The Computational Complexity Column

Eric Allender
Rutgers University, Department of Computer Science
Piscataway, NJ 08854-8019 USA
allender@cs.rutgers.edu

Exploration of the connections between computational complexity, descriptive complexity, and logic remains one of the most active and important areas of theoretical computer science. In this edition of the Computational Complexity Column, we hear from one of the leaders in this area.

Progress in Descriptive Complexity

Neil Immerman[1]

1. Introduction

In the beginning, there were two measures of computational complexity: time and space. From an engineering standpoint, these were very natural measures, quantifying the amount of physical resources needed to perform a computation. From a mathematical viewpoint, time and space were somewhat less satisfying, since neither appeared to be tied to the inherent mathematical complexity of the computational problem.

In 1974, Ron Fagin changed this. He showed that the complexity class NP — those problems computable in nondeterministic polynomial time — is exactly the set of problems describable in second-order existential logic. This was a remarkable insight, for it demonstrated that the computational complexity of a problem can be understood as the richness of a language needed to specify the problem. Time and space are not model-dependent engineering concepts, they are more fundamental.

Although few programmers consider their work in this way, a computer program is a completely precise description of a mapping from inputs to outputs. We follow database terminology and call such a map a *query* from input structures to output structures. Typically a program describes a precise sequence of steps that compute a given query. However, we may choose to describe the query in some other precise way. For example, we may describe queries in variants of first- and second-order mathematical logic.

Fagin's Theorem gave the first such connection. Using first-order languages, this approach, commonly called descriptive complexity, demonstrated that virtually all measures of complexity can be mirrored in logic. Furthermore, as we will see, the most important classes have especially elegant and clean descriptive characterizations.

In this column I will survey a few of the connections between descriptive and computational complexity. I will also describe some recent progress and applications. Further detail can be found in [**12**].

[1]Comp. Sci. Dept., UMass, Amherst 01003, USA, immerman@cs.umass.edu, http://www.cs.umass.edu/∼immerman. Research supported by NSF CCR-9505446

2. Notation and Logical Background

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

$$K_w \quad = \quad \langle \{1, 2, \ldots , |w|\}, M^w, \leq \rangle$$

consisting of a universe $|K_w| = \{1, 2, \ldots , |w|\}$ of bit positions in the string, the monadic relation M^w defined so that $M^w(i)$ holds iff bit i of w is one, and $\leq$ is the usual total ordering on $|K_w|$. We write $\|K_w\|$ to denote the cardinality of the universe of structure K_w.

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 $M(x)$, i.e., the bit at that position is a one:

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

A graph is a logical structure $G = \langle \{1, 2, \ldots , v\}, E^G \rangle$ whose universe is the set of vertices and E^G is the binary edge relation. This first-order formula says that every vertex exactly two outgoing edges:

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

We next show that addition of natural numbers is first-order expressible. We will see that the first-order queries characterize the problems computable in constant parallel time. Thus the following may be thought of as an addition algorithm that runs in constant parallel time.

Proposition 1. *Addition of natural numbers, represented in binary, is first-order expressible.*

Proof. We use the well-known "carry-look-ahead" algorithm. In order to express addition, we first express the carry bit,

$$\varphi_{carry}(x) \equiv (\exists y . x < y)[A(y) \wedge B(y) \wedge (\forall z . x < z < y) A(z) \vee B(z)]$$

The formula $\varphi_{carry}(x)$ holds if there is a position y to the right of x where $A(y)$ and $B(y)$ are both one (i.e. the carry is generated) and for all intervening positions z, at least one of $A(z)$ and $B(z)$ holds (that is, the carry is propagated). Let $\oplus$ be an abbreviation for the commutative and associative "exclusive or" operation.

We can express φ_{add} as follows,

$$\begin{aligned} \alpha \oplus \beta \quad &\equiv \quad \alpha \leftrightarrow \neg \beta \\ \varphi_{add}(x) \quad &\equiv \quad A(x) \oplus B(x) \oplus \varphi_{carry}(x) \end{aligned}$$

Note that the formula $\varphi_{add}(x)$ has the free variable x. Thus, φ_{add} is a description of n bits: one for each possible value of x. $\square$

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, 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.

$$(\exists R)(\exists Y)(\exists B)(\forall x)\Big[\big(R(x) \vee Y(x) \vee B(x)\big) \;\wedge\; (\forall y)\Big(E(x,y) \;\rightarrow \\ \neg\big(R(x) \wedge R(y)\big) \;\wedge\; \neg\big(Y(x) \wedge Y(y)\big) \;\wedge\; \neg\big(B(x) \wedge B(y)\big)\Big)\Big]$$

A *query* is a polynomially bounded mapping from the set of finite structures of one vocabulary to those of another vocabulary. For example, the formula φ_{add} from Proposition 1 defines a query from pairs of strings to strings. A *boolean query* is a set of finite structures of some vocabulary. A complexity class is a set of boolean queries: those queries that can be recognized in a given amount of computation. Descriptive Complexity began with Fagin's descriptive characterization of the complexity class NP.

THEOREM 2 ([**7**]). *A boolean query T is in* NP *iff there exists a second-order existential formula, Φ such that $T = \{K \mid K \models \Phi\}$.*

To capture complexity classes P and below, first-order logic is more appropriate. 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.

We will see that the set of first-order definable boolean queries is a weak complexity class. 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:

$$E^\star(x,y) \quad \equiv \quad x = y \vee E(x,y) \ \vee \ (\exists z)(E^\star(x,z) \wedge E^\star(z,y)) \tag{3}$$

Consider the following map $\varphi : R \mapsto \varphi(R)$ on binary relations:

$$\varphi(R,x,y) \quad \equiv \quad E(x,y) \vee (\exists z)(R(x,z) \wedge R(z,y)) \tag{4}$$

Since R appears only positively in φ, this operator has a least fixed point which we take as the meaning of the inductive definition (3). Thus, we can write, $E^\star \ \equiv \ (\mathrm{LFP}_{R,x,y}\,\varphi)$.

$\mathrm{LFP}(\varphi)$ is the union of the sequence: $\varphi(\emptyset) \subseteq \varphi(\varphi(\emptyset)) \subseteq \varphi(\varphi(\varphi(\emptyset))) \subseteq \cdots$. 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.

We can transform formulas such as φ (4) into a block of restricted quantifiers. Iterating φ then just corresponds to writing another copy of this quantifier block. We write $\mathrm{FO}[t(n)]$ to denote those properties expressible for structures of size n by formulas that consist of a block of restricted quantifiers repeated $t(n)$ times. Such a formula no matter what the length has a fixed number of variables — the same variables that appear in φ.

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 5 ([**14, 15, 24**]). *A query 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,* $\mathrm{P} \ = \ \mathrm{FO(LFP)} \ = \ \mathrm{FO}[n^{O(1)}]$.

Theorems 2 and 5 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 6. 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,* $(\mathrm{P} = \mathrm{NP}) \ \Leftrightarrow \ \mathrm{FO(LFP)} = \mathrm{SO}$.

We mention two other natural operators that let us capture important complexity classes. Let $\varphi(x_1, \ldots, x_r, x'_1, \ldots x'_r)$ be a formula with 2r free variables. We can think of φ as representing an edge relation over a vertex set $V = |K|^r$ consisting of all r-tuples from the universe of K. Define the transitive closure operator, $(\mathrm{TC}_{\bar{x},\bar{x}'}\,\varphi)$, denoting the reflexive, transitive closure of the binary relation φ. $\mathrm{NSPACE}[\log n]$ is an important complexity class that includes most path queries. 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 3. Every time we reapply the inductive equation, the paths considered double in length. That is why FO(TC) $\subseteq$ FO[$\log n$].)

THEOREM 7 ([**16, 17**]). *The complexity class nondeterministic logarithmic space is equal to the set of boolean queries describable in first-order logic with the addition of a transitive closure operator. This is a subclass of the set of queries describable by first-order formulas iterated* $\log n$ *times. In symbols,*

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

Suppose we are given a first-order formula $\varphi(R, x_1, \dots x_r)$, 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 $\varphi(\emptyset), \varphi^2(\emptyset), \dots,$ or $\emptyset$ if there is no such fixed point. Since this sequence must repeat after at most 2^{n^r} 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 8 ([**13, 14, 24**]). *A query 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 a first-order formula iterated exponentially, or by a second-order formula iterated polynomially. In symbols,*

$$\mathrm{PSPACE} \; = \; \mathrm{FO(PFP)} \; = \; \mathrm{FO}[2^{n^{O(1)}}] \; = \; \mathrm{SO(TC)} \; = \; \mathrm{SO}[n^{O(1)}]$$

3. Parallel Complexity

Quantification is a parallel operation. The query $(\exists x)M(x)$ can be executed using n processors in constant parallel time. Processor p_i checks whether $M(i)$ holds for $i = 0, 1, \dots, n-1$. Any p_i for which $M(i)$ does hold should write a one into a specified location in global memory that was originally zero.

A *concurrent random access machine* (CRAM) consists of a large number of processors, all connected to a common, global memory. The processors are identical except that they each contain a unique processor number. At each step, any number of processors may read or write any word of global memory. If several processors try to write the same word at the same time, then the lowest numbered processor succeeds.

The CRAM is a special case of the concurrent read, concurrent write, parallel random access machine (CRCW-PRAM). Let CRAM[$t(n)$] be the set of queries computable in parallel time $O(t(n)$ by a CRAM using at most polynomially many processors.

THEOREM 9. ([**18**]) *For all polynomially bounded, constructible $t(n)$,*

$$\mathrm{FO}[t(n)] \quad = \quad \mathrm{CRAM}[t(n)] \; .$$

This means that the quantifier depth of an optimal description of a query corresponds exactly to the parallel time needed to compute the query, using polynomially much hardware. Parallel time — one of the most fundamental computational resources — is quantifier depth.

The amount of hardware needed to compute a query is closely tied to the number of variables needed to describe the query. When the amount of hardware is measured purely as space, this result is tight.

THEOREM 10. [**20**] *For $k = 1, 2, \ldots$,* $\quad \mathrm{DSPACE}[n^k] = \mathrm{VAR}[k+1]$.

The fundamental mysteries of complexity theory are the tradeoffs between parallel time and hardware. A consequence of Theorems 9 and 10 is that these mysteries can all be understood as the descriptive tradeoff between quantifier depth and number of variables.

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:

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

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 [**4**]. It was known that for infinite structures this gave a strict hierarchy [**22**]; 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 FO(LFP) is expressible as a single fixed point of a first-order formula. We say that the "hierarchy" collapses to its first level.

THEOREM 11 ([**15**]). *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 FO(TC). In fact the transitive closure "hierarchy" also collapses to its first-level.

THEOREM 12 ([**17**]). *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 that nondeterministic space is a "one-sided" class, not closed under complementation. Thus the following corollary of Theorem 12 was quite surprising.

COROLLARY 13 ([**17, 23**]). *For* $s(n) \geq \log n$*,* NSPACE$[s(n)]$ *is closed under complementation.*

4. Lower Bounds and Ordering

Theorems 2, 5, 7, and 8 hold out the prospect that we might 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, A, B. There are two players, Samson and Delilah. At each move, Samson chooses some element of the universe of one of the structures and Delilah must respond with an element from the other structure. If at the end of the game the map from the elements chosen from A to those chosen from B is an isomorphism of the induced substructures then Delilah wins, otherwise Samson wins. For most logical languages L there is a corresponding game G_L with the following fundamental property. Write $A \equiv_L B$ to mean that for all $\varphi \in L$, $A \models \varphi$ iff $B \models \varphi$.

$$\text{(Delilah has a winning strategy for game } G_L(A,B)) \quad \Leftrightarrow \quad A \equiv_L B \tag{14}$$

We use Ehrenfeucht-Fraïssé games to prove that certain properties are not expressible in certain logics. If A and B disagree on property Φ and yet we can construct a winning strategy for Delilah on $G_L(A,B)$ then we have proved that Φ is not expressible in L.

Consider the following lower bound on quantifier-depth for the reachability query for and-or graphs (REACH$_a$), a natural P-complete query.

THEOREM 15 ([**13**]). REACH$_a$ *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* $2^{\sqrt{\log n}}$*.*

If Theorem 15 could be shown with ordering it would follow that REACH_a is not in $\mathrm{FO}[(\log n)]$ and thus that P strictly contains $\mathrm{NSPACE}[\log n]$.

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 vertex i 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 5, 7, and 8 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 16 ([**4**]). *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 5 which says that FO(LFP) captures polynomial time for ordered structures.

Consider the language $\mathrm{FO}(\mathrm{wo}{\leq})(\mathrm{LFP}, C)$ 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.

It was conjectured that $\mathrm{FO}(\mathrm{wo}{\leq})(\mathrm{LFP}, C)$ is equal to order independent P. Instead, in [**3**] it was proved that $\mathrm{FO}(\mathrm{wo}{\leq})(\mathrm{LFP}, C)$ 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.

Theorem 17 ([**3**]). *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.*

Some beautiful work by Martin Grohe and his students has shown that there are large and important classes of graphs on which the language $\mathrm{FO}(\mathrm{wo}{\leq})(\mathrm{LFP}, C)$ does capture order independent P [**11, 8, 10**]. For these classes of graphs, isomorphism is testable in polynomial time using the natural algorithm for equivalence in a sublanguage of $\mathrm{FO}(\mathrm{wo}{\leq})(\mathrm{LFP}, C)$. These results include natural classes of graphs for which no polynomial-time graph isomorphism algorithm had been previously known.

Most important complexity classes below P have had their languages without ordering, or with some partial orderings, separated [**13, 14, 9, 6, 5**]. No such separation had been proved for the languages FO(wo$\le$)(LFP) and FO(wo$\le$)(PFP). In 1991 Abiteboul and Vianu explained why by proving

THEOREM 18 ([**1**]). *The following conditions are equivalent:*

1. FO(wo$\le$)(LFP) $=$ FO(wo$\le$)(PFP)
2. FO(LFP) $=$ FO(PFP)
3. $P = PSPACE$

Theorem 18 is proved by showing that if two structures G and H are FO(wo$\le$)(LFP)-equivalent, then they are FO(wo$\le$)(PFP)-equivalent as well. Thus, ordering is not the problem, but Ehrenfeucht-Fraïssé games are unlikely to help separate P from PSPACE.

5. Model Checking

Descriptive Complexity has been applied recently to computer-aided verification. When we design a program, distributed protocol, or circuit, we do so in a formal language. This might be a programming language such as Java or a circuit design language such as VHDL, or a software design language such as Statecharts.

Such a formal design determines a logical structure, $K = \langle S, p_1, \dots, p_k, R\rangle$, called a Kripke structure or transition system. Here S consists of the set of possible global states of the design, the set of all possible gate values for the circuit, or all possible assignments of values to the variables of a program, or all possible assignments of states to each processor in the distributed protocol. Binary relation R is the next move relation: $R(s, s')$ means that the move from s to s' is a possible atomic transition of the circuit or program.

The size of S is often exponential in the size of the design. This phenomenon is called the *state explosion problem.* For this reason, structure K is often represented symbolically.

Once we have our formal design, a representation of the transition system can be automatically generated. We may now wish to write some simple correctness conditions in a formal language. For example, we might want to express the following:

1. If the Restart button is pressed, we eventually restart.
2. Doors are not opened between stations.
3. Division is performed correctly.
4. The motor is not turned on during maintenance.

These conditions express the fact that (1) if we press the Restart button on a personal computer it will eventually restart — without our having to unplug it and plug it in again; (2) the subway train controller meets some simple safety conditions; (3) Intel's new processor does its arithmetic correctly; and (4) the mixer in a nuclear waste storage container (which is designed to keep its contents mixed and thus less volatile) cannot turn on (and thus cause injury) during maintenance.

Now that we have formally expressed our design and our statement concerning its behavior, we can simply press a button and be told whether or not $K \models \varphi$, that is, does the design satisfy the desired property?

This is the model checking problem: given K and φ, test whether K satisfies φ. Model checkers are currently being hailed as great debugging aides. This they are. We can write a simple property that our system should satisfy and press the button. If it does not satisfy the property then we will be told so. Typical model checking programs will present a counter-example run of the system, i.e., they produce the explicit bug. If the model does satisfy the property, then we will be told so. Note that this is better than just not finding a bug. Our model checker has automatically proved that our design has a certain desirable property.

The connection between model checking and descriptive complexity is clear. Since the models involved are huge, it is useful to write queries in languages that only express feasible queries. A popular and quite expressive temporal logic for model checking is CTL*. The model checking problem for CTL* is PSPACE complete and that is PSPACE in the size of the huge Kripke structure. However, it was shown in [**21**] that CTL* can be embedded in linear time in the simple language FO^2(TC) — the sublanguage of FO(TC) in which there are at most two domain variables. The catch is that for very complex formulas in CTL*, the translation might also include a linear number of boolean variables. However, in practice, most queries that people seem to want to write involve zero or very close to zero booleans.

In [**2**], it is shown that the embedding of CTL* in FO^2(TC) lands in a fragment that can be efficiently model checked in time $O(\|K\||\varphi|2^{n_b})$: linear in the size of the structure and the formula, but exponential in the number of booleans in the formula.

This is useful, both because n_b tends to be tiny, and because the language involved is closely tied to reachability queries which are the bread and butter of model checking. One nice feature is that we can look at the formula, count the number of booleans, and automatically say whether the query can be checked efficiently or not.

In summary, descriptive complexity reveals a simple and elegant view of computation. New insights and even practical algorithms have resulted from this approach.

References

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

[2] N. Alechina and N. Immerman, "Efficient Fragment of Transitive Closure Logic," manuscript, 1999.

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

[4] 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).

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

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

[7] 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).

[8] M. Frick and M. Grohe, "Checking First-Order Properties of Locally Tree-Decomposable Graphs," (1999).

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

[10] M Grohe and J. Mariño, "Definability and Descriptive Complexity on Databases of Bounded Tree-Width," to appear in *Intl. Conf. on Database Theory*(1999)

[11] M. Grohe, "Finite Variable Logics in Descriptive Complexity Theory," *Bulletin of Symbolic Logic* , 4(4) (1998), 345- 398.

[12] N. Immerman, *Descriptive Complexity*, 1998, Springer Graduate Texts in Computer Science, New York.

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

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

[15] N. Immerman, "Relational Queries Computable in Polynomial Time," *Information and Control,* 68 (1986), 86-104.

[16] N. Immerman, "Languages That Capture Complexity Classes," *SIAM J. Comput.* **16**, No. 4 (1987), 760-778.

[17] N. Immerman, "Nondeterministic Space is Closed Under Complementation," *SIAM J. Comput.* **17**, No. 5 (1988), 935-938.

[18] N. Immerman, "Expressibility and Parallel Complexity," *SIAM J. of Comput.* 18 (1989), 625-638.

[19] 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.

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

[21] N. Immerman and M. Vardi. Model Checking and Transitive Closure Logic. *Proc. 9th Int'l Conf. on Computer-Aided Verification* (CAV'97), Lecture Notes in Computer Science, Springer-Verlag 291 - 302, 1997.

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

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

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