Descriptive Complexity: a Logician's Approach to Computation

Neil Immerman*

Computer Science Dept.
University of Massachusetts
Amherst, MA 01003
immerman@cs.umass.edu

Appeared in *Notices of the American Mathematical Society* 42(10) (1995), 1127 - 1133.

A basic issue in computer science is the complexity of problems. If one is doing a calculation once on a medium-sized input, the simplest algorithm may be the best method to use, even if it is not the fastest. However, when one has a subproblem that will have to be solved millions of times, optimization is important. A fundamental issue in theoretical computer science is the computational complexity of problems. How much time and how much memory space is needed to solve a particular problem? Here are a few examples of such problems:

1. Reachability: Given a directed graph and two specified points s, t, determine if there is a path from s to t. A simple, linear-time algorithm marks s and then continues to mark every vertex at the head of an edge whose tail is marked. When no more vertices can be marked, t is reachable from s iff it has been marked.

2. Min-triangulation: Given a polygon in the plane and a length L, determine if there is a triangulation of the polygon of total length less than or equal to L. Even though there are exponentially many possible triangulations, a dynamic programming algorithm can find an optimal one in $O(n^3)$ steps.

3. Three-Colorability: Given an undirected graph, determine whether its vertices can be colored using three colors with no two adjacent vertices having the same color. Again there are exponentially many possibilities but in this case no known algorithm is faster than exponential time.

*Research supported in part by NSF grant CCR-9207797.

Computational complexity measures how much time and/or memory space is needed as a function of the input size. Let TIME$[t(n)]$ be the set of problems that can be solved by algorithms that perform at most $O(t(n))$ steps for inputs of size n. The complexity class Polynomial Time (P) is the set of problems that are solvable in time at most some polynomial in n.

$$\mathrm{P} \quad = \quad \bigcup_{k=1}^{\infty} \mathrm{TIME}[n^k]$$

Even though the class TIME$[t(n)]$ is sensitive to the exact machine model used in the computations, the class P is quite robust. The problems Reachability and Min-triangulation are elements of P.

Some important computational problems appear to require more than polynomial time. An interesting class of such problems is contained in nondeterministic polynomial time (NP). A nondeterministic computation is one that may make arbitrary choices as it works. If any of these choices lead to an accept state then we say the input is accepted. As an example, let us consider the three-colorability problem. A nondeterministic algorithm traverses the input graph arbitrarily assigning to each vertex a color: red, yellow, or blue. Then it checks whether each edge joins vertices of different colors. If so, it accepts.

A nondeterministic computation can be modeled as a tree whose root is the starting configuration. Every choice forms a branching node. The computation accepts if any of the leaves is an accepting configuration. The nondeterministic time of this computation is the length of the path from root to accepting leaf – not the exponentially larger size of the tree of all possible choices.

Continuing the example, given a graph G with v vertices, the nondeterministic three-colorability algorithm defines a computation tree that has 3^v branches – one for each possible coloring of the vertices. Each such branch ends in an accepting leaf if and only if the corresponding coloring is a valid three coloring. It follows that there exists an accepting leaf just if G is three-colorable. Thus, three-colorability can be checked in nondeterministic $O(n)$ time – where n is the number of vertices plus edges. Three-colorability is in NP.

The three-colorability problem as well as hundreds of other well-known combinatorial problems are NP complete. (See [8] for a survey of many of these.) This means that not only are they in NP, but they are the "hardest problems" in NP: all problems in NP are reducible to each NP-complete problem. (A reduction from problem A to problem B is a polynomial-time mapping from any input to A to an input to B that has the same answer. It follows that if A is reducible to B and B is in P then A is in P.) At present, the fastest known algorithm for any of these problems is exponential. An efficient algorithm for any one of these problems would translate to an efficient algorithm for all of them.

The P =?NP question is an example of our inability to determine what can or cannot be computed in a certain amount of computational resource: time, space, parallel time, etc. The only truly effective tool that we currently have for this is Cantor's diagonalization argument. This is very useful for proving hierarchy theorems: i.e., that more of a given computational resource enables us to compute more.

$$\mathrm{TIME}[n] \underset{\neq}{\subset} \mathrm{TIME}[n^2]; \qquad \mathrm{NTIME}[n] \underset{\neq}{\subset} \mathrm{NTIME}[n^2]; \qquad \mathrm{SPACE}[n] \underset{\neq}{\subset} \mathrm{SPACE}[n^2]$$

However, there are no known techniques for comparing different types of resources, e.g. time versus nondeterministic time, times versus space, etc.

These notions of complexity might not seem fundamental, but rather tied to the sort of machine that the algorithm will be performed on. On the contrary, the notions of time, space, parallel time and even nondeterministic time are fundamental and have many equivalent formulations. One such formulation which I will now describe involves no machines at all, but relies instead on classic notions from mathematical logic.

Complexity theory typically considers yes/no problems: this is the examination of the difficulty of computing a particular bit of the desired output. Yes/no problems are properties of the input: the set of all inputs to which the answer is "yes" have the property in question. Rather than asking the complexity of checking if a certain input has a property T, in Descriptive Complexity we ask how hard is it to express the property T in a formal language. It is plausible that properties that are harder to check might be harder to express. What is surprising is how closely mathematics mimics the physical world: when we use first-order logic, descriptive complexity exactly captures the important complexity classes.

In Descriptive Complexity we view inputs as finite logical structures, e.g., a binary string $w = w_1 w_2 \ldots w_{|w|}$ is coded as

$$\mathcal{A}_w \quad = \quad \langle \{1, 2, \ldots, |w|\}, S^w, \leq \rangle$$

consisting of a universe $U = \{1, 2, \ldots, |w|\}$ of the bit positions in the string, the monadic relation S^w defined so that $S^w(i)$ holds iff the i^{th} bit of w is one, and $\leq$ is the usual total ordering on U.

A graph is a logical structure $\mathcal{A}_G = \langle \{1, 2, \ldots, v\}, E^G \rangle$ whose universe is the set of vertices and E^G is the binary edge relation. A *graph problem* is a set of finite structures whose vocabulary consists of a single binary relation. Similarly, we may think of any problem T in some complexity class $\mathcal{C}$ as a set of structures of some fixed vocabulary.

Recall that in first-order logic we can quantify over the universe. We can say, for example, that a string ends in a one. The following sentence does this by asserting the existence of a string position x that is the last position and asserting $S(x)$, i.e., the bit at that position is a one:

$$(\exists x)(\forall y)(y \leq x \ \wedge \ S(x))$$

As another example we can say that there are exactly two edges leaving every vertex:

$$(\forall x)(\exists y z)(\forall w)(y \neq z \wedge E(x, y) \wedge E(x, z) \wedge (E(x, w) \ \rightarrow \ w = y \vee w = z))$$

For any structure $\mathcal{A}$, we use the notation $\mathcal{A} \models \varphi$ to mean that φ is true in $\mathcal{A}$.

In second-order logic we also have variables X_i that range over relations over the universe. These variables may be quantified.

A second-order existential formula (SO$\exists$) begins with second order existential quantifiers and is followed by a first-order formula. As an example, the following second-order existential sentence, Ψ_3, says that the graph in question is three-colorable. It does this by asserting that there are three unary relations, Red (R), Yellow (Y), and Blue (B), defined on the universe of vertices. It goes on to say that every vertex has some color and no two adjacent vertices have the same color.

$$\begin{aligned}\Psi_3 \quad \equiv \quad & (\exists R)(\exists Y)(\exists B)(\forall x)\Big[(R(x) \vee Y(x) \vee B(x)) \;\wedge\; (\forall y)\Big(E(x,y) \rightarrow \\ & \qquad \neg(R(x) \wedge R(y)) \;\wedge\; \neg(Y(x) \wedge Y(y)) \;\wedge\; \neg(B(x) \wedge B(y))\Big)\Big]\end{aligned}$$

Descriptive Complexity began with the following theorem of Ronald Fagin. Fagin's theorem says that NP is equal to the set of problems describable in second-order, existential logic. Observe that Fagin's Theorem characterizes the complexity class NP purely by logic, with no mention of machines or time.

Theorem 1 ([6]) *A set of structures T is in* NP *iff there exists a second-order existential formula, Φ such that $T = \{\mathcal{A} \mid \mathcal{A} \models \Phi\}$.*

To capture complexity classes P and below, first-order logic is more appropriate. Since first-order logic is weaker than second-order it is natural to let formulas grow with the size of inputs.

For example, one can ask, "How long a first-order formula is needed to express graph connectivity?" A graph G is connected iff $G \models (\forall xy)P(x,y)$ where $P(x,y)$ means that there is a path from x to y. We are interested in paths of length less than $n = |V^G|$. First-order formulas of size $O(\log n)$ are both necessary and sufficient to express connectivity.

In order to characterize complexity classes as in Fagin's theorem but via first-order logics, one must look at the coding of inputs. A graph or other structure is given to a computer in some order, e.g. vertices $v_1, v_2, \ldots, v_n$. Furthermore, algorithms may use the ordering, e.g., searching through each vertex in turn. To simulate the machine, the logical languages need access to the ordering. (This was also necessary for Fagin's theorem, but in SO$\exists$ we may existentially quantify a total ordering on the universe.) From now on, we will assume that all first-order languages include a binary relation symbol "$\leq$" denoting a total ordering on the universe. With this proviso, we can relate computational complexity to first-order descriptive complexity.

Let FO-SIZE$[s(n)]$ be the set of properties expressible by uniform sequences of first-order formulas, $\{\varphi_i\}_{i \in \mathbf{Z}^+}$, such that the n^{th} formula has $O(s(n))$ symbols and expresses the property in question for structures of size n. (Uniform means that the map $n \mapsto \varphi_n$ has very low complexity, e.g., SPACE[$\log n$]. Later we will see that purely syntactic uniformity conditions suffice.)

The following theorem shows that FO-SIZE is a good measure of space. In the following, NSPACE is the nondeterministic version of space in which again we allow algorithms to make arbitrary choices and we only count the amount of space used in an accepting path. In 1970, Walter Savitch proved that NSPACE[$s(n)$] is contained in SPACE[$s(n)^2$], a result that is not obvious [20]. It is not known whether Savitch's theorem is optimal, nor is it known whether SPACE[$s(n)$] is equal to NSPACE[$s(n)$]. The following theorem closely relates the descriptive measure FO-SIZE to space, by fitting it within the bounds of Savitch's theorem:

Theorem 2 ([10]) *For $s(n) \geq \log n$, any problem in nondeterministic space $s(n)$ can be expressed by first-order formulas of size $(s(n))^2/\log n$. Any problem so expressible is in deterministic space $(s(n))^2$. In symbols,*

$$\mathrm{NSPACE}[s(n)] \quad \subseteq \quad \text{FO-}\mathit{SIZE}[(s(n))^2/\log n] \quad \subseteq \quad \mathit{SPACE}[(s(n))^2]$$

One way to increase the power of first-order logic is by allowing inductive definitions. This is formalized via a least fixed point operator (LFP).

As an example, suppose that we want to define the transitive closure of the edge relation of a graph. This would be useful if we were given a directed graph $G = (V^G, E^G, s, t)$ and we wanted to assert that there is a path from s to t. Let $E^\star$ be the reflexive, transitive closure of the edge relation E. Given $E^\star$ we can describe the reachability property simply, "$E^\star(s,t)$". We can define $E^\star$ inductively as follows:

$$E^\star(x,y) \quad \equiv \quad x = y \vee E(x,y) \ \vee \ (\exists z)(E^\star(x,z) \wedge E^\star(z,y)) \tag{1}$$

Equation 1 asserts that $E^\star$ is a fixed point of the following map $R \mapsto \varphi(R)$ of binary relations:

$$\varphi(R,x,y) \quad \equiv \quad E(x,y) \vee (\exists z)(R(x,z) \wedge R(z,y))$$

Since R appears only positively, that is, without any negation signs, in φ, this operator has a least fixed point which we take as the meaning of the inductive definition 1. Thus, we can write,

$$E^\star \quad \equiv \quad (\mathrm{LFP}_{R,x,y}\, \varphi)$$

In fact, if we consider the sequence $\varphi(\emptyset) \subseteq \varphi(\varphi(\emptyset)) \subseteq \varphi(\varphi(\varphi(\emptyset))) \subseteq \cdots$, then LFP($\varphi$) is the union of this sequence. Since the sequence is monotone, and there are at most n^k tuples in a k-ary relation, LFP is in fact a polynomial iterator of formulas. Thus LFP is a particularly natural way to iterate first-order formulas, letting their size grow while the number of variables they use remain constant.[1] The following theorem says that if we add to first-order logic the power to define new relations by induction, then we can express exactly the properties that are checkable in polynomial time. This is exciting because a very natural descriptive class – first-order logic plus inductive definitions – captures a natural and important complexity class. Polynomial time is characterized using only basic logical notions and no mention of computation.

Theorem 3 ([11, 12, 22]) *A problem is in polynomial time iff it is describable in first-order logic with the addition of the least fixed point operator. This is equivalent to being expressible by a first-order formula iterated polynomially many times. In symbols,* $\mathrm{P} \quad = \quad (\mathrm{FO} + \mathrm{LFP}) \quad = \quad \mathrm{FO}[n^{O(1)}]$.

Theorems 1 and 3 cast the P =?NP question in a different light. (In the following we are using the fact that if P were equal to NP, then NP would be closed under complementation. It would then follow that every second-order formula would be equivalent to a second-order existential one.)

Corollary 4 P *is equal to* NP *iff every second-order expressible property over finite, ordered structures is already expressible in first-order logic using inductive definitions. In symbols,*

$$(\mathrm{P} = \mathrm{NP}) \qquad \Leftrightarrow \qquad (\mathrm{FO} + \mathrm{LFP}) = \mathrm{SO}$$

We mention two other natural operators that let us capture important complexity classes. Let $\varphi(x_1, \ldots, x_k, x'_1, \ldots x'_k)$ be a formula with 2k free variables. We can think of φ as representing an edge relation over a vertex set $V = U^k$ consisting of all k-tuples from the original universe. Define the transitive closure operator, writing $(\mathrm{TC}_{\bar{x},\bar{x}'}\varphi)$ to denote the reflexive, transitive closure of the binary relation φ. NSPACE[$\log n$] is an important complexity class that includes most path problems. (The Reachability problem mentioned at the beginning of this article is complete for NSPACE[$\log n$].) The following theorem says that this complexity class is captured by (FO + TC). (Note that transitive closures are a special kind of inductive definition, see Equation 1. Every time we reapply the inductive equation, the paths considered double in length. That is why $(\mathrm{FO} + \mathrm{TC}) \subseteq \mathrm{FO}[\log n]$.)

Theorem 5 ([13, 14]) *The complexity class nondeterministic logarithmic space is equal to the set of problems describable in first-order logic with the addition of a transitive closure operator. This is a subclass of the set of problems describable by first-order formulas iterated* $\log n$ *times. In symbols,*

$$\mathrm{NSPACE}[\log n] \quad = \quad (\mathrm{FO} + \mathrm{TC}) \quad \subseteq \quad \mathrm{FO}[\log n]$$

[1]We write FO[$t(n)$] to denote those properties expressible for structures of size n by formulas that consist of a block of restricted quantifiers repeated $t(n)$ times. What can be expressed with a bounded number of variables is exactly what can be computed with polynomially much "hardware", i.e., polynomial-space and polynomially many processors. The depth of nesting of quantifiers needed to express a property is precisely the parallel time needed to check it. See [15] for a more detailed survey of these results.

Suppose we are given a first-order formula $\varphi(R, x_1, \ldots x_k)$, where R is a new relation variable, but in which R need not occur only positively. Then the least fixed point of φ may not exist. However, we may describe its "partial fixed point" (PFP) which is equal to the first fixed point in the sequence $\varphi(\emptyset), \varphi^2(\emptyset), \ldots$, or $\emptyset$ if there is no such fixed point. Since this sequence must repeat after at most 2^{n^k} steps, PFP may be thought of as an exponential iterator.

The following theorem shows that the arbitrary iteration of first-order formulas – which is the same as iterating them exponentially – allows the description of exactly all properties computable using a polynomial amount of space.

Theorem 6 ([10, 11, 22]) *A problem is in polynomial space iff it is describable in first logic with the addition of the partial fixed point operator. This is equivalent to being expressible by first-order formulas of polynomial size and also equivalent to being expressible by a first-order formula iterated exponentially. In symbols,*

$$\text{PSPACE} \quad = \quad (\text{FO} + \text{PFP}) \quad = \quad \text{FO-}\mathit{SIZE}[n^{O(1)}] \quad = \quad \text{FO}[2^{n^{O(1)}}]$$

Complementation

One of the early successes of descriptive complexity was in response to questions concerning database query languages. A very popular model of databases is the relational model. In this model, databases are exactly finite logical structures. Furthermore, query languages are based on first-order logic.

As an example, suppose that we have an airline database. One of its relations might be FLIGHTS, with arguments: flight number, origin, departure time, destination, arrival time. The query, "What are the direct flights from JFK to LAX leaving in the morning?" could be phrased as:

$$(\exists t_d, t_a)(t_d < 12 \wedge \text{FLIGHTS}(x, \text{JFK}, t_d, \text{LAX}, t_a))$$

Note that this query has one free variable, x. The response to the query should be the set of x's such that x is the flight number of a direct flight from JFK to LAX that leaves before noon.

Of course not all queries that we might want to express are first-order. For example, the reachability query – is there a route, with no fixed limit on the number of hops, taking me from s to t? – is not first-order. For this and related reasons, Chandra and Harel proposed a hierarchy of query languages above first-order logic based on alternating applications of quantification, negation, and the least fixed point operator [3]. It was known that for infinite structures this gave a strict hierarchy [18]; the same was conjectured for finite structures.

There is a big difference between infinite and finite structures. Over an infinite structure, a least fixed point might never reach a stage of the induction at which it has completed. Over a finite structure, there is a finite stage where the fixed point is realized. Furthermore, one can verify within the induction that this stage has been reached. Thus, by saying that we have reached the final stage and that tuple $\bar{t}$ is not present, we can express the negation of $(\mathrm{LFP}\,\varphi)(\bar{t})$. It follows that the "hierarchy" proposed by Chandra and Harel is not a hierarchy at all. Every formula in $(\mathrm{FO} + \mathrm{LFP})$ is expressible as a single fixed point of a first-order formula. We say that the "hierarchy" collapses to its first level.

Theorem 7 ([12]) *Every formula in first-order logic with the addition of the least fixed point operator is equivalent to a single application of least fixed point to a first-order formula.*

A related question could be asked about a proposed hierarchy of $(\mathrm{FO} + \mathrm{TC})$. In fact the transitive closure "hierarchy" also collapses to its first-level.

Theorem 8 ([14]) *Every formula in first-order logic with the addition of the transitive closure operator is equivalent to a single application of transitive closure to a first-order formula.*

Deterministic complexity classes are closed under complementation. It had been long believed in the computer science community that nondeterministic space is a "one-sided" class, not closed under complementation. In particular, it was believed that the complement of the Reachability problem, that is, the set of graphs $G = (V, E, s, t)$ such that there is no path from s to t, was not recognizable in NSPACE$[\log n]$. It is conjectured that SPACE$[\log n]$ is strictly contained in NSPACE$[\log n]$ and the most likely way to prove that appeared to be via showing that NSPACE$[\log n]$ is not closed under complementation [17]. Thus the following corollary of Theorem 8 was quite surprising.

Corollary 9 ([14, 21]) *For $s(n) \geq \log n$,* NSPACE$[s(n)]$ *is closed under complementation.*

Coincidently, Corollary 9, which had been open since 1964, was proved independently by Róbert Szelepcsényi at almost exactly the same time, but using different methods.

Lower Bounds and Ordering

One tantalizing feature of Theorems 1, 2, 3, 5, and 6 is that they hold out the prospect that we can settle questions such as P =?NP via logical methods. In particular, there are quantifier games due to Ehrenfeucht and Fraïssé that characterize the expressive power of logical languages. Ehrenfeucht-Fraïssé games are played on a pair of structures, $\mathcal{A}, \mathcal{B}$. There are two players, the Spoiler and the Duplicator. At each move, the Spoiler chooses some element of the universe of one of the structures and the Duplicator must respond with an element from the other structure. If at the end of the game the map from the elements chosen from $\mathcal{A}$ to those chosen from $\mathcal{B}$ is an isomorphism of the induced substructures then the Duplicator wins, otherwise the Spoiler wins. For most logical languages $\mathcal{L}$ there is a corresponding game $G_{\mathcal{L}}$ with the following fundamental property. Write $\mathcal{A} \equiv_{\mathcal{L}} \mathcal{B}$ to mean that for all $\varphi \in \mathcal{L}$, $\mathcal{A} \models \varphi$ iff $\mathcal{B} \models \varphi$.

$$\text{(Duplicator has a winning strategy for game } G_{\mathcal{L}}(\mathcal{A},\mathcal{B})) \quad \Leftrightarrow \quad \mathcal{A} \equiv_{\mathcal{L}} \mathcal{B} \tag{2}$$

We use Ehrenfeucht-Fraïssé games to prove that certain properties are not expressible in certain logics. If $\mathcal{A}$ and $\mathcal{B}$ disagree on property S and yet we can construct a winning strategy for the Duplicator on $G_{\mathcal{L}}(\mathcal{A},\mathcal{B})$ then we have proved that S is not expressible in $\mathcal{L}$.

SO∃(monadic) is a subset of SO∃ in which the only second-order variables are relations that take one argument. We can thus assert the existence of colorings of the vertices. Fagin used Ehrenfeucht-Fraïssé games to prove that

Theorem 10 ([7]) *Graph Connectivity is not describable by a second-order existential formula in which all relational variables are monadic.*

However, non-connectivity is in SO∃(monadic). The following formula says that there exists a set C of vertices that is not empty and not the whole universe that is closed under edge connections. Thus Theorem 10 implies that SO∃(monadic) is not closed under complementation.

$$\text{Not-Connected} \;\equiv\; (\exists C)(\exists xy)(C(x) \wedge \neg C(y) \wedge (\forall xy)(C(x) \wedge E(x,y)) \rightarrow C(y))$$

Proving lower bounds on SO∃ when relational variables need not be monadic appears hard. Lower bounds for first-order logic may be more tractable. A lower bound corresponding to Theorem 2 was presented for the AGAP problem. AGAP is a generalization of the graph reachability problem to and/or graphs and is complete for P. The following theorem proves a lower bound on the size of first-order formulas needed to express the AGAP property.

Theorem 11 ([10]) *The AGAP property is describable by linear-size first-order formulas that do not include the ordering relation. However, it is not describable by such formulas of size less than* $2^{\sqrt{\log n}}$

The relationship between space and time is not well understood. We do know that

$$\text{NSPACE}[\log n] \quad \subseteq \quad \text{P} \quad \subseteq \quad \bigcup_{k=1}^{\infty} \text{SPACE}[n^k]$$

From Theorem 2 we know that NSPACE[$\log n$] $\subseteq$ FO-SIZE[$\log n$]. Theorem 11 is a lower bound on languages without ordering. If Theorem 11 could be shown with ordering it would follow that AGAP is not in FO-SIZE[$\log n$] and thus that P strictly contains NSPACE[$\log n$].[2]

The Ehrenfeucht-Fraïssé games become much less useful as a proof technique when working with ordered graphs. This is because in a fairly weak ordered language we can express the property of a vertex v_i that it is the i^{th} vertex in the ordering. Once a language has this strength, two structures will be equivalent iff they are identical. (If we can assert about a graph G that vertex 17 has an edge to vertex 289, then any graph that agrees on all such sentences is identical to G.)

On the other hand, if we just remove the ordering, then Theorems 2, 3, 5, and 6 all fail. For example, it is easy to show that without an ordering we cannot count. In fact, if EVEN represents the query, "The size of the universe is even," then:

Theorem 12 ([3]) *In the absence of the ordering relation, first-order logic with the addition of the fixed point operator cannot describe the problem EVEN.*

All the graph properties that we want to express are order independent. Thus, it would be very nice to have a language that captures all polynomial-time computable order-independent properties. This would be analogous to Theorem 3 which says that (FO + LFP) captures polynomial time for ordered structures.

Before 1989, examples involving the counting of large, unstructured sets were the only problems known to be in order-independent P but not in (FO(wo$\leq$)+LFP). Consider the language (FO(wo$\leq$) + LFP + COUNT) in which structures are two-sorted: their universe is partitioned into an unordered domain $D = \{d_1, d_2, \ldots, d_n\}$ and a separate number domain: $N = \{1, 2, \ldots, n\}$. We have the database predicates defined on D and the standard ordering defined on N. The two sorts are combined via counting quantifiers:

$$(\exists i\, x)\varphi(x)$$

meaning that there exist at least i elements x such that $\varphi(x)$. Here i is a number variable and x is a domain variable.

For quite a while, it was an open question whether the language (FO(wo$\leq$) + LFP + COUNT) was equal to order independent P.

Instead, in [2] it was proved that (FO(wo$\leq$)+LFP+COUNT) is strictly contained in order-independent P. In the following lower bound, the graphs are "almost ordered." They consist of $n/4$ groups of 4 vertices each and there is a given total ordering on the groups.

[2]In fact, it would follow that no P-complete problem is in SPACE[$(\log n)^k$] for any k. From this, we could conclude that P-complete problems do not admit the kind of extensive speed-up via parallelism that problems such as reachability, matrix multiplication, and matrix inversion do [15].

Theorem 13 ([2]) *There is an order-independent property of graphs, T, that is very easy to compute – in a complexity class well below* P *– but, in the absence of ordering, is not expressible in first-order logic with the addition of the fixed point operator and counting quantifiers.*

Most important complexity classes below P have had their languages without ordering, or with some partial orderings, separated [10, 11, 9, 5, 4]. No such separation had been proved for the languages $(\mathrm{FO}(\mathrm{wo}{\leq}) + \mathrm{LFP})$ and $(\mathrm{FO}(\mathrm{wo}{\leq}) + \mathrm{PFP})$. In 1991 Abiteboul and Vianu explained why by proving

Theorem 14 ([1]) *The following conditions are equivalent:*

1. *In the absence of ordering, first-order logic plus the least fixed point operator describes all the properties describable by first-order logic plus the partial fixed point operator.*
2. *In the presence of ordering, first-order logic plus the least fixed point operator describes all the properties describable by first-order logic plus the partial fixed point operator.*
3. *P = PSPACE*

Theorem 14 is proved by showing that if two structures G and H are $(\mathrm{FO}(\mathrm{wo}{\leq}){+}\mathrm{LFP})$-equivalent, then they are $(\mathrm{FO}(\mathrm{wo}{\leq}) + \mathrm{PFP})$-equivalent as well. Thus, ordering is not the problem, but Ehrenfeucht-Fraïssé games won't help separate P from PSPACE.

Descriptive complexity reveals a simple but elegant view of computation. Nonetheless, the basic problems of how to compare different computational resources remain mysterious.

References

[1] S. Abiteboul and V. Vianu, "Generic Computation And Its Complexity," *23rd ACM STOC* (1991), 209-219.

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

[3] A. Chandra and D. Harel, "Structure and Complexity of Relational Queries," *21st Symp. on Foundations of Computer Science,* 1980, (333-347). Also appeared in a revised form in *JCSS* **25**, 1982, (99-128).

[4] K. Etessami and N. Immerman, "Tree Canonization and Transitive Closure," *IEEE Symp. Logic In Comput. Sci.* (1995), 331-341.

[5] K. Etessami and N. Immerman, "Reachability and the Power of Local Ordering," *Eleventh Symp. Theoretical Aspects Comp. Sci.* (1994), 123 - 135.

[6] R. Fagin, "Generalized First-Order Spectra and Polynomial-Time Recognizable Sets," in *Complexity of Computation,* (ed. R. Karp), *SIAM-AMS Proc. 7,* 1974, (27-41).

[7] R. Fagin, "Monadic generalized spectra," *Zeitschr. f. math. Logik und Grundlagen d. Math.* **21** (1975), 89-96.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability,* Freeman, 1979.

[9] E. Grädel and G. McColm, "On the Power of Deterministic Transitive Closures," *Information and Computation* (119:1) (1995), 129-135.

[10] N. Immerman, "Number of Quantifiers is Better than Number of Tape Cells," *JCSS* (22:3) (1981), 65-72.

[11] N. Immerman, "Upper and Lower Bounds for First Order Expressibility," *JCSS* **25**, No. 1 (1982), 76-98.

[12] N. Immerman, "Relational Queries Computable in Polynomial Time," *Information and Control,* 68 (1986), 86-104. A preliminary version of this paper appeared in *14th ACM STOC Symp.,* (1982), 147-152.

[13] N. Immerman, "Languages That Capture Complexity Classes," *SIAM J. Comput.* **16**, No. 4 (1987), 760-778. A preliminary version of this paper appeared in *15th ACM STOC Symp.,* (1983) 347-354.

[14] N. Immerman, "Nondeterministic Space is Closed Under Complementation," *SIAM J. Comput.* **17**, No. 5 (1988), 935-938. Also appeared in *Third Structure in Complexity Theory Conf.* (1988), 112-115.

[15] N. Immerman, "Descriptive and Computational Complexity," in *Computational Complexity Theory,* ed. J. Hartmanis, *Proc. Symp. in Applied Math.,* 38, American Mathematical Society (1989), 75-91.

[16] N. Immerman, "DSPACE[n^k] = VAR[$k+1$]," *Sixth IEEE Structure in Complexity Theory Symp.* (1991), 334-340.

[17] S.Y. Kuroda, "Classes of Languages and Linear-Bounded Automata," *Information and Control* **7** (1964), 207-233.

[18] Yiannis N. Moschovakis, *Elementary Induction on Abstract Structures,* North Holland, 1974.

[19] S. Patnaik and N. Immerman, "Dyn-FO: A Parallel, Dynamic Complexity Class," *ACM Symp. Principles Database Systems* (1994), 210-221.

[20] W.J.Savitch, "Relationships Between Nondeterministic and Deterministic Tape Complexities," *J. Comput. System Sci.* **4** (1970), 177-192.

[21] R. Szelepcsényi, "The Method of Forced Enumeration for Nondeterministic Automata," *Acta Informatica* **26** (1988), 279-284.

[22] M. Vardi, "Complexity of Relational Query Languages," *14th ACM STOC Symp.,* (1982), 137-146.