all known polytime algorithms for CSP problems and extract properties that make CSP problems NP-hard. This is where the second part of the title – "a study through Datalog and group theory" – comes in. We present conjectures about this class which would end in showing the dichotomy.

1 Introduction

We start with a basic overview of the framework explored in this paper; for an accompanying pictorial description, see Figure 1. A more detailed presentation of the work and its relationship to earlier work is given in the next section.

It is well-known that if P≠NP, then NP contains problems that are neither solvable in polynomial time nor NP-complete. We explore the following question: What is the most general subclass of NP that we can define that may not contain such in-between problems? We investigate this question by means of syntactic restrictions. The logic class SNP is contained in NP, and can be restricted with three further requirements: *monotonicity*, *monadicity*, and *no inequalities*. We show that if any two out of these three conditions are imposed on SNP, then the resulting subclasses of SNP are still general enough to contain a polynomially equivalent problem for every problem in NP, and in particular for the in-between problems in NP. We thus address the question by imposing all three restrictions simultaneously: The resulting subclass of SNP is called *MMSNP*.

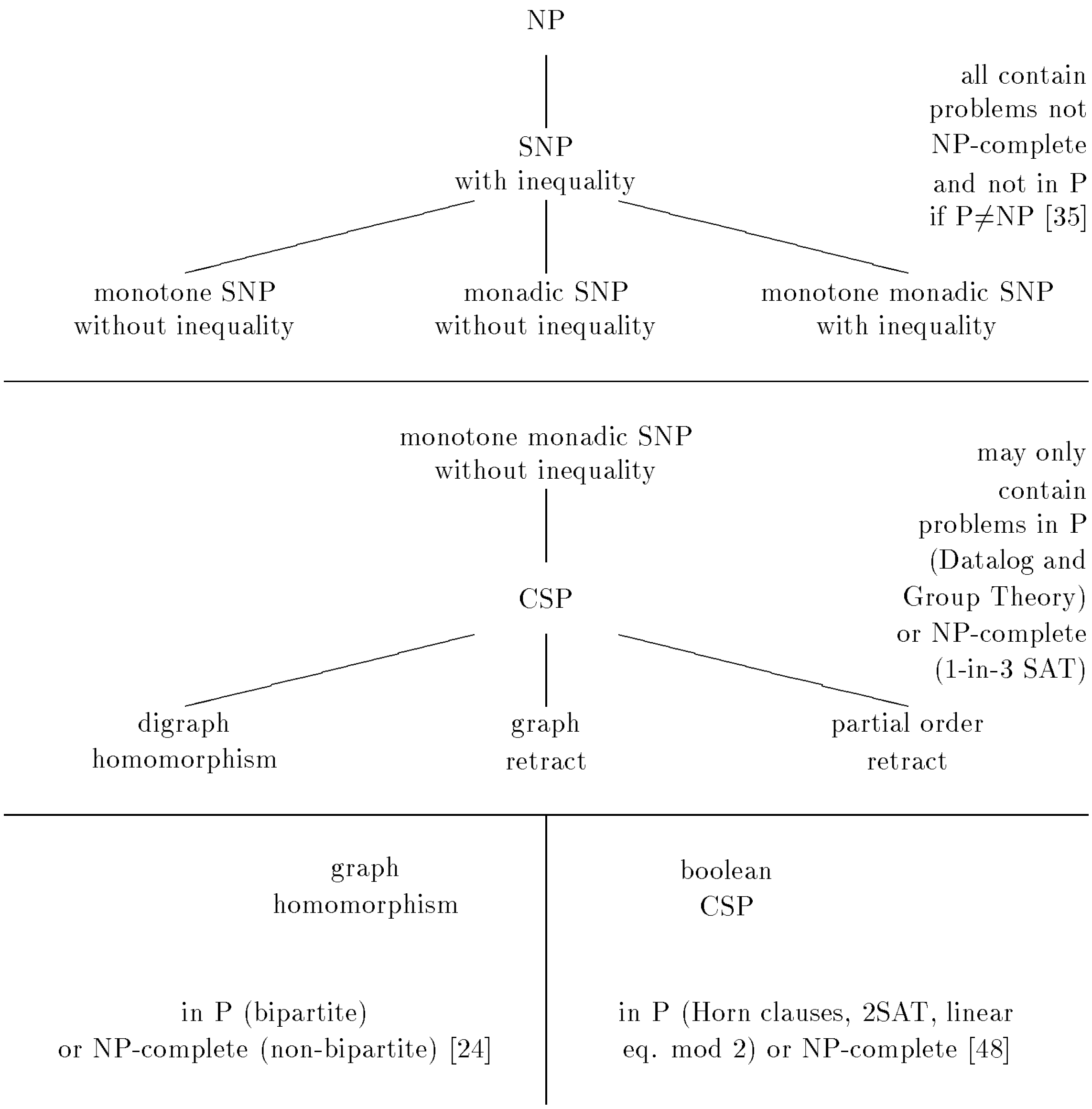

Figure 1: Summary

We examine MMSNP, and observe that it contains a family of interesting problems. A constraint-satisfaction problem is given by a pair I (the *instance*) and T (the *template*) of finite relational structures over the same vocabulary. The problem is satisfied if there is a homomorphism from I to T. It is well-known that the constraint-satisfaction problem is NP-complete. In practice, however, one often encounters the situation where the template T is fixed and it is only the instance I that varies. We define *CSP* to be the class of constraint-satisfaction problems with respect to fixed templates; that is, for every template T, the class CSP contains the problem P_T that asks for an instance I over the same vocabulary as T whether there is a homomorphism from I to T. The class CSP is contained in MMSNP. We show that CSP is in a sense the same as all of MMSNP: Every problem in MMSNP has an equivalent problem in CSP under randomized polynomial time reductions.

The class CSP in turn has some interesting subclasses: the *graph-retract*, *digraph-homomorphism*, and *partial order-retract* problems. We show that in fact every problem in CSP has a polynomially equivalent problem in each of these three subclasses, so that all three of them are as general as CSP. Some special cases were previously investigated: For CSP with a Boolean template it was shown that there are three polynomially solvable problems, namely Horn clauses, 2SAT, and linear equations modulo 2, while the remaining problems are NP-complete; For the graph-homomorphism problem, it was shown that bipartite graph templates are polynomially solvable and non-bipartite graph templates are NP-complete. Could it then be that every problem in CSP is either polynomially solvable or NP-complete?

Some representative problems that were previously observed as belonging to CSP are k-satisfiability, k-colorability, and systems of linear equations modulo q; a polynomially solvable problem that can be less obviously seen to belong to CSP is labeled graph isomorphism. We notice here that at present, all known polynomially solvable problems in CSP can be explained by a combination of Datalog and group theory. More precisely, we define *bounded-width* problems as those that can be defined by Datalog programs, and *subgroup* problems as those whose relations correspond to subgroups or cosets of a given group; both subclasses are polynomially solvable. Two decidable subclasses of the bounded-width case are the *width 1* and the *bounded strict width* problems; in fact the three polynomially solvable cases with a Boolean template are width 1, strict width 2, and subgroup, respectively. The three main polynomial cases, namely width 1, bounded strict width, and subgroup, are all characterized by algebraic closure properties, and so is convex programming, a polynomial constraint satisfaction problem over the reals.

We finally observe that the only way we know how to show that a problem is not bounded-width requires the problem to have a property that we call the *ability to count*; once a problem has the ability to count, it seems that it must necessarily contain the general subgroup problem for an abelian group as a special case; when a new type of subset of a group, which we call *nearsubgroup*, is also allowed in a subgroup problem, the resulting problem reduces to subgroup problems, at least for solvable groups; and if an allowed non-subgroup subset is not a nearsubgroup, then the subgroup problem becomes NP-complete. Does this sequence of observations lead to a classification of the problems in CSP as polynomially solvable or NP-complete?

2 Preliminaries

A large class of problems in AI and other areas of computer science can be viewed as *constraint-satisfaction problems* [10, 34, 40, 41, 42, 43, 45]. This includes problems in machine vision, belief maintenance, scheduling, temporal reasoning, type reconstruction, graph theory, and satisfiability.

We start with some definitions. A *vocabulary* is a set $V = \{(R_1, k_1), \ldots, (R_t, k_t)\}$ of relation names and their arities. A *relational structure* over the vocabulary V is a set S together with relations R_i of arity k_i on the set S. An instance of constraint satisfaction is given by a pair I, T of finite relational structures over the same vocabulary. The instance is satisfied if there is a homomorphism from I to T, that is, there exists a mapping h such that for every tuple $(x_1, \ldots, x_k) \in R_i$ in I we have $(h(x_1), \ldots, h(x_k)) \in R_i$ in T. Intuitively, the elements of I should be thought of as variables and the elements of T should be thought of as possible values for the variables. The tuples in the relations of I and T should be viewed as constraints on the set of allowed assignments of values to variables. The set of allowed assignments is nonempty iff there exists a homomorphism from I to T. In what follows, we shall use the homomorphism and variable-value views interchangeably in defining constraint satisfaction problems.

It is well-known that the constraint-satisfaction problem is NP-complete. In practice, however, one often encounters the situation where the structure T (which we call the *template*) is fixed and it is only the structure I (which we call the *instance*) that varies.

For example, the template of the 3SAT problem has domain $\{0, 1\}$ and four ternary relations C_0, C_1, C_2, C_3 that contain all triples except for $(0, 0, 0)$ in the case of C_0, except for $(1, 0, 0)$ in the case of C_1, except for $(1, 1, 0)$ in the case of C_2, and except for $(1, 1, 1)$ in the case of C_3. The tuples in the instance describe the clauses of the problem. For example, a constraint $C_2(x, y, z)$ imposes a condition on the three variables x, y, z that is equivalent to the clause $\overline{x} \vee \overline{y} \vee z$.

As a second example, the template of the 3-coloring problem is the graph K_3; i.e., it has domain $\{r, b, g\}$ and a single binary relation E that holds for all pairs (x, y) from the domain with $x \neq y$. The tuples in the instance describe the edges of the graph. Thus, the variables $x_1, x_2, \ldots, x_n$ can be viewed as vertices to be colored with r, b, g, and the constraints $E(x_i, x_j)$ can be viewed as describing the edges whose endpoints must be colored differently. If we replace the template K_3 by an arbitrary graph H, we get the so-called *H-coloring* problem [24].

As a third example, given an integer $q \geq 2$, the template of the linear equations modulo q problem has domain $\{0, 1, \ldots, q-1\}$, a monadic relation Z that holds only for the element 0, and a ternary relation C that holds for the triples (x, y, z) with $x + y + z = 1 \pmod{q}$. It is easy to show that any other linear relations on variables modulo q can be expressed by introducing a few auxiliary variables and using only the Z and C relations.

In this paper we consider constraint-satisfaction problems with respect to fixed templates. We define *CSP* to be the class of such problems. It is easy to see that CSP is contained in NP. We know that NP contains polynomially solvable problems and NP-complete problems. We also know that if P≠NP, then there exist problems in NP that are neither in P nor NP-complete [35]. The existence of such "intermediate" problems is proved by a diagonalization argument. It seems, however, impossible to carry this argument in CSP. This motivates our main question:

Dichotomy Question: *Is every problem in CSP either in P or NP-complete?*

Our question is supported by two previous investigations of constraint-satisfaction problems that demonstrated dichotomies. Schaefer [48] showed that there are essentially only three polynomially solvable constraint-satisfaction problems on the set $\{0, 1\}$, namely, (0) 0-valid problems (problems where all-zeros is always a solution, and similarly 1-valid problems); (1) Horn clauses (problems where every relation in the template can be characterized by a conjunction of clauses with at most one positive literal per clause, and similarly anti-Horn clauses, with at most one negative literal per clause); (2) 2SAT (problems where every relation in the template can be characterized by a conjunction of clauses with two literals per clause); (3) linear equations modulo 2 (problems where every relation in the template is the solution set of a system of linear equations modulo 2). All constraint-satisfaction problems on $\{0, 1\}$ that are not in one of these classes are NP-complete. The NP-complete cases include one-in-three SAT, where the template has a single relation containing precisely $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$, and not-all-equal SAT, where the template has a single relation that contains all triples except $(0, 0, 0)$ and $(1, 1, 1)$.

Hell and Nešetřil [24] showed that the H-coloring problem is in P if H is bipartite and NP-complete for H non-bipartite. Bang-Jensen and Hell [8] conjecture that this result extends to the digraph case when every vertex in the template has at least one incoming and at least one incoming and at least one outgoing edge: if the template is equivalent to a cycle then the problem is polynomially solvable, otherwise NP-complete.

The issue that we address first is the robustness of the class CSP. We investigate the dichotomy question in the context of the complexity class *SNP*, which is a subclass of NP that is defined by means of a logical syntax [32, 44], and which, in particular, includes CSP. We show that SNP is too general a class to address the dichotomy question, because every problem in NP has an *equivalent* problem in SNP under polynomial time reductions. Here two problems are said to be equivalent under polynomial time reductions if there are polynomial time reductions from one to the other, in both directions. We then impose three syntactic restrictions on SNP, namely *monotonicity*, *monadicity*, and *no inequalities*, since CSP is contained in SNP with these restrictions imposed. It turns out that if only two of these three restrictions are imposed, then the resulting subclass of SNP is still general enough to contain an equivalent problem for every problem in NP.

When all three restrictions are imposed, we obtain the class *MMSNP*: *monotone monadic* SNP *without inequality*. This class is still more general than CSP, because it strictly contains CSP. We prove, however, that every problem in MMSNP has an equivalent problem in CSP, this time under randomized polynomial time reductions (we believe that it may be possible to derandomize the reduction).

Thus, CSP is essentially the same as the seemingly more general class MMSNP. In the other direction, there are three special cases of CSP, namely, the *graph-retract*, the *digraph-homomorphism*, and the *partial-order-retract* problems, that turn out to be as hard as all of CSP, again under polynomial time reductions. The equivalence between CSP and classes both above and below it seems to indicate that CSP is a fairly robust class.

We then try to solve the dichotomy question by considering a more practical question:

Primary Classification Question: *Which problems in CSP are in P and which are NP-complete?*

In order to try to answer this question, we consider again Schaefer's results for constraint-satisfaction problems on the set $\{0,1\}$ [48]. Schaefer showed that there are only three such polynomially solvable constraint-satisfaction problems. We introduce two subclasses of CSP, namely *bounded-width* CSP and *subgroup* CSP, respectively, as generalizations of Schaefer's three cases. Bounded-width problems are problems that can be solved by considering only bounded sets of variables, which we formalize in terms of the language Datalog [50]. Both Horn clauses and 2SAT fall into this subclass; we show that linear equations modulo 2 do not. Subgroup problems are group-theoretic problems where the constraints are expressed as subgroup constraints. Linear equations modulo 2 fall into this subclass. Not only are these subclasses solvable in polynomial time, but, at present, *all* known polynomially solvable constraint-satisfaction problems can be explained in terms of these conditions.

Assuming that these conditions are indeed the only possible causes for polynomial solvability for problems in CSP, this poses a new classification problem:

Secondary Classification Question: *Which problems in CSP are bounded-width problems and which are subgroup problems?*

The main issue here is that it is not clear whether membership in these subclasses of CSP is decidable.

Our results provide some progress in understanding the bounded-width and subgroup subclasses. For example, for the bounded-width problems, our results provide a classification for the 1-width problems in CSP (these are the problems that can be solved by *monadic* Datalog programs). We also identify a property of problems, which we call *the ability to count*. We prove that this property implies that the problem cannot be solved by means of Datalog. Once a constraint-satisfaction problem has the ability to count, it is still possible in many cases to solve it by group-theoretic means.

While all known polynomially solvable problems in CSP can be reduced to the bounded-width and group-theoretic subclasses, not all such problems belong to those classes from the start. For example, we show that under some conditions non-subgroup problems can be reduced to the subgroup subclass. These conditions are stated in terms of the new notion of *nearsubgroup*, and delineating the boundary between polynomially solvable and NP-complete group-theoretical problems seems to require certain progress in finite-group theory.

The remainder of the paper is organized as follows. Section 3 introduces the logic class MMSNP as the largest subclass, in some sense, of SNP, that is not computationally equivalent to all of NP. Section 4 introduces the class CSP as a subclass of MMSNP which is essentially equivalent to MMSNP. Section 5 studies three subclasses of CSP, the graph retract, digraph homomorphism, and partial order retract problems, essentially equivalent to all of CSP. Section 6 considers classes of problems in CSP that are polynomially solvable. Section 6.1 considers the bounded width problems which are those that can be solved by means of Datalog, and their relationship to two-player games. Sections 6.1.1 and 6.1.2 examine two special subclasses of the bounded width case for which membership is decidable, namely the width 1 case with its connection to the notion of tree duality, and the strict width l case with its connection to the Helly property. Section 6.2 examines which problems are not of bounded width via a notion called the ability to count. Section 6.3 considers the group-theoretic case, and introduces the notion of nearsubgroup in an attempt to understand the boundary between tractability and intractability. Section 7 explores further directions for a possible complete classification. An appendix explores the connection between constraint satisfaction and Tom Etter's link systems.

3 Monotone Monadic SNP without inequality

The class SNP [32, 44] (see also [14]) consists of all problems expressible by an existential second-order sentence with a universal first-order part, namely, by a sentence of the form $(\exists S')(\forall \mathbf{x})\Phi(\mathbf{x}, S, S')$, where Φ is a first-order quantifier-free formula. That is, Φ is a formula built from relations in S and S' applied to variables in $\mathbf{x}$, by means of conjunctions, disjunctions, and negation. Intuitively, the problem is to decide, for an input structure S, whether there exists a structure S' on the same domain such that for all values in this domain for the variables in $\mathbf{x}$ it is true that $\Phi(\mathbf{x}, S, S')$ holds. We will refer to the relations of S as *input relations*, while the relations of S' will be referred to as *existential relations*. The 3SAT problem is an example of an SNP problem: The input structure S consists of four ternary relations C_0, C_1, C_2, C_3, on the domain $\{0, 1\}$, where C_i corresponds to a clause on three variables with the first i of them negated. The existential structure S' is a single monadic relation T describing a truth assignment. The condition that must be satisfied states that for all x_1, x_2, x_3, if $C_0(x_1, x_2, x_3)$ then $T(x_1)$ or $T(x_2)$ or $T(x_3)$, and similarly for the remaining C_i by negating $T(x_j)$ if $j \leq i$. We are interested in the following question:

Which subclasses of NP have the same computational power as all of NP?

That is, which subclasses of NP are such that for every problem in NP there is a problem in the subclass equivalent to it under polynomial time reductions. More precisely, we say that two problems A and B are *equivalent* under polynomial time reductions if there is a polynomial time reduction from A to B, as well as a polynomial time reduction from B to A. It turns out that every problem in NP is equivalent to a problem in SNP under polynomial time reductions. This means that for every problem A in NP, there is a problem B in SNP such that there is a polynomial time reduction from A to B, as well as a polynomial time reduction from B to A. In fact, we now show that this is the case even for restrictions of SNP. We start by assuming that the equality or inequality relations are not allowed in the first order formula, only relations from the input structure S or the existential structure S'. For *monotone* SNP *without inequality*, we require that all occurrences of an input relation C_i in Φ have the same polarity (the polarity of a relation is positive if it is contained in an even number of subformulas with a negation applied to it, and it is negative otherwise); by convention, we assume that this polarity is negative, so that the C_i can be interpreted as constraints, in the sense that imposing C_i on more elements of the input structure can only make the instance "less satisfiable". Note that 3SAT as described above has this property. For *monadic* SNP *without inequality*, we require that the existential structure S' consist of monadic relations only. This is again the case for 3SAT described above. For *monotone monadic* SNP *with inequality*, we assume that the language contains also the equality relation, so both equalities and inequalities are allowed in Φ. (If we consider that equalities and inequalities appear with negative polarity, then only inequalities give more expressive power, since a statement of the form 'if $x = y$ then $\Phi(x, y)$' can be replaced by '$\Phi(x, x)$'.)

We have thus taken the class SNP, and we are considering three possible syntactic restrictions, namely *monotonicity*, *monadicity*, and *no inequality.* We shall later be especially interested in SNP with all three syntactic restrictions imposed. However, for now, we are only considering the cases where only two of these three syntactic restrictions are simultaneously imposed.

Theorem 1 *Every problem in NP has an equivalent (under polynomial time reductions) problem in monotone monadic SNP with inequality.*

Proof. Hillebrand, Kanellakis, Mairson and Vardi [30] showed that monadic Datalog with inequality (but without negation) can verify a polynomial time encoding of a Turing machine computation; the machine can be nondeterministic. A Datalog program is a formula Φ that consists of a conjunction of formulas of the form $R_0(\mathbf{x}_0) \leftarrow R_1(\mathbf{x}_1) \wedge \cdots \wedge R_k(\mathbf{x}_k)$, where the $\mathbf{x}_i$ may share variables. The relation R_0 cannot be an input relation, and monadicity here means that R_0, as a relation that is not an input relation, must be monadic or of arity zero; furthermore an R_i may be an inequality relation. There is a particular $\hat{R}_0$ of arity zero that must be derived by the program in order for the program to accept its input; this means that the input is rejected by the Datalog program if $(\exists \mathbf{R})(\forall \mathbf{x})(\Phi(\mathbf{R}, \mathbf{S}, \mathbf{x}) \wedge \neg \hat{R}_0)$. Notice that this formula Φ' is a monotone monadic SNP with inequality formula. Here $\mathbf{S}$ describes the computation of a nondeterministic Turing machine, including the input, the description of the movement of the head on the tape of the machine, and the states of the machine and cell values used during the computation. We would now like to assume that the computation of the machine is not known ahead of time; that is, only the input to the machine is given, the movement of the head and the cell values are not known, and are quantified existentially. Unfortunately, the description of the movement of the head does not consist of monadic relations, and may depend on the input to the machine. We avoid this difficulty by assuming that the Turing machine is *oblivious*, i.e., the head traverses the space initially occupied by the input back and forth from one end to the other, and accepts in exactly n^k steps for some k. We can then assume that the movement of the head is given as part of the input, since it must be independent of the input for such an oblivious machine. Thus only the states of the machine and cell values used during the computation must be quantified existentially, giving a monotone monadic SNP with inequality formula that expresses whether the machine accepts a given input. A particular computation is thus described by a choice of states and cell values, which are described by monadic existential relations that are then used as inputs to the Datalog program. The condition that must be satisfied is that if a state is marked as being the (n^k)th state (this is determined by a deterministic component of the machine), then it must also be marked as being an accepting state (this depends on the nondeterministic choice of computation). The monotone monadic SNP with inequality formula will thus reject an instance if it does not describe an input followed by the correct movement of the head for the subsequent oblivious computation, accept the instance if the number of cells allowed for the computation is smaller than n^k, and otherwise accept precisely when the machine accepts. $\square$

Theorem 2 *Every problem in NP has an equivalent (under polynomial time reductions) problem in monadic SNP without inequality.*

Proof. Since the existence of an equivalent problem in monotone monadic SNP with inequality for every problem in NP was previously shown, it is sufficient to remove inequalities at the cost of monotonicity.

To remove inequalities at the cost of monotonicity, introduce a new binary input relation *eq*, augment the formula by a conjunct requiring *eq* to be an equivalence relation with the property that if an input or existential monadic relation holds on some elements, then it also holds when an element in an argument position is replaced by an element related to it under *eq*; finally replace all occurrences of $x \neq y$ by $\neg eq(x, y)$. Thus the formula no longer contains inequalities, but it contains an input relation that appears with both positive and negative polarity, i.e., it is no longer monotone. The formula is therefore a monadic SNP without inequality formula. □

Theorem 3 *Every problem in NP has an equivalent (under polynomial time reductions) problem in monotone SNP without inequality.*

Proof. Since the existence of an equivalent problem in monotone monadic SNP with inequality for every problem in NP was previously shown, it is sufficient to remove inequalities at the cost of monadicity.

To remove inequalities at the cost of monadicity, the intuition is that up to equivalence, certain *marked* elements form a *succ* path with *pred* as its transitive closure. Introduce a monadic input relation *special*, a binary input relation *succ*, a monadic existential relation *marked*, a binary existential relation *eq*, and a binary existential relation *pred*. Require now every *special* element to be *marked*, and every element related to a *marked* element under *succ* (in either direction) to be *marked*. Require that *pred* be transitive but not relate any element to itself, that two elements related by *succ* be related by *pred* (in the same direction), that *eq* be an equivalence relation, that any two *special* elements be related by *eq*, that *pred* be preserved under the replacement of an element by an element related to it by *eq*, and that if two elements are related by *eq* and if each has a related element under *succ* (in the same direction), then these two other elements are also related by *eq*. Finally, restrict the original formula to *marked* elements, replace $x \neq y$ by $\neg eq(x, y)$, and consider that a relation holds on some elements if it is imposed on elements related to them by *eq*. Note that on elements that are forced to be *marked*, the relations *eq* and *pred* can be defined in at most one way, giving a *succ* path (up to *eq*, with *pred* as its transitive closure). □

From these three theorems, by Ladner's result [35] that if P≠NP then there exist problems in NP that are neither in P nor NP-complete, it follows that:

Theorem 4 *If* P $\neq$ NP, *then there are problems in each of monotone monadic SNP with inequality, monadic SNP without inequality, and monotone SNP without inequality, that are neither in* P *nor* NP-*complete.*

We now consider the class *MMSNP*, which is *monotone monadic SNP, without inequality*. That is, in MMSNP we impose all three restrictions simultaneously (instead of just two at a time as in the three subclasses of SNP considered above). It seems impossible to carry out Ladner's diagonalization argument in MMSNP. Thus, the dichotomy question from the introduction applies also to this class.

4 Constraint Satisfaction

Let S and T be two finite relational structures over the same vocabulary. A *homomorphism* from S to T is a mapping from the elements of S to elements of T such that all elements related by some relation C_i in S map to elements related by C_i in T. If T is the substructure of S obtained by considering only relations on a subset of the elements of S, and the homomorphism h from S to T is just the identity mapping when restricted to T, then h is called a *retraction*, and T is called a *retract* of S. If no proper restriction T of S is a retract of S, then S is a *core*, otherwise its core is a retract T that is a core. It is easy to show that the core of a structure S is unique up to isomorphism.

A *constraint-satisfaction problem* (or *structure-homomorphism problem*) will be here a problem of the following form. Fix a finite relational structure T over some vocabulary; T is called the *template*. An *instance* is a finite relational structure S over the same vocabulary. The instance is satisfied if there is a homomorphism from S to T. Such a homomorphism is called a *solution*. We define *CSP* to be the class of constraint-satisfaction problems. We can assume that T is a core and include a copy of T in the input structure S, so that the structure-homomorphism problem is a structure-retract problem.

Remark: It is possible to define constraint satisfaction with respect to infinite templates. For example, digraph acyclicity can be viewed as the question of whether a given digraph can be homomorphically mapped to the transitive closure of an infinite directed path. We will not consider infinite templates in this paper. If we allow infinite structures T, then the constraint-satisfaction problems are just the problems whose complement is closed under homomorphisms, with the additional property that an instance with satisfiable connected components is satisfiable. Note that all problems in monotone SNP, without inequality, have a complement closed under homomorphisms.

We shall later show that CSP is strictly contained in MMSNP, and that every problem in MMSNP has an essentially equivalent problem in CSP, up to randomized polynomial time reductions. To obtain this result, we need a preliminary result that will often be used later in the paper as well.

We say that a structure S has *girth* greater that k if for any choice of at most k occurrences of relations R_i of arity r_i in S, the total number of elements mentioned by these k occurrences is at least $1 + \sum(r_i - 1)$. That is, k or fewer occurrences of relations never form a cycle.

Theorem 5 *Fix two integers k, d. Then for every structure S on n elements there exists a structure S' on n^a elements (where a depends only on k and d) such that the girth of S' is greater than k, there is a homomorphism from S' to S, and for every structure T on at most d elements over the same vocabulary as S, there is a homomorphism from S' to T if and only if there is a homomorphism from S to T. In brief, every instance S of the constraint-satisfaction problem defined by T can be replaced by an instance S' of high girth. Furthermore, S' can be constructed from S in randomized polynomial time.*

Proof. The transformation that enforces large girth is an adaptation of a randomized construction of Erdős [11] of graphs with large girth and large chromatic number. Given a structure S on n elements, define S' by making $N = n^s$ copies of each element, where

s is a large constant. If a relation R of arity r was initially imposed on some r elements, then it could a priori be imposed on N^r choices of copies. Impose R on each such choice with probability $N^{1-r+\epsilon}$, where ϵ is a small constant. We thus expect to impose R on $N^{1+\epsilon}$ copies. If R has arity $r=1$, impose R on all copies of the element.

Finally, remove one relation from each cycle with at most k relations, i.e., minimal sets of relations R_i of arity r_i involving $t \leq \sum(r_i - 1)$ elements all together. Now, given such a cycle, it must correspond to a cycle that existed before the copies were made. The number of possible such short cycles is at most n^a for some constant a. Each such short cycle could occur in N^t choices of copies. For each such choice, the probability that it occurs is $\prod N^{1-r_i+\epsilon}$, so the expected number of occurrences is $n^a N^t \prod N^{1-r_i+\epsilon} \leq N^{a/s+k\epsilon} = N^{\epsilon'}$, and hence no more than twice this much with probability at least $1/2$ by Markov's inequality.

It is clear that if before making copies, there is a homomorphism from S to some T, then there is a homomorphism from S' to T as well: S' maps to S by a homomorphism. To obtain a converse, suppose that a homomorphism maps S' to T. If we consider the N copies of a particular element, then at least N/d of the copies agree on the image in T. If we select these copies, for each of the n elements, then the expected number of copies of a relation R of arity r is $(N/d)^r N^{1-r+\epsilon} = N^{1+\epsilon}/b$ for some constant b, and hence the probability that the number of copies is not even half this much is only $e^{-N^{1+\epsilon}/c}$ for some constant c by the Chernoff bound, since the occurrences of copies are independent. The total number of occurrences of relations R in the instance is n^r, and the number of possible choices of subsets of size N/d for the copies of the elements involved is at most 2^{rN}, and hence the probability that some choice of subsets will involve only $N^{1+\epsilon}/2b$ copies of some relation is at most $n^r 2^{rN} e^{-N^{1+\epsilon}/c}$, hence very small. Once $N^{1+\epsilon}/2b$ copies are present, the removal of $2N^{\epsilon'}$ of them is insignificant, provided s is large enough and ϵ is small enough. Therefore any choice of values in T for elements that appears on N/d of the copies will give a mapping from S to T. ∎

It remains open to derandomize this construction. The key question seems to be whether the construction of Erdős can be derandomized, i.e., whether given a fixed integer k, for integers n, there is a deterministic algorithm running in time polynomial in n that produces a graph of size polynomial in n, chromatic number at least n, and girth at least k. It may be possible to use quasi-random graphs for this purpose.

It is easy to see that CSP is contained in MMSNP. Let T be a template. Then there is a monadic monotone existential second-order sentence ϕ_T (without inequality) that expresses the constraint-satisfaction problem defined by T. For each element a in the domain of the template T, we introduce an existentially quantified monadic relation T_a; intuitively, $T_a(x)$ indicates that a variable x has been assigned value a by the homomorphism. The sentence ϕ_T says that the sets T_a are disjoint and that the tuples of S satisfy the constraints given by T.

It can be shown that CSP is strictly contained in MMSNP. Nevertheless, as the following two theorems show, in terms of the complexity of its problems, CSP is just as general as MMSNP.

We begin with a simple example of an MMSNP problem that is not a constraint-satisfaction problem: testing whether a graph is triangle-free. If it were a constraint-satisfaction problem, there would have to exist a triangle-free graph to which one can map all triangle-free graphs by a homomorphism. This would require the existence of a triangle-free graph containing as induced subgraphs all triangle-free graphs such that all non-adjacent vertices are joined both by a path of length 2 and a path of length 3 (since a homomorphism can add edges or collapse two vertices); there are $2^{\Omega(n^2)}$ such graphs on n vertices, forcing T to grow exponentially in the size of S. On the other hand, this monotone monadic SNP problem can be solved in polynomial time, and is hence equivalent to a trivial constraint-satisfaction problem.

A more interesting example is the following: testing whether a graph can be colored with two colors with no monochromatic triangle (it can easily be related to the triangle-free problem to show that it is not a constraint-satisfaction problem). However, it can be viewed as a special case of not-all-equal 3SAT, where each clause is viewed as a triangle, and it is essentially equivalent to this NP-complete constraint-satisfaction problem.

Theorem 6 *CSP is contained in MMSNP. Every problem in MMSNP is polynomially equivalent to a problem in CSP. The equivalence is by a randomized Turing reduction from the CSP problem to the MMSNP problem and by a deterministic Karp reduction from the MMSNP problem to the CSP problem.*

Proof. We just saw that CSP is contained in MMSNP; we prove the second statement. In fact, we can use a randomized Karp reduction if we only consider connected instances of the constraint-satisfaction problem; disconnected instances require simply a solution for each connected component.

Consider an MMSNP problem that asks for a input structure S whether there exists a monadic structure S' such that for all $\mathbf{x}$, $\Phi(\mathbf{x}, S, S')$. We write Φ in conjunctive normal form, or more precisely, as a conjunction of negated conjunctions. We can assume that each negated conjunction describes a biconnected component. For consider first the disconnected case, so that we have a conjunct of the form $\neg(A(\mathbf{x}) \wedge B(\mathbf{y}))$, where $\mathbf{x}$ and $\mathbf{y}$ are disjoint variable sets. We can then introduce an existential zero-adic relation p, and write instead $(A(\mathbf{x}) \to p) \wedge (B(\mathbf{y}) \to \neg p)$. The case where A and B share a single variable z is treated similarly; we introduce an existential monadic relation q, and replace $\neg(A(\mathbf{x}, z) \wedge B(\mathbf{y}, z))$ by $(A(\mathbf{x}, z) \to q(z)) \wedge (B(\mathbf{y}, z) \to \neg q(z))$. If the conjunction cannot be decomposed into either two disconnected parts, or two parts that share a single articulation element z, we say that it describes a biconnected component. Before carrying out this transformation, we assume for each negated conjunction that every replacement of different variables by the same variable is also present as a negated conjunction. For instance, if $\neg A(x, y, z, t, u)$ is present, then so is $\neg A(x, x, z, z, u)$. This must be enforced beforehand, since biconnected components may no longer be biconnected when distinct variables are collapsed. We also assume that if an input relation R appears with all arguments equal, say as $R(x, x, x)$, then it is the only input relation in the negated conjunction; otherwise, we can introduce an existential monadic p, replace such an occurrence of R by $p(x)$, and add a new condition $R(x, x, x) \to p(x)$; in other words, x is in this case an articulation element.

The next transformation is the main step; we enforce that each negated conjunction contains at most one input relation, and that the arguments of this input relation are different variables. For each negated conjunction, introduce a relation R whose arity is the number of distinct variables in the negated conjunction. Intuitively, this relation stands for the conjunction C of all input relations appearing in the negated conjunction. We replace the conjunction C by the single relation R; also, if the corresponding conjunction C' for some other negated conjunction is a sub-conjunction of C, then we include the negated conjunction obtained by replacing C' with the possibly longer C, using new variables if necessary, and then replace C by R. Here we are not considering the case where it might be necessary to replace two arguments in R by the same argument; this will be justified because such instantiations were handled beforehand.

We must argue that the new MMSNP problem of the special form is equivalent to the original problem. Clearly every instance of the original problem can be viewed as an instance of the new problem, simply by introducing a relation R on distinct input elements whenever the conjunction C that is represented by R is present on them in the input instance.

On the other hand, the converse is not immediately true. If we replace each occurrence of R by the appropriate conjunction C, then some additional occurrences of R may be implicitly present. Consider, for example, the case where triangles $E(x,y) \wedge E(y,z) \wedge E(z,x)$ have been replaced by a single ternary relation $R(x,y,z)$. Then an instance of the new problem containing $R(x_1,y_1,x_2)$, $R(x_2,y_2,x_3)$ and $R(x_3,y_3,x_1)$ also contains the triangle represented by $R(x_3,x_2,x_1)$, when each R is replaced by the conjunction that it stands for.

To avoid such hidden occurrences of relations, we show that every instance of the new problem involving the R relations can be transformed into an equivalent instance of large *girth*; the girth is the length of the shortest cycle. Fix an integer k larger than the number of conjuncts in any conjunction C that was replaced by an R. We shall ensure that for any choice of at most k occurrences of relations R_i of arity r_i in the instance, the total number of elements mentioned by these k occurrences is at least $1+\sum(r_i-1)$, so the girth is greater than k; this implies that such k occurrences define an acyclic sub-structure, so any biconnected R' implicitly present in the union of k such occurrences must be entirely contained in one of the R_i, and then the condition stated by R' was already stated for this R_i as well.

Large girth can be enforced by applying the previous theorem, provided the new problem can be viewed as a constraint-satisfaction problem. The new problem is very close to a constraint-satisfaction problem. In fact, it is a constraint-satisfaction problem if there are no zero-ary existential relations: Construct the structure T by introducing one element for each combination of truth assignments for existential monadic relations on a single element, except for those combinations explicitly forbidden by the formula; impose a relation R on all choices of elements in T except for those combinations explicitly forbidden by the formula. To remove the assumption that there are no existential zero-ary relations, do a case analysis on the possible truth assignments for such relations, and make T the disjoint union of the T_i obtained in the different cases. The only difficulty here is that we must ensure that a disconnected instance still maps to a single T_i, so we introduce a new binary relation that holds on all pairs of elements from the same T_i, and consider only connected instances of the constraint-satisfaction problem. This concludes the proof. As mentioned before, solving disconnected instances is equivalent to solving all connected components of the constraint-satisfaction problem. □

The construction in the last theorem can also be used to show the following result, which will be proven useful later. The *containment* problem asks whether given two problems A and B over the same vocabulary, every instance accepted by A is also accepted by B.

Theorem 7 *Containment is decidable for problems in MMSNP.*

Proof. (This problem becomes undecidable when the antecedent of the containment is generalized to monotone binary SNP without inequality, using Datalog to encode Turing machines as before). To decide whether A is contained in B for MMSNP problems, first assume that A and B are written in the canonical form involving biconnected components from the above proof. Also remove the existential quantifier in A (since it is in the antecedent of an implication), so that A is now a universal formula which is monotone except for monadic relations. Now, if B has an instance with no solution that satisfies A, then making copies of elements of the instance as before, we can assume that the only biconnected components that arise are those explicitly stated in the conditions for B, so go through the forbidden biconnected components stated in A and remove all negated conjunctions in B that mention them (since the stated condition will never arise on instances satisfying A). Here we must assume that B stated explicitly for each element mentioned in a negated conjunction which monadic relations are true or false. Now we can assume that A holds, and we are left to decide whether B is a tautology; this can be decided by considering the instance consisting of one element for each possible combination of truth and falsity of monadic input relations, and then imposing all other kinds of relations on all elements. □

5 Graphs, Digraphs, Partial Orders

We have seen that CSP has the same computational power as all of MMSNP. We ask the following question:

Which subclasses of CSP have the same computational power as all of CSP?

The *graph-retract problem* is an example of a constraint-satisfaction problem. Fix a graph H, and for an input graph G containing H as a subgraph, ask whether H is a retract of G. (Note that when G and H are disjoint, we get the *graph homomorphism* or H-coloring problem mentioned in the introduction [24].)

The *digraph-homomorphism problem* is another example of a constraint-satisfaction problem: this is the case where the template is a digraph. For an oriented cycle (cycle with all edges oriented in either direction), the *length* of the cycle is the absolute value of the difference between edges oriented in one direction and edges oriented in the opposite direction. A digraph is *balanced* if all its cycles have length zero, otherwise it is *unbalanced*. The vertices of balanced digraphs are divided into levels, defined by $level(v) = level(u) + 1$ if (u, v) is an edge of the digraph.

A *partial order* is a set with a reflexive antisymmetric transitive relation $\leq$ defined on it. If reflexive is replace by antireflexive, we have a *strict partial order*. We may also consider homomorphism and retract problems for partial orders.

Theorem 8 *Every constraint-satisfaction problem is polynomially equivalent to a bipartite graph-retract problem.*

Proof. First ensure that the structure T defining the problem in CSP is a core. This ensures that each element in T is uniquely identifiable by looking at the structure T, up to isomorphisms of T, i.e., we can include a copy of T in an instance S and then assume that the elements of the copy of T in S must map to the corresponding elements of T. Next, assume that T can be partitioned into disjoint sets A_j so that for each relation C_i, the possible values for each argument come from a single A_j, and the possible values for different arguments come from different A_j; this can be ensured by making copies A_j of the set of elements of T and allowing an equality constraint between copies of the same element in different A_j. Now, the bipartite graph H consists of a single vertex for each A_j which is adjacent to vertices representing the elements of A_j; a single vertex for each relation C_i which is adjacent to vertices representing the tuples satisfying C_i; a bipartite graph joining the tuples coming from each C_i to the elements of the tuples from the A_j; and two additional adjacent vertices, one of them adjacent to all the elements of sets A_j, and the other one adjacent to all the tuples for conditions C_i.

To see that the resulting retract problem on graphs is equivalent to the given constraint-satisfaction problem, observe in one direction that an instance of the constraint-satisfaction problem can be transformed into an instance of the graph retract problem, by requiring each element to range over the copy A_1 (just make it adjacent to the vertex for A_1 in H), and then to impose a constraint C_i of arity r on some elements, create a vertex adjacent to the vertex for C_i, and make this vertex adjacent to r vertices, each of which is adjacent to the vertex for the appropriate A_j (the r values for j are distinct); then make sure that the value chosen in A_j is the same as the value for the intended element in A_1, using an equality constraint. In the other direction, an instance of the graph retract problem can be assumed to be bipartite, since H is bipartite; furthermore, each vertex can be assumed to be adjacent to either an A_j or C_i, since all other vertices can always be mapped to the two additional vertices that were added for H at the end of the construction. Then each vertex adjacent to vertex A_j can be viewed as an element ranging over A_j, and each vertex adjacent to vertex C_i can be viewed as the application of C_i on certain elements. □

Given a bipartite graph H, we say for two vertices x, y on the same side of H that x *dominates* y if every neighbor of y is a neighbor of x. We say that H is *domination-free* if it has no $x \neq y$ such that x dominates y. If R is a graph, we say that H is R-free if H contains no induced subgraph isomorphic to R. We are interested in the cases where R is either $K_{3,3}$, a complete bipartite graph with three vertices on each side, or $K_{3,3}$ minus a single edge. The following result will be used later in the study of the partial order-retract problem.

Theorem 9 *Every constraint-satisfaction problem is polynomially equivalent to a domination-free $K_{3,3}$-free $K_{3,3}\backslash\{e\}$-free bipartite graph-retract problem.*

Proof. We first show that every constraint-satisfaction problem is equivalent to a $K_{3,3}$-free $K_{3,3}\backslash\{e\}$-free bipartite graph-retract problem. We then show how domination-freedom can in addition also be achieved.

We know that every constraint-satisfaction problem can be encoded as a bipartite graph-retract problem. To achieve $K_{3,3}$-freedom and $K_{3,3}\backslash\{e\}$-freedom, we encode the bipartite graph-retract problem again as a bipartite graph-retract problem, by reusing essentially the same reduction.

So we are given a bipartite graph-retract problem with template $H = (S, T, E)$, which we shall show polynomially equivalent to another bipartite graph-retract problem with template $H' = (U, V, F)$. We introduce five new elements r, s, t, s', t', and define H' by $U = \{r\} \cup S \cup T$, $V = \{s, t, s', t'\} \cup E$, and $F = (\{r\} \times (\{s', t'\} \cup E)) \cup (S \times \{s, s'\}) \cup (T \times \{t, t'\}) \cup \{(u, e) : u \in S \cup T, e \in E, u \in e\}$.

Let $G = (S', T', E')$ be an instance for H. (We can assume G bipartite since it otherwise cannot map to H, and that we know that S' maps to S and T' maps to T because it is connected to the subgraph H, any other component of G can be mapped to a single edge of H.) We define an instance $G' = (U', V', F')$ for H' by letting $U' = \{r\} \cup S' \cup T'$, $V' = \{s, t, s', t'\} \cup E'$, and defining F' by letting r be adjacent to all of E', s adjacent to all of S', t adjacent to all of T', and each $e \in E'$ adjacent to the two vertices in $S' \cup T'$ it joins in G. It is immediate in the instance G' for H' that S' must map to S, T' to T, and E' to E with of $e \in E'$ mapping to the element of E joining the images of the two vertices incident on e in G, so the retractions mapping G to H and those mapping G' to H' correspond to each other.

In the other direction, let $G' = (U', V', F')$ be an instance for H'. Since s', t' dominate s, t respectively, no element need ever be mapped to s, t, other than s, t themselves. So we can require that the neighbors of s, t map to elements of S, T respectively, and then remove s, t from H' and G'. We can then assume that every element of U' that is not required to map to S or T maps to r, since r is adjacent to what remains of V. We can now remove r from H' and G'. Now if a vertex in V' is only adjacent to vertices that map to S we map it to s', if only to vertices that map to T we map it to t', and if to both it must map to E, thus defining a retract problem instance for H.

It only remains to show that H' is $K_{3,3}$-free and $K_{3,3}\backslash\{e\}$-free, and then to enforce dominance-freedom. Suppose that H' contains H_0 which is either a $K_{3,3}$ or a $K_{3,3}\backslash\{e\}$. Then every vertex v in the right side of H_0 must belong to the 3-side of a $K_{2,3}$. This immediately gives $v \neq s, t$ because any pair of neighbors of s is only adjacent to s, s', and similarly for t. So we can remove s, t in looking for H_0. Two vertices in S, T respectively share only one neighbor, so H_0 involves at most one of S, T, and we can remove one them, say T. Two vertices in S have only s' as a common neighbor, so H_0 can have at most one vertex u in S. But this leaves only two vertices r, u in one side, so there is no H_0.

The last step enforces dominance-freedom. Suppose that x dominates y. Let H_1 be the graph consisting of an 8-cycle $C_8 = (1, 2, 3, 4, 5, 6, 7, 8)$, a 4-cycle $C_4 = (1', 2', 3', 4')$, and additional edges joining each i' to both i and $i+4$. We join H_1 to H', with $y = 1'$ as common vertex in H_1 and H'. This does not introduce any new dominated vertices, and y is no longer dominated. Furthermore H_1 contains no $K_{3,2}$. So we only need to show that joining an H_1 at a vertex y gives an equivalent retract problem. If an instance for H' maps to H', it also maps to H' with H_1 joined; if it maps to H' with H_1 joined, since no vertices are forced to map to H_1 other than y, we can map all of H_1 to y and one of its neighbors in H', so the instance maps to H'. In the other direction, consider an instance for H' with H_1 joined. Certain vertices are required to map to specific vertices in C_8. If a vertex is adjacent to two vertices at distance 2 in C_8, then it must map to their unique common neighbor in C_8. So if a vertex v is adjacent to vertices in C_8, we may assume it is adjacent to either just one vertex in C_8 or two opposite vertices in C_8; in either case, we may assume that such v maps to the unique vertex in C_4 having these adjacencies, and remove C_8 from the template. So the template is now H' with C_4 joined at a vertex $y = 1'$. Now for $3'$, we may insist that its neighbors map to $\{2', 4'\}$, and no other vertex maps to $3'$ since $1'$ now dominates $3'$, so we may remove $3'$ from the graph. Then if a vertex is labeled $2'$, $4'$, or $\{2', 4'\}$, its neighbors must map to $1'$, and we may remove $2'$ and $4'$ from the graph since every neighbor of $y = 1'$ in H' now dominates them. So we have reduced the instance for H' with H_1 joined to an instance for H' alone, as desired. $\Box$

Theorem 10 *Every constraint-satisfaction problem is polynomially equivalent to a balanced digraph-homomorphism problem.*

Proof. We encode the graph-retract problem as a balanced digraph-homomorphism problem. Draw the bipartite graph with one vertex set on the left and the other on the right, and orient the edges from left to right. What remains is to distinguish the different vertices on each side; we describe the transformation for vertices in the right, a similar transformation is carried out for vertices in the left. In an oriented path, let 1 denote a forward edge and 0 a backward edge. If there are k vertices $0, 1, \ldots, k-1$ on the right, add starting at the ith vertex an oriented path $(110)^i 1 (110)^{k-i-1} 11$. The intuition is that none of these paths maps to another one of them, and that if a digraph maps to two of them, then it maps to $(110)^k 11$, hence to all of them. Furthermore, the question of whether a digraph maps to an oriented path is polynomially solvable, see sections 6.1.1 and 6.1.2. $\Box$

Theorem 11 *Every constraint-satisfaction problem is polynomially equivalent to an unbalanced digraph-homomorphism problem.*

Proof. First assume that the given constraint-satisfaction problem consists of a single relation R of arity k; multiple relations can always be combined into a single relation by taking their product, adding their arities. Now, define a new constraint satisfaction problem whose domain consists of k-tuples from the original domain; thus R is now a monadic relation. In order to be able to state an equality constraint among different components of different tuples, define a 'shift' relation $S(t, t')$ on tuples $t = (x_1, x_2, \ldots, x_{k-1}, y)$ and $t' = (z, x_1, x_2, \ldots, x_{k-1})$; one can use such shifts to state that certain components of certain tuples coincide.

We have thus reduced the general constraint-satisfaction problem to a single monadic and a single binary relation. It is clear that any instance of the original problem can be represented using tuples on which the constraint R is imposed, and the relation S allows us to state that components of different tuples take the same value; similarly, the new problem only allows us to impose constraints from the original problem. We wish to have a single binary relation alone, i.e., a digraph. Define the following dag D. It has vertices corresponding to the tuples from the constraint-satisfaction problem just constructed. For each relation $S(t, t')$ that holds, introduce a new vertex joined by a path of length 1 to t and by a path of length 2 to t'. For each relation $R(t)$ that holds, introduce a new vertex joined by a path of length 3 to t. This completes the dag.

We show that the digraph homomorphism problem for D is equivalent to the original constraint-satisfaction problem. In one direction, given an instance of the original problem involving S and R, tag each element with a reverse path of length 2 followed by a path of length 1 followed by a reverse path of length 2. This ensures that the element can be mapped precisely to vertices in D representing elements of the domain; note here that we are using the fact that each t' is related to some t by S in the domain of the constraint-satisfaction problem. To state $S(t, t')$ and $R(t)$ on such elements, use incoming paths of length 1, 2, 3 as above for D. In the other direction, suppose that we have an instance of the digraph homomorphism problem. We can assume that the instance is a dag, since D is a dag. We can also assume that if a vertex has both incoming and outgoing edges, then it has only a single incoming and a single outgoing edge, because this holds in D, so we could always collapse neighbors to enforce this. The input dag now looks like a bipartite graph (A, B, P), with disjoint (except at their endpoints) paths of different lengths joining vertices in A to vertices in B (all in the same direction). We can assume that the paths have length at most 3, since D has no path of length 4. We can also assume that a vertex at which a path of length 3 starts necessarily starts just this path, because this is the case in D. We can also assume that a vertex can at most start a single path of length 2, since this is the case in D. We also assume that a vertex starts at most a single path of length 1; the only way two different paths of length 1 could go in different directions would be if one of them mapped on the path of length 2 out of an out-degree-2 vertex v in D; but then, since the endpoint has no outgoing edges, all its neighbors would necessarily map to v, and so we could have mapped this endpoint to the other neighbor of v along the path of length 1. We can also assume that the only vertices that will map to an attached path of length 3 are vertices on a path of length 3; the reason is that if a directed graph containing no path of length 3 can be mapped to a reverse path of length 3, then it can be mapped to a reverse path of length 2 followed by a path of length 1 followed by a reverse path of length 2, and this configuration can be found in D from a fixed endpoint of a path of length 3 without using this path (we used this same configuration before). We can now assume that vertices in A and B map to vertices in the two corresponding sides of the bipartite graph corresponding to D. For vertices in B, this is clear if they have incoming paths of length 3, or of length 2 since we have assumed that they do not map to a vertex inside a path of length 3, or of length 1 since we can assume that they do not map to a vertex inside a path of length 2; the same is then clear for vertices in A. But then the vertices in B can be viewed as elements of the original constraint satisfaction problem and the vertices in A can be viewed as imposing constraints on them. ∎

Theorem 12 *Every constraint-satisfaction problem is polynomially equivalent to a bipartite graph-retract problem, but now only allowing 3 specific vertices of the template H to occur in the input G (but not just 2 vertices, which is polynomially solvable).*

Proof. We encode a digraph-homomorphism problem. The encoding introduces three special vertices r, b, g, which may occur in G, three additional vertices r', b', g' (which cannot appear in G), a vertex a adjacent to r', b', g', replaces each vertex of the dag with a vertex adjacent to r, replaces each edge in the dag with a path $0, 1, 2, 3, 4, 5, 6$ of length 6, where the intermediate vertices in positions 2 and 4 are adjacent to b and g respectively, while those in positions $1, 3, 5$ are adjacent to a, and finally links r', b', g' to all the vertices linked to r, b, g respectively. The proof is here a straightforward encoding argument.

A bipartite graph with just two distinguished vertices can always be retracted to just a path joining the two vertices, namely a shortest such path; this is then the core, which defines a polynomially solvable problem. ∎

Theorem 13 *Every constraint-satisfaction problem is polynomially equivalent to a balanced digraph-homomorphism problem, but now for a balanced digraph with only 5 levels (but not just 4 levels, which is polynomially solvable).*

Proof. The digraph-homomorphism problem can be encoded as a balanced digraph-homomorphism problem with only 5 levels. Given an arbitrary digraph without self-loops, represent all vertices as vertices at level 1, and all edges as vertices at level 5. If vertex v has outgoing edge e, join their representations by an oriented path 111011. If vertex v has incoming edge e, join their representations by an oriented path 110111. If neither relation holds, join their representations by an oriented path 11011011. The key properties are that neither of the first two paths map to each other, and that a digraph maps to the third path if and only if it maps to the first two. Now given a digraph, we can decompose it into connected components by removing the vertices at levels 1 and 5. Each such component either maps to none of the three paths (in which case no homomorphism exists), or to all three of them (in which case it imposes no restriction on where the boundary vertices at levels 1 and 5 map), or to exactly one of the first two paths (in which case it indicates an outgoing or incoming edge in the original graph). Thus every instance of the new problem can be viewed as an instance of the original problem, given the fact that mapping digraphs to paths is polynomially solvable, see sections 5.1.1 and 5.1.2. □

Another case of interest is that of *reflexive graphs*, i.e., graphs with self-loops. The homomorphism problem is not interesting here, since all vertices may be mapped to a single self-loop. We consider the reflexive graph-retract problem, as well as two other related problems. The *reflexive graph-list* problem is the homomorphism problem where in addition we may require that some vertex maps to a chosen subset of the vertices in the template. The *reflexive graph-connected list* problem allows only subsets that induce a connected subset of the vertices in the template. The following results are from Feder and Hell [18, 19].

Theorem 14 *Every constraint-satisfaction problem is polynomially equivalent to a reflexive graph-retract problem. The reflexive graph-retract problem is NP-complete for graphs without triangles other than trees. The reflexive graph-list problem is polynomially solvable for interval graphs, NP-complete otherwise. The reflexive graph-connected list problem is polynomially solvable for chordal graphs, NP-complete otherwise. The graph-retract problem for connected graphs with some self-loops is NP-complete if the vertices with self-loops induce a disconnected subgraph.*

For partial orders and strict partial orders, the homomorphism problem is easy, since the core is either a single vertex, in the case of partial orders, or a total strict order, in the case of strict partial orders. We examine the corresponding retract problem. For strict partial orders, even if the strict partial order is bipartite, the problem is equivalent to the bipartite graph-retract problem and hence to all of CSP. For partial orders, there are applications to type reconstruction, see Mitchell [41], Mitchell and Lincoln [42], O'Keefe and Wand [43]. Pratt and Tiuryn [45] showed that the bipartite partial order-retract problem is polynomially solvable if the underlying graph is a tree (in fact in NLOGSPACE), NP-complete otherwise. We can give here an alternative proof of this result here. If the underlying graph is a tree, we have a directed reflexive graph whose underlying graph is a tree. If we then associate a boolean variable with each subtree, the problem is just a 2SAT problem, because if a set of subtrees pairwise intersect, then they jointly intersect. For the NP-completeness result, define an undirected reflexive graph on the same set as the bipartite partial order, making x, y adjacent if there exist s, t such that $s \leq x, y \leq t$; in this case, this means that $x \leq y$ or $y \leq x$. This gives a reflexive graph-retract problem on a graph without triangles and not a tree, which is NP-complete by the preceding theorem. We examine now the general case. Let the *depth* of a partial order be the number of elements in a total sub-order.

Theorem 15 *Every constraint-satisfaction problem is polynomially equivalent to a partial order-retract problem, even if only the top and bottom elements of the partial order can occur in an instance. The equivalence holds even for depth 3 partial order-retract problems.*

Proof. We prove the equivalence to the domination-free bipartite graph-retract problem, which was shown equivalent to all of CSP above. We shall assume that only the top and bottom elements of the partial order can be used in an instance; to extend the result to the case where all elements can be used, we can simply consider the core of the partial order. Let $H = (S, T, E)$ be a domination-free bipartite graph. Define the corresponding partial order $P = (Q, \leq)$ as follows. Let Q be the set of all bipartite cliques $A \times B \subseteq E$, with $A, B \neq \emptyset$. Let $A \times B \leq A' \times B'$ if $A \subseteq A'$ and $B' \subseteq B$. If $N(v)$ denotes the set of neighbors of v in H, then the bottom elements are $\{a\} \times N(a)$ for $a \in S$ and the top elements are $N(b) \times \{b\}$ for $b \in T$.

Given an instance G for H, we can assume that G is bipartite since H is bipartite, and that G and H share at least a vertex since otherwise G can be mapped to a single edge in H, so that we know which side of G maps to S and which to T. Replace adjacency in G with $\leq$ from the side mapping to S to the side mapping to T, replace any occurrence of an H vertex in G by the corresponding bottom or top element in P, and ask whether this partial order maps to P. We can assume that top and bottom elements map to top and bottom elements respectively, and on these elements the $\leq$ relation in P corresponds to edges in H, so solving the problem on P solves the instance G for H.

In the other direction, given an instance R for P, where R and P only share top and bottom elements of P, we can assume that such elements are also top and bottom in R, since everything below a bottom element must map to that bottom element and everything above a top element must map to that top element. We can also assume that top and bottom elements in R map to top and bottom elements in P. To map such elements, determine the bipartite $\leq$ relation on them, and map them by solving the problem as a bipartite graph-retract problem for H, with the natural correspondence between bottom and top elements of P and vertices in the S, T sets of H. Clearly, if the bipartite graph-retract problem does not have a solution, neither does the partial order-retract problem. If the bipartite graph-retract problem has a solution, it only remains to map the middle vertices. If a middle element is between bottom and top elements that were mapped to subsets $A \subseteq S$ and $B \subseteq T$, map that middle element to $A \times B$ in P, completing the retraction.

In order to prove the depth 3 result, we first determine the core of the partial order. Clearly the core must contain every $\{a\} \times N(a)$ for $a \in S$ and every $N(b) \times \{b\}$ for $b \in T$, because these elements can be used in an instance. As a result, it must also contain all maximal bipartite cliques $A \times B$, because such maximal bipartite cliques are the only bipartite cliques above the corresponding elements $\{a\} \times N(a)$ for $a \in A$ and below the corresponding elements $N(b) \times \{b\}$ for $b \in B$. Notice that the maximal bipartite cliques are precisely those bipartite cliques $A \times B$ with $A = N(B)$ and $B = N(A)$, where $N(C)$ denotes here the vertices adjacent to all of C. We now observe that the maximal bipartite cliques are indeed the entire core, because the mapping $f(A \times B) = N(N(A)) \times N(A)$ is an appropriate retraction.

Having identified the core of the partial order as the partial order on maximal bipartite cliques, we use the fact that the domination-free bipartite graph can also be assumed to be $K_{3,3}$-free and $K_{3,3} \backslash \{e\}$-free. Now, if $A_1 \times B_1 < A_2 \times B_2 < A_3 \times B_3 < A_4 \times B_4$ for maximal bipartite cliques, then the containments on the A_i and those on the B_i must be strict, so $|A_i| \geq i$ and $|B_i| \geq 5 - i$. In particular, $A_2 \times B_2$ must contain a $K_{2,3}$ and $A_3 \times B_3$ must contain a $K_{3,2}$, and furthermore their union must contain a $K_{3,3} \backslash \{e\}$ by maximality of the bipartite cliques. This establishes the depth 3 claim. Notice also that $K_{3,3}$-freedom implies that a bipartite clique $A \times B$ must have $|A| \leq 2$ or $|B| \leq 2$, and since for maximal bipartite cliques A and B uniquely determine each other, the partial order has size polynomial in the size of the bipartite graph. □

Therefore, the dichotomy question for MMSNP is equivalent to the dichotomy question for CSP, which in turn is equivalent to the dichotomy questions for graph-retract, digraph-homomorphism problems, and partial order-retract problems.

6 Special Classes

Schaefer [48] showed that there are only three polynomially solvable constraint-satisfaction problems on the set $\{0, 1\}$, namely Horn clauses, 2SAT, and linear equations modulo 2; all constraint-satisfaction problems on $\{0, 1\}$ that do not fit into one of these three categories are NP-complete. For general constraint-satisfaction problems, we introduce two classes, namely *bounded width* and *subgroup*, and examine two subclasses of the bounded width class, namely the *width* 1 and *bounded strict width* classes. These are generalizations of Schaefer's three cases, since Horn clauses have width 1, 2SAT has strict width 2, and linear equations modulo 2 is a subgroup problem. At present, all known polynomially solvable constraint satisfaction problems are simple combinations of the bounded-width case and the subgroup case.

Remark: A similar situation of only three polynomially solvable cases was observed for Boolean network stability problems by Mayr and Subramanian [39], Feder [15]; the three cases there are monotone networks, linear networks, and nonexpansive networks, in close correspondence with Horn clauses, linear equations modulo 2, and 2SAT respectively; it is the generalization of the nonexpansive case to metric networks that leads to characterizations along the lines of the bounded strict width case described below.

We describe the work on network stability in the context of constraint-satisfaction in more detail here. Say that a template is *functional* if all relations have some arity $k + l$ with $k, l \geq 0$ and are described by a function $f(x_1, x_2, \ldots, x_k) = (y_1, y_2, \ldots, y_l)$, called a *gate*, where the x_i are called *inputs* and the y_i are called *outputs*. A *network stability* problem is a constraint-satisfaction problem where the template is functional, and where the input structure has the property that every element participates in exactly two relation occurrences, one as an input and one as an output. The input structure is then called a *network*. The work in [39, 15] established the following. Let the *constant* gates be $f() = e$, and the *absorption* gate be $f(x) = ()$.

Theorem 16 *Every network stability problem over a Boolean functional template containing the constant and absorption gates is NP-complete, with the exception of the following polynomially solvable cases:*

(1) Monotone functional templates, where every output y_j of every gate is a monotone function of the inputs x_i. For the general case with AND and OR gates, the problem is P-complete and determining whether there is a solution other than the zero-most and one-most solutions is NP-complete.

(2) Linear functional templates, where every output y_i of every gate is a linear function of the inputs x_i modulo 2.

(3) Adjacency-preserving functional templates, where every gate f has the property that changing the value of just one of the x_i inputs can affect at most one of the y_j outputs. Here the set of solutions can be described by a 2SAT instance because the median of three solutions, obtained by taking coordinate-wise majority, is also a solution.

The case (3) extends to a non-Boolean domain case by assuming that the template has an associated distance function on the elements satisfying the triangle inequality, such that for every gate f, if $f(x_1, x_2, \ldots, x_k) = (y_1, y_2, \ldots, y_l)$ and $f(x'_1, x'_2, \ldots, x'_k) = (y'_1, y'_2, \ldots, y'_l)$, then $\sum d(y_j, y'_j) \leq \sum d(x_i, x'_i)$. The functional template is then called nonexpansive and the associated network is metric; this case is also polynomially solvable. Here the structure of the set of solutions is a strict width 2 problem because the solutions form a 2-isometric subspace, where the corresponding 2-mapping property is obtained with the imprint function, yielding the 2-Helly property (see [15] for the definitions of 2-isometric subspace and imprint function).

It is the structure presented in this theorem and its connection to Schaefer's work that initially led to the work presented here. In this paper, we are not considering special type instances, such as network stability or planar graph coloring: the template is fixed, the instance is not constrained.

6.1 Bounded-Width Problems

A problem is said to have *bounded width* if its complement (i.e., the question of non-existence of a solution) can be expressed in Datalog. More precisely, it is said to have *width* (l, k) if the corresponding Datalog program has rules with at most l variables in the head and at most k variables per rule, and is said to have *width* l if it is has width (l, k) for some k. For a related notion of width, see Afrati and Cosmadakis [4].

Datalog is the language of logic programs without function symbols [50]. The following Datalog program checks that an input graph is not 2-colorable:

$$
\begin{array}{lll}
oddpath(X, Y) & :- & edge(X, Y) \\
oddpath(X, Y) & :- & oddpath(X, Z), edge(Z, T), edge(T, Y) \\
not2colorable & :- & oddpath(X, X).
\end{array}
$$

In this example, *edge* is an input binary relation, *oddpath* is a binary relation computed by the program, and *not2colorable* is a 0-ary relation computed by the program. The first rule says that a single edge forms an odd path, the second rule tells that adding two edges to an odd path forms an odd path, and the third rule says that the input graph is not 2-colorable if the graph contains an odd cycle. In Datalog programs for constraint-satisfaction instances we assume that there is a distinguished predicate p of arity zero (*not2colorable* in the above example) that must be derived when no solution exists for the instance. We say that such a program *solves* the problem.

The example above shows that 2-colorability has width $(2, 3)$, since it can be solved by a Datalog program with at most 2 variables in rule heads and at most 3 variables per rule. Also, 3SAT-Horn can be shown to have width 1. It is not hard to show that bounded-width problems are in monotone SNP without inequality. Furthermore, problems of width 1 are in MMSNP.

It is easy to see that all bounded-width problems are in P, since the rules can derive only a polynomial number of facts. Thus, we would like to know:

Which problems in CSP have bounded width?

The predicates from the instance are called EDB predicates, and the new auxiliary predicates are called IDB predicates. Given a Datalog program with EDB predicates corresponding to the constraints of a constraint-satisfaction problem defined by a template T, we assign to each predicate in the program a relation on values from T. The EDB predicates already have an assigned relation in T. For IDB predicates, we initially assign to them the empty relation, then add tuples as follows. Given a rule, involving at most k variables, we consider the assignments of values from T to these k variables such that the constraints imposed on them by the current relations assigned to predicates in the body of the rule are satisfied. For these satisfying assignments for the body, we consider the induced assignments on the at most l variables in the head, and add all these tuples to the relation associated with the head of the rule. This process of adding tuples over values from T to the relations associated with IDB predicates must eventually terminate, mapping each IDB to a relation on values from T.

The relation associated with the distinguished p at the end of this process must be the empty relation. Otherwise, we could design an instance that has a solution yet for which p can be derived, simply by viewing the derivation tree that made p nonempty as an instance, contrary to the assumption that the Datalog program only accepts instances with no solution.

Even if we know that a constraint-satisfaction problem has width (l, k), there could be many Datalog programs that express the complement of the problem. Thus, it seems that to answer the question above we need to consider all possible (l, k)-programs. Surprisingly, it suffices to focus on very specific Datalog programs.

Theorem 17 *For every constraint satisfaction problem P there is a canonical Datalog (l, k)-program with the following property: if any Datalog (l, k)-program solves P, then the canonical one does.*

Proof. Intuitively, the canonical program of width (l, k) infers all possible constraints on l variables at a time by considering k variables at a time. This canonical program infers constraints on the possible values for the variables in the instance, both considered l at a time and k at a time, as follows. Initially, all constraints from the instance can be viewed as constraints on variables, k at a time. Now, a constraint on k variables can be projected down to a constraint on an l-subset of these k variables. In the other direction, a constraint on an l-subset can be extended up to a constraint on the k variables. This process can be iterated, and if a constraint on variables ever becomes the empty set, we can infer that the instance has no solution. This inference process can easily be described by a Datalog (l, k)-program, and in fact the inferences carried out by this canonical program contain all inferences performed by any Datalog

(l, k)-program, with the interpretation of IDB's as relations on values from T defined above. $\square$

Consider now the following two-player game on the structures S, the instance, and T, the template. Player I selects k variables, and asks Player II to assign to them values from T. Then Player I keeps l out of these k assigned variables, extends this set of l to a new set of k, and asks Player II to assign values to the new $k-l$ variables, back to the earlier situation with k assigned variables. The game proceeds from there as before. Player I wins if at some point, some of the k assigned values violate a constraint from the instance.

Theorem 18 *The canonical (l, k)-program for a constraint-satisfaction problem accepts an instance precisely when Player I has a winning strategy in the associated (l, k)-two-player game.*

Proof. Suppose the canonical program accepts an instance (such an instance necessarily fails to have a solution). Consider the corresponding derivation tree. Each node of the tree corresponds to a relation on at most k elements from S. This relation has an associated set of tuples from T as defined above. Player I traverses a path from the root to a leaf; at each step he holds a tuple that is not in the associated set. He starts at the root, where the relation has arity zero; there he holds the arity-zero tuple, which does not belong to the associated empty set. In general, at a given node v where Player I holds a tuple of arity at most l not in the associated set, Player I selects the rule corresponding to the node v and its children, and asks Player II to assign values to the remaining variables in the rule, up to a total of at most k. It cannot be that all the resulting assignments to at most l variables corresponding to the children of the node v are tuples in the sets associated with them, because then the original tuple would have been in the set associated with v. So Player I can select some child of v such that the assignment to its at most l variables is not in the associated set. When a leaf is reached, Player I holds an assignment violating a given constraint in the instance.

For the converse, suppose that Player I has a winning strategy. The playing of the game depending on the moves by Player II can then be viewed as a tree. For instance, at the root, after Player I has made its initial choice k elements, the children correspond to the possible choices of l elements out of these k that can be made by Player I, depending on the assignment of values to the k elements by Player II. At the next level, the extension of the l assigned values to a tuple of k variables chosen by Player I is again considered, until at the leafs we have assignments to at most l variables that violate a constraint. This tree is then precisely a derivation tree by which the canonical program can accept the instance. $\square$

For related games, see Afrati, Cosmadakis, and Yannakakis [5], Kolaitis and Vardi [33], and Lakshmanan and Mendelzon [36].

This notion of bounded width for constraint-satisfaction problems can also be extended to allow infinite Datalog programs (allowing infinitely many IDB's, infinitely many rules, and infinitely many conjuncts per rule); such programs have been studied before under the name L^{ω}. For constraint-satisfaction problems on a *finite* domain, infinite programs are no more powerful than finite programs; the reason here is that the canonical program has IDB's corresponding to constraint sets, but there are only finitely many possible constraint sets.

Remark: For constraint-satisfaction problems, it can be shown that Datalog is equivalent to Datalog$(\neq, \neg)$, finite or infinite. This equivalence holds more generally for problems closed under homomorphisms, finite and infinite cases separate. For such problems, it turns out that monadic SNP with inequality is no more powerful than monotone monadic SNP without inequality, and the same holds for (binary) SNP with inequality compared with monotone (binary) SNP without inequality [16].

6.1.1 Width 1 and Tree Duality

Horn clauses have width $(1, k)$, where k is the maximum number of variables per Horn clause. To see this, express Horn clauses as implications with a conjunction of positive literals in the antecedent and at most one positive literal in the consequent. An instance has no solution if it implicitly contains a clause with an empty antecedent, which stands for 'true' or 1, and an empty consequent, which stands for 'false' or 0. This situation can be detected by Player I by selecting an appropriate clause with an empty consequent, then Player II must assign value 0 to some variable in the antecedent, then Player I selects some appropriate clause with this variable as the consequent, and so on, until a clause with an empty antecedent is reached, then Player I wins.

Using Theorem 17 and Theorem 7, we can prove:

Theorem 19 *The questions of whether a constraint-satisfaction problem has width $(1, k)$ or whether it has width 1 are decidable.*

Proof. The canonical program describes in that case a monotone monadic SNP problem, and we have seen that containment for such problems is decidable. In fact, we have also seen that it is never necessary to look at conditions for a monotone monadic SNP problem that do not define a biconnected component contained in biconnected components of the statement of the problem, but these are only single relations in the case of constraint-satisfaction problems, so we can assume that the Datalog program looks only at single relations from the input, so k need not be larger than the largest arity, hence width 1 is decidable. There is another way to see this. By Theorem 5, we may assume that an instance recognized by the monadic Datalog program has high girth, larger than the size of any rule. We may then assume that the body of the rule is a tree, and then the rule can be replaced by rules having only one EDB per rule, by introducing additional monadic IDB relations, so k is bounded by the largest arity. □

Let S be a connected structure. An element x is an *articulation element* if the structure S can be decomposed into two nonempty substructures that share only x. If we decompose a structure into substructures by identifying all its articulation elements, we say that the resulting substructures without articulation elements are *biconnected components*. A *tree* is a structure whose biconnected components consist each of a single relation occurrence. Following and generalizing the terminology of Hell, Nešetřil and Zhu [25], we say that a constraint-satisfaction problem defined by a template T has *tree duality* if: A structure S can be mapped to T if and only if every tree that can be mapped to S can be mapped to T.

Theorem 20 *A constraint-satisfaction problem has tree duality if and only if it has width* 1.

Proof. Trees are precisely the objects generated by derivation trees of Datalog programs with at most one variable in the head and at most one EDB per rule. We observed in the proof of the previous theorem that if a problem has width 1, then it has a Datalog program of this form. Then an instance S does not map to T if and only if it is accepted by the Datalog program, i.e., some tree maps to S but does not map to T. Thus width 1 implies tree duality. Conversely, if tree duality holds, then width 1 follows by considering the Datalog program that generates all the trees that do not map to T; this is an infinite program, but we have observed before that infinite programs can be transformed into finite ones for constraint-satisfaction problems, e.g., the canonical program from Theorem 17. ∎

In fact, as observed in the proof of Theorem 7, width 1 can be decided as follows.

Tree duality decision procedure:

Given a constraint-satisfaction problem with template T, let U be the structure defined as follows. The elements of U are the nonempty subsets A of the elements of T. For a relation R of arity k, impose $R(A_1, A_2, \ldots, A_k)$, the A_i not necessarily distinct, if for every $1 \leq i \leq k$ and every a_i in A_i there exist elements a_j in the remaining A_j such that $R(a_1, a_2, \ldots, a_k)$ in T. Then tree duality holds if and only if U maps homomorphically to T.

For example, 2SAT will in particular enforce $x \vee y$ and $\overline{x} \vee \overline{y}$ on $x = y = \{0, 1\}$, hence no solution exists, showing that 2SAT does not have width 1. In general, the question of whether a constraint-satisfaction problem has bounded width (or width l, width (l, k), beyond the case $l = 1$) is not known to be decidable.

We give here an alternative proof of the correctness of the above decision procedure based on tree duality and its equivalence to the existence of a Datalog program for the problem with one EDB relation per rule and at most one variable in the head of each rule, that infers constraints on the possible values for elements of the structure.

Theorem 21 *The above decision procedure correctly decides tree duality.*

Proof. Suppose that tree duality holds. Consider the structure U defined in the decision procedure whose elements are nonempty sets A. We show that U can be mapped to T. Every tree that maps to U has elements that are nonempty sets A. The tree can be mapped to T by choosing one element from the root of the tree, one consistent element from each of its children, and so on. Thus, by the definition of tree duality, U maps to T. Conversely, suppose that U maps to T, and let S be a structure such that every tree that maps to S maps to T. If we use the Datalog program on S, then every element of S will be assigned a nonempty set A by the program, otherwise the derivation tree would provide a tree that maps to S but not to T. This gives a mapping from S to U, and by composition from S to T. Therefore tree duality holds. ∎

Thus tree duality has a simple decision procedure. Consider the special case where the template T is an oriented path (a path each of whose edges may be oriented in either direction). This T was shown to have tree duality by Hell and Zhu [28]; in fact, they showed that it satisfies the stronger path duality property that S maps to T if and only if every oriented path that maps to S maps to T. We give a simple proof of tree duality via the above decision procedure. Suppose that the elements of the path T are numbered $1, 2, \ldots, r$ in order. For every nonempty subset A of $\{1, \ldots, r\}$, map A to the least numbered element of A. If there is an edge from A to B, then the least elements of A and B cannot be the same element a, since otherwise T would contain an edge from a to $a+1$ and one from $a+1$ to a. So either a is the least element of

A, $a+1$ the least element of B, and T has an edge from a to $a+1$, or a is the least element of B, $a+1$ the least element of A, and T has an edge from $a+1$ to a. Thus T has an edge from the image of A to the image of B, as required.

A more general case is the case of oriented trees, which defines both polynomial and NP-complete problems, as well as problems that have not yet been classified [26].

We say that a constraint-satisfaction problem defined by a template T with T a core has *extended tree duality* if: A connected structure S with one element s preassigned a value t in T can be mapped to T if and only if every tree that can be mapped to S can be mapped to T in such a way that the elements of the tree that map to s end up mapping to t. Just like tree duality could be decided by the existence of a mapping from a particular structure U to T, extended tree duality can be decided by the existence of a mapping from a particular structure U' to T, where U' is the substructure of U consisting of the union of the connected components containing the singletons $\{t\}$ for t in T. The proof of correctness is similar to the tree duality case. Since the extended tree duality property involves one special element s, and problems with tree duality have width 1, problems with extended tree duality have width 2. (The converse is false, e.g. 2SAT.)

Theorem 22 *For a constraint-satisfaction problem with template T, tree duality is equivalent to the existence of a homomorphism from U to T. If T is a core, then extended tree duality is equivalent to the existence of a homomorphism from the substructure U' to T. Problems with extended tree duality have width* 2.

Say that two elements are *related* if they both belong to the same set A in the connected component U'. A special case of extended tree duality for digraphs is the case where there is a total ordering of the vertices of T such that if (a,b) and (c,d) are two edges of T with $a < b$ and $c < d$, with a, c related and b, d related, then (a,d) is also an edge of T. In this case we can map a nonempty set A in U' to the least element of A under the ordering. This case is essentially the same as the extended $\underline{\text{X}}$-property from [23, 25].

More generally, an (l,k)-*tree* is a structure given by a derivation tree of a Datalog program whose rules have at most l variables in the head and at most k variables per rule. Combinatorially, a structure S is an (l,k)-tree if: There exists a tree t whose nodes are sets of elements of S of cardinality at most k, where a node and a child in t share at most l elements of S, the nodes in which an element of S participates form a subtree of t, and each relation occurrence in S involves elements contained in a single node of t. In the literature, when S is a graph and S is an (l,k)-tree, then it is said to have *tree-width* $k-1$ (see e.g. [38, 47]).

Along the general lines of duality of graph homomorphisms (see Hell, Nešetřil and Zhu [27]), a constraint-satisfaction problem defined by a template T has (l,k)-*tree duality* if: A structure S can be mapped to T if and only if every (l,k)-tree that can be mapped to S can be mapped to T. The following is immediate from the definition of acceptance by a Datalog program.

Theorem 23 *A constraint-satisfaction problem has (l,k)-tree duality if and only if it has width (l,k).*

Recall from section 3 the definition of the length of an oriented cycle and of balanced and unbalanced digraphs. Hell and Zhu [29] show that in the case where T is an unbalanced cycle, it is sufficient to test oriented paths, which are $(1,2)$-trees, and oriented cycles, which are $(2,3)$-trees; therefore unbalanced cycles define a problem of width $(2,3)$. The property that cycles of the instance S must satisfy is that their length is a multiple of the length of T. Therefore, in the special case where the length is 1, cycles need not be tested, only oriented paths, and the problem has width $(1,2)$. We show here that unbalanced cycles have extended tree duality, hence width 2, and that when their length is 1, they have tree duality, hence width 1. Suppose that a cycle has length $m \geq 1$. Enumerate the vertices $0, 1, \ldots, k$ in order around the cycle, where vertices 0 and k coincide, in such a way that there is a *level* function such that every edge (i,j) satisfies $level(j) = level(i) + 1$, with $level(0) = m$, $level(k) = 0$, and $level(i) \geq 1$ for $i \neq k$. Then define the total ordering mentioned above for extended tree duality to be the lexicographical ordering on pairs $(level(i), i)$. Notice that if two elements are related, their levels must differ by a multiple of m. In the case where $m = 1$, this imposes no restriction, i.e., we may use the structure U instead of U'. The same construction can be used to establish tree duality for balanced cycles such that if we denote by l an occurrence of a lowest level element and by h an occurrence of a highest level element, then the ordering $lhlh$ does not occur when the cycle is traversed once, so that the complete ordering of l and h is l^+h^+.

In general, for an arbitrary constraint-satisfaction problem, if we only consider instances that are (l,k)-trees with l, k fixed, then the problem is solved in polynomial time by the canonical Datalog (l,k)-program, since this program has as derivation trees precisely the (l,k)-trees that do not have a solution for the constraint-satisfaction problem.

Theorem 24 *On (l,k)-tree instances, constraint-satisfaction problems are polynomially solvable, for l, k fixed.*

For more general results on polynomially solvable problems in the case of graphs of bounded tree-width, see e.g. [1, 9].

6.1.2 Bounded Strict Width and the Helly Property

The canonical algorithm for problems of width (l,k) involved inferring all possible constraints on l variables at a time by considering k variables at a time. We may in addition require that if this inference process does not reach a contradiction (the empty set), then it should be possible to obtain a solution by greedily assigning values to the variables one at a time while satisfying the inferred l-constraints. We say that a constraint-satisfaction problem that can be solved in this way has *strict width* (l,k), and say that it has *strict width* l if it has strict width (l,k) for some k. It turns out that strict width l is equivalent to strict width (l,k), for all $k > l$, so we can assume $k = l + 1$.

This intuition behind strict width l can also be captured in two other ways. First, we can require that if we have an instance and after assigning specific values to some of the variables we obtain an instance with no solution, then some l out of the specific value assignments chosen are sufficient to give an instance with no solution. We refer to this property as the l-Helly property. Second, we could require that there exists a function g that maps $l+1$ elements from the domain of the structure T to another element, with the property that if all but at most one of the $l+1$ arguments are equal to some value b then the value of g is also b, and, furthermore, for all constraints C_i in T, if we have $l+1$ tuples satisfying C_i, then the tuple obtained by applying g component-wise also satisfies C_i. We call this property the l-mapping property.

Theorem 25 *Strict width l, the l-Helly property, and the l-mapping property are equivalent. These properties are polynomially decidable (for a fixed l).*

Proof. The proof of the equivalence of the various formulations of bounded strict width shows first that the correctness of the greedy (l,k)-algorithm implies that the Helly property for l must hold. The reason is that the final step of finding a solution uses only the inferred constraints on l variables at a time, hence these constraints must characterize precisely the solutions, showing that the Helly property for l holds. This in turn implies that the instance stating the existence of g must have a solution. The reason is that l out of the constant value assignments imposed on g can always be satisfied, because each of these constant value assignments has l out of the $l+1$ arguments equal for g; there must be an argument position that is not the one exceptional argument for any of the l constant value assignments being considered; so we can always return the value of this argument, satisfying all conditions. Since the constant value assignments considered l at a time are satisfiable, then all together they must be satisfiable by the Helly property for l, hence g exists.

Finally, the existence of g implies the correctness of the greedy $(l,l+1)$-algorithm; it implies in fact the correctness of a more restrictive algorithm that eliminates variables one by one in arbitrary order (by considering just the $l+1$-subsets containing a chosen variable to infer a constraint on the remaining l variables) and assigns values in reverse order. Consider the elimination of the first variable x_1. We claim that any solution x' of the resulting instance on the remaining variables $x_2,\ldots,x_n$ can be extended to a solution for x_1. We must show that there exists a value for x_1 satisfying all constraints involving x_1 in conjunction with x'. We first consider constraints involving only l of the variables in x'. If no value of x_1 satisfies the constraints involving x_1 and the chosen l variables, then this would result in the inference step in forbidding the value assignment on l variables induced by x', and x' would not have been a solution of the resulting instance. Therefore constraints involving only x_1 and l of the variables in x' can be satisfied (we are including here in the inference step constraints obtained as projections of constraints involving x_1 and variables possibly different from the chosen l). Consider now constraints involving only x_1 and k of the variables in x', for $k \geq l+1$, where we assume inductively that $k-1$ can be handled. Consider $l+1$ particular variables out of the chosen k from x'. For each choice of one variable (say the ith one) out of these $l+1$, if we ignore it, then a value x_1^i for x_1 can be found by inductive assumption. But then, we can set $x_1 = g(x_1^1, x_1^2, \ldots, x_1^{l+1})$ and satisfy the constraints on x_1 and all k chosen variables. The reason is that any such constraint involves l variables including x_1. One of the $l-1$ variables other than x_1 may have been ignored in choosing x_1^i, but a value for it can be found since otherwise x_1^i would not have been considered consistent. Furthermore, a variable was ignored in choosing at most one x_1^i, so g applied to the $l+1$ values for this variable gives the correct majority value. Since the constraint is closed under g, the value x_1 obtained by applying g satisfies all the constraints, completing the induction. To handle the elimination of subsequent variables analogously, it is only important to observe that the constraints obtained by the inference step are also closed under g, since they are obtained by intersection and projection of constraints closed under g. This proves the correctness of the algorithm.

Note that the existence of g is itself an instance of the constraint-satisfaction problem, hence strict width l is decidable and in fact polynomially decidable, for fixed l. It is not known to be decidable when l is not fixed. $\Box$

2-colorability is an example of a constraint-satisfaction problem with strict width 2. If a graph is bipartite, but after having two-colored some of the vertices there is no two-coloring consistent with this partial coloring, then either two vertices on different sides of the bipartite graph were given the same color, or two vertices in the same side were given different colors. Also, 2SAT has strict width 2, and so does integer programming with two variables per inequality with variables ranging over a fixed range.

Feder [17] showed that digraph-homomorphism for oriented cycles is either polynomially solvable or NP-complete. The proof is a good illustration of some of the techniques we have been using up to this point; we give here a sketch of the proof.

Theorem 26 *Every oriented cycle digraph-homomorphism problem is either polynomially solvable or NP-complete.*

Proof. (Sketch.) The case where a template is an oriented path, which we saw in the previous section has width 1, also has strict width 2. To see this, number again the vertices $1, 2, \ldots, r$ and consider the mapping $g(x, y, z) = \mathrm{median}(x, y, z)$. In the case where the template is a directed graph that maps to a cycle C, it is sufficient to consider only argument lists for g such that all arguments map to the same vertex in C. For unbalanced oriented cycles, which we saw have extended tree duality, we also have strict width 2; the argument considers the lexicographic ordering on pairs $(level(i), i)$ as before and uses the $\mathrm{median}(x, y, z)$ function on three arguments whose levels differ by a multiple of the length of the cycle. The same argument applies to the balanced cycles not containing the ordering $lhlh$ as mentioned before, so that the complete pattern of l and h is l^+h^+. There is a family of balanced cycles that does not have extended tree duality (it can encode 2SAT) yet has strict width 2: these are the balanced cycles with two l and two h elements that form the pattern $lhlh$ along the cycle. Consider the four paths $1 = l_1h_1$, $2 = l_2h_1$, $3 = l_2h_2$, $4 = l_1h_2$ on the cycle; given three paths $i-1$, i, $i+1$ out of these four (modulo 4), the *middle path* is i. The mapping $g(x, y, z)$, with the three arguments at the same level, is defined as follows: (1) If x, y, z belong to three different paths, return the one that belongs to the middle path; (2) If x, y, z belong to the same path, return the one in the middle position on the path; (3) If exactly two out of x, y, z belong to the same path, return the one of these two occurring earliest on the path. In each of the classes of the form $l^+h^+l^+h^+$, other than the polynomial class $lhlh$, there are both polynomial and NP-complete templates that are cores; the cases $(l^+h^+)^{\geq 3}$ are NP-complete for cores. It is shown in [17] that the remaining case $l^+h^+l^+h^+$, all problems are either in P or NP-complete, completing the classification of oriented cycles; the proof uses the 2-Helly property for paths, which have strict width 2, and a generalization of Schaefer's classification of boolean satisfiability to a k-partite version [17, 48]. $\Box$

Problems with bounded strict width are a special case of bounded-width CSP. For such problems we have a more efficient algorithm.

Theorem 27 *A simplified version of the canonical algorithm runs in parallel $O^*(n)$ time using a polynomial number of processors (the O^* notation ignores polylogarithmic factors). In the case $l = 2$, this algorithm can be implemented in parallel $O^*(\sqrt{n})$ time, because variables can be eliminated in parallel.*

Proof. First, observe that the number of constraints that could be inferred is $O(n^2)$. Therefore the number of steps that infer at least $n^{1.5}$ constraints is only $O(\sqrt{n})$. Suppose that a step would infer only $n^{1.5}$ constraints, i.e., only these many pairs of values for pairs of variables are discarded. Suppose that we 'charge' such a pair of values eliminated for two variables x_j, x_k to a variable x_i such that the inference on these three variables causes the pair of values to be eliminated. Since there are n variables, they are charged $n^{0.5}$ eliminations each in average, and thus at least half of them are charged at most $2n^{0.5}$ eliminations, involving at most $4n^{0.5}$ other variables. But then, it must be possible to choose a set of $n^{0.5}/8$ variables that are charged only $2n^{0.5}$ eliminations each involving two other variables not in the set. But then these $n^{0.5}/8$ variables can be eliminated simultaneously, and hence this type of step need only be performed only $O(\sqrt{n})$ times as well. All inferences will thus be obtained by the standard algorithm in $O(\sqrt{n})$ steps (consider eliminating a chosen pair last). To obtain a solution, either a value assignment for one variable restricts the values for $n^{0.5}$ others, or there are $n^{0.5}$ pairwise unconstrained variables; alternatively, a maximal independent set computation works. □

6.2 Problems with the Ability to Count

Which problems in CSP do not have bounded width? Say that a constraint-satisfaction problem has *the ability to count* if the following two conditions hold: (1) The template T contains at least the values $0, 1$, as well as a ternary relation C that includes at least the triples $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$, as well as a monadic relation Z that includes at least 0; (2) If an instance consists of only the constraints C and Z, with all constraints partitioned into two sets A and B such that A contains one more C constraint than B, and furthermore each variable appears in exactly two constraints, one from A and one from B, then the instance has no solution.

Intuitively, we can think of $C(x,y,z)$ as $x+y+z=1$, and of $Z(x)$ as $x=0$, with an obvious contradiction for instances of the special form, since adding the constraints from A and subtracting those from B yields $0=1$. Some problems of this form include linear equations over an abelian group (finite or infinite), where 0 is the identity of the group and 1 is any other element. In particular, this includes the problem for linear equations modulo 2. Another example of such problem is linear programs over the nonnegative reals. For this last case, inexpressibility in Datalog($\neq$, *succ*) was shown by Afrati et al. [5] using Razborov's monotone circuit lower bound for matching.

We shall first show that if a constraint-satisfaction problem has the ability to count, then it does not have polynomial size monotone circuits. We begin by citing Razborov's lower bound for matching [46]. In fact, Razborov's lower bound is not just for matching: it applies to any monotone problem such that certain particular instances are 'yes' instances, certain other instances are 'no' instances, and the remaining instances may be either 'yes' or 'no'. Matching is just a specific application of the result. The exact statement of Razborov's result is the following:

Theorem 28 *Consider a monotone problem on bipartite graphs such that (1) if the instance has a perfect matching, then the answer is 'yes', and (2) if the instance contains a bipartite connected component with a different number of vertices in the two sides, then the answer is 'no'. Then monotone circuits for the problem have size* $m^{\Omega(\log m)}$.

The lower bound for matching is thus obtained by requiring a 'yes' answer for the instances (1), and a 'no' answer for all remaining instances, not just instances (2). The other extreme case is also interesting, namely the problem that requires a 'no' answer for the instances (2), and a 'yes' answer for all remaining instances, not just instances (1). This gives a lower bound for systems of linear equations over the integers, the rationals, or the reals, as follows. View a complete bipartite graph with n vertices on each side as an instance with n^2 variables corresponding to the n^2 edges. View each vertex as stating the constraint that the sum of the variables corresponding to edges incident on the vertex is equal to 1. If the bipartite graph is not complete, then each missing edge is viewed as a constraint stating that the corresponding variable is equal to 0. Now if an instance contains a bipartite connected component with k vertices in the left side and k' vertices in the right side, and with $k \neq k'$, then the kk' variables corresponding to the possible edges joining the two sides must add to k according to the left side, and to k' according to the right side, so an instance (2) is indeed a 'no' instance. On the other hand, if all connected components have $k = k'$, then we can pair-up the k and k' vertices in the two sides; for each such pair (u, v), there is an odd length path from u to v, so we can assign to the edges on the path the values 1 and -1 in alternation, so that the sum is 1 for edges incident on u, and on v, but 0 for edges incident on all other vertices; doing this for all chosen pairs (u, v) and adding up the values corresponding to the different paths gives a sum 1 for each of the k and k' vertices. So the answer is 'yes' for every instance other than (2).

This gives the basic idea for getting a monotone circuit size lower bound on problems that have the ability to count, since both problems just considered have the ability to count. However, the theorem cannot applied directly for other problems that have the ability to count, such as linear equations modulo 2. Nevertheless, a result just slightly stronger than Razborov's will yield what we need: we just weaken (2) a little bit.

Theorem 29 *Consider a monotone problem on bipartite graphs such that (1) if the instance has a perfect matching, then the answer is 'yes', and (2) if the instance contains a subgraph not connected to the rest of the graph with one more vertex on the left than on the right, then the answer is 'no'. Then monotone circuits for the problem have size $m^{\Omega(\log m)}$.*

Proof. Razborov's proof involves a choice of a random bipartite E_- on two vertex sets A and B with m elements each by selecting random subsets $A' \subseteq A$ and $B' \subseteq B$ and linking all vertices in A' to all vertices in B', as well as all vertices in $\overline{A'}$ to all vertices in $\overline{B'}$. The random E_- is used to bound probabilities for three events, namely (stating equations directly from [46]) $P(E_- \in [E]) \geq 2^{-s}$ (30), $P(E_- \in A(f_m)) \leq m^{-1/2}$ (31), and $P(E_- \in S) \leq h(t, r, s, m)$ (32). We replace E_- by an event E' chosen from a smaller space, conditioning on $|A'| = |B'| + 1$ as well as $|(|A'|) - (|\overline{A'}|)| > s$ (the latter for convenience only). Formula (30) measures the probability that a fixed matching of size s will be contained in E_-. To measure this for E', we can first choose $|A'|$ from the appropriate distribution, then assume that A' and B' are fixed while the s edges of the matching are chosen at random. From the assumptions on $|A'|$ and $|B'|$, it is clear that the first of these edges will fall in E' with probability at least $1/2$, and that if this has happened for the first $i - 1$ edges, then it will happen for the ith edge with probability at least $1/2$ as well, as long as $i \leq s$. Hence (30) can only become stronger for E'. Formula (31) measures the probability that E_- has a perfect matching;

but this probability is zero for E'. Only (32) becomes weaker; an E_- will satisfy the conditioning with probability $\Omega(1/\sqrt{m})$, so for E' the bound increases by a factor of $\sqrt{m}$; this factor carries over to the final $m^{\Omega(\log m)}$ lower bound on monotone circuit size (where it is insignificant). □

This result can now be applied to linear equations modulo q, with $q \geq 2$. Relate linear equations to bipartite graphs as before. If the bipartite graph has a perfect matching, then giving value 1 to the edges in the matching and value 0 to all other edges satisfies the linear equations. If the bipartite graph has a subgraph not connected to the rest of the graph with k vertices on the left and k' on the right, and $k = k' + 1$, then the kk' variables involved must add up to k and to k', yet now $k \neq k'$ holds even modulo q. We are now ready to apply the theorem to any problem with the ability to count.

Theorem 30 *If a constraint-satisfaction problem has the ability to count, then monotone circuits for it have size $m^{\Omega(\log m)}$.*

Proof. We first represent the complete bipartite graph with n vertices on each side by a bipartite graph with vertices of degree 2 or 3, as follows. Replace every vertex of degree n with a path on $2n - 1$ vertices $0, 1, 2, \ldots, 2n - 2$, with the n incident edges attached to the n vertices in even positions, achieving thus the degree constraint. Notice that in a perfect matching, the $n - 1$ vertices in odd positions must be matched to $n - 1$ out of the n vertices in even positions, leaving exactly 1 vertex in even position left to be matched to a vertex not on that path. Thus perfect matchings for subgraphs of the complete bipartite graph and perfect matchings for subgraphs of the new graph obtained by removing edges other than those on the paths are in 1-to-1 correspondence. Similarly, subgraphs of the complete bipartite graph with one more vertex in the left correspond to subgraphs of the new graph obtained by removing edges other than those on the paths, also with one more vertex in the left.

So we have reduced the previous theorem to the case of graphs that are subgraphs of a bipartite graph G with vertices of degree at most 3. Now view each edge as a variable, a vertex with three incident edges x, y, z as a constraint $C(x, y, z)$, and a vertex with two incident edges x, y as a constraint $C(x, y, z)$, where z is an auxiliary variable that is also constrained by $Z(z)$. Removing an edge x not on one of the paths corresponds to adding a constraint $Z(x)$. Clearly, if the graph has a perfect matching, then setting the variables in the matching to 1 and all other variables to 0 gives a solution for the problem with the ability to count. Similarly, if the graph has a subgraph with one more vertex in the left, this gives an instance of the problem with the ability to count that was required not to have a solution. The only technical point here is that removing an edge x incident to two vertices corresponds to having $C(x, y, z)$, $C(x, y', z')$, and $Z(x)$, so x participates in three constraints instead of just two as was required for problems with the ability to count with no solution. However, we can replace x with two variables x, x' constrained by $C(x, y, z)$, $C(x', y', z')$, $Z(x)$, and $Z(x')$, use the fact that the resulting instance has no solution by the definition of the ability to count, and then observe that identifying x and x' cannot make the constraint-satisfaction problem have a solution if it had no solution before identifying x and x'. □

Remark: The idea of obtaining monotone circuit lower bounds for problems that are strictly between the required 'yes' and 'no' instances was used previously by Tardos [49], who obtained a truly exponential monotone circuit lower bound for the Lovasz

θ function, a polynomially computable function strictly between the NP-complete functions maximum clique and chromatic number.

Afrati et al. [5] showed that if a problem does not have polynomial size monotone circuits, then it is not expressible in Datalog($\neq$, *succ*). Thus problems with the ability to count do not have bounded width. If we just want to show that a problem does not have bounded width, then it suffices to show that it does not have a Datalog (l, k)-program for any fixed l, k. This can be proved more easily via two-player games, by an argument similar again to an argument used by Afrati et al. [5] for linear programs related to matching.

Theorem 31 *If a constraint-satisfaction problem has the ability to count, then it does not have bounded width.*

Proof. An instance of the special form with no solution, as in the definition of the ability to count, can be viewed as a bipartite graph with no perfect matching, where the C constraints in A and B are viewed as two vertex sets, and the variables without a Z constraint imposed on them are viewed as edges joining the two vertices in A and B representing the constraints where they occur. Clearly no perfect matching exists, since there is one more C constraint in A than in B. However, we can construct such an instance of size n such that if two players play the game with l, k about $\sqrt{n}$, where Player I selects edges and Player II indicates whether they are in the matching or not, then Player II can ensure that if the two or three edges incident on a vertex are ever selected together, then exactly one of them is claimed to be in the matching, corresponding to satisfaction of C. Ignore first the degree constraint, and consider a complete $k+1$ by k bipartite graph. Then, intuitively, as long as fewer than k edges are currently in the matching, an unmatched vertex can always be matched by Player II. To bound the degree, we replace each vertex of degree d in this graph by a path on $2d-1$ vertices, with the d incident edges attached to alternating vertices in the graph, hence all vertices now have degree 2 or 3. (This last transformation is similar to the one given in the proof of the previous theorem.) Here $|A|$, $|B|$ and the number of variables (edges) are quadratic in k.

Hence rules with about $\sqrt{n}$ variables per rule will be needed to recognize certain instances with no solution on n variables. In the case of abelian groups, we can improve this lower bound to about n. The basic idea is that the complete bipartite graph considered above can be viewed as describing constraints $\sum_j x_{ij} = 1$ for all $1 \leq i \leq k+1$ and $\sum_i x_{ij} = 1$ for all $1 \leq j \leq k$, giving an obvious contradiction $\sum_i \sum_j x_{ij} = 1 + \sum_j \sum_i x_{ij}$. This equation is of the form $\sum_i y_i = 1 + \sum_i y_i$, where the y_i are added in different order in the left and right hand side. If we consider the graph consisting of a path joining the y_i in the order from the left hand side and another path joining the y_i in the order from the right hand side, then in the case where the order of summation was exchanged we have essentially a grid. For a square grid, a bipartition of the vertices into two sets of about equal size must have about $\sqrt{n}$ edges across the bipartition. On the other hand, an expander increases this quantity to about n, e.g., in the case where the order of y_i in the two sides is chosen independently at random. The game played by the two players consists of selecting vertices (variables y_i) and edges (variables representing partial sums); the removal of these may disconnect the graph, and Player II can always ensure that values are assigned so that the connected components with fewer than some constant fraction of the vertices have a solution consistent with the boundary constraints; a contradiction can only be reached by growing such components, but then the fact that the graph is an expander forces the number k of edges out of a sufficiently large component to be about n. We don't know whether such a bound can be obtained in the more general case via matchings. □

Thus, the polynomial solvability of linear equations modulo 2 cannot be explained in terms of Datalog.

It seems possible that the ability to count is the converse of having bounded width. The intuition for this is that the non-existence of solutions can be attributed to the presence of the same variable in different constraints, something that cannot be remembered by Datalog if these occurrences in two different places are not ordered, and it seems that to keep track of equality of variables in different order one needs the ability to count.

Say that a core T *can simulate* a core T' if for every relation C_i in T' there is an instance S_i that defines on some variables in S_i a relation C_i' over the domain of T whose core is precisely C_i. (Here we can bound $|S_i| \leq |T|^{|C_i|}$.)

We are then saying that it may be that a constraint-satisfaction problem is not of bounded width if and only if it can simulate a problem that has the ability to count.

6.3 Subgroup Problems

In this section we study subgroup problems, whose template has as elements the elements of a finite group G, and a relation of arity k in the template must be a subgroup or coset in the direct product G^k. We first show that if a problem gets the ability to count with the two relations $x = 0$ and $x+y+z = 1$ modulo p, then it can simulate the general subgroup problem for $G = Z_p$. Furthermore, if a set that is not a subgroup or a coset in G^k is added to the general subgroup problem for an abelian G, the problem becomes NP-complete. The natural way to obtain more general problems is to allow non-abelian groups G. Here the subgroup problem with subgroups and cosets in G^k is polynomially solvable. Here again, if a subset is allowed whose projection into some abelian section of G^k is not a subgroup or a coset, the problem can simulate one-in-three SAT and is NP-complete. We call a subset containing the identity 1 whose projection into abelian sections forms subgroups a nearsubgroup. So non-nearsubgroups added to the general subgroup problem for G give NP-completeness. On the other hand, by a result of Michael Aschbacher, the intersection of nearsubgroups is a nearsubgroup, so nearsubgroups alone cannot simulate one-in-three SAT and are thus unlikely to give NP-completeness. In fact, we identify a 2-element property, which Aschbacher shows holds for solvable groups, which implies polynomiality for nearsubgroups added to the subgroup problem. We finally identify a weaker nearsubgroup intersection property that also implies polynomiality for nearsubgroups added to the subgroup problem, and here Aschbacher has found no finite group counterexample.

The simplest example of a problem that has the ability to count is the constraint-satisfaction problem whose template is the integers modulo p for some prime p, with two of the relations in the template given by $x = 0$ and $x + y + z = 1$, modulo p. As far as we know, any problem with a finite template that has the ability to count can simulate this problem for some prime p. We wish to study the interaction between these relations on Z_p and other relations in the template. We begin with a simple observation.

Theorem 32 *Suppose that a problem with template Z_p for p prime contains at least the two relations $x = 0$ and $x + y + z = 1$ modulo p, thus getting the ability to count. Then from these two relations, every relation that is a subgroup or a coset of a subgroup of some power Z_p^k can also be obtained.*

Proof. Every subgroup or coset of a subgroup of Z_p^k is an intersection of sets defined by linear equations $\sum_{i=1}^k a_i x_i = b$ modulo p. So it suffices to show that every such linear equation can be obtained. To obtain such a linear equation, it suffices to obtain all equations of the form $x = a$, $y = ax$, and $x + y = z$, modulo p, since every linear equation can be defined by combining these three basic types. In fact we only need $x = 1$ and $x+y = z$, since $x = a$ is $x = 1+1+\cdots+1$ and $y = ax$ is $y = x+x+\cdots+x$, with a terms in the sums. To get $x = 1$, just set $x + y + z = 1$, $y = 0$, $z = 0$. To get $x + y = z$, just set $x + y + t = 1$, $z + u + t = 1$, $u = 0$. □

We shall later see that if in addition to all these subgroups and cosets for Z_p and its powers, the template also has a subset of Z_p^k that is not a subgroup or coset, then the constraint-satisfaction problem for that template is NP-complete. Thus if we wish to examine constraint-satisfaction problems that have at least the ability to count modulo p but are not NP-complete, it does not make sense to add new constraints in Z_p. It may still make sense however, to examine templates where the p elements defining Z_p are only a subset of all the elements. Furthermore, the remaining elements should interact in a natural manner with the p elements defining Z_p, to avoid defining subsets of Z_p^k that are neither subgroups or cosets. One way of achieving this is to encode Z_p as a problem such as digraph-homomorphism; the encoding adds extra elements to the template, but only defines linear equations modulo p on some p specific elements. This approach, however, does not create a template that is essentially different from Z_p itself, just an encoding of Z_p. There only seems to be one way of obtaining a template with more that the p elements for Z_p without interfering with the structure of Z_p: simply view Z_p as a subgroup of a larger group, not necessarily abelian, and allow subgroup and coset relations on the larger group. Here we have the following.

Let the general subgroup problem for a finite group G be the constraint-satisfaction problem with template G whose relations are subgroups and cosets of subgroups of G^k. We shall need to bound k by some constant to obtain a finite template, but we will always allow k to be as large as needed for the argument at hand.

Theorem 33 *The general subgroup problem for a finite group G is polynomially solvable.*

Proof. The result follows immediately from a known algorithm that finds generators for a group, obtain by Babai [6], Furst et al. [21]; see also Theorem II.12 in Hoffmann [31].

The main observation is that given a group H with known generators and a chain of subgroups $H = H_0 > H_1 > \cdots > H_r = \{1\}$, one can obtain distinct representatives from each coset of each H_i in H_{i-1} as follows. Select two elements x, x' among the generators of H_0 that belong to the same coset of H_1, say $x' = xy$ with $y \in H_1$, then discard x' and add y to the list of generators. Iterate until there is only one generator in each coset of each H_i in H_{i-1}, and carry out the process for products xy of two current generators as well. The fact that only products of pairs are needed to obtain representatives for all cosets of each H_i in H_{i-1} requires proof, see Theorem II.8 in [31].

In our application, we have n elements that must be assigned values in G, so a solution is an element of $H = G^n$. Each relation in an instance defines a subgroup or more generally a coset $a_i J_i$ in H, for some subgroup J_i of H, with $1 \leq i \leq s$ if there are s relations in an instance. Let $H_i = J_1 \cap J_2 \cap \ldots \cap J_i$ for $1 \leq i \leq s$, then fix each of the n components to 1 successively until $H_r = H_{s+n} = \{1\}$ is obtained. Obtain now representatives for all cosets of each H_i in H_{i-1} using the algorithm above.

To solve the constraint-satisfaction problem, observe that the first relation $a_1 J_1$ is a coset of $H_1 = J_1$ in H_0, so we may select a representative a for this coset from the above representation, and then look for a solution of the form ax with a fixed and x in H_1. Having fixed a, a condition $ax \in a_i J_i$ now becomes $x \in a^{-1} a_i J_i = b_i J_i$. Now we proceed with $b_2 J_2$ and H_2 as we did before for $a_1 J_1$ and H_1. Here it might be that no coset representative b for H_2 in H_1 is in $b_2 J_2$, in which case the problem has no solution. If such a representative b exists, we may again look for a solution of the form by with y in H_2, and proceed as before to $H_3, H_4, \ldots, H_s$. In the end, we just need to select an element of H_s, with no constraints, and we may just take 1.

This gives a polynomial time algorithm because n, r, $|G|$, and $|H_i|/|H_{i-1}|$ are polynomially bounded. □

Labeled graph isomorphism has polynomial time algorithms obtained by the authors mentioned above, and the algorithm described in the preceding theorem is essentially the same as the algorithm of Furst et al. [21] as described in Hoffmann [31]. In labeled graph isomorphism, two graphs have been colored with each color occurring a bounded number of times, and we look for a color-preserving isomorphism. For simplicity, assume that each color occurs in k of the vertices of each of the two graphs. Let G be the group of permutations on $2k$ elements, corresponding to the $2k$ vertices of the same color in the two graphs. The constraint that vertices in one graph map to vertices in the other is a coset in G, while the constraint that adjacent vertices map to adjacent vertices is a subgroup of G for adjacent vertices of the same color, and of G^2 for adjacent vertices of different colors. Thus labeled graph isomorphism can be viewed as a subgroup constraint-satisfaction problem.

Let G be a finite group, and consider the general subgroup problem for G. Suppose that we consider adding a non-subgroup constraint, where we mean a subset of G^k that is neither a subgroup or a coset. We have stated before that for Z_p, this makes the problem NP-complete. In fact this is still true for any abelian group G. However, it turns out that for non-abelian groups, it is sometimes possible to include a non-subgroup constraint and still have a polynomially solvable problem. The key notion turns out to be what we call a *nearsubgroup*; nearsubgroups coincide with subgroups in the abelian case, but not in general for an arbitrary non-abelian group.

Let G be a finite group, and let K be a subset of G such that $1 \in K$. We say that K is a *nearsubgroup* if for all $b \in G$ such that $1 \in bK$, for all subgroups M of G, and for all normal subgroups N of M such that $M^* = M/N$ is abelian, the set $bK^* = \{aN \subseteq M : bK \cap aN \neq \emptyset\}$ is a subgroup of M^*. In words, the intersections of K with the abelian sections of G form subgroups.

An alternative definition is the following. Let G be a finite group, and let K be a subset of G such that $1 \in K$ and if $x, y \in K$, then $xyx \in K$. We call this condition the *cycles condition* because given that $1 \in K$, it is equivalent to stating that if $b, bx \in K$, then $bx^2 \in K$; and furthermore, this implies that $bx^i \in K$ for all i, thus obtaining a coset $b < x >$ of a cyclic group such that $b < x > \subseteq K$, where $< x >$ denotes the subgroup generated by x. Suppose that K satisfies the cycles condition. If M is a subgroup of G, and N is a normal subgroup of M with M/N isomorphic to $E_4 = Z_2^2$, then there is no $b \in G$ such that $bK \cap M$ meets exactly three of the four cosets of N in M. In this case, K is a nearsubgroup.

It will be useful to consider nearsubgroups with the following stronger 2-*element property*. Here K satisfies the cycles condition, and if S is the set of 2-elements in G, b is such that $1 \in bK$ then $S \cap < S \cap bK > \subseteq bK$. That is, the 2-elements in bK generate a subgroup whose 2-elements are in bK.

Our interest in these notions comes from the following three theorems. A non-nearsubgroup is a bK with $1 \in K$ such that K is not a nearsubgroup.

Theorem 34 *Let G be a finite group. Consider the general subgroup problem for G. Include also a single subset of G^k for some k that is a non-nearsubgroup. Then the subgroup problem with this additional non-nearsubgroup constraint can simulate one-in-three SAT and is therefore NP-complete.*

Proof. Let bK be the non-nearsubgroup in G^k. We can treat k-tuples as single elements, and so assume that bK is a non-nearsubgroup in G. We can simulate the constraint $x \in K$ by $y = bx$ and $y \in bK$, because $y = bx$ is a coset $(x, y) \in (1, b)H$ where $H = \{(z, z) : z \in G\}$/ So K itself is a set in the problem, with $1 \in K$, and not a nearsubgroup.

Suppose that K does not satisfy the cycles condition. If for $y \in K$, we also have $y^{-1} \in K$, then the cycles condition can be restated as $x, y \in K$ implies $xy^{-1}x \in K$. Setting $a = y$ and $az = x$, we have $a, az \in K$ but $az^2 \notin K$; we also have this if $y^{-1} \notin K$ for some $y \in K$, letting $a = y$ and $az = 1$. Furthermore, we can use the set $K' = a^{-1}K$ as a constraint by $x = ay$ and $x \in K$. So we have a constraint set K with $1, z \in K$ but $z^2 \notin K$. We pass to $< z >$, the group generated by z, which is Z_n for some n. We wish to obtain a set with just two elements $1, z$ in $Z_k =< z >$ for some $k|n$, $k \geq 3$.

We start with K itself, and gradually reduce the size of K or the integer n down to a smaller k. If $z^k, z^{k+1} \in K$ for some $k \neq 0$, then $K' = K \cap (z^{-k}K)$ still has $1, z \in K'$ but with K' strictly contained in K, unless $x \in K$ if and only if $xz^k \in K$, in which case we may pass to the smaller Z_k which is isomorphic to $Z_n / < z^k >$. So we may assume that $z^k, z^{k+1} \in K$ only for $k = 0, 1$. If K contains some z^k with $k \neq 0, 1$ relatively prime to n, then we set $K' = K \cap (z^k K^{-1})$, so that $1, z^k \in K'$ but $z \notin K'$, so K' is strictly smaller than K and we may rename z^k to z. Similarly, if K contains some z^k with $k \neq 0, 1$ and $k - 1$ relatively prime to n, then we set $K' = K \cap (z^{k+1}K^{-1})$, so that $z, z^k \in K'$ but $1 \notin K'$, so K' is strictly smaller than K and we may rename z^k to 1 by a simple transformation. Now, if K contains some z^k with $k \neq 0, 1$ with $k - 1, k$ not relatively prime to n, then arguing in the smaller groups generated by z^{k-1} and z^k we may assume that K contains z^{2k} and z^{2k-1}, but this is only possible if $2k - 1 = n$, contrary to the assumption that k is not relatively prime to n.

So we may indeed assume that K contains precisely $1, z$ in $Z_n =< z >$ with $n \geq 3$. Now consider $x, y, t \in K \cap Z_n$ with the coset constraint $xyt = z$. Then one of x, y, t is z and the other two are 1, thus defining one-in-three SAT and giving NP-completeness.

For the other case, suppose that K meets exactly three of the four cosets of N in M. We may then pass to $E_4 = M/N = \{1, a, b, ab\}$, and assume $K = \{1, a, b\}$. Then consider $x, y, z \in \{1, a\}$ with $xyz = a$, which are all subgroup and coset constraints; furthermore, add $t \in \{1, b\}$ with $(x, t) \in \{(1, 1), (a, b)\}$, which are still subgroup constraints; finally $yt = u$ with $u \in K$. Then one of x, y, z is a and the other two are 1, again defining one-in-three SAT and giving NP-completeness. □

Theorem 35 *Let G be a finite group. Consider the general subgroup problem for G Include also any number of subsets bK of G^k, where the sets K are nearsubgroups. Then the subgroup problem with these additional nearsubgroup constraints cannot simulate one-in-three SAT (and is thus unlikely to be NP-complete).*

Proof. Since the intersection of nearsubgroups is a nearsubgroup by a result of Aschbacher [3], it suffices to show that a single one-nearsubgroup K of G^3 cannot represent the one-in-three SAT relation via some cK. Suppose that it does, so that cK contains three elements (a, b, b), (b, a, b), (b, b, a) with $a \neq b$. We may assume $b = 1$, and then multiply by $(a^-1, 1, 1)$, so the three triples are $(1, 1, 1)$, $(a^-1, a, 1)$ and $(a^{-1}, 1, a)$ in K; But the product (a^-2, a, a) of the last two is also in K since the last two also commute, so we have a triple (a^{-1}, a, a) together with $(a, 1, 1)$, $(1, a, 1)$, $(1, 1, a)$, i.e., the core will still contain some (x, a, a) for $x = 1$ or $x = a$, which does not define one-in-three SAT. □

Consider now a problem whose constraints are subsets of G^k containing the identity element 1; the only constraints that do not contain 1 are single-element subsets $\{a\} \in G$. If every such a is of odd order, we call this the *odd problem* for G. If every such a is of order a power of 2, we call this the 2-*element problem* for G.

Theorem 36 *Consider a problem on a finite group G with arbitrary constraints. This problem reduces to the odd problem and the 2-element problem together, with constraints aR corresponding to the constraints R in the original problem. Furthermore, (1) The odd problem reduces to the subgroup problem for G if all constraints satisfy the cycles condition; (2) the 2-element problem reduces to the subgroup problem for G if all constraints K satisfy $S \cap < S \cap K > \subseteq K$, where S is the set of 2 elements, and also $< x > \in K$ for $x \in K$. Therefore the problem for G with nearsubgroup satisfying the 2-element property reduces to the subgroup problem for G, and is thus polynomially solvable.*

Proof. The first step takes arbitrary constraints on G and reduces them to odd problems and 2-element problems. The basic idea is that if we have r constraints, and s of them already contain the identity element 1, then we shall force one more of these constraints to contain 1. Repeating this step eventually forces all constraints to contain 1, and then 1 is a solution. Let K be the chosen set not containing 1, and ignore the remaining $r - s - 1$ constraints. For $K \subseteq G^k$ itself, we may just try each of the possible values for k variables involved. Suppose we just consider them one at a time. We have thus reduced the problem to a problem where all constraints contain 1 except for a single constraint that assigns a value to a single variable. If this constant is of odd order or of order a power of two, we have an odd problem or a 2-element problem respectively. If this constant is of order rs, with r odd and s a power of 2, then it can be written as (a, b) in $Z_r \times Z_s$, where a generates Z_r and b generates Z_s. We initially replace b with a variable constrained to Z_s, so we only have the odd order constant a, and hence an odd problem. After solving the odd problem, we obtain a solution s, and we may look for a solution of the form sx. This means that in $Z_r \times Z_s$ we want the element $(1, s_i^{-1}b)$, and this is now a 2-element problem.

It remains to reduce the odd problem and the 2-element problem to the subgroup problem in the cases mentioned in the theorem. We consider first the odd problem, where the constraints are subsets K of G^k containing 1 and satisfying the cycles condition. Additional constraints just assign a single odd order value to a variable. We replace each variable with two variables. If K is a subset of G^k, we replace it with the subgroup H of G^{2k} generated by the pairs (x, x^{-1}) in K. If a is a single odd order constant, we replace it by the pair $(a^{\frac{1}{2}}, a^{-\frac{1}{2}})$, where $a^{\frac{1}{2}}$ denotes $a^{\frac{r+1}{2}}$, r being the order of a. After a solution is found for the resulting subgroup problem, we replace each pair (x, y^{-1}) in the solution by the product xy to obtain a solution for the original problem. If the original problem had a solution, we can choose s a power of 2 such that $x^s = x$ for odd order elements x and x^s has odd order for all x; raising the solution to the power s gives an odd order solution t, and we may use the pair $(t^{\frac{1}{2}}, t^{-\frac{1}{2}})$ as a solution to the new problem. Conversely, if the new problem has a solution, then the odd constant pairs $(a^{\frac{1}{2}}, a^{-\frac{1}{2}})$ give the right product value a; furthermore, a pair (x, y^{-1}) in H can be written as $(x_1x_2 \cdots x_k, x_1^{-1}x_2^{-1} \cdots x_k^{-1})$ with the x_i in K, and then $xy = x_1x_2 \cdots x_kx_k \cdots x_2x_1$ is in K by the cycles condition, as desired.

For the 2-element problem, the constraints are subsets K of G^k containing 1 and satisfying the condition $S \cap < S \cap K > \subseteq K$. Additional constraints just assign a single 2-element value to a variable. The main point is that the set K and the subgroup $< S \cap K >$ are indistinguishable as far as their 2-elements are concerned, i.e., $S \cap < S \cap K > = S \cap K$. So we replace the set K with the subgroup $< S \cap K >$, and insist for either problem that the solution consist of 2-elements. As before, we can choose r odd such that $x^r = x$ for 2-elements x and x^r is a 2-element for all x; raising a solution to either problem to the power r guarantees that the solution is a 2-element, as desired. $\square$

Summarizing, non-nearsubgroups give NP-completeness, nearsubgroups give non-NP-completeness unless the reduction is not a simulation of one-in-three SAT, and nearsubgroups with the 2-element property give polynomiality by a reduction to the subgroup case.

We first showed that nearsubgroups are the same as subgroups in the abelian case. We moved on to the non-abelian case, and still showed that nearsubgroups are the same as subgroups for 2-groups. This led us to consider the case of odd order groups, where we encountered a nearsubgroup that is not a subgroup: the elements are triples from Z_p with product operation $(i, j, k)(i', j', k') = (i + i' + jk', j + j', k + k')$ and the nearsubgroup K consists of the elements of the form $(\frac{1}{2}jk, j, k)$. For odd order groups, nearsubgroups immediately satisfy the 2-element property. From this we inferred that for groups that are the product of a 2-group and an odd order group, in particular for nilpotent groups, nearsubgroups satisfy the 2-element property, and so the constraint-satisfaction problem for nearsubgroups is polynomially solvable.

We then asked whether it might always be the case that nearsubgroups satisfy the 2-element property, so that the constraint-satisfaction problem for nearsubgroups is polynomially solvable. Michael Aschbacher found a counterexample, where $K = I$ is the set of involutions (elements of order 2) plus the identity element 1, in a rank 1 simple Lie group of even characteristic with at least one element of order 4.

Because of this example, we considered the case of groups with no element of order

4, and showed for such groups that when the 2-Sylow subgroups have at most four elements, then nearsubgroups satisfy the 2-element property; Aschbacher proved a general theorem that implies that nearsubgroups satisfy the 2-element property for all groups with no element of order 4. Since all groups where we had previously shown that nearsubgroups satisfy the 2-element property are solvable, and Aschbacher's counterexample is not a solvable group, we asked whether there might be any counterexample for solvable groups. Here we could understand the case of a dihedral group, $< a, b >$ with $a^2 = b^n = 1$ and $ab = b^{-1}a$; then the only nearsubgroups that are not necessarily subgroups are given for r, s dividing n by the elements of the form b^{ir} and ab^{js}, provided that the same power of 2 divides r and s. Aschbacher showed that nearsubgroups satisfy the 2-element property for solvable groups. Finally, since it is not possible to simulate one-in-three SAT and obtain NP-completeness with nearsubgroups alone, we asked whether nearsubgroups could give a non-nearsubgroup by intersection; Aschbacher showed that this is not possible, the intersection of nearsubgroups is a nearsubgroup.

Summarizing our findings and those of Michael Aschbacher [3],

Theorem 37 *Let G be a finite group.*

(1) If G is abelian, or a 2-group, then its nearsubgroups are subgroups.

(2) Let I be the involutions plus 1. If G has two involutions whose product has order 4, then I is not a nearsubgroup. Otherwise, I is a nearsubgroup. If in addition I generates no element of order 4, then I satisfies the 2-element property, otherwise it does not. There exist groups G that meet this condition for I being a nearsubgroup but not satisfying the 2-element property.

(3) If G has no element of order 4, or if it is solvable, then its nearsubgroups satisfy the 2-element property.

(4) The intersection of nearsubgroups is a nearsubgroup.

The fact that there are finite groups G with nearsubgroups that do not satisfy the 2-element property, such as the ones in (2) above, creates an interesting situation. Consider the nearsubgroup problem for G. We know that this problem cannot simulate one-in-three SAT, and is thus unlikely to be NP-complete. On the other hand, only nearsubgroups with the 2-element property seem to be transformable into subgroups so as to obtain a subgroup problem, so the problem might not be polynomially solvable. This is our best candidate for a constraint-satisfaction problem that might be neither polynomially solvable nor NP-complete.

On the other hand, we have identified a property, the *nearsubgroup intersection property*, that would imply that the problem with nearsubgroups is polynomially solvable. We know of no counterexamples to the nearsubgroup intersection property, and it is implied by the 2-element property. The nearsubgroup intersection property is as follows. Let G be a finite group with a subgroup H of index 2. Then there exists an element g in $G \setminus H$ such that g belongs to every nearsubgroup K of G such that the 2-elements in the intersection of K^* and $G^* \setminus H^*$ generate G^*. Here $G^* = G/N$ where N is some normal subgroup of G (dependent on K) such that H contains N, and kN is contained in K for every k in K; and H^*, K^* are the corresponding induced subsets of G^* obtained from H, K. The *weak nearsubgroup intersection property* considers only nearsubgroups K with $N = 1$. That is, for a finite group G with a subgroup H of index 2, there exists an element g in $G \setminus H$ such that g belongs to every nearsubgroup

K of G such that the 2-elements in the intersection of K and $G \setminus H$ generate G. The following theorem can be proved using the weak nearsubgroup intersection property; we have chosen to use the stronger definition because it leads to a simpler proof and algorithm.

Theorem 38 *Suppose that the nearsubgroup intersection property holds. Then the 2-element problem is polynomially solvable for nearsubgroups, and therefore the problem for nearsubgroups is polynomially solvable.*

Proof. We have already shown that to solve the problem with nearsubgroup constraints, it is sufficient to solve the 2-element problem. In the 2-element problem, we have a space which is the direct product of groups $G_1, G_2, \ldots, G_n,$, where $G_1 =< r >$ is a group generated by a 2-element r, we would like to obtain this element r in a solution, and all constraints are nearsubgroups of the direct product of a bounded number of G_i. Call G the product of all the G_i, and let H be the subgroup of index 2 obtained by replacing G_1 with $G_1' =< r^2 >$. If we can obtain a solution g in $G \setminus H$, this solution will have r^{2i+1} in G_1, and then a solution having r itself can be easily obtained since $r = (r^{2i+1})^j$ for some j. So suppose we are looking for a solution g in $G \setminus H$. Consider the first nearsubgroup constraint K, which constrains, say, the direct product of the first k groups G_i. Call N the product of the remaining G_i. By taking the factor group defined by N, we move to G^*, and obtain H^* and K^*. It is then sufficient to look for a solution in the group generated by the 2-elements in the intersection of K^* and $G^* \setminus H^*$. This gives a subgroup of G, and we may obtain generators for this subgroup. By carrying out this process of further restricting G by considering each nearsubgroup constraint K in turn, we obtain a decreasing chain of subgroups (we may need to look at the same nearsubgroup more than once; this process is similar to how the subgroup problem was solved). If at the end of this process, the subgroup H still has index 2 in G, then the nearsubgroup intersection property guarantees the existence of a solution with all nearsubgroup constraints. To actually find such a solution, we may successively fix the values in $G_2, G_3, \ldots, G_n$, test that H still has index 2, until we find a complete solution. □

7 Conclusions and Further Directions

Every known polynomially solvable problem in CSP can be explained by a combination of Datalog and group theory. In fact, only three specific cases combine to give all known polynomially solvable problems. The three cases are width 1, bounded strict width, and subgroup problems.

These three cases have something in common, which is best illustrated by the following characterizations, where we also include a fourth case where the template is over the reals.

(1) A problem has width 1 if and only if there is a function f that maps nonempty subsets of the template to elements of the template, such that for every relation R in the template, of arity k, the following holds. Let $S_1, S_2, \ldots, S_k$ be subsets with the property that for every x_i in S_i, there exist x_j in S_j for $j \neq i$ such that $(x_1, x_2, \ldots, x_k)$ is in R. Then $(f(S_1), f(S_2), \ldots, f(S_k))$ is in R. The case of extended width 1 is slightly more general, because it only considers a fraction of the subsets of the template. The existence of solutions can also be described in terms of tree duality.

(2) A problem has strict width l if and only if there is a function g that maps $l+1$-tuples from the template to elements of the template, such that for every relation R in the template, of arity k, the following holds. First, if all but at most one of some $l+1$ elements x_i are equal to some specific element x, then $g(x_1, x_2, \ldots, x_{l+1}) = x$. Second, let x_{ij} be elements with $(x_{1j}, x_{2j}, \ldots, x_{kj})$ in R for every j. Then $(g(x_{11}, x_{12}, \ldots, x_{1(l+1)}), g(x_{21}, x_{22}, \ldots, x_{2(l+1)}), \ldots, g(x_{k1}, x_{k2}, \ldots, x_{k(l+1)}))$ is in R. The existence of solutions can also be described in terms of the Helly property.

(3) A problem is a subgroup problem if and only if we can define a group operation on the elements of the template such that for every relation R in the template, of arity k, the following holds. If the three tuples $(b_1, b_2, \ldots, b_k)$, $(b_1x_1, b_2x_2, \ldots, b_kx_k)$, $(b_1y_1, b_2y_2, \ldots, b_ky_k)$ are in R, then the tuple $(b_1x_1y_1, b_2x_2y_2, \ldots, b_kx_ky_k)$ is also in R. This means that R is a coset of a subgroup of G^k. The case of nearsubgroups can be viewed as mapping b, bx, by, instead of into bxy, into $bxyz$, where z is an element of the commutator group of the subgroup generated by x and y. The algorithm that finds solutions depends on the fact that when G acts on itself every subgroup defines a partition into cosets; we do not know whether a partition theory can also be developed for nearsubgroups.

(4) Convex programming, a constraint satisfaction problem over the reals solvable in polynomial time with the ellipsoid method, has the property that if $(x_1, \ldots, x_k)$ and $(y_1, \ldots, y_k)$ are both in some convex set, then so is $(h_\alpha(x_1, y_1), \ldots, h_\alpha(x_k, y_k))$, where $h_\alpha(x, y) = \alpha x + (1-\alpha)y$ and $0 \leq \alpha \leq 1$. Here convex programming duality is of interest in the study of solutions.

These four characterizations are all *closure properties*, i.e., there exists a function (f, g, group operation, or h_α) such that every relation R in the template is closed under component-wise application of the function. Furthermore, there is always an associated structure theory for the study of the existence of solutions (tree duality, Helly property, coset partition, convex programming duality, respectively). Schaefer [48] used such closure properties when he classified the polynomially solvable and NP-complete problems in boolean CSP. He was thus in a sense showing that Horn clauses, 2SAT, and linear equations modulo 2, the three polynomially solvable cases, are width 1, strict width 2, and subgroup, respectively. We do not know whether polynomial solvability for CSP is always necessarily tied to a closure property.

The algorithmic significance of these closure properties is an interesting question. For problems of width 1, the f mapping is not needed to solve the problem in polynomial time, but can be used to obtain a solution directly once the Datalog program has found nonempty sets associated with each variable. For problems of strict width l, once the Datalog program has found nonempty sets associated with l-tuples of variables satisfying the Helly property, the g mapping can be used to obtain a solution efficiently in parallel. There are however problems where the f and g mappings help find fast algorithms without running the Datalog program. Here the following results from Feder and Hell [18] are good examples. The connected list problem for reflexive graphs is polynomially solvable for chordal graphs, NP-complete otherwise. For chordal graphs, this problem has width 1. By using the perfect elimination ordering for perfect graphs, a fast parallel algorithm can be found for this problem; and here the f mapping is based on the existence of a perfect elimination ordering. Similarly, the arbitrary list problem for reflexive graphs is polynomially solvable for interval graphs, NP-complete otherwise. For interval graphs, this problem has strict width 2. By using the interval representation for interval graphs, a reduction to 2SAT can be found for this problem, giving again a fast parallel algorithm; and here the g mapping is based on the existence of an interval representation. Thus the f and g mappings can help understand the structure of a problem and lead to fast parallel algorithms. For subgroup problems, the situation is more drastic: we do not know of any algorithm that does not involve finding generators, and here the group operation is used directly.

This raises an important question. For subgroup problems, the set of solutions has a polynomial number of generators. This means that there exists a polynomial number of solutions such that if we take the closure under the mapping that maps s, sx, sy to sxy, then we obtain all solutions. Now we may ask: Does there exist a polynomial number of generators for the width 1 and bounded strict width cases, when closure under f and g mappings is considered? We study first the bounded strict width case. Here there does exist a polynomial number of generators, namely, if a problem has strict width l, then for each choice of l variables $x_1, x_2, \ldots, x_l$, determine the assignments of values to these variables for which a solution exists, and choose one solution for each assignment. This produces at most n^l generators. To see that these are generators, suppose that we have a solution, and consider the values it assigns to $l+1$ variables $x_1, \ldots, x_{l+1}$. For each choice of a variable x_i, there exists a generator that assigns the correct value to the remaining x_j, but not necessarily to x_i. Find $l+1$ such generators, one for each i, and then applying the g mapping to them will assign the correct value to the $l+1$ variables. Inductively, we can then obtain the correct assignment for all variables. Now consider the width 1 case. Here we write $f(S) = t$ for a set of solutions S and a solution t if for each variable x_i, letting S_i be the set of values taken by x_i in S, and letting t_i be the value of x_i in t, we have $f(S_i) = t_i$. In general, the number of generators needed for this f mapping is exponential. For Horn clauses, with $f(\{0\}) = 0$, $f(\{1\}) = 1$, and $f(\{0,1\}) = 0$, the mapping $f(S) = t$ corresponds to set intersection. Even for independent set, a special case of Horn clauses, the set of generators with the f mapping must contain at least the maximal independent sets, and there can be an exponential number of them; independent set, on the other hand, is also a special case of 2SAT, and the number of generators with the g mapping is polynomial. For linear programming, the generators with the h_α mappings are the vertices of the polytope, and there can be an exponential number of them. Notice now that Horn clauses and linear programming are in general P-complete, while neither bounded strict width problems nor subgroup problems seem to be P-complete; it might be that P-completeness is tied to the non-existence of a closure property that allows for a small (polynomial or at least non-exponential) set of generators.

Our attempt to classify the problems in CSP and establish a dichotomy is based on the following two conjectures.

Conjecture 1 *A constraint-satisfaction problem is not in Datalog if and only if the associated core T can simulate a core T' consisting of two relations C, Z that give the ability to count. This is equivalent to simulating either Z_p or one-in-three SAT.*

Conjecture 2 *A constraint-satisfaction problem is NP-complete if and only if the associated core T can simulate a core T' consisting of the single relation C defining one-in-three SAT.*

The first conjecture indicates a sharp line out of Datalog and into group theory: Since one-in-three SAT can simulate Z_2, and when the two linear equations $x = 0$,

$x + y + z = 1$ modulo p give the ability to count, then every linear equation modulo p can be simulated, it follows that the conjecture basically says that a problem not in Datalog is at least as powerful as the general subgroup problem for Z_p.

So we assume that a template not in Datalog contains at least the general subgroup problem for Z_p. We move then on to the second conjecture, which indicates a sharp line into NP-completeness. We thus try to determine what can make a problem that can simulate Z_p be able to simulate one-in-three SAT, and thus NP-complete. We first observe that on the p elements that simulate Z_p, any relation that is not a subgroup or a coset in a power of Z_p makes it possible to simulate one-in-three SAT. It is thus not possible to interfere with Z_p itself and avoid NP-completeness. This suggests that the only way to enlarge the Z_p problem and still obtain a problem that cannot simulate one-in-three SAT may be to view Z_p as a subgroup of a larger, not necessarily abelian, group. It seems that any other approach to extending Z_p would simply combine Z_p with other problems without interfering with Z_p itself, either by taking the product of Z_p with another problem, or encoding Z_p in a special class such as digraph-homomorphism.

Suppose then that we have the general subgroup problem for a finite group, which is still polynomially solvable. It is no longer true that adding a non-subgroup makes it possible to simulate one-in-three SAT. We can show that adding a non-nearsubgroup makes it possible to simulate one-in-three SAT. So we only allow nearsubgroups. The intersection of nearsubgroups gives nearsubgroups. So nearsubgroups do not make it possible to simulate one-in-three SAT, which by the second conjecture would mean that adding nearsubgroups to a subgroup problem cannot make the problem NP-complete. We can show that if we restrict our attention further to nearsubgroups with the 2-element property, then nearsubgroups can be replaced with related subgroups, the resulting problem reduces to subgroup problems, and is thus polynomially solvable.

But then nearsubgroups have the 2-element property for many groups, including solvable groups and groups with no element of order 4, by results of Aschbacher. It may then be that a subgroup problem always remains polynomially solvable when we add nearsubgroups. However, Aschbacher identifies a group with a nearsubgroup that does not have the 2-element property. It does not seem possible to represent this nearsubgroup as a subgroup, so the problem does not immediately reduce to a subgroup problem, although it still contains the general subgroup problem for the chosen group; this may mean that the problem is not polynomially solvable. On the other hand, the fact that nearsubgroups cannot simulate one-in-three SAT may mean that the problem is not NP-complete.

We thus have our best candidate for a problem in CSP that may be neither polynomially solvable nor NP-complete, which is the following. Consider the general subgroup problem for a finite group, and focus on the set of involutions, including the identity, which we wish to add as a new constraint. If some element of order 4 is the product of two involutions, then involutions are not a nearsubgroup, and the problem is NP-complete. Otherwise, the involutions are a nearsubgroup. If no element of order 4 is generated by involutions, then the involutions have the 2-element property, and the problem is polynomially solvable. Otherwise, the involutions do not have the 2-element property. Aschbacher observed that there are such finite groups, where the involutions form a nearsubgroup without the 2-element property. Could it be that for such groups the involutions define a problem that is neither polynomially solvable nor NP-complete? On the other hand, the following may hold:

Conjecture 3 *The nearsubgroup intersection property holds for all finite groups. Therefore the constraint satisfaction problem with nearsubgroup constraints is polynomially solvable.*

This is the current state of the attempt to classify the problems in CSP and obtain a dichotomy. Considering the three main conjectures, it may be that there is a direct approach towards proving the first one, concerning the ability to count. One might start by showing that if a subgroup problem does not have the ability to count, then it can be solved with Datalog, beginning with abelian groups; similarly, one could consider linear programming, a constraint satisfaction problem over the reals that gets the ability to count by representing bipartite matching or linear equations, and show that if a linear programming template does not have the ability to count, then it can be solved with Datalog.

The second conjecture cannot be approached directly, as we cannot show non-NP-completeness without showing P$\neq$NP. It might still be possible to show that if a CSP problem cannot simulate one-in-three SAT, then it belongs to a class that is unlikely to contain NP-complete problems, such as co-NP; this would establish a dichotomy, namely NP-complete versus in NP$\cap$co-NP. An approach along these lines was successful in showing that graph isomorphism is not NP-complete unless the polynomial hierarchy collapses. The approach there was via interactive proofs for graph non-isomorphism; interactive proofs have already been shown to be relevant in the study of constraint satisfaction problems, see [20, 7].

It might be possible to answer the third conjecture with the theory of nearsubgroups as studied by Michael Aschbacher [3].

A basic questions remain open. Find a deterministic construction of small graphs of high chromatic number and high girth, in particular with the size of the graph polynomial in the chromatic number, with the girth lower bounded by a constant. This would help derandomize the reduction between equivalent MMSNP and CSP problems.

The class NP consists of all problems expressible by an existential second-order sentence with an arbitrary first-order part (see [14], the first-order part need not be universal as for SNP). What is the complexity of problems in *MMNP*, the class *monotone monadic NP without equality or inequality*?

Appendix: Constraint Satisfaction and Link Systems

In his paper "Process, System, Causality, and Quantum Mechanics" [12], Tom Etter proposes link systems as a unifying framework for i-o systems, Markov processes, and quantum mechanics; he also suggests that link systems also provide an interpretation for relativity and for relational databases. In this paper, we have proposed constraint satisfaction as a unifying framework for problems that can be solved with Datalog such as Horn clauses and 2-satisfiability; for NP-complete problems such as 3-coloring and 1-in-3-satisfiability; for problems from group theory such as systems of linear equations and labeled graph isomorphism; for linear programming, graph matching, and a family of matroid parity problems; for network stability and stable matching.

In this note we intend to expose a direct connection between constraint satisfaction and link systems, so that this work and the work of Etter may benefit from each other.

A *link system* is a stochastic process on a set of random variables $x_1, x_2, \ldots, x_n$, together with a set of *links*. A link is a condition of the form $x = y$ on two variables x and y of a stochastic process, thereby crating a new process in which the unconditional probability $p(E)$ of any event E is $p(E|x = y)$ in the old process. The duplicate variable y is then dropped.

A *proper link system* is a link system in which all links are between different independent parts of the stochastic process, and no two links involve the same variable. Etter is primarily interested in proper link systems.

A *white variable* is a random variable with a uniform distribution. An *i-o system* is a link system where every link is of the form $x = w$, where w is a white variable, with x and w independent. Notice that when the process is conditioned on $x = w$, so that x and w are linked and w is dropped, the distribution of x in the resulting stochastic process is the same as in the original process, because w is white.

Suppose that x and y are independent random variables. Then the probability that x and y will both have some value k is the product of the probabilities that they will have value k separately, i.e., $p(x = y = k) = p(x = k)p(y = k)$. Let's call $p(x = k)$ and $p(y = k)$ the *unlinked* probabilities of x being k and of y being k. Now suppose we impose the link condition $x = y$. The *linked* probability is then the probability, conditioned by $x = y$, that both $x = k$ and $y = k$. The probability that x is k, as a function of k, is proportional to $p(x = k)p(y = k)$.

In short, linked probabilities are always quadratic in unlinked probabilities. If the distributions of x and y are identical, which is the quantum situation, then linked probabilities are the squares of unlinked probabilities. That is, quantum amplitudes are simply unlinked probabilities, which explains the square root law. The main additional element that appears in quantum mechanics is that probability theory must be extended by allowing cases to count negatively. Much of the trouble that we have when looking for an interpretation of quantum mechanics is that we are looking for an i-o system, where one of the two linked variables is white, when instead it is the case that the two variables are identically distributed.

This is in essence a summary of some of the basic ideas in Etter's paper. He also shows that the velocity law of relativity, which adds velocities as $(v+v')/(1+vv')$, where the speed of light is 1, can be viewed as linking two variables x and x' corresponding to coin tosses, where the velocity is the probability of heads minus the probability of tails.

In a second paper, "Quantum Mechanics as a Branch of Mereology" [13] proposes an approach to link systems from relational databases instead of probability theory. The word *mereology* means the mathematical theory of parts and wholes. The idea is to use the record count for database tables instead of probabilities. Linking is then defined in the usual sense for combining tables in relational databases. For example, let A be a two-field table with record count $A(x, y)$, and let B be another two-field table with record count $B(y, z)$. When A and B are linked at y to obtain a three-field table, the record counts multiply, giving record count $A(x, y)B(y, z)$ for the entry given by the triple x, y, z. When we hide the linked field y, we obtain a two-field table C, adding the record counts over y, so the record count is $C(x, z) = \sum_y A(x, y)B(y, z)$. In short, when A, B, C are viewed as matrices, we have $C = AB$. That is, linking is intimately tied to matrix multiplication. Etter uses this connection to examine the two core laws of quantum mechanics, the Born probability rule, and the unitary dynamical law (whose best known form is Schrodinger's equation), formulated by von Neumann in the language of Hilbert space in terms of matrices as $\mathrm{prob}(P) = \mathrm{trace}(PS)$ and $S'T = TS$ respectively. Etter observes that the algebraic forms of these two core laws occur as completely general theorems about links.

We now turn to the connection with constraint satisfaction. Given an instance I of constraint satisfaction, suppose we create an auxiliary instance I' where every variable in I has been replaced by many variables, so that every variable occurs in precisely one constraint. The instance I' always has a solution, namely satisfying each constraint separately. Now view I' as a stochastic process, by requiring for each constraint $(x_1, \ldots, x_k) \in R$ that the variables x_i be distributed so as to give a uniform distribution over the satisfying assignments for R. Now I can be viewed as the system obtained from I' by linking the many variables that came from the same variable. The resulting distribution is then the uniform distribution over the set of satisfying assignments to the instance I.

This is in essence the connection between constraint satisfaction and link systems. We may then ask what part of the framework of Etter makes sense in the context of constraint satisfaction. A first observation is that the use of Datalog allows for rules such as $C(x,z){:}{-}A(x,y), B(y,z)$. When conjunctions and disjunctions are replaced by multiplication and addition respectively, the rule becomes $C(x,z) = \sum_y A(x,y)B(y,z)$, which we saw arises in link systems.

Consider next the network stability problem. Here the constraints are gates, i.e., they involve $k+l$ variables via a functional constraint $f(x_1, x_2, \ldots, x_k) = (y_1, y_2, \ldots, y_l)$. In the instance I', prior to the linking, the x_i variables are independently uniformly distributed. All the links are of the form, $y = x$, matching an output y to an input x, where the input x is uniformly distributed, hence a white variable. Therefore network stability is just an i-o system, in Etter's terminology. Stable matching is known to be the special case of network stability where all variables are boolean and all gates are the X gate $X(x_1, x_2) = (x_1\overline{x_2}, x_2\overline{x_1})$, in addition to the constant generating gates with $k = 0$ and $l = 1$, as well as the absorption gate with $k = 1$ and $l = 0$. For every network there is an underlying digraph, namely the digraph with a vertex for every constraint C, and a directed edge from C to C' if an output of C is an input to C'. This digraph need not be acyclic in general, and for example stable matching problems give digraphs that are usually not acyclic. Then the network stability may not have any solutions, or more than one solution. When the digraph is acyclic, the network is called a circuit, and there is then a unique solution obtained by evaluating the gates in the order they appear in the digraph.

Notice that in the case of network stability, variables occur in disjoint linked pairs, as in Etter's proper linked systems. In general, we have not made the assumption that every variable occurs in precisely two constraints in this paper. Without this assumption, we have conjectured that the constraint satisfaction problems are either solved with Datalog or have the ability to count; in the latter case, we either have linear equations or one-in-three satisfiability. Suppose we make the same conjecture in the case where every variable occurs in precisely two constraints; then one-in-three satisfibility becomes a polynomially solvable problem, namely graph matching.

In fact graph matching generalizes to a family of matroid parity problems as a constraint satisfaction problem with the property that every variable occurs in only two constraints. To see this assume that each relation in the template is a matroid. Then all relations occurring in an instance form a single matroid that decomposes into a collection of small matroids, one for each relation. The constraints that pair up the variables define a matroid parity problem. Notice that the bases exchange property acts once again as a closure property. By limiting the number of relations involving a single variable, we are in essence limiting fanout. In the case of network stability, every variable occurs in one relation as an input and in one relation as an output. This limits fanout unless we have a copy gate $f(x) = (x, x)$ that creates fanout. Mayr and Subramanian observed that in the boolean case, forbidding such a copy gate gives rise precisely to the adjacency-preserving case. In the case where the relations are arbitrary, not necessarily gates, the constraint that each variable appears in only two constraints limits fanout unless we have a copy relation $R(x, x, x)$ that creates fanout. Here we observe that in the boolean case, if we consider only relations where all tuples have the same number of zeros and the same number of ones, then forbidding such a copy relation gives rise precisely to the matroid parity problems just described. It is also interesting whether limiting fanout has relevance in the quantum context, as suggested by applications in quantum cryptography.

Now let us look at the definition of the ability to count in detail. This definition involves a system I' consisting of two disjoint parts A and B that do not share any variables, and the links that produce I involve a pair of variables, one from A and one from B, with each variable participating in exactly one pair. This is precisely the main case of interest for Etter, where removing the links disconnects the system into two independent parts A and B.

Finally, when we looked at group theoretic problems, specifically in the odd order problem, we saw that to turn the nearsubgroup problem into a subgroup problem, each variable x had to be replaced by a pair $(x^{1/2}, x^{-1/2})$. That is, just like in quantum mechanics, we have a square root law. It would be interesting to see whether the fact that a square root law arises in group theory is not a coincidence, since group theory has central importance in quantum mechanics. In fact, could it be that nearsubgroups have some interpretation in quantum physics?

Tom Etter has observed that given that constraint satisfaction is the problem of mapping a structure S to a fixed template T, it makes sense to exchange the roles of S and T, and look for a homomorphism from T to S. He has observed that this duality has something in quantum mehanics that seems related to this strange reversal, which is the Foch-space representation of the wave-particle duality that reverses the role of vector and tensor component in Hilbert space. He suspects that duality will turn out to be a key concept for unifying quantum mechanics with relativity. In the context of constraint satisfaction, the simplest example of this form of duality is the case where the template is a clique K_r on r vertices. Mapping S to K_r is the problem of coloring a graph with r colors, while mapping K_r to S is the problem of whether a graph has a clique of size r.

Acknowledgements

We benefitted greatly from early discussions with Yatin Saraiya and with Peter Winkler. Christos Papadimitriou suggested the connection between constraint satisfaction and two problems with a fixed structure that were previously studied, graph homomorphism and the boolean domain case. Milena Mihail suggested that quasirandom graphs may derandomize the representation of monotone monadic SNP in CSP. Tom McFarlane showed us the paper of Tom Etter on link systems. We also had very valuable conversations with Miki Ajtai, Michael Aschbacher, Yossi Azar, Laszlo Babai, Ron Fagin, Jim Hafner, Pavol Hell, Rajeev Motwani, and Moni Naor. The comments of two anonymous referees were very helpful in the final writing of this paper.

References

[1] S. Arnborg, J. Lagergren, and D. Seese, "Easy problems for tree-decomposable graphs," J. of Algorithms 12 (1991), 308–340.

[2] M. Aschbacher, "Finite group theory," Cambridge studies in advanced mathematics 10.

[3] M. Aschbacher, "Near subgroups of finite groups," manuscript.

[4] F. Afrati and S. S. Cosmadakis, "Expressiveness of restricted recursive queries," Proc. 21st ACM Symp. on Theory of Computing (1989), 113–126.

[5] F. Afrati, S. S. Cosmadakis, and M. Yannakakis, "On Datalog vs. polynomial time," Proc. 10th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (1991), 13–25.

[6] L. Babai, "Monte Carlo algorithms in graph isomorphism testing," (1979).

[7] R. Bačík and S. Mahajan, "'Semidefinite programming and its applications to NP problems," (1995).

[8] J. Bang-Jensen and P. Hell, "The effect of two cycles on the complexity of colourings by directed graphs," Discrete Applied Math 26 (1990), 1–23.

[9] H. L. Bodlaender, "Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees," J. of Algorithms 11 (1990), 631–643.

[10] R. Dechter, "Constraint networks," In Encyclopedia of Artificial Intelligence, 1992, 276–285.

[11] P. Erdős, "Graph theory and probability," Canadian J. of Math. 11 (1959), 34–38.

[12] T. Etter, "Process, system, causality, and quantum mechanics – a psychoanalysis of animal faith," to appear in International Journal of General Systems, special issue on General systems and the emergence of physical structure from information theory.

[13] T. Etter, "Quantum mechanics as a branch of mereology," extended abstract for PhysComp paper.

[14] R. Fagin, "Generalized first-order spectra, and polynomial-time recognizable sets," in R. Karp (ed.), Complexity of Computations, AMS, 1974.

[15] T. Feder, "Stable Networks and Product Graphs," doctoral dissertation, Stanford University (1991), also Memoirs of the Amer. Math. Soc. 555.

[16] T. Feder, "Removing inequalities and negation for homomorphism-closed problems," manuscript.

[17] T. Feder, "Classification of homomorphisms to oriented cycles and of k-partite satisfiability problems," manuscript.

[18] T. Feder and P. Hell, "List problems for reflexive graphs," manuscript.

[19] T. Feder and P. Hell, "Homomorphism problems on graphs with some self-loops," manuscript.

[20] U. Feige and L. Lovász, "Two-prover one-round proof systems: their power and their problems," 24th Annual ACM Symp. on Theory of Computing (1994), 422–431.

[21] M. Furst, J. E. Hopcroft, and E. Luks, "Polynomial-time algorithms for permutation groups," Proc. 21st IEEE Symp. on Found. of Comp. Sci. (1980), 36–41.

[22] D. M. Goldschmidt, "2-fusion in finite groups," Annals of Math. 99 (1974), 70–117.

[23] W. Gutjahr, E. Welzl, and G. Woeginger, "Polynomial graph colourings," Discrete Appl. Math.35 (1992), 29–46.

[24] P. Hell and J. Nešetřil, "On the complexity of H-coloring," J. Comb. Theory, Series B 48 (1990), 92–110.

[25] P. Hell, J. Nešetřil, and X. Zhu, "Duality and polynomial testing of tree homomorphisms," Transactions of the AMS, in print.

[26] P. Hell, J. Nešetřil, and X. Zhu, "Complexity of tree homomorphisms," Discrete Applied Math, in print.

[27] P. Hell, J. Nešetřil, and X. Zhu, "Duality of graph homomorphisms," Combinatorics, Paul Erdos is Eighty, Bolyai Society Mathematical Studies, vol. 2, 1995.

[28] P. Hell and X. Zhu, "Homomorphisms to oriented paths," Discrete Mathematics 132 (1994), 107–114.

[29] P. Hell and X. Zhu, "The existence of homomorphisms to oriented cycles," SIAM J. on Discrete Math. 8 (1995), 208–222.

[30] G. G. Hillebrand, P. C. Kanellakis, H. G. Mairson, and M. Y. Vardi, "Tools for Datalog boundedness," Proc. 10th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (1991), 1–12.

[31] C. M. Hoffmann, "Group-Theoretic Algorithms and Graph Isomorphism," Lecture Notes in Comp. Sci. 136 (1982), Springer-Verlag.

[32] P. G. Kolaitis and M. Y. Vardi, "The decision problem for the probabilities of higher-order properties," Proc. 19th ACM Symp. on Theory of Computing (1987), 425–435.

[33] P. G. Kolaitis and M. Y. Vardi, "On the expressive power of Datalog: tools and a case study," Proc. 9th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (1990), 61–71.

[34] V. Kumar, "Algorithms for constraint-satisfaction problems," AI Magazine 13 (1992), 32–44.

[35] R. E. Ladner, "On the structure of polynomial time reducibility," J. Assoc. Comput. Mach. 22 (1975), 155–171.

[36] V. S. Lakshmanan and A. O. Mendelzon, "Inductive pebble games and the expressive power of Datalog," Proc. 8th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (1989), 301–310.

[37] A. Lubiw, "Some NP-complete problems similar to graph isomorphism," SIAM J. Comput. 10 (1981), 11–21.

[38] J. Matoušek and R. Thomas, "Algorithms finding tree-decompositions of graphs," J. of Algorithms 12 (1991), 1–22.

[39] E. Mayr and A. Subramanian, "The complexity of circuit value and network stability," J. Comput. Syst. Sci. 44 (1992), 302–323.

[40] P. Meseguer, "Constraint satisfaction problem: an overview," AICOM 2 (1989), 3–16.

[41] J. C. Mitchell, "Coercion and type inference (summary)," in Conf. Rec. 11th ACM Symp. on Principles of Programming Languages (1984), 175–185.

[42] P. Lincoln and J. C. Mitchell, "Algorithmic aspects of type inference with subtypes," in Conf. Rec. 19th ACM Symp. on Principles of Programming Languages (1992), 293–304.

[43] M. Wand and P. M. O'Keefe, "On the complexity of type inference with coercion," in Conf. on Functional Programming Languages and Computer Architecture (1989).

[44] C. H. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," J. Comput. Syst. Sci. 43 (1991), 425–440.

[45] V. Pratt and J. Tiuryn, "Satisfiability of inequalities in a poset," manuscript.

[46] A. A. Razborov, "Lower bounds on monotone complexity of the logical permanent," Math. Notes of the Academy of Sciences of the USSR 37 (1985), 485–493.

[47] N. Robertson and P. Seymour, "Graph minors. II. Algorithmic aspects of tree-width," J. of Algorithms 7 (1985), 309–322.

[48] T. J. Schaefer, "The complexity of satisfiability problems," Proc. 10th ACM Symp. on Theory of Computing (1978), 216–226.

[49] E. Tardos, "The gap between monotone and non-monotone circuit complexity is exponential," Combinatorica 7–4 (1987), 141–142.

[50] J. D. Ullman: *Principles of Database and Knowledge-Base Systems*, Vol I. Computer Science Press, 1989.