Experimental Descriptive Complexity

Marco Carmosino[1,⋆], Neil Immerman[1,⋆], and Charles Jordan[2,⋆⋆]

[1] Computer Science Dept.
University of Massachusetts, Amherst
[2] Division of Computer Science
Hokkaido University
{mcarmosi,immerman}@cs.umass.edu, skip@ist.hokudai.ac.jp

Abstract. We describe our development and use of DescriptiveEnvironment (DE). This is a program to aid researchers in Finite Model Theory and students of logic to automatically generate examples, counter-examples of conjectures, reductions between problems, and visualizations of structures and queries.

DescriptiveEnvironment is available for free use under an ISC license at `http://www.cs.umass.edu/~immerman/de` . We encourage researchers and students at all levels to experiment with it. Please tell us of your insights, progress, suggestions, or extensions of DE.

Dedication

Dexter Kozen is a man of enormous energy. He plays soccer, rugby and ice hockey; he volunteers at the Cayuga Heights Fire Department; he raises three boys with Fran. In his spare time, Dexter proves theorems, teaches classes and writes books.

How has he done it all? One aspect is that when Dexter is engaged on a project, sleep is apparently unnecessary. Another is his plasticity: with an amazing ability to quickly master any relevant tool — whether it is a body of mathematics or a piece of software — that will add insight to what he is investigating, Dexter can shape shift easily. Between his flexibility, grace (watch out for him on the soccer field!) and his wide-ranging interests, Dexter has produced many elegant results. His results are deep and his impact has been wide.

We doubt that he is really turning sixty (In the words of Garnet Rogers, "What's Wrong With This Picture?" [Rog94][1]), but we wish him happy birthday all the same.

1 Introduction

When Neil Immerman was a graduate student at Cornell, he discovered Descriptive Complexity[2]. His dream was that rather than writing complex, error-prone

⋆ This research supported in part by NSF grants CCF 1115448; CCF 0830174.
⋆⋆ Supported by a Grant-in-Aid for JSPS Fellows under Grant No. 2100195209.

[1] Dexter has a rock band and it's easy to imagine them performing this song.

[2] Immerman originally called it "First Order Expressibility"; sometime later his advisor, Juris Hartmanis, suggested "Descriptive Complexity".

R.L. Constable and A. Silva (Eds.): Kozen Festschrift, LNCS 7230, pp. 24–34, 2012.

programs, one could simply express the desired task in logic — leading to simple, correct and flexible code. However, it turns out that it is not particularly easy to write involved specifications in logic. Furthermore, just constructing examples and counter-examples of conjectures can be challenging. One tool to help with this would be a program that would automatically construct the models described by logical formulas. This is the genesis of DescriptiveEnvironment (DE).

For many years, DE was simply a glimmer in Immerman's eye. When Charles Jordan was an undergraduate at the University of Massachusetts in the early 2000s, he built a prototype of DE as a senior project. During his early years as a graduate student in computer science at Hokkaido University, Jordan refined DE. During a current visit to UMass Amherst, Jordan, along with graduate student Marco Carmosino, have extended DE.

In this paper we will introduce DE and present its basic functionality. In particular we will discuss **queries**: the formalism for expressing properties, transforming structures, and building reductions between problems. We begin by introducing some terminology from logic and Descriptive Complexity.

2 Descriptive Complexity: Mathematical Background

This background material is condensed from [Imm99]. The reader desiring more detail should consult that book as well as [EF99, Lib04].

This paper is not meant to be an introduction or survey on Descriptive Complexity. To satisfy the reader's curiosity we will provide the following very brief description of this subject. Beyond that, we point the reader to the following short survey: [Imm95], or to the above three books to find out about the subject in depth.

Descriptive Complexity began with Fagin's Theorem which says that a property, e.g., a graph property, is in NP iff it is expressible in second-order existential logic, i.e., NP = SO$\exists$ [Fag74]. Since that time, essentially all important complexity classes have been characterized via standard logical languages.

Two fundamental translations from computational complexity to logic are the following: (i) The parallel time needed to check whether an input structure has a property is equal to the depth needed to describe that property in a first-order inductive definition; and (ii) the amount of memory needed to check whether an input structure has a property is characterized by the number of distinct variables needed to describe that property in first-order logic.

In general, the computational complexity of checking whether an input has a given property can be exactly characterized as the richness of a logical language needed to express that property.

Thus the secrets of computation can be understood with logic. For this reason, it is of great use to have tools such as DE to help us manipulate and reason about the objects in question, i.e., structures and queries.

In the rest of this section we recall standard notation from mathematical logic. A **relational vocabulary**, $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$ is a tuple of relation

symbols of given arity and constant symbols. For example, the following DE commands create the vocabularies "graph", consisting of one binary relation symbol, E, and two constant symbols, s, t; and "set" consisting of a single unary relations symbol, S.

```
graph is new vocabulary{E:2, s, t}.
set is new vocabulary{S:1}.
```

A **structure** with vocabulary τ is a tuple,

$$\mathcal{A} \quad = \quad \langle |\mathcal{A}|, R_1^{\mathcal{A}}, \ldots, R_r^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_s^{\mathcal{A}} \rangle$$

whose universe is the nonempty set $|\mathcal{A}|$. For each relation symbol R_i of arity a_i in τ, $\mathcal{A}$ has a relation $R_i^{\mathcal{A}}$ of arity a_i defined on $|\mathcal{A}|$, and for each constant symbol, c_j, $c_j^{\mathcal{A}}$ is an element of $|\mathcal{A}|$.

The following command creates line10, a structure of vocabulary graph that is a line graph on 10 vertices:

```
line10 is new structure{graph, 10, E:2 is x2=x1+1, s is 0, t is 9}.
```

In DE, the universe of a structure with size $n+1$ is $\{0, 1, \ldots, n\}$. The definition of E has two free variables, x1 and x2, and the above command defines the edge relation in line10 to be $\{(x_1, x_2) \mid 0 \leq x_1, x_2 \leq 9, x_2 = x_1 + 1\}$ (see Fig. 1). In DE, the definition of a relation symbol of arity a assumes that the free variables are $\{x_1, \ldots, x_a\}$.

The next instruction creates primes100, a structure of vocabulary set that is the set of all primes less than 100:

```
primes100 is new structure{set,100, S:1 is (1< x1 &
                  \A x, y.(x<=x1 & y<=x) : ((x*y=x1)->(x=1|y=1)))}.
```

The symbols `\A` and `\E` denote $\forall$ and $\exists$, and so S(x1) holds when x1 is prime.

Of course we can ask DE to print a given relation as follows, or to draw a given structure (Fig. 1).

```
primes100.S.
:{(2), (3), (5), (7), (11), (13), (17), (19), (23), (29), (31),
  (37), (41), (43), (47), (53), (59), (61), (67), (71), (73), (79),
  (83), (89), (97)}
```

2.1 Queries

A **query** is any mapping $I : \text{STRUC}[\sigma] \to \text{STRUC}[\tau]$ from structures of one vocabulary to structures of another vocabulary, that is polynomially bounded. A **boolean query** is a map $I_b : \text{STRUC}[\sigma] \to \{0, 1\}$. In Descriptive Complexity, computations are queries and decision problems are boolean queries.

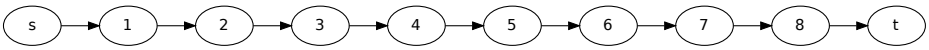

Fig. 1. The result of DE command: `draw(line10)`

Let σ and τ be any two vocabularies where $\tau = \langle R_1^{a_1}, \ldots, R_r^{a_r}, c_1, \ldots, c_s \rangle$. A **first-order query**, $I : \mathrm{STRUC}[\sigma] \to \mathrm{STRUC}[\tau]$,

$$I = \langle k, \varphi_0, \varphi_1, \ldots, \varphi_r, \psi_1, \ldots, \psi_s \rangle$$

is an $r+s+2$-tuple consisting of a positive natural number k called the **dimension** of the query, plus formulas from the first-order language of σ, defining the universe of the image structure together with the relations and constants defined on the image structure.

For each structure $\mathcal{A} \in \mathrm{STRUC}[\sigma]$, these formulas describe a structure $I(\mathcal{A}) \in \mathrm{STRUC}[\tau]$,

$$I(\mathcal{A}) \quad = \quad \langle |I(\mathcal{A})|, R_1^{I(\mathcal{A})}, \ldots, R_r^{I(\mathcal{A})}, c_1^{I(\mathcal{A})}, \ldots, c_s^{I(\mathcal{A})} \rangle \ .$$

The universe of $I(\mathcal{A})$ is a first-order definable subset of $|\mathcal{A}|^k$.

$$|I(\mathcal{A})| \quad = \quad \left\{ \langle b^1, \ldots, b^k \rangle \in |\mathcal{A}|^k \ \middle| \ \mathcal{A} \models \varphi_0(b^1, \ldots, b^k) \right\}$$

(Usually we will take $\varphi_0 \equiv \mathbf{true}$, thus letting $|I(\mathcal{A})|$ be the set of all k-tuples from $|\mathcal{A}|$.)

Each relation $R_i^{I(\mathcal{A})}$ is a first-order definable subset of $|I(\mathcal{A})|^{a_i}$,

$$R_i^{I(\mathcal{A})} \quad = \quad \left\{ (\langle b_1^1, \ldots, b_1^k \rangle, \ldots, \langle b_{a_i}^1, \ldots, b_{a_i}^k \rangle) \in |I(\mathcal{A})|^{a_i} \ \middle| \ \mathcal{A} \models \varphi_i(b_1^1, \ldots, b_{a_i}^k) \right\} .$$

Each constant symbol $c_j^{I(\mathcal{A})}$ is a first-order definable element of $|I(\mathcal{A})|$,

$$c_j^{I(\mathcal{A})} \quad = \quad \text{the unique } \langle b^1, \ldots, b^k \rangle \in |I(\mathcal{A})| \text{ such that } \mathcal{A} \models \psi_j(b^1, \ldots, b^k) \ .$$

For example, the following creates a binary first-order query I from graphs to sets. The universe formula for I is true, denoted in DE as `\t`:

```
I is new query{graph,string,2,\t,S:1 is E(x1,x2)}.
```

Applying I to line10 results in a set of 100 potential elements, populated by exactly the 9 edges of line10:

```
set100 is I(line10).
set100.S.
:{(1), (12), (23), (34), (45), (56), (67), (78), (89)}
```

Here is a slightly more interesting query:

```
R is new query{graph,graph,2,\t,E:2 is (x1=x3 & E(x2,x3))
               | (x1+1=x2 & x2=x3 &   x3=x4), s is x1=0 & x2=0,
               t is x1=max & x2=max}.
```

Here `max` denotes the maximum element of the universe. Let REACH be the set of graphs that have a path from s to t. Observe that for an undirected graph G, R(G) is in REACH iff G is connected. Thus R is a first-order reduction from connectivity of undirected graphs to REACH.

In general, if S and T are finite sets of structures of vocabulary σ and τ, respectively, and R is a first-order query from STRUC[σ] to STRUC[τ], then R is a **first-order reduction** from S to T iff for all finite structures $\mathcal{A} \in$ STRUC[σ],

$$\mathcal{A} \in S \quad \Leftrightarrow \quad R(\mathcal{A}) \in T \; .$$

One of the reasons that complexity theory has been so successful in characterizing the complexity of problems is that naturally arising computational problems tend to be complete for important complexity classes such as NP, P, PSPACE, NSPACE[$\log n$], DSPACE[$\log n$], and a few others. This is true in spite of the fact that a well-known theorem of Ladner says that for any two of these classes that are distinct, there are intermediate problems, i.e., in but not complete for the larger class, but not in the smaller class [Lad75]. Contrast this with the fact that there are thousands of natural NP-complete problems, many dozens of P-complete problems, but only about four known natural problems that are in NP but not known to be in P nor NP complete.

A related phenomenon is that all those natural complete problems, originally shown complete via fairly powerful reductions, e.g., polynomial-time many-one reductions, tend to remain complete under first-order reductions. Remarkably, the extremely weak first-order projections (fop), quantifier-free reductions, and even quantifier-free projections (qfp) also usually suffice [Imm99, Val82].

For example, consider the first-order reduction R above. It is a qfp. It is quantifier-free because the definitions of the new relations and constants – E, s, t – are quantifier free. It is a projection because each bit of the output structure R(G) depends on at most one bit of the input structure G.

Relatively simple queries suffice to construct many of the objects of interest. Furthermore, it is possible to program via reductions, i.e., build a carefully constructed and optimized program for a complete problem, C, and then solve other problems by simply writing the reduction to C. An extremely successful example of this point of view occurs when $C =$ SAT: We already have general automatic problem solving via SAT solvers.

Here is another example of the potential value of being able to reason about simple reductions. (See [Imm99] for details.) The problem REACH mentioned above is complete via qfps for the complexity class NSPACE[$\log n$] = NL. Let REACH$_d$ be the subset of REACH containing graphs of out-degree at most one. (See Fig. 1 for a simple example.) REACH$_d$ is complete via qfps for

DSPACE[$\log n$] = L. The following containments are well known and easy to prove:

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP}\,.$$

Everyone knows that is is open whether P = NP, but in fact it is open whether L = NP. Since three-colorability of graphs (3-COLOR) is complete for NP via logspace reductions (in fact, fops), it follows that 3-COLOR is reducible to REACH_d via qfps iff L= NP. In symbols,

$$\text{3-COLOR} \leq_{\mathrm{qfp}} \mathrm{REACH}_d \quad \Leftrightarrow \quad \mathrm{L} = \mathrm{NP}$$

Thus reasoning about first-order and quantifier-free reductions is valuable for proving upper and lower bounds on complexity.

3 Order-Independent P: A Motivating Example for DE

In 1982 Immerman and Vardi independently characterized polynomial time as the set of properties expressible in first-order logic plus the power to define new relations by induction: P = FO(LFP) [Imm86, Var82].

But that was for ordered structures. When a graph or other logical structure is encoded in a computer, the vertices appear in some order. Furthermore, algorithms exploit this ordering by searching, for example, along the first edge leaving a particular vertex. To even the playing field, the languages we use to describe computational properties, starting with FO, include some arbitrary total ordering relation on the universe. (This is handled in DE by having the universe of all structures of size $n+1$ be $\{0, 1, \ldots, n\}$, where ordering, addition and multiplication on the integers is available.)

Graph properties are order-independent in the sense that they would not change if we changed the order in which the vertices are stored in your computer. However, since programs and formulas in FO(LFP) may use that arbitrary ordering, they may be computing a property of the ordered graph that depends on that ordering and so is not a graph property at all.

It is of great interest to capture **order-independent P** with a logical language. Whether this is possible remains open. If one simply removes ordering from FO(LFP), the resulting language is too weak. It cannot even count whether there are an even number of vertices in a graph. It was natural to ask whether FO(LFP, COUNT) – first-order logic with fixed point and counting quantifiers, but no ordering – captures order-independent P.

In 1981, Immerman and Eric Lander began investigating this question. It suffices to check it on graphs. They showed that this result holds for all trees and almost all graphs even when the formulas in question use only two variables [IL90].

If a graph is ordered, then each of its vertices has a unique name, i.e., the first vertex, the second vertex, etc. It is natural to consider graphs with unary relations representing colorings of the vertices. The *color-class size* of a graph is the number of vertices of the most popular color, e.g., if a graph had two

green vertices, one blue vertex, three red vertices and two yellow vertices then it would have color-class size 3. In some sense being ordered is the same as being of color-class size one.

In the paper [IL90], Immerman and Lander also showed that on graphs of color-class size at most 3, FO(LFP, COUNT) captures order-independent P, and three variables are necessary and sufficient.

Five years later, Jin-yi Cai (another Cornellian) and Immerman were trying to extend this result to larger color classes, in particular, trying to figure out whether this result held for color-class size 4 graphs. It doesn't; the CFI gadget discovered by Cai and Immerman and independently Fürer [CFI92], is a basis for the counterexample.

Fact 1. *There is a linear-time computable property of graphs of color-class size 4 that is not expressible in* FO(LFP, COUNT). *To express this property for graphs on n vertices in* FO(LFP, COUNT) *requires $\Omega(n)$ distinct variables, while any fixed formula in* FO(LFP, COUNT) *only has a bounded number of variables.*

Although it remains open whether it is possible to capture order-independent P, there has been great progress on this problem. An operator giving the rank of a matrix makes the CFI property expressible, so FO(LFP, rank) is a strict extension of FO(LFP, COUNT) that is a new candidate for capturing order-independent P [DGH09]. Furthermore, in a break-through result, Grohe has shown that FO(LFP, COUNT) captures order-independent P for all classes of graphs having an excluded minor [Gro10].

4 Basic Functionality of DE

The CFI gadget took many blackboards, pieces of paper, and years to discover. With DE, we could have found it back then in minutes; today it would only take seconds. We have already seen a few DE commands. We now explain more of what DE can do.

As we have seen, DE lets us easily define vocabularies, structures, and queries. For example, let agraph be the vocabulary of graphs with two colors:

```
agraph is new vocabulary{A:1, E:2}.
```

In addition to defining structures explicitly as we did for line10 and set100 in §2, DE uses the tool Mace4[3] to generate a structure satisfying a given first-order sentence. For example, the following command calls Mace4 to create an undirected bipartite graph whose vertices all have degree at least 2:

```
bip2 is mace(agraph, \A x: \E y1,y2.y1!=y2: (E(x,y1) & E(x,y2)) &
                     \A x,y: (E(x,y)->(E(y,x) & (A(x)<->~A(y))))).
```

[3] Available at `http://www.cs.unm.edu/~mccune/prover9/`

The result is the smallest such graph: the complete bipartite graph $K_{2,2}$.

Built into DE are the boolean queries `minisat()` and `zchaff()` for calling SAT solvers MiniSat[4] and zChaff[5] to check the satisfiability of a propositional formula.

The vocabulary `sat` for propositional formulas is

```
sat is new vocabulary{P:2, N:2}.
```

Here $P(c, x_i)$ means that variable x_i occurs positively in clause c and $N(c, x_i)$ means that literal $\overline{x_i}$ occurs in c. The default is that a clause that has no literals is ignored.

DE makes it easy to use SAT solvers to solve a wide class of problems. For example, consider the following 3-ary reduction from 3COLOR to SAT:

```
thrcoltosat is new query{graph, sat, 3, x1<=3,
      P:2 is x1=3 & x2=0 & x3=x6 & x4=0 & x5<3,
      N:2 is x4=0 & E(x2,x3) & (x1=x5 & (x6=x2 | x6=x3)) & x1<3}.
```

Here, boolean variable $\langle 0, c, x\rangle$ means "vertex x is color c", clause $\langle 3, 0, x\rangle$ says that "vertex x is some color" and if $E(x, y)$ then clause $\langle a, x, y\rangle$ (for $0 \leq a \leq 2$) says that, "then x and y are not both color a".

We can then use the SAT solver to check 3-colorability. For example, if we have the DIMACS-format graph `myciel3.col`[6], we can use DE to check if it is 3-colorable:

```
myciel3 is load("myciel3.col").
mycielfmla is thrcolortosat(myciel3).
minisat(mycielfmla).
:\f
```

Here `mycielfmla` is a boolean formula that is satisfiable iff the graph myciel3 is 3-colorable. In this case, it is not satisfiable and thus we can conclude that myciel3 is not 3-colorable.

DE currently allows formulas from SO$\exists$(TC)[7], i.e., in addition to first-order logic, second-order existential logic, a transitive closure operator, and arithmetic are all available. Thus queries using TC and even second-order quantification may be written and evaluated. In the following two boolean queries, reach is defined using TC and isat is defined using second-order existential quantification. Not surprisingly, it is faster to use minisat and zchaff than to evaluate isat directly:

```
reach is new bquery{graph, TC[x,y:E(x,y)](s,t)}.

isat is new bquery{sat, \E S:1:\A z,x:\E y:(~P(z,x) & ~N(z,x)) |
                                (P(z,y) & S(y)) |(N(z,y) & ~S(y))}.
```

[4] `http://minisat.se`

[5] Available at `http://www.princeton.edu/~chaff/zchaff.html` . Due to licensing issues, users who wish to use zChaff must download it themselves.

[6] Available from `http://mat.gsia.cmu.edu/COLOR/instances.html`

[7] Except when using Mace4, when we are restricted to first-order.

5 A Motivating Example for ReductionFinder

Fact 2. [Imm88, Sze88] *For all* $s(n) \geq \log n$, NSPACE$[s(n)]$ = *co*-NSPACE$[s(n)]$.

When Immerman proved Fact 2, he was motivated by the fact that REACH is complete for NSPACE$[\log n]$ via quantifier-free projections (qfp) [Imm99]. Thus he knew that NSPACE$[\log n]$ (NL) is closed under complementation iff there is a qfp from $\overline{\text{REACH}}$ to REACH.

In fact, his original proof of Fact 2 was just that: a dimension 8 qfp from $\overline{\text{REACH}}$ to REACH. The construction, somewhat analogous to the much simpler reduction R from §2.1, proceeds by stepping through distances d, from 1 to n, and target vertices t, from 0 to $n-1$, computing the exact number n_d of vertices reachable from s in distance at most d, using the previously computed constant, n_{d-1}.

Immerman was struck at the time by how constraining – and thus useful – it was to have to build a qfp. It seemed either easy or impossible in the sense that at most steps of building the reduction there was only one possible thing to do.

This feeling led many years later to the building of ReductionFinder, a program that repeatedly uses a SAT solver to search for a small quantifier-free reduction between two problem specifications [CIE10]. At present writing, ReductionFinder is good at finding very small quantifier-free reductions – dimension 8 is way beyond the current capability. Still we feel that this approach has enormous potential. We are currently making the interface between DE and ReductionFinder.

6 Education

One of the differences between experts and students or novices is the experts' ability to visualize concepts relevant to their field. This ability aids in the construction and interpretation of diagrams, which allow communication between experts and discovery of new results. Students do not gain as much knowledge from examining domain-specific diagrams as experts do; they simply lack the context and skills to interpret diagrams.

In Descriptive Complexity, our diagrams are generally graphs depicting parts of logical structures before and after queries, or gadgets like the CFI construction. There are many high-quality open-source tools for drawing and labeling graphs. We have implemented a script that transforms the saved output of DE structures into files for GraphViz[8].

To make the process as easy as possible, we have added a "draw" command to DE, so that users can make changes to defined structures and then immediately observe their effect. This process mimics the visualization skills of the expert, and will allow students to gain a deeper understanding of how structures are defined by logical formulas.

[8] `http://www.graphviz.org/`

In the future, we will extend "draw" to operate on queries, showing a before-and-after view of the logical structure, and a typeset presentation of the relevant logical formulas in one image. This will not only help students understand example queries through modifying and plotting them, it will help researchers examine and verify the results of more complex queries.

Ehrenfeucht-Fraïssé games and their generalizations are a vital tool for Descriptive Complexity. It would be easier to teach students to play these games if we had software to play them automatically. We will build an EF-game engine for DE, because it is convenient to do so: all of the relevant datatypes are already available. Playing against an automated Samson or Delilah, each step of the game will be drawn to the screen. We expect that students will develop an intuition for EF-games and their generalizations faster and easier when a computer implementation of the game with graphics is readily available.

In the longer term, DE can be used as the backend for a web-based system. This would be the most convenient and easy format for students, and would allow the use of a rich web UI for playing EF games and playing with example queries and structures.

7 Conclusions

We have explained some of the motivations for building DE, together with some of its functionality. We encourage anyone to use it. Please send us your examples, questions, suggestions, improvements, and extensions.

In the near future we anticipate adding more features and visualization tools. In particular, we can hardly wait until DE is able to play Ehrenfeucht-Fraïssé games. It will also aid in visualizing a query by drawing the input and output structures, and letting the user choose points in the input or output and having the display highlight where they lead to or come from. We will add sliders to help students and researchers visualize the computations of transitive closures and fixed points.

There are many great programs and systems related to logic that are being developed and used that we have omitted in this introduction to DE. Our plan is to have an easy-to-use, extensible testbed. If there are great tools available, such as SAT solvers, Mace, ReductionFinder, that we can call to make DE more useful, then we are delighted to do so. The intent of DE is to make the research and teaching of logical methods easier and more fun.

References

[CFI92] Cai, J., Fürer, M., Immerman, N.: An Optimal Lower Bound on the Number of Variables for Graph Identification. Combinatorica 12(4), 389–410 (1992)

[CIE10] Crouch, M., Immerman, N., Moss, J.E.B.: Finding Reductions Automatically. In: Blass, A., Dershowitz, N., Reisig, W. (eds.) Fields of Logic and Computation. LNCS, vol. 6300, pp. 181–200. Springer, Heidelberg (2010)

[EF99] Ebbinghaus, H.-D., Flum, J.: Finite Model Theory, 2nd edn. Springer, Heidelberg (1999)

[DGH09] Dawar, A., Grohe, M., Holm, B., Laubner, B.: Logics with Rank Operators. In: IEEE Symp. Logic In Comput. Sci., pp. 113–122 (2009)

[Fag74] Fagin, R.: Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In: Karp, R. (ed.) Complexity of Computation. SIAM-AMS Proc., vol. 7, pp. 43–73 (1974)

[Gro10] Grohe, M.: Fixed-Point Definability and Polynomial Time on Graphs with Excluded Minors. In: IEEE Symp. Logic In Comput. Sci., pp. 179–188 (2010)

[Imm99] Immerman, N.: Descriptive Complexity. Springer Graduate Texts in Computer Science, New York (1999)

[Imm95] Immerman, N.: Descriptive Complexity: A Logician's Approach to Computation. Notices of the American Mathematical Society 42(10), 1127–1133 (1995)

[Imm88] Immerman, N.: Nondeterministic Space is Closed Under Complementation. SIAM J. Comput. 17(5), 935–938 (1988)

[Imm87] Immerman, N.: Languages That Capture Complexity Classes. SIAM J. Comput. 16(4), 760–778 (1987)

[Imm86] Immerman, N.: Relational Queries Computable in Polynomial Time. Information and Control 68, 86–104 (1986); A preliminary version of this paper appeared in ACM Symp. Theory of Comput., pp. 147–152 (1982)

[IL90] Immerman, N., Lander, E.S.: Describing Graphs: A First-Order Approach to Graph Canonization. In: Selman, A. (ed.) Complexity Theory Retrospective, pp. 59–81. Springer, Heidelberg (1990)

[Lad75] Ladner, R.: On the structure of polynomial time reducibility. J. Assoc. Comput. Mach. 22(1), 155–171 (1975)

[Lib04] Libkin, L.: Elements of Finite Model Theory. Springer, Heidelberg (2004)

[Rog94] Rogers, G.: What's Wrong With This Picture?, `http://www.garnetrogers.com/lyrics/What's%20Wrong%20With%20This%20Picture.txt`

[Sze88] Szelepcsényi, R.: The Method of Forced Enumeration for Nondeterministic Automata. Acta Informatica 26, 279–284 (1988)

[Var82] Vardi, M.: Complexity of Relational Query Languages. In: ACM Symp. Theory of Comput., pp. 137–146 (1982)

[Val82] Valiant, L.: Reducibility By Algebraic Projections. L'Enseignement mathématique T. XXVIII(3-4), 253–268 (1982)