The Finite Model Theory Toolbox of a Database Theoretician

Leonid Libkin
University of Edinburgh
libkin@inf.ed.ac.uk

ABSTRACT

For many years, finite model theory was viewed as the backbone of database theory, and database theory in turn supplied finite model theory with key motivations and problems. By now, finite model theory has built a large arsenal of tools that can easily be used by database theoreticians without going to the basics such as combinatorial games. We survey such tools here, focusing not on how they are proved, but rather on how to apply them, as-is, in various questions that come up in database theory.

Categories and Subject Descriptors. F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages; H.2.4 [**Database Management**]: Systems—*Relational databases*

General Terms. Theory, Languages, Algorithms

Keywords. finite models, expressive power, complexity, games, order, types, logics, query languages

1. Introduction

Since database query languages are logic-based (relational calculus *is* first-order logic), answering relational queries amounts to evaluating formulae over finite relational structures. Dealing with logical formulae over finite structures is the subject of finite model theory. So not surprisingly, finite model theory played a central role in the development of database theory (it was even called *the backbone of database theory* [44]), and database-related questions have traditionally provided the main motivation for finite model theory research. Even the first formal definition of a central database concept – that of a *query* – was given by Chandra and Harel in what is now viewed as one of the seminal finite model theory papers [4].

Even a decade ago, PODS routinely published papers the core of which could be classified as pure finite model theory. But the subject of finite model theory has finally come of age; several texts have appeared [7, 14, 24, 28], and a database theoretician no longer needs to be playing complicated combinatorial games to get the results he or she needs. Finite model theory has developed a large arsenal of tools – many of them motivated by database problems – that can be routinely used to get the results we (database theoreticians) need.

To become familiar with such tools, one needs to go over a finite model theory text, and such texts concentrate on proofs and underlying principles as much as on the applicability of tools. So my goal is to present, in this short survey, the key tools of finite model theory that can be used by a database researcher. I shall not explain how they are proved, but concentrate instead on the statements and examples of their applicability. This is not supposed to be a comprehensive survey of finite model theory, and thus the list of references is not meant to be exhaustive; the reader is referred to the above mentioned texts for additional information, historical comments, and references.

After giving basic notations, we consider standard tools for proving expressivity bounds on first-order query languages: games, locality, zero-one laws. We then move away from first-order, and look at counting and aggregate extensions (that resemble the counting power of SQL) and at fixed-point extensions. We then revisit first-order in a setting where nontrivial conditions on values stored in the database (e.g., arithmetic operations and comparisons) can be used in queries. After that we look at different applications of finite model theory tools: instead of proving that some queries are inexpressible in a logic, the new set of tools will help us show equivalence of languages. Such tools are based on computing with types and the composition method, and they are often used for languages over trees (e.g., XML documents). We conclude by outlining the basics of descriptive complexity (showing that the logic in which a query is expressed and the number of variables used in a query tell us a lot about the complexity of query evaluation), as well as satisfiability properties of queries, which can be used in static analysis questions and answering queries over incomplete data.

PODS'09, June 29–July 2, 2009, Providence, Rhode Island, USA.

2. Notations

We shall use the terminology of logic rather than relational databases and shall refer to *vocabularies* instead of relational schemas. A vocabulary σ is a set of relational symbols with associated arities. A structure of vocabulary σ is $\mathfrak{A} = \langle A, (R^{\mathfrak{A}})_{R\in\sigma}\rangle$, where A, called the universe of $\mathfrak{A}$, is a nonempty set (always assumed to be finite), and each $R^{\mathfrak{A}}$ is an interpretation of a relation from σ: if R is k-ary, then $R^{\mathfrak{A}} \subseteq A^k$. We shall normally omit the superscript $\mathfrak{A}$ in interpretations of relations.

Note that some elements of A may not be present in any of the relations $R^{\mathfrak{A}}$; this is normal for the definition of a structure, but of course in databases the universe is assumed to be the set of all atomic values present in the database (this set of values is sometimes referred to as the *active domain* of a database). This little difference won't affect any of the results as we can always add a unary relation interpreted as A; then the active domain and the universe coincide.

Sometimes we look at vocabularies that have constant symbols in addition to relation symbols. A constant symbol is interpreted in $\mathfrak{A}$ as an element of A. When we write $a \in \mathfrak{A}$, we actually mean $a \in A$ where A is the universe of $\mathfrak{A}$.

We often deal with graphs for the simplicity of exposition; in that case the vocabulary contains one binary relation E, and structures are $G = \langle A, E\rangle$, where A is the set of nodes and E is the set of edges.

An *m-ary query* over σ-structures is a mapping Q that associates with each structure $\mathfrak{A}$ a subset of A^m so that Q is closed under isomorphism: if h is an isomorphism between $\mathfrak{A} = \langle A, (R^{\mathfrak{A}})_{R\in\sigma}\rangle$ and $\mathfrak{B} = \langle B, (R^{\mathfrak{B}})_{R\in\sigma}\rangle$, then $h(Q(\mathfrak{A})) = Q(\mathfrak{B})$. A 0-ary query can return two possible values and thus is naturally associated with a *Boolean query*, i.e., a class of σ-structures closed under isomorphism. An example of a binary query is the transitive closure of a graph; examples of Boolean queries are connectivity test for graphs, and a query EVEN testing if the cardinality of A is even.

We mostly look at *first-order logic* (FO) over σ, which is obtained by closing atomic formulae $R(\bar{x})$ under the Boolean connectives $\vee, \wedge, \neg$ and quantification $\forall x, \exists x$. We also look at second-order extensions, in particular, monadic extensions, that permit quantification over sets $\forall X, \exists X$. A sentence is a formula without free variables. We write $\varphi(\bar{x})$ to indicate that $\bar{x}$ is the tuple of free variables of φ.

By the *quantifier rank* of a formula, $\mathsf{qr}(\varphi)$, we mean the depth of quantifier nesting in φ; that is, $\mathsf{qr}(\varphi) = 0$ if φ is atomic; $\mathsf{qr}(\varphi \vee \psi) = \mathsf{qr}(\varphi \wedge \psi) = \max(\mathsf{qr}(\varphi), \mathsf{qr}(\psi))$; $\mathsf{qr}(\neg\varphi) = \mathsf{qr}(\varphi)$; and $\mathsf{qr}(\exists x\varphi) = \mathsf{qr}(\forall x\varphi) = \mathsf{qr}(\varphi) + 1$.

3. Classics – FO definability

In the early days of database theory, people liked to prove inexpressibility results. The basic query language – relational calculus – is rather limited, so it is natural to ask whether some natural queries (e.g., the transitive closure of a graph) can be expressed in it. Ever since the negative answer to this question given by Fagin [8], database theory has seen much activity in the development of tools to prove such results. And since relational calculus has precisely the power of FO, it is natural to ask first what logicians had to offer.

3.1 Forget the standard tools

Logicians use tools such as compactness. Compactness says that if we have a set Φ of sentences $\{\varphi_i | i \in I\}$ of the same vocabulary, and each finite subset of Φ has a model, then Φ has a model. Now imagine a property $\mathcal{P}$ that we want to prove inexpressible in FO. The usual argument goes as follows. Assume $\mathcal{P}$ is expressible by a sentence φ. Then construct a set of sentences Φ so that each model of Φ does not satisfy $\mathcal{P}$ and yet each finite subset of Φ has a model satisfying $\mathcal{P}$. By compactness the latter will tell us that $\Phi \cup \{\varphi\}$ has a model; this contradiction then implies that $\mathcal{P}$ is not FO-expressible. In fact this is how one shows that graph connectivity is not FO-expressible.

So why aren't database theoreticians happy with it? Because compactness deals with infinite structures: in fact the model satisfying the entire family Φ in the statement of the compactness theorem is (almost always) infinite. And as database people, we want to deal with finite structures.

But maybe a finite version of compactness holds? That is, if each finite subset of Φ has a *finite* model, then Φ has a *finite* model. The problem is, it doesn't. Just look at the sentences λ_n saying that a structure has n distinct elements: $\exists x_1, \ldots, x_n \bigwedge_{i\neq j} \neg(x_i = x_j)$. Each finite set of such sentences has a finite model, but the set $\{\lambda_n | n \in \mathbb{N}\}$ doesn't. So compactness is not our tool of choice.

That said, we'll give one example when compactness proves something about *finite* models. Namely, we show that over the empty vocabulary (i.e., just sets), the query EVEN (even cardinality) is not FO-expressible. Assume, to the contrary, that φ expresses EVEN over finite structures, and consider the set $\Phi_0 = \{\lambda_n | n \in \mathbb{N}\} \cup \{\varphi\}$. Each finite subset of Φ_0 has a model (just pick a large enough even number), and hence by compactness Φ_0 has a model. Another well-known result in logic implies that it has a *countable* model, i.e., just a countable set. But we could have applied the same argument to $\Phi_1 = \{\lambda_n | n \in \mathbb{N}\} \cup \{\neg\varphi\}$ to get a countable set satisfying $\neg\varphi$. Now we have two countable sets, which are isomorphic as structures; one satisfies φ and the other $\neg\varphi$. This contradiction implies that φ cannot express EVEN.

But such proofs are rare in database theory, and we now move to different tools designed for finite structures.

3.2 Games

For much of the early history of applications of finite model theory, games were the tool of choice. They are commonly known as *Ehrenfeucht-Fraïssé games*. Such a game is played on two structures $\mathfrak{A}$ and $\mathfrak{B}$ of the same vocabulary by two players, called the *spoiler* and the *duplicator* (less

imaginative names such as player 1 and player 2 are also often used). Think of the spoiler as someone trying to show that $\mathfrak{A}$ and $\mathfrak{B}$ differ, and of the duplicator as someone trying to show that they are the same. Even if $\mathfrak{A}$ and $\mathfrak{B}$ are not isomorphic, the games goes only for a fixed number of rounds, and this gives the duplicator a chance of winning.

The game goes as follows. In each round i, the spoiler picks a structure and an element of that structure. The duplicator goes to the other structure and picks an element there. So if the spoiler picks $\mathfrak{A}$ and an element $a_i \in \mathfrak{A}$, the duplicator responds with an element $b_i \in \mathfrak{B}$; and if the spoiler picks $\mathfrak{B}$ and $b_i \in \mathfrak{B}$, then the duplicator responds with an element $a_i \in \mathfrak{A}$. After n rounds, we have points $a_1, \ldots, a_n$ played in $\mathfrak{A}$ and $b_1, \ldots, b_n$ played in $\mathfrak{B}$. The duplicator wins the game if the mapping $a_i \mapsto b_i$ is a partial isomorphism between $\mathfrak{A}$ and $\mathfrak{B}$. For example, if the structures are graphs, it means that $a_i = a_j$ iff $b_i = b_j$ and that $E(a_i, a_j)$ iff $E(b_i, b_j)$ for all $i, j \leq n$. We say that the duplicator *has a winning strategy* in the n-round game if he can win no matter how the spoiler plays. In that case, we write $\mathfrak{A} \equiv_n \mathfrak{B}$.

The reason this is important is due to the following: $\mathfrak{A} \equiv_n \mathfrak{B}$ iff $\mathfrak{A}$ and $\mathfrak{B}$ agree on all FO sentences of quantifier rank up to n. So now we have a nice tool to prove that a property $\mathcal{P}$ is not FO-expressible: come up with families of structures $\mathfrak{A}_n, \mathfrak{B}_n$, $n \in \mathbb{N}$, so that:

1. all $\mathfrak{A}_n$'s satisfy $\mathcal{P}$; no $\mathfrak{B}_n$ satisfies $\mathcal{P}$; and
2. $\mathfrak{A}_n \equiv_n \mathfrak{B}_n$ for all n.

Why does this work? Assume $\mathcal{P}$ is expressible in FO by a sentence φ of quantifier rank n. Then $\mathfrak{A}_n \models \varphi$ and $\mathfrak{B}_n \models \neg\varphi$ by 1), but 2) tells us that $\mathfrak{A}_n$ and $\mathfrak{B}_n$ have to agree on φ.

So why not just stop there? The method of games looks nice, and it is in a certain sense complete: any inexpressibility result – even relative to a class of structures – can in principle be proved by games. The problem with the technique is that, even if we find good classes of structures $\mathfrak{A}_n$ and $\mathfrak{B}_n$, it is often *hard* to prove that $\mathfrak{A}_n \equiv_n \mathfrak{B}_n$.

To illustrate this, we start with a very simple example, where playing the game is actually easy, and then show how the complexity rises quickly as structures get a bit more complicated. The easy example is again the query EVEN on sets, i.e. structures of the empty vocabulary. Note that in the n-round game on any two sets with at least n elements, the duplicator has a very simple winning strategy: if the spoiler plays an already played element, the duplicator does the same in the other set, and if the spoiler plays a new element, so does the duplicator: the sizes of the sets ensure that in n rounds, the duplicator won't run out of elements to play.

So to show that EVEN is not expressible, we can take, for example, $\mathfrak{A}_n$ to be a $2n$-element set and $\mathfrak{B}_n$ to be a $(2n+1)$-element set; by what we just saw, $\mathfrak{A}_n \equiv_n \mathfrak{B}_n$. So far so good, but what if we have something in the vocabulary? After all, databases without relations aren't very interesting!

Let's try to look at a little extension: suppose we deal not with sets, but with orders, i.e. graphs with one binary relation interpreted as a linear order. We denote an n-element linear ordering by L_n. Can we prove that EVEN is not expressible over linear orders?

A moment's reflection shows that the previous proof does not work. But the following was observed by several authors, e.g., [37]:

Theorem 3.1. *For every $m, k \geq 2^n$, we have $L_m \equiv_n L_k$.*

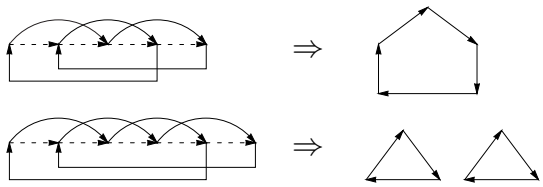
In particular, EVEN is not expressible over orders: we take L_{2^n} as $\mathfrak{A}_n$, and L_{2^n+1} as $\mathfrak{B}_n$.

But it is more important to observe that a small addition to the structure leads to an "exponential" blowup in the complexity of the proof. In fact one does not even need an order relation: the successor relation would do. And what if we have two successor relations? Or three? Game-based proofs become very heavy combinatorially. In fact, [10] suggested that we build a *library* of winning strategies for the duplicator. We now start working towards such a library, first by showing a few simple tricks, and then creating powerful tools for proving inexpressibility results.

3.3 Inexpressibility results: tricks

We have shown very little so far – only that EVEN cannot be expressed over sets and linear orders – but with that, we can already derive surprisingly strong bounds on the expressiveness of FO. We are about to show, with a simple trick, that graph connectivity, acyclicity, and the transitive closure query are not FO-definable.

For graph connectivity, we start with linear orders. The following query is easily definable: for each element in the order, put an edge to its 2nd successor; also put edges between the last element of the order and the 2nd element, and the penultimate element and the first element. This construction is illustrated below for orders on 5 and 6 elements.



It is now easy to observe that: a) the construction we presented is expressible in FO; and b) the resulting graph is connected for orders of odd size, and contains two connected components for orders of even size. Hence, if we could express graph connectivity in FO, we would be able to express EVEN on linear orders, contradicting Theorem 3.1.

This trick, observed by [19], also gives an easy proof that testing whether a graph is acyclic is not FO-definable. In this case, simply put one back edge, from the last element to the first. The resulting graph is acyclic for orders of even size, and cyclic for orders of odd size. With the transitive closure query one can check if a graph is connected: add an edge (x, y) for each edge (y, x), compute the transitive closure, and see if the resulting graph is complete. So we get:

Corollary 3.2. *Connectivity, acyclicity, and transitive closure queries are not FO-expressible.*

While the reduction-to-EVEN trick is nice, it is just a trick, and not yet a tool that can be applied in many situations. We shall now be looking at such tools, starting with those based on the locality of FO.

3.4 Inexpressibility tools: locality

As a warm-up, consider again the inexpressibility of transitive closure. Suppose we now start with a *successor* relation, i.e. a graph of the form $\{(a_1, a_2), (a_2, a_3), \ldots, (a_{n-1}, a_n)\}$, where all the a_i's are distinct. When we view it as a graph, all the in- and out-degrees of nodes are either 0 or 1: in fact, the in-degree of a_1 and the out-degree of a_n are 0, and all other in- and out-degrees are 1. In the transitive closure, we have all the edges (a_i, a_j) for $i < j$. In particular, for each number k from $\{0, \ldots, n-1\}$ there is a node whose in- or out-degree is k. Thus, the transitive closure query takes a graph whose degrees are either 0 and 1 and produces a graph which realizes a "large" number of degrees: large here means depending on the input.

It turns out that FO-definable queries cannot exhibit such a behavior. For now, consider queries Q on graphs; that is, both the input and the output of a query are graphs. If such a query were definable in a logic, it would be by a formula $\varphi(x, y)$ with two free variables. The first locality-based tool is captured by the following definition.

Definition 3.3. *A query Q has the* bounded number of degrees property (BNDP) *if there is a function $f_Q : \mathbb{N} \to \mathbb{N}$ such that for each graph G whose in- and out-degrees are bounded by a number k, the number of different in- and out-degrees in $Q(G)$ is at most $f_Q(k)$.*

Theorem 3.4. ([6]) *Every FO-definable query has the BNDP.*

The result is not limited to graph queries: it holds for all FO-definable queries under the appropriate notion of a degree in m-ary relations.

The BNDP is a very simple tool to use to prove that fixed-point queries cannot be defined in FO: indeed, it is often easy to produce many different degrees in the output with such queries (typically, each stage of the fixed-point computation generates a new element of the degree-set). The transitive closure is one example, as we just saw. As another application, consider the *same-generation* query expressed by the Datalog program below:

$$\begin{array}{lll} sg(x, x) & :- & \\ sg(x, y) & :- & e(x', x), e(y', y), sg(x', y') \end{array}$$

That is, if $e(\cdot, \cdot)$ is the parent-child relation, then x and y are in the same generation if so are their parents or if $x = y$. Now consider a full binary tree of depth n. In it, all nodes have degrees 0, 1, or 2, but in the output of the same-generation query we would have all degrees $1, 2, 4, \ldots, 2^n$ present – hence it violates the BNDP and is not FO-expressible.

The BNDP itself is based on two *locality* tools that have found numerous applications. They originate from results by Gaifman [12] and Fagin, Stockmeyer, Vardi [10] (which adapted results of Hanf [20] to finite models). Again, we present these notions for graphs to keep the notation simple, but they extend to queries on arbitrary structures.

Given a graph G, the distance $d(a, b)$ between two nodes is the length of the shortest path between them, if we forget about the orientation of edges (i.e., we can traverse an edge (u, v) in the direction from u to v, and from v to u). The distance $d(\bar{a}, b)$ for $\bar{a} = (a_1, \ldots, a_n)$ is the minimum of the distances $d(a_i, b)$.

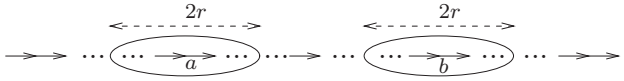
If $G = \langle A, E\rangle$ is a graph and $\bar{a} = (a_1, \ldots, a_n) \in A^n$, then the *radius r ball around $\bar{a}$* is the set $B_r^G(\bar{a}) = \{b \in A \mid d(\bar{a}, b) \leq r\}$, and the *$r$-neighborhood of $\bar{a}$ in G* is the subgraph induced by $B_r^G(\bar{a})$, with $\bar{a}$ being distinguished nodes. The latter means that if we consider an isomorphism $h : N_r^G(a_1, \ldots, a_n) \to N_r^{G'}(b_1, \ldots, b_n)$, then we must have $h(a_i) = b_i$ for all i.

Definition 3.5. *An m-ary query Q, for $m > 0$, is called* Gaifman-local *if there exists a number $r \geq 0$ such that for every graph G, two tuples $\bar{a}, \bar{b} \in A^m$ cannot be distinguished by Q whenever $N_r^G(\bar{a})$ and $N_r^G(\bar{b})$ are isomorphic.*

By "cannot be distinguished" we mean that $\bar{a} \in Q(G)$ iff $\bar{b} \in Q(G)$. This notion applies to all FO-queries:

Theorem 3.6. ([12]) *Every FO-definable query is Gaifman-local.*

The canonical example of using Gaifman-locality is proving that transitive closure is not FO-definable. Suppose it were, by a query Q; then choose r as in the definition and consider a very long chain, as below, with two points at distances bigger than $2r$ from each other, and from the endpoints:



Then r-neighborhoods of (a, b) and (b, a) are isomorphic, since each is a disjoint union of two chains of length $2r$. We know that (a, b) belongs to the output of Q; hence by Gaifman-locality, (b, a) is in the output as well, which contradicts the assumption that Q defines transitive closure.

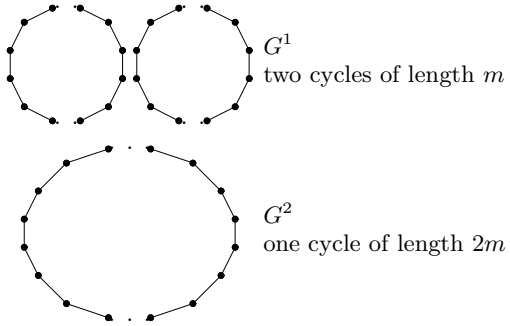
And yet another notion of locality is applicable to FO-queries, and it is often useful in establishing expressivity bounds for Boolean queries. It refers to *pairs* of structures. Again we deal with graphs for simplicity. If $G = \langle A, E\rangle$ and $G' = \langle A', E'\rangle$ are two graphs, we write $G \leftrightarrows_r G'$ if there exists a bijection $f : A \to A'$ such that for every $a \in A$, the neighborhoods $N_r^G(a)$ and $N_r^{G'}(f(a))$ are isomorphic. The $\leftrightarrows_r$ relation says, in a sense, that locally two graphs look the same, with respect to a certain bijection f; that is, f sends each node a into $f(a)$ that has the same neighborhood.

Definition 3.7. *A Boolean query Q is* Hanf-local *if there exists a number $r \geq 0$ such that for every two graphs G and G' satisfying $G \leftrightarrows_r G'$, we have $Q(G) = Q(G')$.*

Theorem 3.8. ([10]) *Every FO-definable Boolean query is Hanf-local.*

The notion can be extended to non-Boolean queries as well [21] but since most of the time Hanf-locality is applied to prove inexpressibility of sentences, we only present this limited version here.

We now give the canonical example of using Hanf-locality, and prove that graph connectivity is not FO-definable. Assume to the contrary that it is; then it is Hanf-local, so we choose the number r as in the definition of Hanf-locality. Now consider two graphs below, for $m > 2r + 1$.



Let f be an arbitrary bijection between the graphs. The d-neighborhood of *any* node a is the same: it is a chain of length $2r$ with a in the middle. Hence, $G^1 \leftrightarrows_r G^2$, and they must agree on Q, but G^2 is connected, and G^1 is not. Thus, graph connectivity is not FO-definable. A similar example shows that testing whether a graph is a tree is not FO-definable. In that case, we take G_1 to be a chain of length $2m$, and G^2 the disjoint union of a chain of length m and a cycle of length m; then $G^1 \leftrightarrows_r G^2$ as long as $m > 2r + 1$.

It is natural to ask how these notions are related. The precise relationship is known (assuming the definition of Hanf-locality that applies to arbitrary m-ary queries):

Theorem 3.9. ([21]) *Each Hanf-local query is Gaifman-local, and each Gaifman-local query has the BNDP.*

3.5 Other uses of locality

Hanf-locality as defined here can be applied only when example structures have the same cardinality (e.g., the graphs G^1 and G^2 in the picture). Sometimes this is an inconvenient restriction, and a more relaxed notion can be used for graphs of bounded degree. Suppose we are looking at graphs whose in- and out-degrees are bounded by $k \in \mathbb{N}$. Then, for each fixed r, we have finitely many possible isomorphism types of neighborhoods of radius r. We denote this set by $\mathcal{N}(k, r)$. For each graph G and a node a, we say that a realizes $\tau \in \mathcal{N}(k, r)$ if the isomorphism type of $N_r^G(a)$ is τ. Now we write $G \leftrightarrows^*_{m,r} G'$ if for each $\tau \in \mathcal{N}(k, r)$, either

1. both G and G' have the same number of nodes realizing τ; or
2. both G and G' have at least m nodes realizing τ.

Thus, the numbers of nodes realizing τ have to be the same up to threshold m; above the threshold they can be arbitrary. Notice that if we remove the second condition, we get the definition of $G \leftrightarrows_r G'$. The applicability of this notion to FO queries is due to the following.

Theorem 3.10. ([10]) *For each FO sentence φ and $k \in \mathbb{N}$, one can find numbers $m, r \in \mathbb{N}$ so that for every two graphs G, G' with degrees bounded by k, we have $G \models \varphi \Leftrightarrow G' \models \varphi$ whenever $G \leftrightarrows^*_{m,r} G'$.*

This result has an algorithmic application. We say that a class of graphs has *bounded degree* if for some $k \in \mathbb{N}$, all degrees in graphs in that class are bounded by k.

Theorem 3.11. ([40]) *Evaluation of FO queries over classes of graphs of bounded degree can be done with linear-time data complexity.*

The idea is simple: take a query φ, and the bound k on degrees; compute m, r as in Theorem 3.10, and construct $\mathcal{N}(k, r)$. Then enumerate functions $f : \mathcal{N}(k, r) \to \{0, \ldots, m, *\}$, and for each such function decide if a graph in which the number of nodes realizing τ is $f(\tau)$ (with $*$ meaning "above the threshold") satisfies φ. Notice that so far we haven't used the input graph. Now go over the input graph G, compute in linear time the number of nodes realizing each τ, and use the result of the precomputation to see if G satisfies φ.

This result is a starting point of a field called algorithmic model theory, that uses properties of logical formulae on various classes of graphs and other structures to come up with efficient algorithms; see [16] for a survey. We finish this section by a key result on locality often used in such applications. It characterizes precisely what can be expressed in FO. We say that a formula $\varphi(x)$ is *r-local* if all quantification in it is of the form $\exists y \in B_r(x)$ or $\forall y \in B_r(x)$, i.e., restricted to the radius-r ball around x.

Theorem 3.12. ([12]) *Every FO sentence is equivalent to a Boolean combination of sentences of the form*

$$\exists x_1 \ldots \exists x_n \; \Big(\bigwedge_{i=1}^{n} \varphi(x) \wedge \bigwedge_{i \neq j} d(x_i, x_j) > 2r \Big),$$

where $\varphi(x)$ is r-local.

In other words, such a basic sentence asserts the existence of a scattered sequence $x_1, \ldots, x_n$ so that the same formula φ is true in the r-neighborhood of each x_i; and every FO sentence is a Boolean combination of such basic sentences.

3.6 Structures with order

In most database applications, we deal with domains that are totally ordered (e.g., numbers by the usual $<$ relation or strings by the lexicographic ordering). The question is then whether the bounds on the expressive power remain valid. More precisely, we now talk about expressibility over structures of the form $(\mathfrak{A}, <)$, i.e., σ-structures $\mathfrak{A}$ expanded

with a binary relation $<$ interpreted as a linear ordering on the universe. Locality tools, as defined above, don't tell us anything about ordered structures: indeed since for every two elements a, a' we have either $a \leq a'$ or $a' \leq a$, radius-1 neighborhood of every point is the whole structure.

The 'reduction-to-EVEN' trick from Section 3.3 actually shows that many queries (e.g., the transitive closure, connectivity, acyclicity) remain inexpressible in the presence of an order. Indeed, in those reductions, we always started with a linear ordering, and then used connectivity or acyclicity of some constructed graphs to check whether the number of elements in an order was even.

But remarkably, locality tools can be applied after all to structures with an order. Notice that when we talk about connectivity, acyclicity, etc., of structures $(\mathfrak{A}, <)$, we don't actually mention the order: the query asks about the structure $\mathfrak{A}$ itself. Such queries are called invariant: more precisely, a query Q over ordered structures is *invariant* if for every structure $\mathfrak{A}$, every tuple $\bar{a}$, and every two linear orderings $<_1$ and $<_2$ on $\mathfrak{A}$, we have $\bar{a} \in Q((\mathfrak{A}, <_1))$ iff $\bar{a} \in Q((\mathfrak{A}, <_2))$.

For queries definable in FO over arbitrary (finite and infinite) structures, invariance is equivalent to FO-definability *without* the ordering. But for queries over finite structures this is not the case [7, 28]: sometimes invariant queries express more than FO alone can express over $\mathfrak{A}$. But nonetheless, they remain locality. More precisely, if Q is an invariant query, it naturally defines a query Q_{inv} on σ-structures: $Q_{\text{inv}}(\mathfrak{A}) = Q((\mathfrak{A}, <))$ for an arbitrarily chosen ordering $<$. We call such queries Q_{inv} *invariant FO-queries*.

Theorem 3.13. ([17]) *Every invariant FO-query is Gaifman-local.*

In particular, by Theorem 3.9, every invariant FO-query has the BNDP. Hence, the bounds shown in the previous section apply to such queries. But it is still open whether all such queries are Hanf-local. A partial result is known for queries on directed trees [32] (it actually proves a much stronger result that we shall see shortly).

3.7 Inexpressibility tools: 0-1 laws

How diverse are classes of structures definable in FO? Not very, it turns out. Suppose we pick a structure at random. What is the probability that it will satisfy a given FO sentence? Say, your property is $\forall x \forall y\ E(x, y)$. Then it's clear that such graphs are rare among all graphs, and thus the probability will tend to 0. Or we can look at the property $\forall x \forall y \exists z\ E(z, x) \wedge \neg E(z, y)$. Turns out that this property is very common, and the probability tends to 1. More generally, for each FO sentence, exactly one of the two possibilities holds: the probability tends to 0, or to 1.

Let us formalize this. Assume that the universe of an n-element structure $\mathfrak{A}$ is $\{0, \ldots, n-1\}$. Let Str_n^σ be the set of all n-element structures of vocabulary σ, where σ contains only relations. For each property (Boolean query) Q, let

$$\mu_n(Q) \;=\; \frac{|\{\mathfrak{A} \in \text{Str}_n^\sigma \mid Q(\mathfrak{A}) = \text{true}\}|}{|\text{Str}_n^\sigma|}.$$

Another way of looking at it is that $\mu_n(Q)$ is the probability that a randomly chosen structure on the set of nodes $\{0, \ldots, n-1\}$ satisfies Q. Randomly here means with respect to the uniform distribution: each tuple is put in each relation with probability $\frac{1}{2}$. We then define the *asymptotic probability* of Q as

$$\mu(Q) \;=\; \lim_{n \to \infty} \mu_n(Q),$$

if the limit exists.

Definition 3.14. *A logic has the* zero-one law *if for every Boolean query Q expressible in it, either $\mu(Q) = 0$ or $\mu(Q) = 1$.*

Theorem 3.15. ([9]) *FO has the zero-one law.*

Thus, in a sense, FO cannot say anything interesting, at least asymptotically. If we have an FO sentence, we know, with high probability, whether it's true or false. But of course in database querying we aren't interested in asymptotic behavior; rather, we need to know the result of a query in each concrete instance.

The zero-one law gives us easy proofs of inexpressibility of counting properties. Consider again the query EVEN. Since $\mu_n(\text{EVEN})$ alternates between 0 and 1 depending on whether n is even or odd, the limit $\mu(\text{EVEN})$ doesn't exist, and thus EVEN cannot be an FO query. Likewise, if we consider a property D_k which is true iff the size of the structure is divisible by k, then the same argument shows that D_k is not FO-definable for all $k > 1$ (of course EVEN is just D_2).

As a slightly different example consider structures with one unary relation U, and a query PARITY testing whether the size of the set U is even. Then $\mu_n(\text{PARITY}) = (\sum_{i \text{ is even}} \binom{n}{i})/2^n$ and $\mu(\text{PARITY}) = \frac{1}{2}$ – hence PARITY is not FO-definable.

Applying the zero-one law, one has to be careful about vocabularies and interpretations of relations. For instance, with constants, the zero-one law fails (the probability of a loop on a constant is $\frac{1}{2}$). Likewise, it fails in the presence of orders or successor relations (assuming, for example, graphs: the probability that there is an edge between the largest and the smallest element is again $\frac{1}{2}$). Quite remarkably though, the zero-law holds in the presence of a circular successor relation, i.e. a relation $(a_0, a_1), (a_1, a_2), \ldots, (a_n, a_0)$ [30].

4. Beyond FO

Locality tools tell us that FO cannot do fixed-point computations. The zero-one law easily shows that FO cannot do nontrivial counting. Both counting and fixed-point computations play a prominent role in database queries: the former in standard SQL querying that involves grouping and aggregation, the latter – initially in the study of Datalog, and

more recently in extensions of SQL with recursion. We now consider counting and fixed-point extensions of FO. A quick summary of the section is this:

- locality tools continue to work for counting extensions;
- the zero-one law holds for fixed-point extensions;
- but these are only true if we don't add ordering.

4.1 Counting and aggregation

Since SQL has both counting and aggregation, it is natural to study counting extensions of FO to understand its power. Such extensions are achieved by adding *counting quantifiers* or *counting terms*. Let us illustrate this by an example. Suppose we want to express the PARITY query (whether the cardinality of a set U is even). We do it like this:

$$\exists j \exists i \left((i + i = j) \wedge \exists^{\geq j} x\, U(x) \wedge \forall k (\exists^{\geq k} x\, U(x)) \rightarrow k \leq j \right)$$

There are a couple of new things here. First: the quantifiers $\exists^{\geq j} x$ meaning "exists at least j elements x such that...". Second: quantification over numbers ($\exists i$, $\exists j$) and arithmetic subformulae $i + i = j$. Now let us decipher this formula. The subformula $\exists^{\geq j} x\ U(x) \wedge \forall k (\exists^{\geq k} x\ U(x)) \rightarrow k \leq j$ says that there are at least j elements in U, and if there are k elements in U, then $j \geq k$ – that is, $|\, U \,| = j$. The formula $\exists i\ (i + i = j)$ says that j is even. So the whole formula says that $|\, U \,|$ is even.

Technically speaking, this logic is interpreted over *two-sorted* structures of the form $\mathfrak{A} = \langle A, \{0, \ldots, n-1\}, (R^{\mathfrak{A}})_{R \in \sigma}, Arith \rangle$, where, for A of cardinality n, the set $\{0, \ldots, n-1\}$ is the numerical sort, and $Arith$ refers to a set of arithmetic operations over it. The two sorts are connected by quantifiers $\exists^{\geq j} x$ that bind x but not j. One such logic, FO+Cnt, is defined by taking $Arith$ to be $+$ and $\times$ [25]. It was shown in [34] that queries definable in FO+Cnt (i.e., definable by formulae without numerical-sort variables) are Hanf-local (and thus Gaifman-local, and have the BNDP).

So far, the counting abilities of the logic are very limited: the numerical universe is limited to n, the size of A; even though a k-ary relation may have up to n^k tuples, we cannot count beyond n. So it is natural not to put any restrictions on the numerical sort. In other words, we consider the numerical sort to be all of $\mathbb{N}$, add arbitrary arithmetic predicates, and introduce *counting terms* $\#\bar{x}.\varphi(\bar{x}, \cdots)$ which count the number of tuples $\bar{a}$ (over A) satisfying φ. For instance, the PARITY query is expressed by $P_{\text{even}}(\#x.U(x))$, as we assume that all predicates available on $\mathbb{N}$. To check if the number of edges in a graph is a prime number, we can write $P_{\text{prime}}(\#x, y.E(x, y))$.

We call this logic FO + AllCnt. It turns out that even such a powerful counting does not destroy locality.

Theorem 4.1. ([27]) FO + AllCnt *queries are Hanf-local (and thus Gaifman-local, and have the BNDP).*

But even this form of counting is not exactly the aggregates used in SQL that operate over whole columns and can produce rational numbers. To model aggregates, we need to assume relations with columns of two sorts (one sort is numerical), and the addition of *aggregate terms* $t'(\bar{x}) = \mathsf{Aggr}_{\mathcal{F}} \bar{y}.\ (\varphi(\bar{x}, \bar{y}), t'(\bar{x}, \bar{y}))$. Let us explain how it works. First, $\mathcal{F}$ is an aggregate function, i.e., a family of functions $f_0, f_1, f_2, \ldots$ so that each f_n takes an n-element bag of rational numbers and produces a rational number. Given a tuple $\bar{a}$, construct a set $B = \{\bar{b} \mid \varphi(\bar{a}, \bar{b}) \text{ holds}\}$. If $B = \{\bar{b}_1, \ldots, \bar{b}_n\}$, compute the n-element bag of numbers $t'(\bar{a}, \bar{b}_i)$, $i \leq n$, and let $t(\bar{a})$ be f_n applied to that bag[1].

The logic resulting from enhancing FO + AllCnt by changing the domain from $\mathbb{N}$ and $\mathbb{Q}$ and adding *all* aggregate functions is denoted by FO + Aggr. This addition models both the grouping feature of SQL, arbitrary arithmetic and arbitrary aggregation. And yet we have

Theorem 4.2. ([22]) FO + Aggr *queries are Hanf-local (and thus Gaifman-local, and have the BNDP).*

Hence, queries such as graph connectivity and transitive closure remain inexpressible when aggregates are added. But what if we also add ordering? Unfortunately, the situation is no longer as nice as it was for FO: we lose locality [21]. What's more, even the simple logic FO+Cnt captures, over ordered structures, a complexity class that has not yet been separated from NP. Very little is known about counting logics over ordered structures. For instance, [32] considers invariant queries definable in a simple extension of FO with modulo quantifiers $D_k x \varphi(x, \cdot)$ (meaning that the number of elements satisfying φ is divisible by k). Then, over trees, such queries are Hanf-local, as long as all numbers k are odd.

4.2 Fixed-points

Since FO cannot do fixed-point computations, it is natural to add them to query languages – to answer queries such as reachability (transitive closure) or the same-generation. In the theory community, one normally deals with various flavors of Datalog. Some Datalog features (recursive queries) have now been incorporated into SQL. To compute the transitive closure of a binary relation E, one would write in Datalog

$$\begin{array}{lcl} R(x, y) & :- & E(x, y) \\ R(x, y) & :- & E(x, z), R(z, y) \end{array}$$

The computation is by a fixed-point construction. We start with the empty R, and keep applying the rules. After one step, R becomes equal to E, after 2 steps, it has both E and nodes connected by a path of length 2, and so on. The semantics is defined as the least fixed-point of this construction.

There are multiple extensions of FO and other logics with fixed-point operators. These are described in detail in finite model theory texts and we won't give a formal definition here. But we present two important bounds on the expressiveness of Datalog. An obvious bound is that Datalog-expressible

[1]Since the logic doesn't use a range-restriction condition that would be imposed by the syntax of SQL, it is possible that the bag B is infinite. In such a case we just assume that the value of t is 0.

queries are monotone (assuming no negation, of course). A much less obvious one is:

Theorem 4.3. ([2]) *Datalog has the zero-one law.*

Hence, Datalog still cannot express EVEN. But this quickly changes in the presence of an order, or even a successor relation $S(x, y)$ with predicates $\min(x)$ and $\max(x)$ for the smallest and the largest element; indeed

$$\begin{array}{rcl} odd(x) & :- & \min(x) \\ odd(x) & :- & S(y,x), even(y) \\ even(x) & :- & S(y,x), odd(y) \\ \text{EVEN} & :- & \max(x), even(x) \end{array}$$

computes the query EVEN. In fact, on structures with $S, \min$, and $\max$, Datalog with negation (under inflationary semantics) captures PTIME[36], so proving, for example, that an NP-complete query is not expressible in such a flavor of Datalog amounts to separating PTIME from NP.

5. Back to FO

So far, elements populating our structures had no life; we could only compare them for equality, or sometimes for ordering. But imagine that they are, for example, real numbers. If we have a graph, we could ask, for instance, whether its edges (x, y), viewed as points in $\mathbb{R}^2$, lie on the same circle: $\exists a \exists b \exists r \forall x \forall y\ (E(x, y) \rightarrow (x - a)^2 + (y - b)^2 = r^2)$.

This setting was brought to the fore in the context of *constraint databases* [26]. Those were motivated by geographical and temporal databases, in which potentially infinite sets are described finitely by FO formulae over some structures: e.g., a geographical region can be described by its boundary, given as a piecewise polynomial function. Expressibility questions that arose in that setting initially looked different from those that we have considered; for example, a typical question is whether it is possible to express topological connectivity of a region. But it was quickly noticed that most such questions are easily answered if we can answer the usual, finite, expressibility questions [18].

So the model consists of

- an infinite structure $\mathfrak{M}$ (e.g., $\langle \mathbb{N}, +, \cdot \rangle$ or $\langle \mathbb{R}, +, \cdot, 0, 1, < \rangle$);
- a relational vocabulary σ; and
- finite σ-structures over $\mathfrak{M}$ (e.g., graphs whose nodes are numbers).

The logic is the usual FO but with one addition: we have two kind of quantifiers. Quantifiers of the first kind range over the *active domain* (the set of elements of the finite structure). We denote them by $\exists x \in adom$ and $\forall x \in adom$. The universal quantifiers in our "lies on a circle" example are such. Quantifiers of the second kind range over the entire universe of $\mathfrak{M}$. The existential quantifiers in our example are such, since the coordinates of the center of the circle and its radius needn't be in the active domain. We denote this logic by $\mathrm{FO}(\mathfrak{M}, \sigma)$, and its fragment that only uses the quantifiers ranging over the active domain by $\mathrm{FO}_{\mathrm{act}}(\mathfrak{M}, \sigma)$ (note that it is indeed a fragment since quantification over active domain is FO-expressible).

Now, how can we answer questions about the expressiveness of $\mathrm{FO}(\mathfrak{M}, \sigma)$? We have seen many results about the power of what can be viewed in this terminology as $\mathrm{FO}_{\mathrm{act}}(\mathfrak{M}_<, \sigma)$, where $\mathfrak{M}_< = \langle U, < \rangle$ is a structure that only has a linear order available. Indeed, the 'reduction-to-EVEN' trick applies, and invariant queries over it are Gaifman-local.

To get from $\mathrm{FO}(\mathfrak{M}, \sigma)$ to $\mathrm{FO}_{\mathrm{act}}(\mathfrak{M}_<, \sigma)$ one needs to replace an arbitrary structure by an ordering, and arbitrary quantification by finitary quantification. It turns out that the former is easy, but the latter is much harder.

Many queries we are interested in (e.g., is a graph connected?) are *generic*, meaning that they talk about the isomorphism types of finite σ-structures (on the other hand, the "lies on a circle" query is not generic). It is fairly easy to show that if we have an arbitrary structure $\mathfrak{M} = \langle U, <, \ldots \rangle$, then every generic query definable in $\mathrm{FO}(\mathfrak{M}, \sigma)$ is also definable in $\mathrm{FO}(\langle U, < \rangle, \sigma)$ (see [26, 14] for simple expositions of the proof based on Ramsey Theorem).

But the equality $\mathrm{FO}(\mathfrak{M}, \sigma) = \mathrm{FO}_{\mathrm{act}}(\mathfrak{M}, \sigma)$ is *not* true in all structures: for example, $\mathrm{FO}(\langle \mathbb{N}, +, \cdot \rangle, \sigma)$ expresses all computable queries over σ-structures, by coding them in Peano arithmetic. It turns out that whether $\mathrm{FO}(\mathfrak{M}, \sigma) = \mathrm{FO}_{\mathrm{act}}(\mathfrak{M}, \sigma)$ holds, depends on model-theoretic properties of $\mathfrak{M}$. There are many sufficient conditions, which are surveyed in [14, Chap. 5]. Below, we list examples which are important for spatial and temporal applications.

Theorem 5.1. $\mathrm{FO}(\mathfrak{M}, \sigma) = \mathrm{FO}_{\mathrm{act}}(\mathfrak{M}, \sigma)$ *holds if* $\mathfrak{M}$ *is one of the following:*

- $\langle \mathbb{Q}, +, -, 0, 1, < \rangle$ *(rationals, linear arithmetic);*
- $\langle \mathbb{R}, +, -, 0, 1, < \rangle$ *(reals, linear arithmetic);*
- $\langle \mathbb{R}, +, \cdot, 0, 1, < \rangle$ *(reals, polynomial arithmetic);*
- $\langle \mathbb{N}, + \rangle$ *(Presburger arithmetic).*

Hence, for all the structures in Theorem 5.1, generic queries over finite databases over them are already expressible in the usual finite model theory setting: FO over the ordered finite structure alone.

6. Language equivalence

A different set of finite model theory tools was taking a prominent place in database theory research due to the shift from relational model to semi-structured data models, most notably XML. These tools are based on the *composition method* [31], which allows one to compute *types* of structures. The notion of types has a precise meaning, to be defined shortly. These tools are often used in proving equivalence of two languages. To illustrate the need for such equivalence results, consider a simple case of databases defining strings. The choice of strings is not arbitrary: XML documents are modeled as unranked trees,

and strings can be viewed as the simplest possible case of those. Suppose we have a finite alphabet Σ; a string $w = a_0 \dots a_{n-1} \in \Sigma^*$ of length n can be represented as a structure $\mathfrak{A}_w = \langle \{0, \dots, n-1\}, <, (P_a)_{a \in \Sigma} \rangle$, where $<$ is the usual ordering, and each P_a is the set of positions i labeled with a, i.e., $P_a = \{i < n \mid a_i = a\}$.

We consider logics such as FO or MSO (an extension of FO with quantification over sets) over such structures. It is well-known that for each sentence φ, the set of strings w such that $\mathfrak{A}_w \models \varphi$ is regular. So here is one way of evaluating φ: convert it into an automaton $\mathcal{A}_\varphi$ and run this automaton on w. The complexity, in terms of w, is $O(|w|)$, but the problem is that converting from φ to $\mathcal{A}_\varphi$ requires nonelementary complexity. What is more remarkable is that (modulo some complexity-theoretic assumptions), any algorithm for checking whether $\mathfrak{A}_w \models \varphi$ that runs in linear time in $|w|$ will necessarily be nonelementary in the size of φ [11].

Does it mean that FO querying of strings or trees is impossible if we want to achieve linear data complexity? Not really, but we have to change the *syntax* of the logic. In fact, it is well known how to query strings with linear data and query complexity: one uses the linear-time temporal logic LTL. Its syntax is given by

$$\varphi, \psi := a \mid \varphi \vee \psi \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \psi \quad \text{for } a \in \Sigma$$

Such a formula is evaluated in a position i of a string $w = a_0 \dots a_{n-1}$: we have $(w, i) \models a$ iff $a_i = a$; and $(w, i) \models \mathbf{X}\varphi$ iff $(w, i+1) \models \varphi$; and finally $(w, i) \models \varphi \mathbf{U} \psi$ iff for some $j \geq i$ we have $(w, j) \models \psi$ and $(w, k) \models \varphi$ for all k such that $i \leq k < j$. Finally, we write $w \models \varphi$ iff $(w, 0) \models \varphi$.

It is well known that LTL-definable properties of strings are precisely their FO-definable properties. And yet checking whether $w \models \varphi$, for an LTL formula φ, can be done in time $O(|\varphi| \cdot |w|)$. Of course it means that the translation from FO to LTL is necessarily nonelementary, but LTL happens to be a convenient logic for specifying properties of strings.

We shall now outline techniques behind such equivalence results, for strings and trees. Even though XML documents are modeled as *unranked* trees (in which different nodes can have different number of children), here we concentrate on binary trees for the simplicity of exposition, and refer the reader to [29, 39] for surveys about logics and automata over unranked trees.

6.1 Computing with types

The key advantage of well-structured databases (e.g., trees) is that the truth value of a formula on a structure can be computed from truth-values of other formulae on simpler substructures. This is the main idea of the composition method, and the main concept that we need is that of types. For a structure $\mathfrak{A}$, its (FO) *rank-k type* is $\mathrm{tp}_k(\mathfrak{A}) = \{\varphi \mid \mathfrak{A} \models \varphi \text{ and } \mathsf{qr}(\varphi) = k\}$. If formulae φ range over MSO, then we talk about the MSO-rank-k type $\mathrm{mso\text{-}tp}(\mathfrak{A})$. (The notion of quantifier rank for MSO counts the nesting depth of all – first- and second-order – quantifiers.) We can also add free variables, by talking about $\mathrm{tp}_k(\mathfrak{A}, \bar{a})$ as the set of all formulae $\varphi(\bar{x})$ of quantifier rank at most k so that $\mathfrak{A} \models \varphi(\bar{a})$.

It might appear initially that types are infinite objects, but they aren't. In fact, up to logical equivalence, there are only finitely many different FO (or MSO) sentences of quantifier rank k. So if these sentences are $\varphi_1, \dots, \varphi_M$, then types τ are uniquely identified by formulae $\varphi_\tau = \bigwedge_{i \in I} \varphi_i \wedge \bigwedge_{j \notin I} \neg\varphi_j$, specifying completely which φ_i's hold and which don't. Note that $\mathsf{qr}(\varphi_\tau) = \tau$. Also, each sentence of quantifier rank k is a disjunction of sentences defining rank-k types, and $\mathfrak{A} \equiv_k \mathfrak{B}$ iff $\mathrm{tp}_k(\mathfrak{A}) = \mathrm{tp}_k(\mathfrak{B})$.

The latter observation allows us to compute types of structures from types of their substructures using games. As a simple example, consider a structure $\mathfrak{A}_w$ representing a string w and a structure $\mathfrak{A}_{w \cdot a}$ representing w with letter a added at the end. We claim that $\mathrm{tp}_k(\mathfrak{A}_w)$ uniquely determines $\mathrm{tp}_k(\mathfrak{A}_{w \cdot a})$. Indeed, if $\mathfrak{A}_w \equiv_k \mathfrak{A}_{w'}$, then $\mathfrak{A}_{w \cdot a} \equiv_k \mathfrak{A}_{w' \cdot a}$: the strategy of the duplicator is to mimic the winning strategy guaranteeing $\mathfrak{A}_w \equiv_k \mathfrak{A}_{w'}$ if the play happens in (w, w'), and respond with the last letter to such a move by the spoiler. Thus, the duplicator combines the game on $\mathfrak{A}_w$ and $\mathfrak{A}_{w'}$ with a trivial game on two copies of $\mathfrak{A}_a$ to come up with a game on $\mathfrak{A}_{w \cdot a}$ and $\mathfrak{A}_{w' \cdot a}$ – hence the name 'composition method'.

The same argument works for MSO as well (which has a slightly more complicated game allowing set-moves [7, 28]). This gives us the simplest proof that every MSO sentence φ on strings can be converted into an equivalent automaton $\mathcal{A}_\varphi$. Let $\mathsf{qr}(\varphi) = k$ and let $\mathcal{T} = \{\tau_0, \dots, \tau_N\}$ enumerate rank-k MSO types. Let $\delta : \mathcal{T} \times \Sigma \to \mathcal{T}$ be a function such that for each string w with $\mathrm{mso\text{-}tp}_k(\mathfrak{A}_w) = \tau$, we have $\mathrm{mso\text{-}tp}_k(\mathfrak{A}_{w \cdot a}) = \delta(\tau, a)$; by what we have shown above, the function is well-defined. Assume that τ_0 is the type of the empty string, and that φ is equivalent to $\bigvee_{i \in I} \varphi_{\tau_i}$.

Then the automaton $\mathcal{A}_\varphi$ has $\mathcal{T}$ as the set of states, τ_0 as the initial state, δ as the transition function, and $\tau_i, i \in I$, as final states. This automaton is deterministic, and it actually *computes types* as it runs on a string w: the state it is in after reading w is precisely $\mathrm{mso\text{-}tp}_k(\mathfrak{A}_w)$.

6.2 An example: monadic Datalog

We now use the ideas of the previous section and show how to produce a linear-complexity language equivalent to MSO formulae $\varphi(x)$ on both strings and binary trees. Such *unary* queries $\varphi(x)$ select a set of positions in a string or in a tree. If we view trees as XML documents, such queries amount to information extraction: i.e., choosing certain nodes from trees. The technique of this section is from [13, 33].

We start with the easier case of strings. An easy observation – again by composition – is that $\mathrm{mso\text{-}tp}_k(\mathfrak{A}_{a_0 \dots a_{n-1}}, i)$ is uniquely determined by $\mathrm{mso\text{-}tp}_k(\mathfrak{A}_{a_0 \dots a_{i-1}})$, the letter a_i, and $\mathrm{mso\text{-}tp}_k(\mathfrak{A}_{a_{i+1} \dots a_{n-1}})$. Also $\mathrm{mso\text{-}tp}_k(\mathfrak{A}_w)$ uniquely determines $\mathrm{mso\text{-}tp}_k(\mathfrak{A}_{w^{-1}})$, i.e., w read backwards (again by games; the duplicator doesn't in fact need to change his strategy). Hence, we have a function η that takes a triple (τ, τ', a) and computes the $\mathrm{mso\text{-}tp}_k(\mathfrak{A}_{a_0 \dots a_{n-1}}, i)$

under the assumptions that $\text{mso-tp}_k(\mathfrak{A}_{a_0 \ldots a_{i-1}}) = \tau$, $\text{mso-tp}_k(\mathfrak{A}_{a_{n-1} \ldots a_{i+1}}) = \tau'$ and $a_i = a$.

We now use this to come up with a Datalog program that expresses $\varphi(x)$. The extensional predicates are the successor relation $S(\cdot,\cdot)$, labeling $P_a(\cdot)$, and $\text{First}(\cdot)$ and $\text{Last}(\cdot)$ for the first and the last element. For each type τ, we shall have two intensional unary predicates U^τ (true in i if the rank-k type of $a_0 \ldots a_i$ is τ) and V_τ (true in i if the rank-k type of $a_{n-1} \ldots a_i$ is τ). They are computed by:

$$\begin{array}{lcll} U_{\tau_a}(x) & :- & \text{First}(x), P_a(x); & a \in \Sigma \\ U_{\tau'}(x) & :- & S(y,x), P_a(x), U_\tau(y); & a \in \Sigma,\ \delta(\tau, a) = \tau' \\ V_{\tau_a}(x) & :- & \text{Last}(x), P_a(x); & a \in \Sigma \\ U_{\tau'}(x) & :- & S(x,y), P_a(x), V_\tau(y); & a \in \Sigma,\ \delta(\tau, a) = \tau', \end{array}$$

where τ_a is the type of the string a. Next, if the type $\eta(\tau, \tau', a)$ is consistent with φ, we add a rule

$$\text{ANSWER}(x) \ :- \ U_\tau(y), S(y,x), P_a(x), S(x,z), V_{\tau'}(z)$$

to the program. We also need the "boundary" cases when one of the types is τ_0, e.g., $\text{ANSWER}(x) :- \text{First}(x), S(x,z), P_a(x), V_{\tau'}(z)$ if $\eta(\tau_0, a, \tau')$ implies φ.

In the resulting program, every intensional predicate is monadic, i.e., we are dealing with *monadic Datalog*. Thus, over strings, every MSO formula $\varphi(x)$ can be expressed by a monadic Datalog program; the converse is true as well (and in fact is much easier). Hence, monadic Datalog *captures* MSO. Furthermore, monadic Datalog can be evaluated in time linear in both the size of the program and the size of the string [13].

This technique can be extended to unranked trees; in fact, due to its expressivity and low complexity, monadic Datalog has been successfully used for information extraction from XML documents [13]. For simplicity, we present it here for binary trees. A binary tree is viewed as a structure $T = \langle D, S_0, S_1, (P_a)_{a \in \Sigma} \rangle$, where $D \subseteq \{0,1\}^*$ is the domain, i.e. a prefix-closed finite set of strings over $\{0,1\}$ so that for each $s \in D$, either both $s \cdot 0$ and $s \cdot 1$ are in D, or none is in D. In the vocabulary, we have two successor relations $S_0 = \{(s, s \cdot 0)\}$ and $S_1 = \{(s, s \cdot 1)\}$, and the labeling predicates P_a's, as for strings.

Given a tree T and a node s in its domain, by T_s we mean the subtree rooted at s, i.e., the subtree generated by all the nodes that contain s as a prefix. An *envelope* $\text{env}(T, s)$ is obtained by removing T_s from T, except the node s itself, which is viewed as a distinguished node (technically, an envelope is a structure of a vocabulary expanded with a constant symbol, interpreted as s).

Fix $k > 0$. Let $\mathcal{T} = \{\tau_0, \ldots, \tau_N\}$ be the rank-k MSO types of trees, and $\Theta = \{\theta_0, \ldots, \theta_M\}$ be the rank-k MSO types of trees with an extra element (e.g., types of envelopes will come from from this set). The three composition results we need now are as follows:

1. There is a function $\delta : \mathcal{T} \times \mathcal{T} \times \Sigma \to \mathcal{T}$ so that if $\text{mso-tp}_k(T) = \tau$ and $\text{mso-tp}_k(T') = \tau'$, then for the tree T'' whose root is labeled a, the left subtree is T and the right subtree is T', $\text{mso-tp}_k(T'') = \delta(\tau, \tau', a)$.
2. There are functions $\gamma_i : \mathcal{T} \times \Theta \to \Theta$, for $i = 0, 1$, so that for a tree T and a node s whose last letter is i, with $\text{mso-tp}_k(T_s) = \tau$ and $\text{mso-tp}_k(\text{env}(T, s)) = \theta$, we have $\text{mso-tp}_k(\text{env}(T, s \cdot i)) = \gamma_i(\tau, \theta)$.
3. There is a function $\beta : \mathcal{T} \times \Theta \to \Theta$ so that if $\text{mso-tp}_k(T_s) = \tau$ and $\text{mso-tp}_k(\text{env}(T, s)) = \theta$, then $\text{mso-tp}_k(T, s) = \beta(\tau, \theta)$.

Again, these are proved by simple game composition arguments. So now we can start building a monadic Datalog program that evaluates an MSO formula $\varphi(x)$ of quantifier rank k. We assume that the extensional predicates are S_0, S_1, the labeling predicates, as well as unary predicates Leaf and Root. First, we compute types of subtrees using δ:

$$\begin{array}{lcl} U_{\tau_a}(x) & :- & \text{Leaf}(x), P_a(x) \\ U_\tau(x) & :- & S_0(x,x'), S_1(x,x''), P_a(x), U_{\tau'}(x'), U_{\tau''}(x'') \end{array}$$

for all $\delta(\tau', \tau'', a) = \tau$. Here τ_a is the type of the singleton-tree labeled a. Then we handle envelope predicates $V_\theta(\cdot)$'s:

$$\begin{array}{lcl} V_{\theta_a}(x) & :- & \text{Root}(x), P_a(x) \\ V_\theta(x) & :- & S_0(y,x), S_1(y,z), V_{\theta'}(y), U_\tau(z) \end{array}$$

where θ_a is the type of a single-node tree labeled a, and where $\gamma_0(\theta', \tau) = \theta$ (of course we add symmetric rules for γ_1). Finally, for each pair of types τ, θ such that $\beta(\tau, \theta)$ is consistent with φ, we include the rule

$$\text{ANSWER}(x) \ :- \ U_\tau(x), V_\theta(x)$$

Thus, again, the composition technique and computing with types naturally suggested a language that captures MSO, and has good query evaluation properties (both data and query complexity remain linear over trees). In fact most languages with good query evaluation properties over unranked trees have been obtained with the help of the composition method ([29] surveys several of them).

7. Know the complexity

Expressibility – or inexpressibility – of queries in logical formalisms can tell us a lot about their complexity. This is mainly due to the achievements of a field called descriptive complexity, that provides machine-independent characterizations of complexity classes. Also, knowing the number of variables needed to express a query can give us some additional insights, not in terms of the complexity class, but rather the actual big-O complexity of query evaluation.

7.1 Descriptive complexity

The field of descriptive complexity started with a classical result by Fagin that existential second-order logic captures the class NP [8]. It was followed by the characterization of PTIME on ordered structures as the set of properties defined in least-fixed-point logic [23, 42], and then logical characterizations appeared for many more complexity classes (see

[24] for a comprehensive survey). We list below some of the most relevant classes used in database theory research, and their corresponding logics. Note that in all the results below, except NP, coNP, and PH, logics capture complexity classes only over ordered structures.

- AC^0 – the complexity of the relational calculus. This class can be described as constant parallel time (with polynomial many processors). In fact AC^0 is a bit larger than the class of FO-definable properties, as it assumes an ordered universe and basic arithmetic predicates $(+, \cdot)$ over it. Even then, AC^0 is one of very few complexity classes for which nontrivial bounds have been proved: for example, we know that PARITY, graph connectivity, and many other queries are not expressible in it [1].
- TC^0 – constant parallel time with additional, majority, gates (checking whether the number of 1s is more than the number of 0s). The relevance of this class is due to the fact that it is captured, over ordered structures, by $\mathrm{FO}+\mathrm{Cnt}$ (the first counting approximation of SQL). The class is not yet separated from NP, although it is widely believed to be a small subset of PTIME.
- $\mathrm{DLOG}, \mathrm{NLOG}$ – deterministic and nondeterministic logspace. These classes are captured by extensions of FO with transitive closure operators. For DLOG, the transitive closure has to be deterministic, meaning that on a path, every node has to have outdegree 1.
- PTIME – captured by the extension of FO with a least-fixed-point operator (like in Datalog). This can be seen as capturing the complexity of the extension of SQL with recursion.
- $\mathrm{NP}, \mathrm{coNP}, \mathrm{PH}$ (polynomial hierarchy) – these classes appear mostly in static analysis of queries and constraints (queries with such complexity would be prohibitively expensive). They are captured by existential, universal, and full second-order logic respectively. And while we cannot yet separate those, in the *monadic* case (when second-order variables range over sets), we know that all these logics are different [10].
- PSPACE – again this class typically comes up in static analysis problems; it can be characterized via an FO extension with a *partial* fixed-point (i.e., a fixed-point operator that may or may not converge; if it diverges, the result is assumed to be empty).

We have a chain of inclusions $\mathrm{AC}^0 \subsetneq \mathrm{TC}^0 \subseteq \mathrm{DLOG} \subseteq \mathrm{NLOG} \subseteq \mathrm{PTIME} \subseteq \{\mathrm{NP}, \mathrm{coNP}\} \subseteq \mathrm{PH} \subseteq \mathrm{PSPACE}$. All results talk about *data* complexity, i.e., complexity in terms of the size of the structure, and not the formula. One can also gain a lot of additional insight by incorporating the size of the formula as a parameter; we refer to [15] for a survey of the field of parameterized complexity.

7.2 Count the variables

Often, when we write FO formulae, we use variables as we please. But they turn out to be a useful measure of complexity, and it pays off to use them carefully. Let us consider a simple property: a linear ordering has at least 3 elements. A naive way to write this as an FO sentence is $\exists x \exists y \exists z\ (x < y < z)$. But we can also do it like this: $\exists x \exists y\ (x < y \wedge \exists x\ (x = y \wedge \exists y\ (x < y)))$. So we expressed the same property with just 2 variables. In fact, the same trick can show that every cardinality of a linear ordering can be expressed by a sentence using just two variables.

We refer to a fragment of FO that uses a fixed set of variables $x_1, \ldots, x_k$ by FO^k. One of the key reasons this restriction is important is the following:

Theorem 7.1. ([43]) *An FO^k-definable query can be evaluated in time $O(n^k)$.*

Another reason has to do with fixed-point logics. It can be shown that for each fixed-point formula (whether it is least-, or partial-, or some other fixed-point) with k variables can be expressed as a countable disjunction of FO^{2k} formulae. The logic obtained by closing FO^n's by countable disjunctions and conjunctions is well-studied in finite model theory, as it has good properties and provides a uniform treatment of fixed-point logics (for example, it has the zero-one law; it can be used to relate questions about separation of complexity classes to questions about distinguishing logics; and it can be used to show that least-fixed-point logic captures PTIME on some classes of unordered structures; see [5, 7, 28] for more details). It is also known that over arbitrary finite structures, the FO^k hierarchy is strict, and in a remarkable recent paper [38] a long-standing open question about the strictness of the FO^k hierarchy over *ordered* structures was answered positively.

8. Satisfiability questions

These questions often appear in the study of static analysis questions (for example, is a given specification consistent?) or in the study of incompleteness of information (can we answer a query with certainty?)

The general satisfiability question is this: given an FO formula φ, does it have a model (i.e., is φ satisfiable)? In database theory, we are interested primarily in *finite* satisfiability: given φ, does it have a finite model?

In a paper that is often associated with the birth of finite model theory, Trakhtenbrot [41] proved that finite satisfiability is undecidable (even more, it is not co-r.e., unlike arbitrary satisfiability, which is co-r.e.). This result can be used to show that some problems related to answering queries with certainty (say, over incomplete databases, or in data integration scenarios) are undecidable. For example, assume that we have a view V of an unknown database D, and we know that V is obtained from D by losing tuples, and replacing some values with nulls. Now if Q is an FO query, can we find tuples that are guaranteed to belong to $Q(D)$, assuming that we only have access to V? The answer is that this problem is undecidable. Indeed if V is empty, and Q is a Boolean query (a sentence), finding such certain answers

means that φ is valid in all finite structures, i.e., $\neg\varphi$ is not satisfiable. Hence, this is undecidable (not even r.e., in fact).

But for some classes of FO sentences, satisfiability and finite satisfiability coincide, and can be established by means of a *finite model property*: if φ has a model, then it has a model of size $f(|\varphi|)$, for some computable function f.

Two such cases keep coming up in database research. One is the Bernays-Schönfinkel class that consists of formulae $\exists x_1 \ldots \exists x_m \forall y_1 \ldots \forall y_k\, \alpha$, where α is quantifier-free. It turns out that if such a sentence is satisfiable, then it is satisfiable in a structure whose universe has at most m elements. This can also be used to show that satisfiability for this class is NEXPTIME-complete in general, and PSPACE-complete when arities of relations are fixed.

The second important class is FO^2, i.e., FO with two variables. Each satisfiable sentence is again satisfiable in a finite structure, of a larger size, but still giving the overall NEXPTIME-completeness bound. Moreover, this result is rather robust as the finite model property can be extended to classes of structures which are themselves not definable in FO^2, for example, for structures with an extra order, or an extra equivalence relation, or even unranked trees with an extra equivalence relation [3, 35].

Acknowledgments. I thank Pablo Barceló and Ron Fagin for their comments. The author was supported by EPSRC grants E005039 and F028288, the EU grant MEXC-CT-2005-024502 and the FET-Open Project FoX (grant agreement 233599).

9. References

[1] M. Ajtai. Σ^1_1 formulae on finite structures. *Annals of Pure and Applied Logic*, 24 (1983), 1–48.

[2] A. Blass, Y. Gurevich, and D. Kozen. A zero-one law for logic with a fixed-point operator. *Information and Control*, 67 (1985), 70–90.

[3] M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, L. Segoufin. Two-variable logic on data trees and XML reasoning. In *PODS'06*, pages 10-19.

[4] A. Chandra and D. Harel. Computable queries for relational databases. *JCSS*, 21 (1980), 156–178.

[5] A. Dawar, S. Lindell, and S. Weinstein. Infinitary logic and inductive definability over finite structures. *Information and Computation*, 119 (1995), 160–175.

[6] G. Dong, L. Libkin, and L. Wong. Local properties of query languages. *TCS*, 239 (2000), 277–308.

[7] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1995.

[8] R. Fagin. Monadic generalized spectra. *Zeit. Math. Logik und Grund. der Math.*, 21 (1975), 89–96.

[9] R. Fagin. Probabilities on finite models. *J. Symbolic Logic*, 41 (1976), 50–58.

[10] R. Fagin, L. Stockmeyer, and M.Y. Vardi. On monadic NP vs monadic co-NP. *Inf.&Comput.*, 120 (1994), 78–92.

[11] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *IEEE Symp. on Logic in Computer Science*, 2002, pages 215–224.

[12] H. Gaifman. On local and non-local properties, *Proc. Herbrand Symp., Logic Colloquium '81.*

[13] G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for Web information extraction. *J. ACM*, 51 (2004), 74–113.

[14] E. Grädel, Ph. Kolaitis, L. Libkin, M. Marx, J. Spencer, M.Y. Vardi, Y. Venema, S. Weinstein. *Finite Model Theory and its Applications*. Springer-Verlag, 2004.

[15] M. Grohe. Parameterized complexity for the database theorist. *SIGMOD Record*, 31 (2002), 86–96.

[16] M. Grohe. Logic, graphs, and algorithms. In *Logic and Automata – History and Perspectives*, Amsterdam Univ. Press, 2007.

[17] M. Grohe and T. Schwentick. Locality of order-invariant first-order formulas. *ACM TOCL*, 1 (2000), 112–130.

[18] S. Grumbach and J. Su. Queries with arithmetical constraints. *TCS*, 173 (1997), 151–181.

[19] Y. Gurevich. Logic and the challenge of computer science. In *Current trends in theoretical computer science*, E. Börger, ed., Computer Science Press, 1988, pages 1–57.

[20] W. Hanf. Model-theoretic methods in the study of elementary logic. In *The Theory of Models*, North-Holland, 1965, pages 132–145.

[21] L. Hella, L. Libkin, and J. Nurmonen. Notions of locality and their logical characterizations over finite models. *J. Symbolic Logic*, 64 (1999), 1751–1773.

[22] L. Hella, L. Libkin, J. Nurmonen, and L. Wong. Logics with aggregate operators. *J. ACM*, 48 (2001), 880–907.

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

[24] N. Immerman. *Descriptive Complexity*. Springer, 1998.

[25] N. Immerman and E. Lander. Describing graphs: a first order approach to graph canonization. In *Complexity Theory Retrospective*, Springer-Verlag, Berlin, 1990.

[26] G. Kuper, L. Libkin, and J. Paredaens, eds. *Constraint Databases*. Springer, 2000.

[27] L. Libkin. On counting logics and local properties. *ACM TOCL*, 1 (2000), 33–59.

[28] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[29] L. Libkin. Logics over unranked trees: an overview. *LMCS* 2(3): (2006).

[30] J. Lynch. Almost sure theories. *Annals of Mathematical Logic*, 18 (1980), 91–135.

[31] J. Makowsky. Algorithmic aspects of the Feferman-Vaught Theorem. *APAL*, 126 (2004), 159–213.

[32] H. Niemistö. Locality and order-invariant logics. PhD Thesis, Univ. of Helsinki, 2007.

[33] F. Neven, Th. Schwentick. Query automata over finite trees. *TCS* 275 (2002), 633–674.

[34] J. Nurmonen. On winning strategies with unary quantifiers. *J. Logic and Comput.*, 6 (1996), 779–798.

[35] M. Otto. Two variable first-order logic over ordered domains. *J. Symb. Logic* 66 (2001), 685–702.

[36] C. Papadimitriou. A note on the expressive power of Prolog. *Bull. EATCS*, 26 (1985), 21–23.

[37] J. Rosenstein. *Linear Orderings*. Academic Press, 1982.

[38] B. Rossman. On the constant-depth complexity of k-clique. In *STOC'08*, pages 721–730.

[39] T. Schwentick. Automata for XML – a survey. *JCSS* 73 (2007), 289–315.

[40] D. Seese. Linear time computable problems and first-order descriptions. *Math. Struct. Comp. Sci.*, 6 (1996), 505–526.

[41] B. A. Trakhtenbrot. The impossibility of an algorithm for the decision problem for finite models. *Doklady Academii Nauk SSSR*, 70 (1950), 569–572.

[42] M.Y. Vardi. The complexity of relational query languages. In *STOC'82*, pages 137–146.

[43] M.Y. Vardi. On the complexity of bounded-variable queries. In *PODS'95*, pages 266–276.

[44] V. Vianu. Databases and finite-model theory. In *Descriptive Complexity and Finite Models*, Proc. of a DIMACS workshop. AMS, 1997, pages 97–148.