The Logic in Computer Science Column

BY

Anuj Dawar

Department of Computer Science and Technology
University of Cambridge, Cambridge, CB3 0FD, UK
anuj.dawar@cl.cam.ac.uk

I am pleased to introduce myself as the new editor of the *Logic in Computer Science Column* in the Bulletin. I am stepping into the shoes of Yuri Gurevich, who has been running this column since 1988 and that is a daunting task. I aim to do my best to live up to the standards that he has set. If you wish to contribute a column or have suggestions for other excellent expositors who might be able to, I would love to hear from you. Please do get in touch.

To kick things off, I offer a column of my own. This is based on an invited talk that I gave at the LICS 2025 conference in Singapore in June 2025. It was after returning from the conference that I ran into Quisani, Yuri's old student who wanted to hear what I had talked about. We had a fascinating conversation and you can read all about it below.

Notions of Width: Variables, Pebbles and Supports

Anuj Dawar

Abstract

Given a formula, what is the smallest number of variables with which it can be equivalently written? What seems like an abstruse question in syntactic manipulation turns out to have significance in a variety of areas of theoretical computer science. The number of variables in a formula emerges as an important measure related to notions of width arising in fields as varied as database theory; combinatorics and graph theory; and permutation groups. I explore how these notions are related to each other and the exploration will take us through a diverse landscape of topics, from comonads to lower bounds in circuit complexity.

1 Width

Quisani: Hello, I'm Quisani. I'm a former student of Yuri Gurevich and I often meet him here for chats about Logic in Computer Science. I don't believe I have met you before.
Author: Hello, it's nice to meet you. My name is Anuj Dawar, and I am new here. I've heard a lot about your chats. Yuri has asked me to take over his responsibilities here, so I expect to be coming here regularly and seeing more of you.

Q: Does that mean that Yuri is retiring from his role?

A: Not completely. He will be back from time to time, so you will see more of him. But, I expect to take over from him in due course. It is daunting to step into his shoes, but I hope we can find plenty to chat about from the world of Logic in Computer Science.

Q: I look forward to it. Do you have any new and interesting stories from that world to share today?

A: I have recently returned from the LICS 2025 conference in Singapore [4] where I gave an invited talk and I thought I might tell you what I talked about. The title of my talk was *Notions of Width: Variables, Pebbles and Supports*.

Q: Sounds intriguing. What do you mean by *width*?

A: The word *width* is used to mean many different things, even within the fields of logic and combinatorics. Not all of these uses of the word are related to each other. But, in my talk I was trying to show that many different measures of complexity that go by this name are, in fact, related and sometimes surprisingly so. I start by asking the following question: given a first-order formula φ, say in the language of graphs, what is the smallest number of variables with which φ can be equivalently rewritten?

Q: I suppose I should see this as an algorithmic problem: given φ, calculate the smallest k such that φ is equivalent to a formula with just k variables. This sounds to me like it should be uncomputable, given that equivalence is undecidable.

A: You are quite right. But, I am not so much interested in computing the minimum but using the number of variables as a measure of the complexity of a formula. So, let us say that a formula φ has *width* k if no subformula of φ has more than k *free* variables. This is sometimes called the *syntactic width* of the formula.

Q: Yes, I see. If that is the case, then it is clear that you can rename bound variables in such a way that you use no more than k distinct variables. This means that the formula is equivalent to one in L^k, the fragment of first-order logic that uses only the variables $x_1, \ldots, x_k$. I remember a conversation with Ian Hodkinson [23] about this logic many years ago. We also talked about infinitary logics: $L^k_{\infty\omega}$ which is the closure of L^k under infinitary conjunctions and disjunctions and the logic $L^\omega_{\infty\omega}$ which is obtained as the union of $L^k_{\infty\omega}$ as k varies.

A: Indeed, I studied these logics back when I was a graduate student and you talked about some of that work with Ian. I remember once giving a talk in which I mentioned the logic L^k and a member of the audience asking me: "what is the point of a logic that is not closed under the renaming of bound variables"? And this is one reason I emphasise the characterization in terms of limiting the *width*: the maximum number of free variables in any subformula. That's what's important and not the actual names of the variables. It helps explain the intuition of the resource we are limiting, i.e. the number of things we can simultaneously refer to and connect. But, what I want to tell you about today is how this notion of width connects to many other notions of width that arise in logic, combinatorics and complexity, not to mention database theory and constraint programming. Let me start by telling you a bit about conjunctive queries in databases.

2 Conjunctive Queries

Q: I have heard of *conjunctive queries*. Can you remind me what they are?

A: From the point of view of a logician, it is best to describe this as a fragment of first-order logic. For simplicity, let's assume that we are always working in the language of graphs. That is, we only have one binary relation E and no other relation or function symbols. The formulas of first-order logic can then be described by the following syntax:

$$\varphi := x = y \mid E(x, y) \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists x\varphi \mid \forall x\varphi.$$

Each formula $\varphi(\mathbf{x})$ with free variables $\mathbf{x}$ defines a *query*, which we think of as a function from graphs G to relations $\varphi^G \subseteq G^{\mathbf{x}}$. Conjunctive queries are then those that can be defined by formulas without negation, disjunction or universal quantification. Once we have disallowed these operations from our formulas, we can also dispense with equality. Do you see why?

Q: Yes, when $x = y$ appears in a conjunction, we can simply replace all occurrences of y with x and get rid of y. This way we also get rid of a variable, which is useful since we are being careful with how many we need.

A: Excellent! This leaves us with this highly reduced syntax.

$$\varphi := E(x, y) \mid \varphi \wedge \varphi \mid \exists x\varphi.$$

Can we say anything useful with this? Suppose I want to express the property of a directed graph that it contains a walk of length five. Can you see how to do it?

Q: I can. A walk of length five is just a sequence of six vertices, not necessarily distinct, such that there is an edge from each vertex to its successor in the sequence. I would express it with something like this:

$$\exists x_1 \cdots \exists x_6(E(x_1, x_2) \wedge \cdots \wedge E(x_5, x_6)).$$

A: That certainly works and note how you had to use six variables. But, with a careful re-use of variables, you can get away with using just two. Here's how:

$$\exists x\exists y(E(x, y) \wedge \exists x(E(y, x) \wedge \exists y(\cdots))), \tag{1}$$

where I'm sure you can fill in the $\cdots$.

Q: I see. Instead of having the quantifiers all in front like I did, you push them in as far as you can.

A: Poizat [33] calls the logic L^k an "enemy of the prenex normal form" for this reason.

Q: There is nothing special about the fact that the walk is of length five. For a walk of length n, in prenex normal form I would need $n + 1$ variables but I can

always get away with just two. But this contrast with prenex normal form is not just about conjunctive queries. What's special about them?

A: Conjunctive queries are widely studied in database theory as they correspond to the *select-project-join* queries which are said to be the most common kind of queries on relational databases [1]. And we care about the number of variables because it is tied to the complexity of the natural algorithm for evaluating the query in a database. Let's take the example from (1). Picture the parse tree of the formula, which we can see in Figure 1.

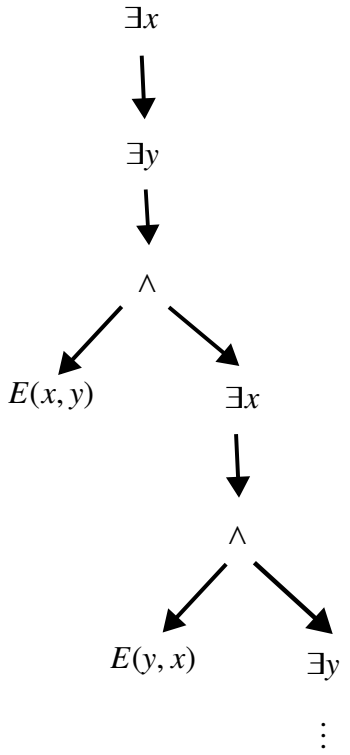

Figure 1: The parse tree of the formula $\exists x \exists y (E(x, y) \wedge \exists x (E(y, x) \wedge \exists y (\cdots)))$.

Now, imagine the process of evaluating this formula bottom up in a graph G. With each node we can associate the relation that is defined in G by the query given by the subformula below that node. The operations at the nodes are *select* (the atomic formulas), *project* (the existential quantifiers) and *join* (for the conjunctions)—hence the name.

Q: I see, and the fact that each subformula has at most two free variables means all the relations are at most binary.

A: Exactly. This limits the complexity of the evaluation process. Most importantly, it limits the space complexity as storing relations of higher arity takes up a lot

of space. There is much work in database theory aimed at optimizing queries. Cutting down the number of variables in a conjunctive query is an important optimization as it greatly reduces the size of the intermediate results that we have to consider. That is a topic for another day. If you want to know how the number of variables may be optimized, I would suggest looking at this paper [7].

For now, I want to look at structural questions about conjunctive queries. Chandra and Merlin [9] noted that the problem of conjunctive query evaluation is the same as *structure homomorphism*.

Q: As I recall, for a pair of structures $\mathbb{A}$ and $\mathbb{B}$ in the same vocabulary, a homomorphism from $\mathbb{A}$ to $\mathbb{B}$ is a function $h : A \to B$ from the universe A of $\mathbb{A}$ to the universe B of $\mathbb{B}$ that preserves all relations.

A: Yes, in particular, if $\mathbb{A}$ and $\mathbb{B}$ are graphs, this means that whenever (u, v) is an edge in $\mathbb{A}$ then $(h(u), h(v))$ is an edge in $\mathbb{B}$. So, what Chandra and Merlin observe is that from any conjunctive query φ we can construct a structure M_φ such that for any $\mathbb{A}$, we have $\mathbb{A} \models \varphi$ if, and only if, there is a homomorphism from M_φ to $\mathbb{A}$. Can you see how to construct M_φ from φ?

Q: I suppose so. If we put φ in prenex normal form, it is a list of existential quantifiers followed by a conjunction of atomic formulas. We create M_φ by taking the collection $x_1, \ldots, x_n$ of quantified variables as elements, and then put an edge between x_i and x_j whenever an atomic formula $E(x_i, x_j)$ appears in the body.

A: That's right. Then, a homomorphism from M_φ to $\mathbb{A}$ is exactly an evaluation of the existential quantifiers of φ in $\mathbb{A}$ which makes all conjuncts true.

Q: It's funny how we considered the prenex normal form of φ, which you said is the opposite of minimizing the number of variables. What does this have to do with the syntactic width of the formula?

A: It turns out that the syntactic width of φ is related to an important structural parameter of the graph M_φ: its *treewidth*. The treewidth of a graph G is sometimes described as a measure of how tree-like G is. Roughly speaking, G has treewidth k if its edges can be covered by subgraphs of at most $k + 1$ vertices in a tree-like fashion. I won't give a detailed definition of it here as it is well-studied in the literature.

Q: Yes, I have heard a lot about it. I understand that it plays an important role in parameterized algorithms.

A: It also has a central role in structural graph theory and often pops up in logic. If you want to know more I recommend the book on *Sparsity* by Nešetřil and Ossona de Mendez [30].

Q: So, how does the treewidth of M_φ connect with the syntactic width of φ?

A: If the treewidth of M_φ is less than k, then φ can be written with at most k variables. Indeed, you can turn a tree decomposition of M_φ into a formula quite directly and the number of variables is just one more than the width of the decomposition. You can find details of this and much related material in this paper by Dalmau et al. [10].

Q: I like how we've related a purely combinatorial parameter from graph theory to a syntactic measure of the complexity of formulas. But, I suppose this is because of the special nature of conjunctive queries. As we noted, they are almost already just structures with a string of existential quantifiers in front. Can you say something about the syntactic width of more general formulas? As I recall, the reason for studying logics like L^k and $L^\omega_{\infty\omega}$ has to do with *fixed-point logics*.

A: You are absolutely right. Let's turn to these next.

3 Fixed-Point and Counting Logics

A: *Fixed-point logic*, which we'll abbreviate FP, is an extension of first-order logic with a mechanism for recursion.

Q: I know it well. I have often discussed this with Yuri [5], and it also came up in my chat with Ian Hodkinson all those years ago. I know that what is called the Immerman-Vardi theorem [24, 35] tells us that a property of finite ordered structures is decidable in polynomial time if, and only if, it is definable in FP. Yuri told me that this was also independently discovered by Livchak [27].

A: That's all true. But, if you don't restrict yourself to ordered structures there are simple properties that are not definable.

Q: Well, yes. Every formula of FP is equivalent to one of $L^\omega_{\infty\omega}$ and Kolaitis and Vardi [26] proved that the latter logic has a 0-1 law so cannot express, for instance, that a structure has an even number of elements.

A: Indeed, simple counting properties are not definable in FP. This led Immerman [24] to propose that an extension of FP with a mechanism for counting, which we now call FPC, might express all polynomial-time properties. Even this turns out not to be the case but it requires a sophisticated construction to come up with such a property. Still, the problems definable in FPC are a really interesting class in their own right and have been much studied (see [11] for some insights). What I want to mention today is that just like every formula of FP is equivalent to one of $L^\omega_{\infty\omega}$, so every formula of FPC is equivalent to one of $C^\omega_{\infty\omega}$—the *infinitary logic with counting*. The idea, as in the former case, is that recursion can be unfolded into an infinitary disjunction while keeping the number of variables bounded.

Q: So, how does counting appear in the infinitary logic?

A: This is a good question and one of the reasons for considering the translation of FPC because the counting mechanism in $C^\omega_{\infty\omega}$ is much easier to explain than the one used in FPC. First of all, C^k is the logic obtained from first-order logic by *(i)* allowing *counting quantifiers*, i.e. we can write $\exists^i x\varphi$ which is to be read as "there exist at least i elements x such that φ"; and *(ii)* restricting ourselves to formulas using only the variables $x_1, \ldots, x_k$.

Q: I suppose, as before, we don't really care about the names of the variables. The important thing is that we are restricting ourselves to formulas in which no subformula has more than k free variables. That is to say, we are again talking about syntactic width.

A: Yes, but the counting quantifiers make a difference. Note that in the formula $\exists^i x\varphi$, the i is some constant. To be precise, there is one such quantifier for each natural number.

Q: But then the counting quantifier $\exists^i$ can be replaced by a sequence of i ordinary existential quantifiers, so the logic C^k is a fragment of first-order logic.

A: That's right, but replacing the counting quantifiers in this way blows up the number of variables. So, while it is true that C^k is a fragment of first-order logic, it is not contained in any fragment of bounded syntactic width.

The fact we are interested in is that for each formula φ of FPC there is a constant k such that if two structures $\mathbb{A}$ and $\mathbb{B}$ are not distinguished by any formula of C^k then they cannot be distinguished by φ: either φ is true in both or false in both. Let's write $\mathbb{A} \equiv^k \mathbb{B}$ to denote that the two structures satisfy the same sentences of C^k.

Q: This reminds me of the discussion about FP and $L^\omega_{\infty\omega}$ we mentioned earlier. There, I remember it was an important result that the corresponding equivalence for L^k was itself definable in FP.

A: Similar facts can be established about FPC and the C^k-equivalence relations. I recommend the book length treatment of this by Martin Otto [32]. In some ways the C^k-equivalence relations are much more interesting though. It turns out that the family of equivalence relations $\equiv^k$ has many natural characterizations. It pops up quite independently in many contexts. It goes by the name of the Weisfeiler-Leman equivalences and has equivalent characterizations in terms of combinatorics, logic, algebra and linear optimization. There is a vast literature on them if you want to read more. A starting point might be these papers [25, 15]. There is also increasing interest in the topic from people working in the machine learning community [29].

From my point of view, each $\equiv^k$ is an approximation of the graph isomorphism relation, with the approximation getting finer as k increases.

Q: And this refinement process doesn't stop at any finite k? I suppose if it did, we would know that graph isomorphism is decidable in polynomial time, since each individual relation $\equiv^k$ is definable in FPC and hence in P.

A: In fact, we do know that there is no k for which $\equiv^k$ is the same as isomorphism. This was proved by Cai, Fürer and Immerman [8] in their landmark paper which showed that FPC is strictly weaker than P.

Q: So, is this parameter k one of the notions of width you were talking about?

A: It is. We sometimes speak of the Weisfeiler-Leman dimension of a graph G to denote the smallest k for which C^k distinguishes G from all non-isomorphic graphs. We also say that a class C of structures has *bounded counting width* (see [16]) if it is definable in $C^k_{\infty\omega}$. I now want to give you another characterization of the equivalence relation $\equiv^k$ that links back to our discussion of conjunctive queries.

4 Counting Homomorphisms

A: For a pair of graphs G and H, let's write $\hom(H, G)$ to denote the *set* of homomorphisms from H to G and $|\hom(H, G)|$ for the cardinality of this set. It is a classical result of Lovász [28] that two finite graphs G_1 and G_2 are isomorphic if, and only if, $|\hom(H, G_1)| = |\hom(H, G_2)|$ for all finite graphs H. Dvořák [21] showed a similar characterization of the equivalence relations $\equiv^k$. To be precise, $G_1 \equiv^k G_2$ if, and only if, $|\hom(H, G_1)| = |\hom(H, G_2)|$ for all finite graphs H of treewidth less than k.

Q: I see. This then connects *counting width* to *treewidth*. And, because we are talking of homomorphisms, it takes us back to conjunctive queries.

A: Yes, for a graph H of treewidth less than k, we can write a conjunctive query φ_H with at most k variables that says of a graph G that *there exists* a homomorphism from H to G. Can you see how we might modify this to a formula that *counts* the number of homomorphisms?

Q: I would do something like this. Convert φ_H into prenex normal form: say $\exists \mathbf{x} \theta$ and then if we want to say that there are at least i homomorphisms, I would then change it to $\exists^i \mathbf{x} \theta$. But, wait a minute, can I do that? In the definition of C^k, we had counting quantifiers to count the number of instantiations of a variable x satisfying a formula θ. Here I want to count the number of instantiations of a tuple $\mathbf{x}$.

A: That's not a problem. One can show that a quantifier counting tuples can be defined using just ordinary counting quantifiers without increasing the number of variables (see [15]). So, we can treat your formula $\exists^i \mathbf{x}\theta$ as a formula of C^k and Dvořák's theorem tells us that if a pair of graphs G and H is distinguished by any formula of C^k then they are distinguished by one of this particular form, which I will call a *counting conjunctive query* of width k.

Q: And then, it easily follows that any formula of $C^k_{\infty\omega}$ is a (possibly infinite) Boolean combination of counting conjunctive queries of width k. I suppose this is a kind of normal form for this logic. That's neat.

A: Dvořák's theorem (and indeed, Lovász') can be put in a much more general framework, in the language of category theory.

Q: I once had a discussion [6] with Yuri and Andreas about the merits of category theory. Are you a fan or a sceptic?

A: I am not a category theorist myself but I have had the fortune to work with some excellent experts on the topic. In particular, in a paper I wrote with Samson Abramsky and my student Pengming Wang [2], we defined what we called the *pebbling comonad*. Essentially this is a way of transforming a structure $\mathbb{A}$ to a structure $P_k\mathbb{A}$ which we can think of as the (infinite) tree unfolding of $\mathbb{A}$ of width k. The transformation is a *comonad* which means it gives rise to a category which we call the *Kleisli category* of P_k in which a morphism from $\mathbb{A}$ to $\mathbb{B}$ is a (standard) homomorphism from $P_k\mathbb{A}$ to $\mathbb{B}$. It turns out that such a morphism exists precisely when every k-variable conjunctive query that is true in $\mathbb{A}$ is true in $\mathbb{B}$. What's more, there is an isomorphism between $\mathbb{A}$ and $\mathbb{B}$ in this category if, and only if, $\mathbb{A} \equiv^k \mathbb{B}$. Thus, in a precise sense, equivalence in C^k is the isomorphism relation that corresponds to morphisms that preserve k-variable conjunctive queries.

Q: That neatly ties in the different notions of width we have been looking at. And, I suppose treewidth is also a natural part of this picture.

A: It turns out that a structure $\mathbb{A}$ admits a *coalgebra* for the comonad P_k if, and only if, it has treewidth less than k.

Q: You said that you obatin a generalization of the theorems of Dvořák and of Lovász from this comonad?

A: Not from this particular comonad, but we do know that this is an example of a much more general phenomenon. Say that a locally finite category $\mathcal{A}$ is *combinatorial* if two objects a and b in $\mathcal{A}$ are isomorphic if, and only if, $|\hom(c, a)| = |\hom(c, b)|$ for all objects c of $\mathcal{A}$. In a joint paper of mine with Tomáš Jakl and Luca Reggio [13], we give fairly general conditions in terms of the existence of pushouts and a proper factorization system for a category to be combinatorial. The theorems of Dvořák and of Lovász and some others from the literature [22]

are examples. In particular, the theorem of Dvořák follows from showing that the Eilenberg-Moore category of the comonad P_k satisfies these conditions.

Q: Why did you call P_k the *pebbling* comonad?

A: The name comes from *pebble games* which are a very common construct in finite model theory. There is a k-pebble one-sided game that characterizes the relation $\mathbb{A} \Rightarrow^k \mathbb{B}$ which holds if every k-variable conjunctive query true in $\mathbb{A}$ is true in $\mathbb{B}$ and there is a k-pebble bijection game which characterizes the relation $\mathbb{A} \equiv^k \mathbb{B}$. The rules of the game involve two players called *Spoiler* and *Duplicator* placing pebbles in turn on the elements of the structure. Hence the name pebble games. For an account of how these games gave rise to the pebbling comonad I recommend reading this paper [12].

Q: You have given me a lot of material to read today.

A: We have covered a lot of ground. Essentially, we have seen that there is a notion of width that gives a gradation of the relation of homomorphism and a corresponding gradation of the relation of isomorphism. These gradations have many different characterizations. For homomorphisms, we looked at the number of variables in a conjunctive query; the treewidth of graphs and the number of pebbles in a one-sided game. For isomorphisms, we have the number of variables in a counting logic formula, the Weisfeiler-Leman dimension of graphs and the number of pebbles in a two-sided bijection game. These are the notions of width we've talked about and they are all intertwined.

I do have another related notion of width I want to introduce you to. This is what I call the *support* of gates in circuits and it involves some surprising connections. This will take us on a journey through some material on circuit complexity and lead us to some quite amazing new results. Do you want to carry on?

Q: I'm up for it.

5 Circuits and Supports

A: I don't know if you are familiar with the basic setup of circuit complexity. In complexity theory, decision problems are languages, for example over a binary alphabet, so $L \subseteq \{0,1\}^*$. You can consider the nth *slice* of the language (i.e. the membership problem in L of strings of length n) as a Boolean function $L_n : \{0,1\}^n \to \{0,1\}$ and any such finite Boolean function can be computed by a circuit C_n. The circuit is made of inputs labelled with Boolean variables $x_1, \ldots, x_n$ and gates labelled by Boolean functions like AND, OR, NOT and sometimes richer functions such as threshold or majority gates and with one designated output gate.

Q: This is all fairly clear. Indeed, we can define such a family of circuits for any language L, whether decidable or not.

A: That's true. To restrict ourselves to decidable languages we would have to impose a *uniformity* condition specifying that C_n is computable from n. But, the hope in circuit complexity is that we can deduce something about the complexity of L by combinatorial properties of the circuits C_n. In particular, if L is in the complexity class P, then there is a circuit family $(C_n)_{n\in\omega}$ deciding L where the size of C_n is bounded by some polynomial in n and moreover the function taking n to C_n is itself computable in polynomial time. The hope is that for NP-complete problems we might be able to show the impossibility of circuits of polynomial size, without worrying about uniformity.

But, as a logician and a finite model theorist, I am interested in decision problems which are not necessarily sets of strings but classes of abstract structures—say graphs.

Q: Of course. The standard way to present a graph as input to a Boolean circuit would be by its adjacency matrix. So, the nth slice of a graph class C would be the collection of graphs in C which have n vertices and this is given by a Boolean function $f_n : \{0,1\}^{n^2} \to \{0,1\}$ and computed by a circuit with n^2 Boolean inputs, one for each entry in the adjacency matrix.

A: Not every such function is a slice of graph class, though. Do you see why?

Q: I do. A given graph can be represented by an adjacency matrix in more than one way. It depends how we order the vertices. So the function f_n might take different values on different representations of the same graph. What we want to do is define an equivalence relation $\sim$ on strings in $\{0,1\}^{n^2}$ where two strings are equivalent whenever they represent the same graph and then require that $f_n(x) = f_n(y)$ whenever $x \sim y$. Isn't this the same thing as saying the f_n is invariant under graph isomorphisms?

A: It is. It will be useful to describe this as a permutation invariance property of f_n. Recall that S_n is the symmetric group on n elements: the collection of all permutations of the set $\{1,\dots,n\}$. Thinking of a string $x \in \{0,1\}^{n^2}$ as a matrix $(x_{ij})_{i,j\in[n]}$, for any permutation $\pi \in S_n$ define $\pi(x)$ to be the string $y = (y_{ij})_{i,j\in[n]}$ where $y_{ij} = x_{\pi(i)\pi(j)}$. Then, say f_n is invariant if $f(x) = f(\pi(x))$ for all $x \in \{0,1\}^{n^2}$ and $\pi \in S_n$.

Q: I can see that this is the same as what I was defining. A permutation in S_n is just a re-ordering of the vertices of the graph and applying the permutation to the adjacency matrix as you have defined means it's still an adjacency matrix for the same graph. So, a family of circuits $(C_n)_{n\in\omega}$ where each C_n computes a Boolean function $f_n : \{0,1\}^{n^2} \to \{0,1\}$ that is invariant defines exactly an isomorphism-

closed class of graphs. And this class is in P if, and only if, the family of circuits satisfies the conditions you mentioned before.

A: Of course it is not obvious, given a circuit C_n whether or not it is invariant. So, we define a syntactic condition that guarantess invariance. Say that C_n is *symmetric* if any permutation in S_n applied to its inputs $(x_{ij})_{i,j\in[n]}$ can be extended to an *automorphism* of C_n. In other words, for each $\pi \in S_n$, there is an automorphism of C_n which, in particular, takes any input labelled x_{ij} to $x_{\pi(i),\pi(j)}$.

Q: . I can see that if a circuit is symmetric, and the automophisms fix the output gate, then the function it computes is invariant. Is the converse true?

A: Well, yes and no. No, in the sense that there are definitely circuits computing an invaraint function that are not symmetric. Yes, in the sense that every invariant function is computed by *some* symmetric circuit.

For the first statement, consider the following example. Take a circuit that simply takes the conjunction of all its inputs. So, it consists of a single AND gate, which is also the output gate and all inputs x_{ij} feed into it. It evaluates to 1 if, and only if, the input graph has all possible edges. This is clearly invariant and symmetric. But, we can compute the same function with a circuit using only AND gates of fan-in 2 by making a binary tree of the gates with the root as the output gate and the inputs at the leaves. This computes the same function and is therefore invariant but clearly not symmetric in the sense we defined.

Q: And, for the second statement, I suppose we can take an invariant circuit and convert it into a symmetric one computing the same function.

A: We can. The construction is a bit technical but not difficult. In particular, it depends to some extent on what functions we allow at the gates. But it is worth pointing out that it involves, in general, an exponential blow-up in the size of the circuit. One thing that the above example shows us though is that fan-in matters. To talk meaningfully of symmetric circuits we should allow gates of unbounded fan-in.

Q: Do we know that the exponential blowup is necessary? Or is it simply that the best way we currently know of doing this translation is exponential?

A: We can show it's impossible to do better. This is not too hard to show if all you have as gates are the standard Boolean basis of AND, OR and NOT gates. It's a bit more involved if you allow threshold or majority gates. In either case, it is a consequence of results in [3] which I want to come back to in a little bit.

Q: So, why are we interested in symmetric circuits? It seems like the real condition we want to think about is invariance and you have just told me that the restriction to symmetric circuits is sufficient but not necessary for invariance.

A: The interest comes from looking at definability in logic. Suppose you have a first-order sentence φ in the language of graphs. Then for each n you can produce from φ a circuit C_n which decides for an input n-vertex graph G whether G satisfies φ. Moreover, the circuit you get from φ is of polynomial-size, bounded depth and it is *symmetric*.

Q: Yes, I see that. I believe there was some work on this in the 1980s [19].

A: That's right. Also some interesting work extending it to infinitary logic by Otto [31]. When you add counting quantifiers to the mix, however, things get even more interesting. Do you see what happens if you start with a formula φ of C^k?

Q: If I follow the inductive construction of the circuit, where existential quantifiers become big OR gates and universal quantifiers are big AND gates...Oh, I see. For a counting quantifier it makes sense to introduce threshold gates. So, formulas of C^k will translate into circuits of bounded depth but with threshold gates. And the size of the circuits is $O(n^k)$.

A: Very good. And formulas of FPC translate into families of symmetric circuits of polynomial size with threshold gates, but possibly of unbounded depth. What's more, we can get a converse statement. Any class of graphs decided by a polynomial-time uniform family of symmetric circuits with threshold gates is definable in FPC. So, it is an exact circuit characterization of the logic FPC. This is the main result of [3].

Q: That really bolsters your claim that FPC is a natural class to consider. Does the logic $C^\omega_{\infty\omega}$ give you a non-uniform version of this?

A: Not quite, but it does give us something interesting. The issue of non-uniformity is dealt with in [3] by allowing *numerical relations*, and we'll leave that aside for now. When we start with infinitary formulas in $C^\omega_{\infty\omega}$, there is no guarantee that the resulting circuit will be of polynomial size. But, it is necessarily of polynomial *orbit size*.

Q: What is orbit size?

A: Suppose C_n is a symmetric circuit taking as input graphs on n vertices. Then S_n embeds into the automorphism group of C_n by definition. For a gate g in C_n, the *orbit* of g is the collection of gates g' which are mapped to g by some automorphism. The orbit size of C_n is then just the maximum size of any orbit of a gate in C_n.

Q: Yes, I think I can see the connection. If I start with a sentence φ of $C^k_{\infty\omega}$, I get an equivalent formula φ_n of C^k for any fixed size n of graphs. I suppose the size of φ_n is not necessarily bounded by a polynomial in n which is why we don't get polynomial-size circuits.

A: But, the circuit we get from φ_n has orbits of size at most n^k. This is because there is a gate $\psi[\mathbf{a}]$ in the circuit for each subformula ψ of φ_n along with an assignment of vertices $\mathbf{a}$ to the free variables of ψ. The orbit of the gate $\psi[\mathbf{a}]$ is then obtained by varying the assignment $\mathbf{a}$.

Q: I see. And since any subformula has at most k free variables, there are at most n^k assignments and this gives you an upper bound on the orbit size. I like the way we have circled back to the notion of syntactic width.

A: In fact, symmetric circuits give us another related notion of width. That is to say, there is a combinatorial parameter of symmetric circuits which is a counterpart to the syntactic width of formulas. It plays a crucial role in the translation [3] of poly-size symmetric circuits into FPC. It's what we call *supports*.

But, before we get to that, I wanted to note that the translation from symmetric circuits into FPC or into $C^{\omega}_{\infty\omega}$ means that when we prove something is not expressible in these logics, it is a circuit lower bound result. Indeed, we can now think of the bijection games that are used to prove inexpressibility in FPC as a method for proving circuit lower bounds. You can read about this in some generality in my paper with Greg Wilsenach [18] where we use this method to prove lower bounds on *algebraic circuits*.

Q: What are algebraic circuits?

A: We'll come back to that in a moment. First I want to tell you about supports.

Do you remember the orbit-stabilizer theorem?

Q: You mean from permutation group theory? Let's see: if you have a group Γ which is a subgroup of S_n, the Γ-orbit of an element $i \in [n]$ is the collection of elements j which map to i under some permutation in Γ and the stabilizer of i is the subgroup Δ_i of Γ consisting of those permutations that fix i. The theorem then says that the index $[\Gamma : \Delta_i]$ is exactly the size of the orbit of i.

A: That's right. More generally, we consider any action of Γ on a set X (it doesn't just have to be $[n]$) and we can look at the size of the orbit of any element $x \in X$. In our case, I want to consider the case where $\Gamma = S_n$ and we look at its action on a symmetric circuit C_n and think about the orbits of gates.

Q: I see, so if the circuit has orbit size at most n^k, that means that the stabilizer group of any gate g has index in S_n of at most n^k. In other words the stabilizer group is big. Many permutations fix g. I can see this is true for the circuits we constructed from formulas. There each gate was of the form $\psi[\mathbf{a}]$ and clearly any permutation of the vertices that fixes all vertices in the tuple $\mathbf{a}$ fixes the gate. Since the tuple is small (at most k elements) there are lots of such permutations. What you're saying is that something like this is true in all symmetric circuits, not just those that we construct from formulas.

A: In fact, it turns out something stronger is true. Say that a set $X \subseteq [n]$ is a support of a group $\Delta \leq S_n$ if every permutation that fixes all the elements of X is in Δ.

Q: So, just like the elements in the tuple **a** form a support for the stabilizer group of the gate $\psi[\mathbf{a}]$.

A: Like that. But, we can show that in any symmetric circuit in which all gates have an orbit of size at most n^k, the stabilizer of every gate has support of size at most k. So, we associate with a circuit C a measure we call its support size, which is the size of the largest support of any gate. This is the parameter that is the counterpart to width, and which we use to show lower bounds.

In particular, we can directly use bijection games to argue that some properties are not decidable by symmetric circuits with small support. This is just an adaptation of their use in showing that the properties are not definable in $C^{\omega}_{\infty\omega}$. But, we can also use them in the context of circuits where there is no immediate connection to logic.

Q: Are we now turning to algebraic circuits?

6 Algebraic Circuits

A: Algebraic circuits, sometimes also called arithmetic circuits, are a model of computation used to study computations over a field (such as the real numbers or complex numbers) where we treat addition and multiplication as operations of unit cost.

Formally, an algebraic 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 polynomial in the ring $K[X]$. The minimal size circuit computing a polynomial p tells us the minimal number of operations needed to compute p. Another way of thinking about it is that the circuit is a compact representation of the polynomial. Potentially much more compact than the usual representation as a sum of monomials.

Q: This seems quite different from the world of Boolean circuits. We are no longer looking at the complexity of decision problems or languages.

A: Still, there is an algebraic analogue of the question of whether $\mathsf{P} \neq \mathsf{NP}$, which is known as Valiant's conjecture. It is usually stated in the form $\mathsf{VP} \neq \mathsf{VNP}$.

Q: What are VP and VNP?

A: Formally, we are looking at the complexity of *families* of polynomials $(p_n)_{n\in\omega}$ with $p_n \in K[x_1, \ldots, x_n]$. The family is in VP if there are circuits C_n computing p_n and the size of the circuits grows only polynomially in n. VNP has a definition as families of polynomials whose coefficients can be computed by polynomial-size circuits. There is a vast literature on algebraic circuits and this tutorial [34] is an excellent entry point.

Q: I take it you want to define a symmetric version of these circuits. But, the inputs here are no longer graphs. What are the symmetries you are looking at?

A: In many of the families of polynomials that are often studied in the field of algebraic complexity, the variables are the entries of a matrix. That is, we can index the variable set X as $X = \{x_{ij} \mid 1 \leq i \leq m; 1 \leq j \leq n\}$. In particular, when they form a square matrix, i.e. $m = n$. The classic examples are the *determinant polynomial*:

$$\mathrm{Det}(X) = \sum_{\sigma\in S_n} \mathrm{sgn}(\sigma) \prod_{i\in[n]} x_{i\sigma(i)};$$

and the *permanent polynomial*:

$$\mathrm{Per}(X) = \sum_{\sigma\in S_n} \prod_{i\in[n]} x_{i\sigma(i)}.$$

Q: Why are these *classic* examples?

A: They are the most studied in the field of algebraic complexity. The determinant is known to be in VP and the permanent is VNP-complete. This means that Valiant's conjecture is equivalent to the statement that the permanent is not in VP. Indeed, much work in algebraic complexity is about understanding the difference between these two families of polynomials.

Q: And I take it there are differences in terms of symmetric circuits?

A: Yes, let's define what we mean by symmetry. Say we have a circuit C computing a polynomial $p \in K[X]$ and Γ is a group of permutations of X. We say that $p[X]$ is Γ-invariant if for each $\pi \in \Gamma$ the polynomial p^π we get by permuting the variables of p according to Γ is identical to p.

And, we say that the circuit C is Γ-symmetric if every permutation of Γ applied to the inputs of C can be extended to an automorphism of C.

Q: So it is very much like the Boolean symmetric circuits we talked about. For $\mathrm{Det}(X)$ and $\mathrm{Per}(X)$, the set $X = \{x_{ij} \mid 1 \leq i, j \leq n\}$ and so the full group of permutations of X looks like S_{n^2}. What is the group Γ?

A: We could, for example, consider the group S_n acting simultaneously on the rows and columns of the matrix as we did for Boolean circuits. So, Γ is the

group of permutations $\pi \in S_n$ acting on X by the action that takes x_{ij} to $x_{\pi(i)\pi(j)}$. Then, both $\mathrm{Det}(X)$ and $\mathrm{Per}(X)$ are Γ-invariant. The determinant and permanent of a matrix are not changed if you permute the entries by applying the same permutation to the rows and columns. We say that these polynomials are *square symmetric*.

Q: And what can we say about symmetric circuits for them?

A: We showed [17] that there are polynomial size square symmetric circuits for $\mathrm{Det}(X)$ but provably none for $\mathrm{Per}(X)$. This shows an exponential separation in the complexity of these two families of polynomials in a restricted model, which is the first such separation I know of. The method of proof for the lower bound really uses the connection with Boolean circuits and $C^{\omega}_{\infty\omega}$. In fact, we show that any family of circuits for $\mathrm{Per}(X)$ has exponential *orbit size*.

Q: I will have to read up more about it. When we talked of Boolean circuits, the inputs were adjacency matrices of graphs. So the natural notion of symmetry was permutations of the vertices, which corresponds to what you called square symmetric. For general matrices, is there any reason to privilege this particular permutation group?

A: It's still quite a natural collection of permutations to consider, but you are right, we could consider others. For instance, consider the group $S_n \times S_n$, i.e. the direct product of S_n with itself. This has an action on the matrix X of variables where the pair (π, σ) takes x_{ij} to $x_{\pi(i)\sigma(j)}$. In other words we can apply a permutation to the rows of the matrix and possibly a different permutation to its column. Then $\mathrm{Per}(X)$ is invariant under this group action, but $\mathrm{Det}(X)$ is not. We say that $\mathrm{Per}(X)$ is *matrix symmetric* but $\mathrm{Det}(X)$ is not.

Q: I see that. In fact the determinant of a matrix is not changed if you apply one permutation to the rows and another permutation of the same sign to the columns. But, if you apply permutations of different signs to the rows and columns, it would change the sign of the determinant.

A: When you evaluate a polynomial $p(X)$ at values of X which are in the set $\{0, 1\}$, you can understand it as computing a parameter of a graph, provided that $p(X)$ has the right symmetries. So, if $p(X)$ is square symmetric, then substituting for X the adjacency matrix of a graph gives a value which really only depends on the graph.

Q: What about when it is matrix symmetric?

A: In that case, it is best to think of a $\{0, 1\}$ valuation of X as putting in the *biadjacency matrix* of a bipartite graph. That is, we have one set A of vertices corresponding to the rows and another set B corresponding to the columns. Then, any graph property (or more generally any graph parameter) should be invariant under permuting A and B separately.

Q: In this case, there is no reason to restrict ourselves to square matrices. The sets A and B could have different sizes.

A: They could. So, let's go back to the more general case where $X = \{x_{ij} \mid 1 \leq i \leq m; 1 \leq j \leq n\}$ and consider the action of the group $\Gamma = S_m \times S_n$ on this where (π, σ) takes x_{ij} to $x_{\pi(i)\sigma(j)}$ for $\pi \in S_m$ and $\sigma \in S_n$. It turns out that we can give a fairly complete characterization of families of polynomials that have Γ-symmetric circuits in this sense. Again, let's call them *matrix symmetric*.

Q: What's this complete characterization?

A: It's in terms of what we call *homomorphism polynomials*. So, fix an $a \times b$ matrix M with entries in the field K and the set of variables $X = \{x_{ij} \mid 1 \leq i \leq m; 1 \leq j \leq n\}$. Note that a and b are not related to the parameters m and n. We now define the polynomial

$$\hom_M(X) = \sum_{f:[a]\to[m],g:[b]\to[n]} \prod_{i\in[a],j\in[b]:M_{i,j}\neq 0} M_{i,j} x_{f(i),g(j)}.$$

Q: Why is this called a homomorphism polynomial?

A: Consider the special case when the matrix M only has entries in $\{0, 1\}$. We can then think of it as the biadjacency matrix of a graph F_M on a pair of sets of vertices A of size a and B of size B. Then $\hom_M(X)$ is a polynomial in the variables X which, when we substitute for X the biadjacency matrix of a bipartite graph G with vertiex bipartition (U, V) evaluates to the number of homomorphisms from F_M to G. That is, only counting homomorphisms that take A into U and B into V.

Q: I can see that. So, we generalize the homomorphism counting parameter to matrices M which are not necessarily $\{0, 1\}$. I suppose I can think of such matrices as weighted bipartite graphs.

A: And then $\hom_M(X)$ is in some sense counting "*weighted homomorphisms*". These are polynomials that come up quite naturally in algebraic complexity. For instance [20] argue that they give rise to the first natural VP-complete families.

Q: Nice. And how do they come up in characterizing matrix-symmetric families of polynomials?

A: This is a theorem we proved in some recent work that I did with Benedikt Pago and Tim Seppelt [14].

Theorem 1. *A family of polynomials $p(X)$ is computed by a family of matrix symmetric circuits of polynomial orbit size if, and only if, there is a k such that each p is a linear combination of homomorphism polynomials $hom_M(X)$ of graphs F_M of treewidth less than k.*

Q: So bounded treewidth pops up again in the characterization. We're back at the same combinatorial notions of width.

A: Yes, you could look at this theorem as an analogue of the normal form for $C^{\omega}_{\infty\omega}$ we obtain as a consequence of Dvořák's result. So, $C^{\omega}_{\infty\omega}$ defines exactly the properties of graphs decided by families of symmetric circuits with polynomial size orbits. And, from Dvořák's theorem we get that all such properties are obtained as infinite Boolean combinations of homomorphism counts from a class of graphs of bounded treewidth.

Q: I do see the analogy. But, we are looking at somewhat different notions of symmetry.

A: It is indeed just an analogy at the moment. I don't see how you could derive one result from the other. But it is compelling and shows the natural connection between notions of width and symmetry.

7 Conclusion

Q: Well, we have gone over a lot of material and you have given me really a lot to go away and read and digest. I think it is time for a break.

A: We have covered a lot of ground, some of it very new. The take away is that *width* in its various forms is a measure of complexity (of graphs, of formulas, of circuits) which ties to many natural and independently discovered measures in combinatorics, logic and complexity. It turns out to be useful in proving surprising unconditional lower bounds in circuit complexity and other areas.

Q: I will go away and chew on that.

A: And there is so much ground that we didn't cover. There are notions of width in the field of *constraint satisfaction*. There is *submodular width* and *hypertree width*. They are not unrelated to what we have talked about but we didn't have time to go over them.

Q: It will have to be another time. I look forward to carrying on our conversations.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Abramsky, A. Dawar, and P. Wang. The pebbling comonad in finite model theory. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005129.

[3] M. Anderson and A. Dawar. On symmetric circuits and fixed-point logics. *Theory Comput. Syst.*, 60(3):521–551, 2017.

[4] L. Birkedaal and B. Köning, editors. *ACM/IEEE Symp. Logic in Computer Science*. IEEE, 2025.

[5] A. Blass and Yu. Gurevich. Two forms of one useful logic: Existential fixed point logic and liberal datalog. *Bull. EATCS*, 95:164–182, 2008.

[6] A. Blass and Yu. Gurevich. Who needs category theory? *Bull. EATCS*, 124, 2018.

[7] S. Bova and H. Chen. How many variables are needed to express an existential positive query? *Theory Comput. Syst.*, 63:1573–1594, 2019. doi:10.1007/S00224-018-9884-Z.

[8] 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.

[9] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *STOC*, pages 77–90, 1977.

[10] V. Dalmau, Ph.G. Kolaitis, and M.Y. Vardi. Constraint satisfaction, bounded treewidth, and finite variable logics. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming, CP'02*, Lecture Notes in Computer Science, pages 311–326. Springer-Verlag, 2002.

[11] A. Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2(1):8–21, 2015.

[12] A. Dawar. Constraint satisfaction, graph isomorphism, and the pebbling comonad. In Alessandra Palmigiano and Mehrnoosh Sadrzadeh, editors, *Samson Abramsky on Logic and Structure in Computer Science and Beyond*, pages 671–699. Springer Verlag, 2023.

[13] A. Dawar, T. Jakl, and L. Reggio. Lovász-Type Theorems and Game Comonads. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 2021. doi:10.1109/LICS52264.2021.9470609.

[14] A. Dawar, B. Pago, and T. Seppelt. Symmetric algebraic circuits and homomorphism polynomials. arXiv 2502.06740, 2025.

[15] A. Dawar and D. Vagnozzi. Generalizations of k-Weisfeiler-Leman stabilization. *Moscow Journal of Number Theory and Combinatorics*, 9:229–252, 2020.

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

[17] A. Dawar and G. Wilsenach. Symmetric Arithmetic Circuits. In *47th International Colloquium on Automata, Languages, and Programming*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:18, 2020. doi:10.4230/LIPIcs.ICALP.2020.36.

[18] A. Dawar and G. Wilsenach. Lower bounds for symmetric circuits for the determinant. In *13th Innovations in Theoretical Computer Science Conference, ITCS*, volume 215 of *LIPIcs*, pages 52:1–52:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.52.

[19] L. Denenberg, Y. Gurevich, and S. Shelah. Definability by constant-depth polynomial-size circuits. *Information and Control*, 70:216–240, 1986.

[20] A. Durand, M. Mahajan, G. Malod, N. de Rugy-Altherre, and N. Saurabh. Homomorphism polynomials complete for VP. *Chic. J. Theor. Comput. Sci.*, 2016.

[21] Z. Dvořák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64:330–342, 2010.

[22] M. Grohe. Counting Bounded Tree Depth Homomorphisms. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 507–520. ACM, 2020. doi:10.1145/3373718.3394739.

[23] I. M. Hodkinson. Finite variable logics. *Bull. EATCS*, 51:111–140, 1993.

[24] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

[25] Sandra Kiefer. The Weisfeiler-Leman algorithm: an exploration of its power. *ACM SIGLOG News*, 7(3):5–27, 2020.

[26] Ph. G. Kolaitis and M. Y. Vardi. Fixpoint logic vs. infinitary logic in finite-model theory. In *LICS*, pages 46–57, 1992.

[27] A. Livchak. The relational model for process control. *Automated Documentation and Mathematical Linguistics*, 4:27–29, 1983.

[28] L. Lovász. Operations with structures. *Acta Math. Acad. Sci. Hungar.*, 18:321–328, 1967.

[29] Ch. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*, pages 4602–4609. AAAI Press, 2019. doi:10.1609/AAAI.V33I01.33014602.

[30] Jaroslav Nešetřil and Patrice Ossona De Mendez. *Sparsity: graphs, structures, and algorithms*, volume 28. Springer Science & Business Media, 2012.

[31] 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.

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

[33] B. Poizat. Deux ou trois choses que je sais de L_n. *Journal of Symbolic Logic*, 47(3):641–658, 1982.

[34] A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010. doi:10.1561/0400000039.

[35] M. Y. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM Symp. on the Theory of Computing*, pages 137–146, 1982.