Symmetric Computation

Anuj Dawar
Department of Computer Science and Technology, University of Cambridge, U.K.
anuj.dawar@cl.cam.ac.uk

Abstract

We discuss a recent convergence of notions of symmetric computation arising in the theory of linear programming, in logic and in circuit complexity. This leads us to a coherent and robust definition of problems that are efficiently and symmetrically solvable. This is at once a rich class of problems and one for which we have methods for proving lower bounds. In this paper, we take a tour through results which show applications of these methods in a number of areas.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Complexity theory and logic; Theory of computation → Finite Model Theory; Theory of computation → Complexity classes

Keywords and phrases Descriptive Complexity, Fixed-point Logic with Counting, Circuit Complexity, Linear Programming, Hardness of Approximation, Arithmetic Circuits

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.2

Category Invited Talk

Funding *Anuj Dawar*: Research funded in part by EPSRC grant EP/S03238X/1.

1 Introduction

It has been said that computer science is the science of *abstraction*. Aho and Ullman in their seminal book *Foundations of Computer Science* [1] call it the "mechanization of abstraction". To model a part of the world computationally is to forget (or "abstract away") the features that are unnecessary to the computational task at hand and keep only the essential elements in a suitable data model. For example, a widely used data model in the world of algorithm design is that of graphs, which captures a collection of entities and their pairwise relationships. The relationships could reflect compatibility of kidney donors with patients needing transplants or they could pair riders with drivers in a car-pooling system. Once the details are abstracted away, we can use exactly the same *graph matching* algorithm to find a suitable matching in either system. Yet, in the field of computational complexity, which studies the resources required by algorithms and aims to elucidate why some computational problems are inherently intractable, algorithms are usually modelled as *Turing machines*, a low-level model working on strings of bits. This mismatch between the levels of abstraction at which algorithms are formulated, and at which complexity is analyzed is tied to persistent obstacles in complexity theory.

An important feature that distinguishes an abstract data structure, such as a graph, from its concrete representation, such as a pointer list, are its *symmetries*. In a graph, two vertices may appear identical and therefore interchangeable while their concrete representations are distinguished by some feature (such as actual pointer values) that is hidden by the abstraction. Algorithms that work at the higher level of abstraction must respect the symmetries in the abstract data. We use the term *symmetric computation* to describe computation at the abstract level that respects the inherent symmetries of the data.

The mismatch between algorithms working on high-level data structures and complexity defined in terms of low-level machines is one of the central concerns of the field of *descriptive complexity*, which seeks to formulate a theory of complexity at the level of high-level

descriptions. Usually this takes the form of characterizing complexity in terms of *definability* in a logic. The paradigmatic result being Fagin's theorem [22] that the class of problems definable in existential second-order logic is exactly the class NP. The classic question of descriptive complexity, of whether there is a logic that exactly characterizes polynomial-time computation, first posed by Chandra and Harel [13] can be understood as asking whether it is possible to describe efficient algorithms, whenever they exist, at the level of abstraction of the data?

In the quest for a logic for P (see [24]), *fixed-point logic with counting* (FPC) emerged as a logic of reference. Even though Cai et al. [12] demonstrated three decades ago that the logic does not express all graph properties in P, the logic has been the focus of much research in recent years. This is because it has proved remarkably expressive and at the same time we have powerful techniques for proving inexpressibility results for it. Work in recent years has shown that FPC can be seen as capturing a natural class of *symmetric algorithms* inside P, with equivalent formulations in arising in circuit complexity and the theory of linear programming. Thus, the methods for proving inexpressibility results give techniques for showing lower bounds for such algorithms. In this article, I give a brief survey of such results and methods. The survey does not include any proofs, and not much by way of definitions. I attempt to motivate and state the results, placing them in a wider narrative and provide pointers to the original sources.

2 Counting Width

Put simply, FPC is an extension of first-order logic by means of a mechanism for iteration (usually taken to be an *inflationary fixed-point* construct) and a mechanism for counting. The latter allows us to form numerical terms to denote the cardinality of any definable set. The logic has been extensively studied in the context of descriptive complexity theory. A good account of the logic and work on it in the 1990s can be found in Otto's monograph [34]. It was often said at the time that, though Cai, Fürer and Immerman had shown that FPC cannot express all polynomial-time properties of graphs, all natural properties in P are in FPC. We now know this is not true. In particular, the study of constraint satisfaction problems has turned up a host of natural problems that are in P, but not in FPC. Indeed, the constraint satisfaction problems that are in FPC are exactly the ones of *bounded width* (see [5, 10]).

At the same time, research on FPC since the turn of the century has shown the remarkably rich expressive power of the logic. An important strand of this has been the line of work that shows that FPC captures all of P on classes of sparse graphs, or more generally structures with sparse connectivity. This culminates in the result of Grohe [25] which shows that FPC captures P on any class of graphs which excludes some fixed graph as a minor. Grohe's book [25] which gives the proof of this result also provides an excellent, up-to-date definition of and introduction to the logic FPC in its early chapters. For another overview of the logic and its expressive power, see the survey [15].

The key method for proving that some class $\mathcal{C}$ of structures is not definable in FPC is based on the expressive power of *first-order logic* with counting. To be precise, let C^k denote the fragment of first-order logic where we restrict formulas to have no more than k variables altogether, but we are allowed to use them compactly by allowing *counting quantifiers*. That is, we can write $\exists^i x\theta$ to denote that there are at least i elements x for which $\theta(x)$ holds. It is known that for any formula φ of FPC, there is a fixed value k such that, restricted to structures with at most n elements, φ is equivalent to a formula φ_n of C^k. We write $\equiv^k$ to

denote the relation of elementary equivalence for C^k: two structures $\mathbb{A}$ and $\mathbb{B}$ are said to be C^k-equivalent, written $\mathbb{A} \equiv^k \mathbb{B}$, if every sentence of C^k true in one structure is also true in the other. Thus, to show that a class $\mathcal{C}$ is not definable in FPC, it suffices to show that $\mathcal{C}$ is not closed under $\equiv^k$ for any fixed k. This motivates the following definition [17].

▶ **Definition 1.** *For any isomorphism-closed class of finite structures $\mathcal{C}$, let $\mathcal{C}_n$ denote the collection of structures in $\mathcal{C}$ with at most n elements. We write $\nu_{\mathcal{C}} : \mathbb{N} \to \mathbb{N}$ for the function such that $\nu_{\mathcal{C}}(n)$ is the least k for which $\mathcal{C}_n$ is closed under $\equiv^k$. We call $\nu_{\mathcal{C}}$ the* counting width *of $\mathcal{C}$.*

If $\mathcal{C}$ is definable in FPC, $\nu_{\mathcal{C}}$ is bounded by a constant. On the other hand, there are classes of structures not definable in FPC, indeed problems of very high complexity, which still have bounded counting width. We can understand the property of having bounded counting width as a non-uniform version of FPC definability. For any $\mathcal{C}$, the class $\mathcal{C}_n$ is a finite collection of structures (up to isomorphism) which can be defined by a single sentence φ_n of $C^{\nu_{\mathcal{C}}(n)}$. This sequence of sentences is uniform when it is generated by a single sentence of FPC. It is also not hard to see that we never need more than n variables to express φ_n, since any structure on n elements can be described completely, up to isomorphism, by a sentence of first-order logic with at most n variables. Thus, $\nu_{\mathcal{C}}(n) \leq n$ for any $\mathcal{C}$ whatsoever.

Cai, Fürer and Immerman [12] gave the first construction of a class of graphs of unbounded counting width, which we call the *CFI construction* for short. Indeed, the showed that there is a class $\mathcal{C}$ with $\nu_{\mathcal{C}} = \Omega(n)$. Such lower bounds were then established for a number of specific problems, either by a construction inspired by that of Cai et al. (for instance in [14] or [5]) or by means of reductions (see [5] and [9]). The reductions involved are those definable in a logic such as FPC or fragments of it, such as first-order logic or Datalog. In general, if we can show that a class $\mathcal{C}$ is reducible to a class $\mathcal{D}$ by means of such a reduction I, and I takes structures of size n to structures of size n^d, then $\nu_{\mathcal{C}} = O(\nu_{\mathcal{D}}^d)$. In particular, if we can bound the size of $I(\mathbb{A})$ by a linear function in the size of $\mathbb{A}$, we prove that $\nu_{\mathcal{D}} = \Omega(\nu_{\mathcal{C}})$. As was pointed out in [17], this implies in particular that for any non-uniform constraint satisfaction problem, the counting width is either $O(1)$ or $\Omega(n)$, providing a sharp definability dichotomy.

3 Symmetric Circuits

The claim that definability in FPC is a natural formalization of the notion of solvability by means of a *symmetric* polynomial-time algorithm rests on the characterization of FPC as the class of problems decided by polynomially-uniform *symmetric* circuits with threshold gates.

Circuits models have been studied in the context of computational complexity because they seemed a promising route to proving lower bounds. A circuit is really an unfolding of the behaviour of an algorithm for a fixed size of input. The hope is that the difficulty of the computation that it represents can then be studied purely combinatorially in the structure of the circuit. Formally, we have a decision problem that is a language $L \subseteq \{0,1\}^*$. Such a language can be described by a family of *Boolean functions*: $(f_n)_{n\in\omega} : \{0,1\}^n \to \{0,1\}$, and each f_n can be represented by a *circuit* C_n which is a directed acyclic graph where we think of the vertices as gates suitably labeled by Boolean operators for the internal gates and by inputs $x_1, \ldots, x_n$ for the gates without incoming edges. The operators we allow on the internal gates are the *basis*. The standard Boolean basis $(\wedge, \vee, \neg)$ can sometimes be extended, for instance, with *threshold* or *majority* gates.

From our perspective, circuits provide a very low-level model of computation. When we describe decision problems in a high-level descriptive language, such as FPC, the descriptions can, of course, be translated to circuits. The circuits we get as a result of such a translation

have natural *symmetry* properties. In particular, imagine a circuit C that takes as input an n-vertex (directed) graph. That is to say, the inputs to C are n^2 variables labelled $x_{ij}(1 \leq i, j, \leq n)$ representing the potential edges in the graph. Now, if C decides a property of graphs that is invariant under isomorphisms, the output of the circuit is unchanged if we permute the vertices of the graph. Given an n-vertex graph G, there are many ways that it can be mapped onto the inputs of the circuit C, one for each bijection between $V(G)$ – the vertices of G – and $\{1, \ldots, n\}$. So the output is unchanged under any permutation $\pi \in S_n$ acting on the inputs by the action $x_{ij} \mapsto x_{\pi(i)\pi(j)}$. For circuits obtained from logic, this invariance property is witnessed by a *syntactic* invariance condition. That is, any permutation $\pi \in S_n$ can be extended to an *automorphism* of C which takes each input x_{ij} to $x_{\pi(i)\pi(j)}$. We call circuits with this syntactic invariance condition *symmetric*. The definition extends naturally to relational signatures beyond just graphs. For a relational signature τ, we say that a circuit is τ-symmetric if it accepts at its inputs the encoding of a τ-structure and every permutation of the elements of the structure extends to an automorphism of the circuit. This notion has been studied previously in [21] under the name of generic circuits and in [33] where they were called explicitly order-invariant circuits.

We establish in [2] that the expressive power of FPC is exactly captured by families of polynomially-uniform symmetric circuits in a basis that includes the standard Boolean functions along with either threshold or majority gates. This shows that FPC is, in fact, a natural circuit complexity class and helps to establish the robustness of the class. Indeed, we can, besides threshold or majority gates, also allow gates for arbitrary Boolean functions that are *fully symmetric* (this is observed in [20], where circuit models are considered using gates which are not fully symmetric). Here a function $f : \{0,1\}^n \to \{0,1\}$ is fully symmetric if it is unchanged by *any* permutation of its inputs. Equivalently, $f(x) = f(y)$ whenever x and y are two binary strings with the same number of 1s. Hence, FPC consists of exactly those properties that are decidable by symmetric circuits using any fully symmetric functions in the basis, justifying its claim to be a natural symmetric fragment of P.

One consequence of the circuit characterization of FPC is that we can see the CFI construction as giving a circuit lower bound. Indeed, like most circuit lower bounds, this one also works for the non-uniform circuit families. We can prove that any family of polynomial-size τ-symmetric circuits (even a non-uniform one) accepts a class of structures of bounded counting width. The relationship can be stated more generally, not just for polynomial size circuits.

▶ **Theorem 2** ([2]). *For any $\epsilon > 0$ and any family $(C_n)_{n\in\omega}$ of symmetric circuits of size s where $s(n) \leq 2^{n^{1-\epsilon}}$, the class of structures accepted by the circuits has counting width $O(\frac{\log s}{\log n})$.*

We can, indeed, prove this relationship directly, without going through the translation into logic. The usual method of showing that $\mathbb{A} \equiv^k \mathbb{B}$ for a pair of structures $\mathbb{A}$ and $\mathbb{B}$ is by means of the k-pebble bijection game of Hella [28]. This game can be used directly as a lower bound method for symmetric circuits (see [19] for an exposition). The key idea is that we prove in [2] that in a circuit of size s (within the bounds of Theorem 2), the stabilizer group of each gate has a small *support*, of size $O(\frac{\log s}{\log n})$. We can then show that a symmetric circuit with support of size k cannot distinguish two structures $\mathbb{A}$ and $\mathbb{B}$ on which Duplicator has a winning strategy in the $2k$-pebble bijection game.

4 Linear Programming

Linear programming is a widely used approach to solving combinatorial optimization problems. It provides a powerful framework within which optimization problems can be represented, as well as efficient methods for solving the resulting programs. In particular, it is known since the work of [29] on the ellipsoid method that there are polynomial-time algorithms that solve linear programs. In [3], we show that this can be expressed in FPC. That is to say, we consider a natural representation of linear programs as relational structures, where the variables and the constrants are unordered sets, and we show that there is an FPC sentence that defines those linear programs that are feasible and an FPC interpretation that, in any such program, defines a representation of the optimum value of a linear objective function.

We also consider linear programs that are not given *explicitly*, but rather in terms of a separation oracle. The classic example is that of the linear program that encodes the *maximum matching* problem. Given a weighted graph G, the linear program that encodes its matching problem is exponential in the size of G. However, the ellipsoid method can be used in such cases as long as we have a means of determining, for any vector x, whether it is in the polytope P described by the constraint matrix and, if it is not, a hyperplane that separates x from the polytope. This is known as a *separation oracle* for P. It is shown in [3] that, as long as a separation oracle for a polytope P is itself definable in FPC, then the corresponding linear programming optimization problem can also be defined in FPC. By showing that a separation for the matching problem is definable in FPC we were able to prove the following, settling a long-standing open question [11].

▶ **Theorem 3** ([3]). *The size of a maximum matching in a graph is definable in* FPC.

Symmetry of linear programs is a property that has been studied in the literature in its own right. Yannakakis [38] initiated the study of symmetric extended formulations. Again, let us consider graphs over the vertex set $[n]$. We can consider these as functions $G : X \to \{0, 1\}$ where $X = \{x_{ij} \mid i, j \in [n]\}$ is the set of potential edges. Equivalently, we can think of graph as a 0-1 valued vector in the Euclidean space $\mathbb{R}^X$.

Consider, in particular, $P \subseteq \{0, 1\}^X$ which is the collection of simple cycles of length n. The *convex hull* of P, $\text{conv}(P) \subseteq \mathbb{R}^X$ is known as the *travelling salesman polytope* or the *Hamiltonian cycle polytope*. Solving the travelling salesman problem amounts to optimizing a linear function over P, and determining whether a graph $G : X \to \{0, 1\}$ has a Hamiltonian cycle is the same as determining whether there is a point in P consistent with $x \leq G(x)$ for all $x \in X$. Thus, if we could represent $\text{conv}(P)$ by a set of linear constraints of size polynomial in n, we could solve these NP-hard problems in polynomial time. Yannakakis proved that $\text{conv}(P)$ does not have a polynomial-size *symmetric extended formulation*. That is to say, we cannot obtain it as the projection of a polytope $Q \subseteq \mathbb{R}^{X \times Y}$ using additional variables Y, as long as Q is symmetric. The notion of symmetry is the natural one. Any permutation of $[n]$ has a natural action on X and hence on $\mathbb{R}^X$. The symmetry requirement says that for any such permutation $\pi \in S_n$ we can find a permutation σ of Y such that for $\mathbf{xy} \in R^{X \times Y}$, $\mathbf{xy} \in Q$ if, and only if, $\pi(\mathbf{x})\sigma(\mathbf{y}) \in Q$. While the lower bound proof of Yannakakis relies heavily on the notion of symmetry, it turns out that this is not essential to the result. A long line of work originating with the result of Yannakakis culminated in a proof by Rothvoß [35] that shows that the Hamiltonian cycle polytope does not have polynomial-size extended formulations, symmetric or not. It is worth remarking that these lower bound results on the size of the Hamiltonian cycle polytope are obtained by means of reductions from the *matching* polytope. That is, even though the problem of determining the maximum matching in a graph is known to be in polynomial-time, there is no polynomial-size extended formulation of it that yields a linear program of polynomial-size.

We now consider a different way of representing the set $\mathcal{C}_n \subseteq \{0,1\}^X$ as a linear program. We say that a polytope $P \subseteq \mathbb{R}^X$ *recognizes* $\mathcal{C}_n$ if $\mathcal{C}_n \subseteq P$ and $\{0,1\}^n \setminus \mathcal{C}_n$ is disjoint from P. Now, a class $\mathcal{C}$ of structures that is decidable in polynomial time necessarily is recognized by a polynomial-size family of extended polytopes. That is, a P recognizing it can always be obtained as the projection of some $Q \subseteq \mathbb{R}^{X \times Y}$ of polynomial size. This was already shown by Yannakakis [38] and really amounts to establishing that linear programming is complete for P under (say) logarithmic-space reductions. But, here we have dropped the symmetry requirement. What classes of structures are recognized by *symmetric* families of extended polytopes? It turns out that they are exactly the classes of bounded counting width [6].

▶ **Theorem 4** ([6]). *For any $\epsilon > 0$ and any family $(Q_n)_{n\in\omega} \subseteq \mathbb{R}^{X\times Y}$ of symmetric linear programs of size at most s where $s(n) \leq 2^{n^{1-\epsilon}}$, the class of structures accepted has counting width $O(\frac{\log s}{\log n})$.*

This implies, in particular, that there *is* a family of symmetric linear programs that recognizes the class of graphs with a perfect matching in this sense, but there is still provably not one for the class of graphs with a Hamiltonian cycle, in contrast with the results of Yannakakis.

5 Lift-and-Project Hierarchies

In Section 4, we consider extended formulations of linear programs. Such extended formulations are widely used in applications of linear programming (and its extensions) to combinatorial optimization problems. Typically, we can formulate a combinatorial optimization problem as a $\{0,1\}$-integer linear programming problem. The solutions to this form a set $S \subseteq \{0,1\}^X$. The linear programming relaxation takes this formulation and relaxes the condition that a variable x must take values in the set $\{0,1\}$ and replaces it with $0 \leq x \leq 1$. This defines a polytope $P \subseteq \mathbb{R}^X$ which includes all the points in S. Extended formulations are obtained in an attempt to add additional constraints (including additional variables) so that the projection onto $\mathbb{R}^X$ gives us a better approximation of the convex hull of S. There are several systematic ways of constructing extended formulations that give infinite hierarchies interpolating between P and $\text{conv}(S)$. There are hierarchies, not only of linear programs but also of semidefinite programs. They include the Sherali-Adams, Lovasz-Schrijver and Lasserre hierarchies (see [31]).

These hierarchies have an interesting connection with symmetric computation as we have defined it, and especially with the notion of counting width. The first connection, established independently by Atserias and Maneva [7] and Malkin [32] is in connection with the Sherali-Adams relaxations of the *graph isomorphism* integer program. In essence, they show that the equivalence relation on graphs that is induced by the kth Sherali-Adams relaxation of this program is sandwiched between the equivalences $\equiv^k$ and $\equiv^{k+1}$. One consequence is that the CFI construction can be used to show that no finite level of the hierarchy gives us graph isomorphism exactly. See [26] for an alternative account, which also constructs a family of relaxations that correspond exactly to $\equiv^k$. Taking this further, Atserias and Ochremiak [8] show that relaxations of graph isomorphism based on semi-definite programming cannot do better. In particular, they extend the FPC simulation of the ellipsoid method from [3] to show that the class of feasible semidefinite programs itself has bounded counting width. Note that feasibility of semidefinite programs is not known to be in P, so we would not expect it to be in FPC, so the non-uniform definition of counting width is essential here.

A further connection between lift-and-project hierarchies and counting width is established in the context of constraint optimization problems. These form a very general class of combinatorial optimization problems to which the method of systematic extended formulations

can be applied. Constraint satisfaction problems (CSP) are usually defined as decision problems where we are given a collection V of variables, a domain D of values and *constraints*. A constraint is a pair $(\mathbf{x}, R)$, where $\mathbf{x} \in V^k$ is a tuple of variables and $R \subseteq D^k$ a relation on D. An assignment $h : V \to D$ satisfies the constraint $(\mathbf{x}, R)$ if $h(\mathbf{x}) \in R$. The problem is to decide if all the constraints can be simultaneously satisfied. If we fix the domain D in advance as well as the set Γ of relations on D that can appear in the constraints, we can see $\mathbb{D} = (D, \Gamma)$ as a finite relational structure. The instance is then a structure $\mathbb{A}$ in the same vocabulary and we define CSP($\mathbb{D}$) to be the class of structures $\mathbb{A}$ such that there is a homomorphism from $\mathbb{A}$ to $\mathbb{D}$. This view of CSP as essentially homomorphism problems is due to Feder and Vardi [23]. As noted above, we have a known dichotomy with respect to the counting width of CSP($\mathbb{D}$): either $\nu_{\mathrm{CSP}(\mathbb{D})} = O(1)$ or $\nu_{\mathrm{CSP}(\mathbb{D})} = \Omega(n)$. Interestingly, this lifts to an interesting dichotomy for constraint *optimization* problems formulated as *finite-valued* CSP.

A *finite-valued constraint satisfaction problem* (VCSP) is given by a finite set D and a collection Γ of functions $f : D^k \to \mathbb{Q}_+$. An instance of the problem is a set V of variables along with a set C of constraints, each of which is a triple $c = (\mathbf{x}, f, w)$ with $f \in \Gamma$, $\mathbf{x} \in V^k$ where k is the arity of f and $w \in \mathbb{Q}_+$. The algorithmic problem is to find an assignment $h : V \to D$ of values to the variables which minimizes $\sum_{c \in C} w_c f_c(h\mathbf{x}_c)$. As usual, we can obtain a decision problem from the optimization problem by including with an instance an explicit threshold t. Thus, we think of VCSP(D, Γ) as the decision problem of determining, given (V, C, t) whether there is an assignment $h : V \to D$ such that $\sum_{c \in C} w_c f_c(h\mathbf{x}_c) \leq t$. The following dichotomy is established in [16]:

▶ **Theorem 5.** *For every D and Γ, either $\nu_{\mathrm{VCSP}(D,\Gamma)} = O(1)$ or $\nu_{\mathrm{VCSP}(D,\Gamma)} = \Omega(n)$.*

Moreover, the cases of unbounded counting width co-incide exactly with the cases known to be NP-hard, while the bounded width ones are all definable in FPC.

The lower bound on counting width established in Theorem 5 has interesting consequences for lift-and-project hierarchies. We can define a basic linear programming (BLP) formulation for any VCSP(D, Γ) and show that it can be constructed by an FPC interpretation from an instance of VCSP(D, Γ). Moreover, each fixed level of the standard lift-and-project hierarchies, such as the Lasserre hierarchy, over this program is also given by an FPC interpretation. Using the fact that solvability of semidefinite programs has bounded counting width, we can then establish that if $\nu_{\mathrm{VCSP}(D,\Gamma)} = \Omega(n)$, then any bounded number of levels of the Lasserre hierarchy cannot yield a solution to VCSP(D, Γ). These are strong algorithmic lower bounds obtained by means of the methods of counting width. In particular, the following dichotomy is established in [17].

▶ **Theorem 6.** *If for some (D, Γ), $t : \mathbb{N} \to \mathbb{N}$ is a function such that any instance (V, C) of* VCSP(D, Γ) *is solved exacly by considering the $t(n)^{\text{th}}$ Lasserre lift of the basic linear programming relaxation of (V, C), then $t(n) = \Omega(\nu_{\mathrm{VCSP}(D,\Gamma)}(n))$.*

6 Hardness of Approximation

In the 1990s, the field of computational complexity was transformed by the PCP theorem [4], and the proofs of hardness of approximation that flowed from it. This showed that, assuming P $\neq$ NP, not only is it impossible to have efficient algorithms to solve various NP-hard optimization problems exactly, it is also impossible to have efficient algorithms that solve the problem approximately. In the years since then, this has led to the devleopment of a thriving field studying the hardness of approximation.

For many specific NP-hard optimization problems we know an exact ratio α such that there is a polynomial-time algorithm that solves the problem α-approximately, but no such algorithm guaranteeing a better ratio unless P = NP. For instance, consider the problem Max3SAT, where we are given a Boolean formula φ in 3CNF and we are required to determine the maximum number m of clauses of φ that can be simultaneously satisfied. A trivial algorithm can appoximate the value of m up to a ratio of $\frac{7}{8}$, but unless P = NP, no polynomial-time algorithm can guarantee an approximation ration of $\frac{7}{8} + \epsilon$ for any ϵ [27]. There are other problems where there is a gap between the best known achievable approximation ratio and the best known lower bound. For instance, it is known that the size of a vertex cover in a graph can be approximated up to a factor of 2 by a polynommial-time alogrithm. On the other hand, we know that unless P = NP, there is no algorithm that will achieve an approximation ratio better than $\sqrt{2}$ [30]. It is conjectured that the lower bound can be improved to $2 - \epsilon$, but this remains open.

In [9] we were able to reproduce these lower bounds as *unconditional* inexpressibility results for FPC. That is, we can prove that there is no term of FPC that can define, given a Boolean formula φ in 3CNF, a number that is guaranteed to be within $\frac{7}{8} + \epsilon$ of the maximum number of clauses satisfiable in φ. Similarly, we cannot define in FPC a number guaranteed to be within a factor of $\sqrt{2}$ of the size of the minimum vertex cover in a graph. In short, even if P = NP we can still say that there is no polynomial-time *symmetric* algorithm that achieves such approximation ratios. On the other hand, the algorithms that achieve the upper bounds for these problems are easily seen to be in FPC, i.e. they can be implemented symmetrically. Thus, one can see the hardness of approximation results as telling us something fundamental about the limits of symmetric computation, regardless of whether or not P = NP.

The PCP theorem established the hardness of approximating Max3SAT by giving a reduction from the satisfiability decision problem to itself. It gives a way of translating (in polynomial time) a Boolean formula φ into a formula φ' such that if φ is satisfiable than so is φ' and if φ is not satisfiable than in φ', no more than a $1 - \epsilon$ fraction of the clauses can be simultaneously satisfied (for some explicit constant ϵ). The value of ϵ can then be amplified to be arbitrarily close to $\frac{1}{8}$ by further reductions as in [27]. To prove the undefinability of the approximation in FPC, what we show is that the 3CNF formulas that are satisfiable and those that are at most $\frac{7}{8} + \epsilon$-satisfiable cannot be separated by any class of bounded counting width.

▶ **Theorem 7** ([9]). *For any $\epsilon > 0$, if $\mathcal{C}$ is a class of* 3CNF *formulas that contains all satisfiable formulas and does not contain any that are not $(\frac{7}{8} + \epsilon)$-satisfiable, then $\nu_{\mathcal{C}} = \Omega(n)$.*

Theorem 7 is established by means of a reduction from the problem Max3XOR of maximizing the number of satisfiable clauses in a 3XOR formula. A gap similar to that in Theorem 7 is first established for 3XOR.

▶ **Theorem 8** ([9]). *For any $\epsilon > 0$, if $\mathcal{C}$ is a class of* 3XOR *formulas that contains all satisfiable formulas and does not contain any that are not $(\frac{1}{2} + \epsilon)$-satisfiable, then $\nu_{\mathcal{C}} = \Omega(n)$.*

This is proved by means a variant of the CFI construction. In previous versions of the CFI construction, the aim is to construct a pair of structures $\mathbb{A}$ and $\mathbb{B}$ which are $\equiv^k$-equivalent, but differ minimally with respect to some property of interest (such as satisfiability, or graph 3-colourability). The equivalence with respect to $\equiv^k$ is, of course, proved using Spoiler-Duplicator games. In the present instance, the aim is to show indistinguishability of a pair of structures which differ significantly on some numeric parameter (such as the number of clauses that can be satisfied, or the size of the smallest vertex cover). This poses significant new challenges.

While the work reported in [9] establishes counting-width lower bounds for approximation of Max3SAT, Max3XOR and Vertex Cover, there is a substantial body of work on hardness of approximation that we have only begun to explore from the point of view of definability. It would be interesting to establish bounds for problems like MaxCut and Max2SAT where there are gaps between the best known upper and lower bounds for approximability in the context of polynomial-time algorithms. What would be even more exciting would be to show a bound for symmetric algorithms that was stronger than one known for general polynomial-time alogorithms.

7 Arithmetic Circuits

As a final topic, we turn to another model of computation, that of *arithmetic circuits*. These are intended to model computation at a level where arithmetic operations such as addition and multiplication are of unit cost (see [37]).

Formally, an arithmetic circuit over a field K and a set of variables X is a directed acyclic graph where every input (i.e. node of indegree 0) is labelled by an element of X or an element of K, and every internal node is labelled either $+$ (a sum gate) or $\times$ (a product gate). A distinguished *output* gate can then be seen as computing a polynommial in the ring $K[X]$. In the field of arithmetic circuit complexity, we are concerned with determining for various polynomials, what is the size of the smallest circuit that computes it.

Two polynomials (strictly speaking they are families of polynomials) that are much studied in the field are the *determinant* and the *permanent*. They are both defined on a set of variables X representing the entries of an $n \times n$ matrix, so $X = \{x_{ij} \mid 1 \leq i, j \leq n\}$. The determinant is defined as $\det(X) = \sum_{\pi \in S_n} (-1)^{\mathrm{sgn}(\pi)} \prod_i x_{i\pi(i)}$, where $\mathrm{sgn}(\pi)$ is 0 if π is an even permutation and 1 if it is an odd permutation. The permanent is defined similarly, but without the sign. So, $\mathrm{perm}(X) = \sum_{\pi \in S_n} \prod_i x_{i\pi(i)}$. Written this way, the size of the expressions defining the polynomials is exponential in n, due to the sum over $n!$ permutations. Nonetheless, it is known that there are polynomial-size circuits for computing the determinant, and these can be easily obtained from polynomial time algorithms for computing it. On the other hand, it is conjectured that there are no polynomial-size circuits for computing the permanent. Indeed, this is equivalent to Valiant's conjecture that VP is different to VNP, the analogue of the P $\neq$ NP conjecture for arithmetic circuits [36].

It is clear from their definitions that both $\det(X)$ and $\mathrm{perm}(X)$ are invariant under permutations of the variables X which are induced by the natural action of S_n. In other words, for any permutation $\pi \in S_n$, if we permute X by mapping x_{ij} to $x_{\pi(i)\pi(j)}$, it does not change either $\det(X)$ or $\mathrm{perm}(X)$. So, it makes sense to ask whether these polynomials can be computed by polynomial-size *symmetric* circuits in the sense of Section 3. The definition of such circuits is an easy extension of the idea presented in that section: an arithmetic circuit on the variable set $X = \{x_{ij} \mid 1 \leq i, j \leq n\}$ is symmetric if every permutation $\pi \in S_n$ acting on the indices extends to an automorphism of the circuit. In recent work [18], we have been able to show that $\det(X)$ can, indeed, be computed by polynomial-size symmetric circuits, and $\mathrm{perm}(X)$ provably cannot. The upper bound for the determinant is obtained by showing that known fast parallel algorithms for computing the determinant can be done symmetrically. Note that the standard algorithm based on Gaussian elimination cannot be carried out symmetrically in polynomial time. This is known from the counting width lower bounds on solvability of systems of linear equations.

For the lower bound on computing the permanent, we rely on another CFI-like construction. To be precise, we show that we can construct for each k, a pair of bipartite graphs G and H such that $G \equiv^k H$ but G and H have different numbers of perfect matchings. This is then

related to arithmetic circuits by means of a translation. If we had a polynomial-size arithmetic circuit for computing $\mathrm{perm}(X)$, we could get a polynomial-size circuit with Boolean inputs which computes the number of perfect matchings in a bipartite graph. But, for such circuits, we can show that the output must be invariant under $\equiv^k$ for some constant k.

One interesting aspect of this proof is the role of perfect matchings in a graph. We know that the decision problem of determining whether or not a graph has a perfect matching has bounded counting width. For bipartite graphs, a construction of Blass, Gurevich and Shelah [11] shows that it has width 2, and the result of Anderson et al. [3] shows that even for general graphs, it is constant. Nevertheless, it turns out that the *number* of perfect matchings is not a $\equiv^k$-invariant for any k, even for bipartite graphs.

8 Conclusion

The notion of symmetry in computation arises naturally when we consider algorithms described at a high-level of abstraction and how they are translated to low-level models. The tension between preserving symmetry and efficient implementation rests to some extent on the fact that we cannot efficiently detect symmetries, e.g. we do not know how to efficiently determine if two graphs are isomorphic. What is remarkable is that a number of distinct notions of symmetry, arising in different fields, such as database theory, combinatorial optimization and circuit complexity converge on a common core. At the heart of the theory that emerges from this core is a graded approximation of isomorphism – the equivalence relations $\equiv^{C^k}$ – which has itself been widely studied from many independent directions. This leads to a coherent and robust notion of efficient symmetric computation. On the one hand it is a remarkably powerful model and on the other hand, we have methods for proving lower bounds for it. There seems to be a wealth of possible areas of application for it.

References

1 A. V. Aho and J. D. Ullman. *Foundations of Computer Science, C Edition.* Computer Science Press / W. H. Freeman, 1992.

2 M. Anderson and A. Dawar. On Symmetric Circuits and Fixed-Point Logics. *Theory Comput. Syst.*, 60(3):521–551, 2017. doi:10.1007/s00224-016-9692-2.

3 M. Anderson, A. Dawar, and B. Holm. Solving Linear Programs without Breaking Abstractions. *J. ACM*, 62, 2015.

4 S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof Verification and the Hardness of Approximation Problems. *J. ACM*, 45(3):501–555, 1998.

5 A. Atserias, A. Bulatov, and A. Dawar. Affine Systems of Equations and Counting Infinitary Logic. *Theoretical Computer Science*, 410(18):1666–1683, 2009.

6 A. Atserias, A. Dawar, and J. Ochremiak. On the Power of Symmetric Linear Programs. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785792.

7 A. Atserias and E. N. Maneva. Sherali-Adams relaxations and indistinguishability in counting logics. *SIAM J. Comput.*, 42:112–137, 2013.

8 A. Atserias and J. Ochremiak. Definable Ellipsoid Method, Sums-of-Squares Proofs, and the Isomorphism Problem. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, 2018.

9 Albert Atserias and Anuj Dawar. Definable Inapproximability: New Challenges for Duplicator. In *27th EACSL Annual Conference on Computer Science Logic, CSL*, pages 7:1–7:21, 2018. doi:10.4230/LIPIcs.CSL.2018.7.

10 L. Barto and M. Kozik. Constraint Satisfaction Problems Solvable by Local Consistency Methods. *J. ACM*, 61:3:1–3:19, 2014.

11 A. Blass, Y. Gurevich, and S. Shelah. Choiceless Polynomial Time. *Annals of Pure and Applied Logic*, 100:141–187, 1999.

12 J-Y. Cai, M. Fürer, and N. Immerman. An Optimal Lower Bound on the Number of Variables for Graph Identification. *Combinatorica*, 12(4):389–410, 1992.

13 A. Chandra and D. Harel. Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences*, 25:99–128, 1982.

14 A. Dawar. A Restricted Second Order Logic for Finite Structures. *Information and Computation*, 143:154–174, 1998.

15 A. Dawar. The Nature and Power of Fixed-Point Logic with Counting. *ACM SIGLOG News*, 2(1):8–21, 2015.

16 A. Dawar and P. Wang. A Definability Dichotomy for Finite Valued CSPs. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015*, pages 60–77, 2015.

17 A. Dawar and P. Wang. Definability of semidefinite programming and Lasserre lower bounds for CSPs. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, 2017. doi:10.1109/LICS.2017.8005108.

18 A. Dawar and G. Wilsenach. Symmetric Arithmetic Circuits. forthcoming.

19 Anuj Dawar. On Symmetric and Choiceless Computation. In *Topics in Theoretical Computer Science - The First IFIP WG 1.8 International Conference, TTCS*, pages 23–29, 2015. doi:10.1007/978-3-319-28678-5_2.

20 Anuj Dawar and Gregory Wilsenach. Symmetric Circuits for Rank Logic. In *27th EACSL Annual Conference on Computer Science Logic, CSL*, pages 20:1–20:16, 2018. doi:10.4230/LIPIcs.CSL.2018.20.

21 L. Denenberg, Y. Gurevich, and S. Shelah. Definability by Constant-depth Polynomial-size Circuits. *Information and Control*, 70:216–240, 1986.

22 R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol 7*, pages 43–73, 1974.

23 T. Feder and M.Y. Vardi. Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study Through Datalog and Group Theory. *SIAM Journal of Computing*, 28:57–104, 1998.

24 M. Grohe. The Quest for a Logic Capturing PTIME. In *Proc. 22nd IEEE Symp. on Logic in Computer Science*, pages 267–271, 2008.

25 M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017.

26 M. Grohe and M. Otto. Pebble Games and linear equations. *J. Symb. Log.*, 80:797–844, 2015.

27 J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

28 L. Hella. Logical Hierarchies in PTIME. *Information and Computation*, 129:1–19, 1996.

29 L. G. Khachiyan. Polynomial Algorithms in Linear Programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

30 S. Khot, D. Minzer, and M. Safra. Pseudorandom Sets in Grassmann Graph Have Near-Perfect Expansion. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 592–601, 2018. doi:10.1109/FOCS.2018.00062.

31 M. Laurent. A Comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre Relaxations for 0-1 Programming. *Math. Oper. Res.*, 28:470–496, 2003. doi:10.1287/moor.28.3.470.16391.

32 P. N. Malkin. Sherali-Adams relaxations of graph isomorphism polytopes. *Discrete Optimization*, 12:73–97, 2014.

33 M. Otto. The Logic of Explicitly Presentation-Invariant Circuits. In *Computer Science Logic, 10th International Workshop, CSL '96, Annual Conference of the EACSL*, pages 369–384, 1996.

34 M. Otto. *Bounded Variable Logics and Counting — A Study in Finite Models*, volume 9 of *Lecture Notes in Logic*. Springer-Verlag, 1997.

35 T. Rothvoß. The Matching Polytope has Exponential Extension Complexity. In *Symp. Theory of Computing, STOC 2014*, pages 263–272, 2014.

36 L. G. Valiant. Completeness Classes in Algebra. In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing STOC*, pages 249–261, 1979. doi:10.1145/800135.804419.

37 J. von zur Gathen. Algebraic Complexity Theory. *Ann. Rev. Comput. Sci.*, 3:317–347, 1988. doi:10.1146/annurev.cs.03.060188.001533.

38 M. Yannakakis. Expressing Combinatorial Optimization Problems by Linear Programs. *J. Comput. Syst. Sci.*, 43(3):441–466, 1991.