Model Expansion and the Expressiveness of FO(ID) and Other Logics[*]

Antonina Kolokolova KOL@CS.MUN.CA
Memorial University of Newfoundland
Yongmei Liu YLIU@CS.TORONTO.EDU
Hong Kong University of Science and Technology
David G. Mitchell MITCHELL@CS.SFU.CA
Eugenia Ternovska TER@CS.SFU.CA
Simon Fraser University

Abstract

Model expansion problem is a question of determining, given a formula and a structure for a part of the vocabulary of the formula, whether there is an expansion of this structure that satisfies the formula. Recent development of a problem-solving paradigm based on model expansion by (Mitchell & Ternovska, 2005; Mitchell, Ternovska, Hach, & Mohebali, 2006) posed the question of complexity of this problem for logics used in the paradigm.

We discuss the complexity of the model expansion problem for a number of logics, alongside that of satisfiability and model checking. As the task is equivalent to witnessing leading existential second-order quantifiers in a model checking setting, the paper is in large part a survey of this area together with some new results. In particular, we describe the combined and data complexity of model expansion for FO(ID) (Denecker & Ternovska, 2008), as well as guarded and k-guarded logics of (Andréka, van Benthem, & Németi, 1998) and (Gottlob, Leone, & Scarcello, 2001).

1. Introduction

In model theory, expanding a structure means adding new functions and relations to the structure, while keeping the same domain. Model expansion (abbreviated MX), for a logic $\mathcal{L}$, is the following problem. Given a formula ϕ of $\mathcal{L}$, and a structure $\mathcal{A}$ for part of the vocabulary of ϕ, determine if there an expansion of $\mathcal{A}$ to the full vocabulary of ϕ that makes ϕ true. We may consider the task for a variety of logics, including classical first-order and second-order logic (FO and SO, respectively), various non-classical logics, and restrictions or extensions of standard logics. Here, we examine the complexity of model expansion for a number of restrictions and extensions of classical first-order logic (FO), and a related question about expressiveness. Our study is motivated by a relationship between MX for these logics and practical considerations in declarative programming for search problems.

In particular, we analyze the complexity of model expansion for FO(ID), an extension of FO with inductive definitions, GF$_k$, the k-guarded fragment of FO, and GF$_k$(ID). Table 1.3 on page 4 gives an overview of the complexity and expressibility results presented in this paper.

∗. The first two authors were supported in part by PIMS postdoctoral fellowships at Simon Fraser University; all authors were supported in part by NSERC.

Some of these results follow immediately from well-known theorems such as these of Fagin's (Fagin, 1974) and Grädel's (Grädel, 1992)); others are considered for the first time in this paper. We present these results in the context of the complexity of the closely related, but more well-studied, tasks of finite satisfiability and model checking. This work extends an earlier version (Kolokolova, Liu, Mitchell, & Ternovska, 2006).

1.1 Model Expansion, Constraint Programming, and Descriptive Complexity

A number of somewhat independent lines of work, under labels such as "constraint modeling languages" or "declarative programming for search problems", aim to produce high-level declarative languages for representing decision and search problems, together with tools ("solvers") for using these languages to solve instances of the problems in practice. It is natural to see much of this work as having in common the underlying logical task of model expansion. The idea here is that the formula ϕ, is a problem specification, which is fixed for a particular problem. The given finite structure $\mathcal{A}$ is the instance, and expansions of $\mathcal{A}$ that satisfy ϕ correspond to solutions to the instance.

Example 1.1. *For example, the classical NP problem 3-colourability can be naturally expressed as FO MX as follows: the 3-colourability condition is encoded as a formula, and a graph is given as an input structure. The colours then are the expansion predicates for the input structure.*

Let $\mathcal{A}$ be a graph $G = (V; E)$, and let ϕ be

$$\forall v \forall w (R(v) \vee B(v) \vee G(v)) \wedge \bigwedge_{Q \in R,G,B} (\neg Q(v) \vee \neg Q(w) \vee \neg E(v,w)).$$

Let $\mathcal{B}$ be an expansion of $\mathcal{A}$ to vocab(ϕ). Then $\mathcal{B} \models \phi$ iff $R^{\mathcal{B}}, G^{\mathcal{B}}, B^{\mathcal{B}}$ give a three-colouring of G.

Many of the existing approaches to representing search problems explicitly use logical languages. Examples include answer set programming (ASP) languages, such as those of LPARSE and DLV, NP-Spec (Cadoli, Ianni, Palopoli, Schaerf, & Vasile, 2000), ConSQL (Cadoli & Mancini, 2002) and ASPPS (East & Truszczynski, 2006), and it is easy to see that the underlying task is model expansion[1]. The logic of NP-Spec is an extension of Datalog with negation; the language of ConSQL is a generalization of SQL, and thus the logic involved is a syntactic variant of FO. The MX Project (Mitchell & Ternovska, 2005; Mitchell et al., 2006) is to build languages and systems explicitly based on model expansion for extensions of classical logic.

Examples where the languages are not logics include ESSENCE (?) and ESRA (Flener, Pearson, & Agren, 2004), which have semantics specified denotationally, in the style of general programming languages. These languages both include fragments which are essentially fragments of FO and SO extended with a type system, and so it is not hard to see how significant fragments of these languages can be viewed as logics. For example, in (?, ?), various fragments of the constraint specification language are presented as model expansion

1. although the use in some cases of Herbrand models muddles the water somewhat.

for syntactic variants of FO and SO, to carry out an analysis of expressiveness of the language. In other cases, such as Zinc and OPL, the relationship to logic is less transparent, and for these we may see a logic $\mathcal{L}$ as a *mathematical abstraction* of the system language.

Designers of languages and systems may choose their logic according to their goals. One such goal might be to target problems in a particular complexity class ("capture" the class). Such a "capturing" property is of practical importance as it provides an assurance to the user of the *universality* of a chosen system language for a targeted complexity class. The importance of providing such universality has been realized before, and proven for a fragment of ASP, for NP-Spec and ConSQL.

An area which establishes a connection between logics and complexity classes is Descriptive Complexity (Immerman, 1999). A classic example of a descriptive complexity result is Fagin's theorem, which tells us that, when the logic $\mathcal{L}$ is FO, we can describe precisely the search problems corresponding to the complexity class NP. This is because model expansion for FO is the same as model checking for $\exists$SO, the existential fragment of second-order logic, for which Fagin's result applies. The result generalizes to higher levels of Polynomial Time hierarchy. Results by Grädel (Grädel, 1992) and others show that other logics capture other interesting complexity classes such as feasible computation classes P and NL.

The methods of finite model theory and descriptive complexity could be utilized by the Constraint programming community to a greater degree than they are now. However, the use of classical (i.e., first-order) logic in practical systems (including those tailored for search problems in NP) is often considered problematic.Many people think one cannot build systems with a syntactic variant of FO as a specification language because classical logic (or its extensions) is simply "the wrong kind of language". One argument is that the language is too abstract and does not have all the features one needs for effective modeling (e.g. such features as the ability to say that a relation is a bijection, which of course is FO expressible).

Another argument is that one cannot build an efficient solver for such an expressive language, where validity is not even decidable (although validity is not the intended task for search problems).

It is obvious that a lot needs to be done to make finite model theory and descriptive complexity applicable to constraint programming *in practice.*

Remark: it is interesting the that "convenient language" people clearly do not believe in the need for restriction to the degree the "logic" people do, generally speaking.

1.2 The MX Project

The authors of (Mitchell & Ternovska, 2005) proposed formalizing search problems as model expansion and constricting practical tools based on this formalization. In this framework, problems are axiomatized in a high-level language, and an automatic reduction to a complete problem in a complexity class is performed. For example, reduction to SAT or an extension (or any other NP-complete problem for which an efficient solver exists), can be performed for NP, and similarly for other complexity classes. The work to date includes development of a prototype solver, called MXG, which performs as well (often better) even compared with more mature systems based on other approaches such as Answer Set Programming. The specification language for MXG is essentially multi-sorted FO, extended with a limited form of inductive definitions and other convenience features. A general report on the project is (Mitchell et al., 2006) and MXG is described in (?).

While FO model expansion is sufficient for capturing NP, FO lacks many features that are useful for practical modeling. In particular, this logic lacks a construct to represent induction. The need for inductive definitions comes from the lack of ability of FO logic to handle recursion and non-monotonic constructs, a serious handicap in modeling. Simple and useful properties like transitive closure are not expressible in FO. While induction is possible to *simulate* in ∃SO, to model such properties, a "guess and check" approach is needed. For transitive closure, for example, such modeling is a highly non-trivial task, see e.g. (Kolokolova, 2004) for details.

1.3 Logics for MX

The languages used in the Constraints community generally do not have recursion. On the other hand, most logic programming languages (the languages of ASP, Datalog LITE, and Datalog with negation in case of NP-Spec) represent the opposite extreme — everything needs to be expressed through a recursive construct. It is quite obvious that recursion (including recursion through negation) is needed for effective modeling, however it is our view that such a construct is most useful in combination with a classical part. For this reason, in (Mitchell & Ternovska, 2005), it was proposed to focus on a logic with inductive definitions for an MX-based constraint modeling framework. The formalism they chose for this purpose is ID-logic, developed in (Denecker, 2000; Denecker & Ternovska, 2004a, 2008); early ideas for using inductive definitions in Knowledge Representation appear already in (?), and, in combination with classical logic, in (?, ?). The induction construct of ID-logic allows for natural modeling of monotone and non-monotone inductive definitions, including iterated induction and induction over well-founded order. Such constructions occur in mathematics, as well as in common-sense reasoning. The logic is suitable to model non-monotonicity, causality and temporal reasoning (Denecker & Ternovska, 2004b, 2007), and as such is a good knowledge representation language. Two native solvers for the propositional fragment were developed (Mariën, Mitra, Denecker, & Bruynooghe, 2005) and (Pelov & Ternovska, 2005). East and Truszczynski's system ASPPS (East & Truszczynski, 2006) is a system for solving NP-search problems whose modeling language is a language of propositional schemata with rule-based notation and positive induction.

While the extension of FO with inductive definitions is useful, it can also be useful to restrict its power. One interesting restriction is to fragments which are in the family of the guarded fragments of FO (Andréka et al., 1998). A fragment which is most interesting to us is GF_k (Gottlob et al., 2001), where the conjunction of up to k atoms may act as a guard. That fragment was used to achieve efficient grounding in (Patterson, Liu, Ternovska, & Gupta, 2007). In (?), it was shown that GF_k is a natural abstraction of the *type system* of ESSENCE, and that all quantification in that language is guarded in the sense of GF_k. This fragment also naturally reflects quantification used in SQL and very likely in many other systems. The fragment GF_k (with an additional "upper guard" construct) plays a crucial role in adding arithmetic to the MX framework (?).

Table 1: Complexity of model checking, model expansion and satisfiability problems for some logics. Highlighted are logics and results new to this paper. Here, $\equiv$ denotes a capture, sometimes over a certain class of structures: $\equiv_s$ is over structures with a successor function and $\equiv_{BIT}$ over structures with BIT predicate. The notation "-c" means complete for a respective class; with $*$ means that the logic expresses a complete problem.

Logic	**Model checking**		**Model expansion**		**Satisfiability**
	Combined	**Data**	**Combined**	**Data**	
FO	PSPACE-c *Sto74*	$\equiv_{BIT}$AC^0 *BIS90*	NEXP-c *Var82,MT05*	$\equiv$NP *Fagin74*	Und. *Tra50*
FO(LFP)	EXP-c *Var82*	$\equiv_s$P *Imm82,* *Var82,* *Liv82*	NEXP-c	$\equiv$ NP	Und.
FO(ID)	EXP-c*	$\equiv_s$P*	NEXP-c*	$\equiv$NP *MT05*	Und.
FO^k	P-c	AC^0	NP-c *Var95*	NP-c,$\not\equiv$ NP	Und (k>2), NEXP-c (k=2), EXP-c (k=1)
GF$_k$ **RGF$_k$**	P-c *GO99,* *GLS01*	AC^0	**NEXP-c*** **NP-c***	$\equiv$**NP*(k$\geq$2)**, **NP-c** (k=1) **NP-c**, $\not\equiv$**NP***	Und. (k$\geq$2), 2EXP-c (k=1) *Gra99*
μGF	UP$\cap$ co-UP	P	NEXP-c	NP-c	2EXP-c (inf) *GW99,Gra02*
GF$_k$(ID)	$\in$EXP	P	NEXP-c	k>1: $\equiv$NP	k>1: Und.

2. Preliminaries and definitions

In this section, we review the standard notions of "complexity" and "expressiveness" of a logic, and discuss them in the new context of model expansion. In particular, we define the data and combined complexity for model checking and model expansion, the complexity of finite satisfiability, and the concept of "capturing" a complexity class.

2.1 Logical Structures

A vocabulary is a set σ of relation and function symbols, each with an associated arity. A relational vocabulary has no function symbols. A (relational) structure $\mathcal{A}$ for a vocabulary σ is a tuple containing a universe A (also called the domain of $\mathcal{A}$), and a relation defined over A, for each relation symbol of σ. If R is an n-ary relation (a.k.a. predicate) symbol of vocabulary σ, the relation corresponding to R in a σ-structure $\mathcal{A}$ is a set of n-tuples of domain elements denoted $R^{\mathcal{A}}$. For example, we write

$\mathcal{A} := (A;\ R_1^{\mathcal{A}}, \dots R_n^{\mathcal{A}},\ c_1^{\mathcal{A}}, \dots c_k^{\mathcal{A}})$, where A is the universe and the R_i are relation symbols. A structure is *finite* if its universe is finite. An example of a finite structure for vocabulary $\{E\}$ is a graph $\mathcal{G} := (V; E^{\mathcal{G}})$, where V is a set of vertices, $E^{\mathcal{G}}$ is set of pairs of vertices which represent edges. In this paper, we consider only finite relational structures.

We assume that the reader is familiar with first order logic (FO). We only recall one concept, which is of particular importance to this paper — satisfiability with respect to a structure. Given a structure $\mathcal{A}$, and a FO sentence ϕ (i.e., a formula without free variables), $\mathcal{A} \models\ \phi$ denotes that $\mathcal{A}$ *satisfies* ϕ, or that ϕ *is true* in structure $\mathcal{A}$, or $\mathcal{A}$ *is a model of* ϕ.

2.2 Second Order Logic

Second order logic (SO), in addition to first-order quantification, allows quantification over sets (of tuples). This is expressed through quantification over predicate symbols such as, e.g. $P(x_1, \dots, x_n)$. An example of a SO formula is

$$\exists R \exists B \exists G \forall v \forall w (R(v) \vee B(v) \vee G(v)) \wedge \bigwedge_{Q \in R,G,B} (\neg Q(v) \vee \neg Q(w) \vee \neg E(v,w)).$$

This formula encodes 3-colourability of structures with over a vocabulary consisting of a single binary relation E (also called "graph structures", since E is interpreted as an edge relation of a graph).

For more background on second-order logic please see (Enderton, 2001).

The existential fragment of SO, denoted $\exists$SO, is the fragment where all second-order quanfiers are existential. This fragment is of particular importance to us here.

We need to introduce another logic, which is related to SO.

Definition 2.1. Let $\mathcal{L}$ be any logic. Let $\exists$SO($\mathcal{L}$) be the set of formulas of the form $\exists P_1^{r_1} \dots \exists P_k^{r_k} \phi$ for some k, where $\phi \in \mathcal{L}$ and each $P_i^{r_i}$ is a second order relational variable of arity r_i. The logic $\exists$SO($\mathcal{L}$) may or may not be (equivalent to) a fragment of $\exists$SO, depending on $\mathcal{L}$. When $\mathcal{L}$ is FO, $\exists$SO($\mathcal{L}$) is precisely $\exists$SO.

2.3 Three Tasks of Interest

For a given logic $\mathcal{L}$, we consider the complexity of three problems. We repeat the definition of model expansion for convenience of the reader. For all three problems, we are interested in the decision versions here.

1. *Model Checking* (MC): given $(\mathcal{A}, \phi)$, where ϕ is a sentence in $\mathcal{L}$ and $\mathcal{A}$ is a finite structure for $vocab(\phi)$, does $\mathcal{A}$ satisfy ϕ?
2. *Model Expansion* (MX): given $(\mathcal{A}, \phi)$, where ϕ is a sentence in $\mathcal{L}$, $\mathcal{A}$ is a finite σ-structure where $\sigma \subseteq vocab(\phi)$, is there a finite structure $\mathcal{B}$ for $vocab(\phi)$ which expands $\mathcal{A}$ and satisfies ϕ?
3. *Finite Satisfiability* (FinSat): given a sentence ϕ in $\mathcal{L}$, is there a finite structure $\mathcal{A}$ for $vocab(\phi)$ such that $\mathcal{A} \vDash \phi$?

In model checking, the entire structure is given, in model expansion a part of it is given, and in satisfiability we are looking for a structure to satisfy a formula. Finite satisfiability and model checking have been studied for a long time. Our focus here is the model expansion task, and we study it in comparison with the other two problems. Often, we derive our results from the fact that MX for $\mathcal{L}$ is equivalent to MC for $\exists$SO($\mathcal{L}$). That is, there exists an expansion of a structure $\mathcal{A}$ if there exist interpretations for the expansion predicates, or, equivalently, if $\mathcal{A}$ satisfies the original formula of $\mathcal{L}$ preceded by existential quantifiers for all expansion predicates.

2.4 Data and Combined Complexity

For the tasks of model checking and model expansion, we consider two notions of complexity introduced in (Vardi, 1982). Here, we are following the presentation from (Libkin, 2004). Let *enc*() denote some standard encoding of structures and formulas by binary strings.

Definition 2.2. Let *Comp* be a complexity class and $\mathcal{L}$ a logic.

- The *data complexity* of model checking problem for $\mathcal{L}$ is *Comp* if for every sentence ϕ of $\mathcal{L}$ the language $\{enc(\mathcal{A}) \mid \mathcal{A} \models \phi\}$ belongs to *Comp*.
- The *combined complexity* of model checking for $\mathcal{L}$ is *Comp* if the language $\{(enc(\mathcal{A}), \mathrm{enc}(\phi)) \mid \mathcal{A} \models \phi\}$ belongs to *Comp*.
- The data (combined) complexity of the *model expansion* for $\mathcal{L}$ is the data (resp. combined) complexity of model checking of $\exists SO(\mathcal{L})$. enc(ϕ)

For the task of finite satisfiability, such a distinction does not make sense — the input contains only formulas and no structures. Thus, in this case, there is just one notion of complexity.

For a given logic $\mathcal{L}$, the complexity of model expansion always lies between the complexities of model checking and satisfiability (model existence), regardless of whether we consider data or combined complexity:

$$\mathrm{MC}(\mathcal{L}) \leq \mathrm{MX}(\mathcal{L}) \leq \mathrm{FinSat}(\mathcal{L}).$$

The reason is that in MX we are given more information than in FinSat by providing a part of the structure, and in MC even more information is given by providing the entire structure.

An interesting observation about finite model expansion is that even if the satisfiability problem for a logic $\mathcal{L}$ is undecidable, we always avoid undecidability in MX by fixing a (finite) universe.

2.5 Short Review of Complexity Classes

We need some background in the complexity theory, which we give using the exposition in (Libkin, 2004). Let L be a language accepted by a halting Turing machine M. Assume that for some function $f : \mathbb{N} \rightarrow \mathbb{N}$, the number of steps M makes before accepting or rejecting a string is at most $f(|s|)$, where $|s|$ is the length of s. If M is deterministic, then we write $L \in \mathrm{DTIME}(f)$; if M is nondeterministic, then we write $L \in \mathrm{NTIME}(f)$. We define the classes we are interested in this paper. The class P of polynomial-time computable problems is defined as

$$\mathrm{P} := \bigcup_{\mathrm{k} \in \mathbb{N}} \mathrm{DTIME}(\mathrm{n}^{\mathrm{k}}),$$

and the class NP of problems computable by nondeterministic polynomial-time Turing machine as

$$\mathrm{NP} := \bigcup_{\mathrm{k} \in \mathbb{N}} \mathrm{NTIME}(\mathrm{n}^{\mathrm{k}}).$$

To define NL, we need to talk about space complexity for sublinear functions f. To do that, we use a model of Turing machines with a work tape. We define NL to be the class of languages accepted by such nondeterministic machines where at most $O(\log|s|)$ cells of the work tape are used.

The Polynomial Time hierarchy PH is defined as follows. Let $\Sigma_0^p = \Pi_0^p = \mathrm{P}$. Define inductively $\Sigma_i^p = \mathrm{NP}^{\Sigma_{i-1}^p}$, for $i \geq 1$. That is, languages in Σ_i^p are those accepted by a nondeterministic Turing machine running in polynomial time such that the machine can make "calls" to another machine that computes a language in Σ_{i-1}^p. Such a call is assumed to have a unit cost. We define the class Π_i^p as the class of languages whose complements are in Σ_i^p. Notice that $\Sigma_1^p = \mathrm{NP}$ and $\Pi_1^p = \text{co-NP}$. We define the polynomial hierarchy as

$$\mathrm{PH} := \bigcup_{i \in \mathbb{N}} \Sigma_i^p = \bigcup_{i \in \mathbb{N}} \Pi_i^p.$$

The relationship between the main complexity clases is as follows:

$$\mathrm{NL} \subseteq \mathrm{P} \subseteq \left\{ \begin{array}{c} \mathrm{NP} \\ \text{co-NP} \end{array} \right\} \subseteq \mathrm{PH} \subseteq \mathrm{PSPACE}.$$

None of the containments of any two consecutive classes in this sequence is known to be proper, although it is known that $\mathrm{NL} \subsetneq \mathrm{PSPACE}$.

2.6 Capturing Complexity Classes

A property P of structures is *definable* in logic $\mathcal{L}$ if there is a sentence $\phi_P \in \mathcal{L}$ such that for every structure $\mathcal{A}$, we have that $\mathcal{A} \models \phi_P$ iff $\mathcal{A}$ has property P.

Definition 2.3. Let $\mathcal{D}$ be a class of finite structures, *Comp* be a complexity class, and $\mathcal{L}$ be a logic.

1. $\mathcal{L}$ *captures Comp* on $\mathcal{D}$ ($\mathcal{L} \equiv_{\mathcal{D}} Comp$) if
    (a) for every fixed sentence $\phi \in \mathcal{L}$, the data complexity of evaluating ϕ on structures from $\mathcal{D}$ is a problem in the complexity class *Comp*, and
    (b) every property of structures in $\mathcal{D}$ that can be decided with complexity *Comp* is definable in the logic $\mathcal{L}$.
2. We say that *model expansion for logic* $\mathcal{L}$ *captures a complexity class Comp* if *Comp* is captured by $\exists$SO($\mathcal{L}$) in the sense above.

Fagin's theorem says that $\exists$SO captures NP (Fagin, 1974). Similar results, which we discuss later, exist for P, NL, and the Σ levels of the Polynomial Time hierarchy. For more background on finite model theory and descriptive complexity, please see (Immerman, 1999; Libkin, 2004).

Capturing NP is a property of the logic $\mathcal{L}$. It says that *every* problem in NP can be axiomatized as model expansion for $\mathcal{L}$. NP-completeness of the MX task for $\mathcal{L}$ is a different property. It means that some NP-complete problems can be represented as MX, e.g. 3-colourability. Of course, we can always solve a particular problem in NP by a constructing a reduction to a given complete problem, but we may not be able to represent it in our logic $\mathcal{L}$.

3. Complexity of MX for first-order logic

Complexities of model checking and satisfiability for FO have been known for several decades. The combined complexity of model checking for FO is PSPACE-complete by reduction from the problem QBF (Stockmeyer, 1974). The data complexity of model checking for FO is complete for AC^0; moreover, FO captures AC^0 over structures with *BIT* predicate (or arithmetic structures) (Barrington et al., 1990).

Theorem 3.1. *The combined complexity of MX for FO is NEXP-complete; the data complexity is NP-complete. Moreover, MX for FO captures NP.*

Proof. The NEXP-completeness of combined complexity for MX of FO is implicit in the proof of expression complexity of $\exists$SO from (Vardi, 1982). A different proof is presented in (Mitchell & Ternovska, 2005). The NP-completeness and the capturing NP results follow immediately from Fagin's theorem (Fagin, 1974), since MX for a logic $\mathcal{L}$ is equivalent to model checking for $\exists$SO($\mathcal{L}$) by existentially quantifying over the expansion vocabulary. $\square$

Remark 3.2. This also allows us to capture levels of polynomial-time hierarchy: since complexity of MX for Π_i is Σ_{i+1}, MX for Π_i captures Σ^p_{i+1}.

Remark 3.3. In some cases, the only information about the model that is given as an instance for the model expansion is the size of the model (i.e., the instance vocabulary σ is empty). In that case, it is reasonable to give the size of a model as a number in binary notation. This compact representation leads to an exponential increase in complexity (since the structure itself is exponential in the size of the input).

Although data complexity of model expansion for full first-order logic is NP-complete, there are fragments of FO for which model expansion is feasible. In particular, results from (Grädel, 1992) translate to the following results on Horn and Krom sentences.

Definition 3.4. A *universal Horn formula* is a first-order formula consisting of a conjunction of Horn clauses, preceded by universal first-order quantifiers. Here, a Horn clause is a disjunction of literals, at most one of which is a non-negated occurrence of an expansion predicate. A *universal Krom formula* is defined similarly, but the restriction is at most two occurrences of expansion predicates per clause.

Theorem 3.5. *The data complexity of MX for universal Horn and Krom formulae is, respectively, P-complete and NL-complete. Moreover, MX for universal Horn and Krom captures P and NL, respectively, on successor structures.*

4. Complexity of MX for guarded fragments of FO

The guarded fragment GF of FO was introduced by Andréka *et al.* (Andréka et al., 1998). Here, any existentially quantified subformula ϕ must be conjoined with a guard, i.e., an atomic formula over all free variables of ϕ. An alternation-free portion of guarded fixed-point logic is equivalent to Datalog LITE (?). Gottlob *et al.* (Gottlob et al., 2001) extended GF to the k-guarded fragment GF_k where the conjunction of up to k atoms may act as a guard.

An extensive treatment of GSO, the second-order version of GF, is presented in (?). In this paper, the authors show the relationship between monadic and guarded logics and study their relationship on various structures. This corresponds to the complexity of MX for GF. In particular, they show that GSO is strictly more powerful than monadic second-order logic.

The combined complexity of model checking for GF_k is P-complete (Grädel & Otto, 1999; Gottlob et al., 2001). In particular, model checking for GF_k can be done in time $O(ln^k)$, where l is the size of the formula, and n is the size of the structure (Liu & Levesque, 2003). The finite satisfiability problem for GF is $2EXP$-complete (Grädel, 1999b).

In this section, we show that the combined complexity of MX for GF_k, $k \geq 1$, is the same as that for FO, and MX for GF_k, $k \geq 2$, captures NP just as MX for FO does. We also identify a fragment of GF_k, which we denote by RGF_k, such that the combined complexity of MX for RGF_k is NP-complete; a similar-style version of GF was discussed in (?) in the context of Datalog LITE. Although the data complexity of MX for RGF_k is NP-complete, we show that it does not capture NP. As a corollary of our main results, we show that finite satisfiability for GF_2 is undecidable.

Formally, GF_k is defined as follows:

Definition 4.1. The *k-guarded fragment* GF_k of FO is the smallest set of formulae such that
1. GF_k contains atomic formulae;
2. GF_k is closed under Boolean operations;
3. GF_k contains $\exists \bar{x}(G_1 \wedge \ldots \wedge G_m \wedge \phi)$, if the G_i are atomic formulae, $m \leq k$, $\phi \in GF_k$, and each free variable of ϕ appears in some G_i. Here $G_1 \wedge \ldots \wedge G_m$ is called the *guard* of ϕ.

Since GF_k is closed under negation, we may consider universal quantification to be included as an abbreviation in the usual way.

Definition 4.2. In the context of MX, a fragment of GF_k in which all guards are given by the instance structure is denoted RGF_k.

That is, in RGF_k formulas no expansion predicates appear in guards.

Let FO^k denote FO formulae that use at most k distinct variables. Then it is easy to see that any FO^k formula can be rewritten in linear time into an equivalent one in RGF_k, by using atoms of the form $x = x$ as parts of the guards when necessary. For example, the formula $\exists x \exists y[R(x) \wedge E(x, y)]$ can be rewritten into $\exists x \exists y[R(x) \wedge y = y \wedge E(x, y)]$, where R is an instance predicate, and E is an expansion predicate.

Lemma 4.3. [2] *There is a polynomial-time algorithm that, given an arbitrary $\exists SO$ sentence, constructs an equivalent $\exists SO$ sentence whose first-order part is in GF_2.*

Proof. Suppose ϕ is a $\exists SO$ sentence $\exists X_1 \ldots \exists X_m\, \varphi$, where φ is an FO formula. Let l be the size of ϕ, and let k be the width of φ, that is, the maximum number of free variables in any subformula of φ. We introduce k new predicates $U_1, \ldots, U_k$ such that the arity of U_i is i, $1 \leq i \leq k$. Let φ' be the formula obtained from φ by replacing any subformula $\exists \bar{x}\, \psi(\bar{x})$ with $\exists \bar{x}(U_i(\bar{x}) \wedge \psi(\bar{x}))$ and any subformula $\forall \bar{x}\, \psi(\bar{x})$ with $\forall \bar{x}(U_i(\bar{x}) \supset \psi(\bar{x}))$, where i is the length of $\bar{x}$. Let η be the formula

$$\bigwedge_{i=0}^{k-1} \forall x_1 \ldots \forall x_{i+1}(x_1 = x_1 \wedge U_i(x_2 \ldots x_{i+1}) \supset U_{i+1}(x_1 \ldots x_{i+1})).$$

It is easy to see that any model of η interprets U_i as the i-ary universal relation, $1 \leq i \leq k$. Now let ϕ' be the $\exists SO$ sentence $\exists X_1 \ldots \exists X_m \exists U_1 \ldots \exists U_k\ (\varphi' \wedge \eta)$. Clearly, $\varphi \wedge \eta \in GF_2$, and ϕ' is equivalent to ϕ. Also, both φ' and η are of size $O(l^2)$, and hence ϕ' is of size $O(l^2)$. □

Lemma 4.4. *There exists a polynomial-time algorithm that, given a structure M and an $\exists SO$ sentence ϕ, constructs a structure M' and an $\exists SO$ sentence ϕ_M such that the first-order part of ϕ_M is in GF_1, and $M \models \phi$ iff $M' \models \phi_M$.*

Proof. Suppose M is a structure, and ϕ is an $\exists SO$ sentence. Let n be the size of M, and let l be the size of ϕ. For each domain element a of M, we introduce a new constant symbol c_a. Let M' be the structure that expands M by interpreting c_a as a. Let ϕ' be the $\exists SO$ sentence constructed from ϕ as in the proof of the above lemma. Now let ϕ_M be the sentence obtained from ϕ' by replacing each subformula $\forall x_1 \ldots \forall x_{i+1}(x_1 = x_1 \wedge U_i(x_2 \ldots x_{i+1}) \supset U_{i+1}(x_1 \ldots x_{i+1}))$ with

$$\bigwedge_{a \in dom(M)} \forall x_2 \ldots \forall x_{i+1}(U_i(x_2 \ldots x_{i+1}) \supset U_{i+1}(c_a x_2 \ldots x_{i+1})).$$

Clearly, the first-order part of ϕ_M is in GF_1, $M \models \phi$ iff $M' \models \phi_M$, and the size of ϕ_M is $O(l^2 n)$. □

2. Proved independently by Jurai Stacho

Theorem 4.5. *(1) The combined complexity of MX for GF_k, $k \geq 1$ is NEXP-complete. (2) MX for GF_k, $k \geq 2$ captures NP.*

Proof. (1) follows from Lemma 4.4 and that the combined complexity of MX for FO is in NEXP. (2) follows from Lemma 4.3 and that MX for FO captures NP. □

Lemma 4.6 ((Mitchell & Ternovska, 2005))**.** *3-SAT is poly-time reducible to combined MX for RGF_1.*

Proof. Suppose $\Gamma = \{C_1, \ldots, C_m\}$ is a set of 3-clauses. Let A be the structure with universe $\{a, \neg a \mid a \in atoms(\Gamma)\}$ such that A interprets $Clause$ as the set of clauses in Γ and interprets $Complements$ as the set of complementary literals. Let ϕ be

$$\begin{gathered}\forall x \forall y \forall z (Clause(x, y, z) \supset True(x) \vee True(y) \vee True(z)) \\ \wedge \forall x \forall y (Complements(x, y) \supset (True(x) \equiv \neg True(y))).\end{gathered}$$

Clearly, $\phi \in RGF_1$, and Γ is satisfiable iff A can be expanded to a model of ϕ. □

We quote the following result concerning polynomial-time grounding of RGF_k sentences:

Lemma 4.7 ((Patterson et al., 2007))**.** *Combined MX for RGF_k is polytime reducible to SAT.*

Proof. (Patterson et al., 2007) gives an algorithm that, given a structure A and RGF_k sentence ϕ, constructs in $O(l^2 n^k)$ time a propositional formula ψ which is satisfiable iff A can be expanded to a model of ϕ iff ψ, where l is the size of ϕ, and n is the size of A. □

Theorem 4.8. *(1) The combined complexity of MX for RGF_k is NP-complete. (2) The data complexity of MX for GF_1 and RGF_k is NP-complete. (3) MX for RGF_k, and hence also FO^k, does not capture NP.*

Proof. (1) follows from Lemmas 4.6 and 4.7. (2) follows from Lemma 4.6 and that the data complexity of MX for FO is in NP. For (3), since SAT can be decided in nondeterministic $O(n^2)$ time, by Lemma 4.7, MX for RGF_k can be decided in nondeterministic $O(n^{2k})$ time. By Cook's $NTIME$ hierarchy theorem (Cook, 1973), for any $i > 2k$, there is a problem that can be solved in nondeterministic $O(n^i)$ time but not nondeterministic $O(n^{i-1})$ time. Thus there are infinitely many problems in NP that cannot be expressed by MX for RGF_k. □

Theorem 4.9. *The finite satisfiability problem for GF_k, $k \geq 2$ is undecidable.*

Proof. By the proof of Lemma 4.3, finite satisfiability for FO can be reduced to that for GF_2. □

5. Complexity of ID-logic

One disadvantage of first-order logic as a programming language is its lack of mechanism for recursion and induction. Therefore, it is natural to consider extending first-order logic by adding inductive definitions or fixpoint operators. ID-logic (Denecker & Ternovska, 2008) is an extension of classical logic with inductive definitions, in which (non-monotone) definitions can appear as atomic formulae.

Definition 5.1. An *inductive definition* Δ is a set of rules of the form $\forall\bar{x}(X(\bar{t}) \leftarrow \phi)$ where X is a predicate symbol (constant or variable) of arity r, $\bar{x}$ is a tuple of object variables, $\bar{t}$ a tuple of object variables of length r, ϕ is an arbitrary first-order formula.

The semantics of the logic is defined by the standard truth recursion of classical logic, augmented with one additional rule saying that a valuation I satisfies a definition D if it is the 2-valued well-founded model of this definition, as defined in the context of logic programming.

Example 5.1. *Consider the formula* $\Delta_{even}(E) \wedge \forall x(E(x) \vee O(s(x)))$, *where*

$$\Delta_{even} \equiv \left\{ \begin{array}{cl} E(x) & \leftarrow x = 0 \\ E(s(s(x))) & \leftarrow E(x) \wedge \neg E(s(x)) \end{array} \right\}.$$

This formula states that every number is either even or odd. Definition Δ_{even} is one of possible definitions of even numbers, which is total on natural numbers, but not on integers.

Evaluation of such formulae proceeds as follows. Start with upper and lower bounds on the value of the predicate being defined set to $I_0 = \emptyset$ and $J_0 = U^k$. Then, fixing the negated variables to the values given by the upper (lower) bound, compute the next lower (upper) bound by evaluating the resulting positive definition. Repeat until a fixpoint is reached. If there is no fixpoint, we say that the formula containing this definition does not have a model.

Remark 5.2. Note that in the model-checking context all predicates mentioned in a formula of ID-logic, including those defined with inductive definitions, are provided as part of the structure. That is, a formula is true on exactly those structures that provide interpretations for defined predicates that satisfy the definitions.

5.1 Equivalence of model checking for FO(ID) and FO(LFP)

In this section we show that first-order logic with inductive definitions, denoted FO(ID), can be simulated by first-order logic with least fixed point operator, FO(LFP) and vice versa (using additional relational variables). This allows us to transfer known complexity results for FO(LFP) to FO(ID).

Lemma 5.3. *Model checking for FO(ID) can be simulated by model checking for FO(LFP) on the same structure.*

Proof. Since interpretations of all predicates are provided by the structure, each definition can be evaluated independently. Therefore, it is sufficient to show how to encode a single definition Δ (which can have multiple defined predicates) by a FO(LFP) formula. If a definition is not total on I_0, we need to ensure that there is no model for the whole theory. Then we can use evaluated definitions to construct a FO(LFP) formula corresponding to the original FO(ID) formula.

A definition Δ for a given initialization of open (not occurring in a head of any rule) predicates from I_0 is evaluated as follows.

Replace in Δ all occurrences of X_i by X'_i for new variables X'_i. For example, a rule $\forall\bar{x}(X_i(\bar{t}(\bar{x})) \leftarrow \neg X_j(\bar{t}'(\bar{x})))$ becomes replaced with $\forall\bar{x}(X_i(\bar{t}(\bar{x})) \leftarrow \neg X'_j(\bar{t}'(\bar{x})))$ Let ϕ be a formula encoding Δ after this substitution.

Computing one (double) step of the evaluation (a step corresponding to evaluating ϕ with I and then J giving the values for negated literals) becomes

$$\psi \equiv LFP_{\bar{x},\bar{X}}\phi([LFP_{\bar{x},\bar{X}}\phi]^j/X'_j), \tag{*}$$

by semantics of ID-logic. Here fixpoints are simultaneous on all X_i and the notation $[LFP_{\bar{x},\bar{X}}\phi]^j/X'_j$ means replacing the occurrences of X'_j in ϕ with the fixpoint of X_j in the simultaneous least fixed point of ϕ over all $\bar{X}$.

To simplify the presentation assume, using the fact that simultaneous LFP is equivalent to LFP, that a variable X encodes all variables X_i. Then, the simultaneous LFPs from ψ become just LFPs.

Let Y be a variable encoding the fixpoint of X after the double step $(*)$. This variable is used to initialize X'_i before the next double step. Since after each step ψ the variable Y contains the partial truth assignments on structure I after i^{th} (double) step of the evaluation procedure, Y is monotone. Therefore, there exists a fixpoint of Y defined by ψ, and it is the least fixed point. Therefore, the formula $\Psi_\Delta(\bar{u}) \equiv [LFP_{\bar{y},Y}\psi(Y)]\bar{u}$ computes the values of the defined predicates in ϕ whenever the fixpoint exists. This is also true when Y is treated as a list of predicates $X_1 \dots X_k$ being defined in Δ, in which case LFP in Ψ_Δ is a simultaneous fixed point.

It is possible, though, that the value computed using the upper bound estimation (the innermost LFP of the double step $(*)$) is different from the outer LFP in the double step. If this is the case, then the following formula is false ("consistency check"):

$$CONS_\Delta \equiv \forall \bar{z}([LFP_{\bar{y},Y}\psi(Y)]\bar{z} \leftrightarrow LFP_{\bar{x},\bar{X}}\psi(\bar{x}, LFP_{\bar{y},Y}\psi(Y)/X'_i))[\bar{z}] \tag{1}$$

Suppose now that we have FO(ID) formulas with multiple definitions. Let ϕ' be a first-order formula with occurrences of definitions $\Delta_1 \dots \Delta_m$ for some m. To simplify the presentation, view each definition as defining one predicate P_i. If the fixpoint of Δ_i exists, then $\forall\bar{x}(P_i(\bar{x}) \leftrightarrow \Psi_{\Delta_i}(\bar{x}))$, so occurrences of P_i in ϕ' can be treated as occurrences of Ψ_{Δ_i}. From the point of view of evaluation, it is more efficient to compute $P_i \equiv \Psi_{\Delta_i}$ and then refer just to P_i.

Finally, ϕ' is converted to a formula

$$\Phi \equiv \bigwedge_{i=1}^{m} CONS_{\Delta_i} \wedge \phi'((\forall\bar{x}(P_i(\bar{x}) \leftrightarrow \Psi_{\Delta_i}(\bar{x})))/\Delta_i)$$

That is, Φ is a conjunction of two parts: the conjunction of consistency formulae which ensures that all definitions were total, and ϕ', which is the same as the original formula except all definitions are replaced by the FO(LFP) formulae computing them.

The resulting formula is in FO(LFP), which completes this direction of the proof. □

Example 5.2. *Recall the formula from example 5.1 stating that every number is either even or odd. The following describes a construction of an equivalent FO(LFP) formula.*

A formula corresponding to Δ_{even} becomes, replacing $\neg E$ with $\neg E'$,

$$\{(\phi_E(x, E, E') \equiv (\exists y(x = y \wedge y = 0)) \vee (\exists y(x = s(s(y)) \wedge E(y) \wedge \neg E'(s(y))))\}.$$

Define $\psi_E(z, E') \equiv [LFP_{x,E}\phi_E(x, E, LFP_{E,x}\phi_E(x, E, E')))]z$. *This computes one iteration of the stable operator* ST^2_Δ*, which corresponds to obtaining new pair of values for* I *and* J*.*

Now, $\Psi_\Delta \equiv LFP_{z,E'}\psi_E(z, E')$*. Consistency is checked by the formula* $\forall u\Psi_\Delta(u) \leftrightarrow [LFP_{x,E}\psi_E(x, E, \Psi_\Delta)]u$*. Now, the final formula becomes*

$$(\forall u\Psi_\Delta(u) \leftrightarrow [LFP_{x,E}\psi_E(x, E, \Psi_\Delta)]u) \wedge (\forall x(P(x) \leftrightarrow \Psi_\Delta(x)) \wedge (P(x) \vee O(s(x))).$$

Here, the first conjunct checks that the definition "makes sense", otherwise the formula does not have a model, the second part is a syntactic sugar defining a particular variable $P(x)$ *to represent the defined* E*, and the last part uses* P *outside of the definition* Δ_E*.*

Lemma 5.4. *For every formula* ϕ *of FO(LFP) and a structure* A *there is a formula* ϕ' *of ID-logic such that* ϕ *holds on* A *iff* ϕ' *holds on* A *extended by the relational variables interpreted as the names for the fixpoints.*

Proof. By (Ebbinghaus & Flum, 1995) theorem 9.4.2, every FO(LFP) formula is equivalent to one of the form $\forall u[LFP_{\bar{z},Z}\psi]\tilde{u}$, where $\psi \in \Delta_2$. This can be written as an ID-logic formula $\{Z(\bar{z}) \leftarrow \psi\} \wedge \forall u Z(\tilde{u})$. Now, whenever a structure A is a model of an FO(LFP) formula ϕ, a structure A^+ is a model of the equivalent formula ϕ' of ID-logic. Here, A^+ is A together with a relation variable Z of the arity $|\bar{z}|$, and A^+ interprets Z to be the $LFP_{\bar{z},Z}\psi$. □

Therefore, the following holds:

Theorem 5.5. *The complexity of model checking of ID-logic and FO(LFP) coincide over finite structures.*

Proof. The direction from ID-logic to FO(LFP) follows immediately from lemma 5.3. For the other direction, to check a FO(LFP) formula ϕ on a structure A, use the ID-logic evaluation algorithm on ϕ' to compute the (unique, if it exists) value of Z, then evaluate ϕ' on A^+ with this Z. □

Corollary 5.6. *Combined complexity of the model checking for FO(ID) is complete for EXP. Data complexity of model checking for FO(ID) is complete for P.*

5.2 Complexity of MX for FO(ID)

Intuitively, adding polynomial-time computable predicates under the second-order quantifier of an NP predicate should not add any extra power.

This suggests that both combined and data complexity of model checking for FO(ID) (or, equivalently, FO(LFP)) should coincide with the corresponding complexities of MX for FO.

Theorem 5.7. *Combined complexity of MX for FO(ID) is NEXP-complete. MX for FO(ID) captures NP.*

Proof. We know from Theorem 3.1 that data complexity of MX is hard for NP and the combined complexity is hard for NEXP. Therefore, it is sufficient to show membership in these classes.

The evaluation algorithm proceeds as follows. Use non-determinism to guess the expansion predicates. Now the problem is reduced to evaluating FO(ID) formula on an expanded structure. By the equivalence of FO(ID) with FO(LFP), his can be done in polynomial time of the size of the structure when formula is fixed (by (Immerman, 1982; Vardi, 1982; Livchak, 1982)) and in exponential time when the formula is a part of the input by (Vardi, 1982). In the second case, the size of the expansion predicates can be exponential in the size of the structure (since their arity is not constant), but in NEXP we can guess exponential-size certificates. □

5.2.1 Fragments of FO(ID) with polynomial-time MX.

Recall that MX for universal Horn formulae was P-complete. We would like to add inductive definitions to such formulae so that the complexity of the resulting logic is still in P. The following example shows that allowing unrestricted use of expansion predicates in the inductive definitions makes it possible to encode NP-complete problems.

Example 5.3. *Recall the formula for 3-colourability in example 1.1 on page 1. The only part of this formula which is not Horn is the first disjunction. It can be replaced by the inductive definition with a rule $X(i) \leftarrow Q(i)$ for every colour Q. Now, the first disjunction is equivalent to $\forall v X(v)$. Note that the definition of ID-logic requires that such X were minimal, therefore, this does not introduce spurious positives.*

However, if we disallow any occurrences of the expansion predicates inductive definitions, P-completeness is preserved.

Lemma 5.8. *Adding inductive definitions to universal Horn formulae defined on page 9 preserves data complexity of MX problem to be P-complete, when expansion predicates do not occur in inductive definitions.*

Proof. By theorem 3.5, data complexity of MX problem for universal Horn formulae is P-complete. Therefore, a polynomial-time algorithm for MX of universal Horn formulae can first evaluate all inductive definitions, and then run Grädel's algorithm for evaluating existential second-order Horn formulae replacing all defined predicates by their computed values. □

We can also add expansion predicates in a restricted fashion. First, all expansion predicates occurring in definitions have to be defined (i.e, occur in a head of a rule of some definition). Second, such predicates cannot be defined in terms of each other unless they are in the same definition. Third, the definitions can only occur as conjunction to the rest of the formula. Intuitively, in this case, if expansion predicates in the body of a definition are either given values already, or are being defined in that definition, then the definition can be evaluated. The intuition here is similar to the intuition of RGF_k.

Definition 5.9. Let $\{\bar{X}_1, \ldots, \bar{X}_k\}$ be all expansion predicates occurring in a first-order formula ϕ. Then ϕ is in *RFO(ID)* if (1) for each $\bar{X}_i$ there is a definition Δ_i defining all predicates in $\bar{X}_i$, and Δ_i is conjuncted with the rest of the formula. (2) The only expansion predicates allowed in the body of Δ_i are among $\bar{X}_1, \ldots, \bar{X}_{i-1}$; the body of Δ_1 contains no expansion predicates.

More generally, ϕ is in RuHorn(ID) if there are also expansion predicates $\bar{P}$ which do not occur in the definitions and with all definitions removed, ϕ is universal Horn with respect to $\bar{P}$

Theorem 5.10. *MX problem for RFO(ID) is P-complete.*

Corollary 5.11. *MX problem for RuHorn(ID) is P-complete.*

6. Conclusion and open problems

In this paper, we give a survey of complexity results related to the model expansion framework.

The implication of the kind of research we present in this paper is broader than the MX project. The study of complexity-theoretic implications of extending FO with inductive definitions or restricting it to GF_k is applicable for other constraint languages. We believe that our results will help developing existing constraint languages by e.g. informing the reader which constructs can and cannot be added if the language targets a particular complexity class.

In order to apply our results to a particular constraints language, one needs to represent it as model expansion for a particular logic, and then to analyse that logic for expressiveness. For example, it is easy to relate the results presented in this paper with the language of ESSENCE (?), since it representation as MX has been done (?)

In addition, this research may help the reader to understand what constitutes the core part of a language, i.e., a fragment which is sufficient to do everything the entire language can do. In a sense, it will inform a language developer which constructs can be deleted from the language without losing the expressive power. Such understanding is important for solver development — it is much easier to develop a core solver for a fragment of the language, and to deal with additional constructs by expanding them automatically. Also, it is important to know the core language for pre-processing of specifications.

Model expansion is a very new approach. Many problems are still unsolved, both theoretical and practical. From the theoretical point of view, it would be interesting to extend our complexity results by looking at the model expansion versions of other commonly used logic. Also, we know that $GF(ID)$ (guarded logic with inductive definitions) coincides with μGF on total structures; however, the question is still open whether they coincide everywhere, like FO(ID) and FO(LFP). There, the problem lies in a different treatment of inductive definitions that are not total. A very interesting theoretical question is to find a logic capturing $\mathrm{NTIME}(n^k)$. We believe such a logic will be closely related to GF_k.

References

Andréka, H., van Benthem, J., & Németi, I. (1998). Modal languages and bounded fragments of predicate logic. *J. Phil. Logic*, *49*(3), 217–274.

Barrington, D. M., Immerman, N., & Straubing, H. (1990). On uniformity within NC^1. *Journal of Computer and System Sciences*, *41*(3), 274 – 306.

Cadoli, M., Ianni, G., Palopoli, L., Schaerf, A., & Vasile, D. (2000). NP-SPEC: An executable specification language for solving all problems in NP. *Computer Languages*, *26*, 165–195.

Cadoli, M., & Mancini, T. (2002). Combining Relational Algebra, SQL, and constraint programming.. In *Proc., 4th International Workshop on Frontiers of Combining Systems (FroCos '02), Lecture Notes in Artificial Intelligence, Volume 2309*, pp. 147–161. Springer.

Cook, S. A. (1973). A hierarchy for nondeterministic time complexity.. *Journal of Computer and System Sciences*, *7*(4), 343–353.

Denecker, M. (2000). Extending classical logic with inductive definitions. In *Proc. CL'2000*.

Denecker, M., & Ternovska, E. (2004a). A logic of non-monotone inductive definitions and its modularity properties. In *Proc., LPNMR-04*.

Denecker, M., & Ternovska, E. (2004b). Inductive situation calculus. In *Proc., KR-04*.

Denecker, M., & Ternovska, E. (2007). Inductive situation calculus. *Artificial Intelligence*.

Denecker, M., & Ternovska, E. (2008). A logic of non-monotone inductive definitions. *ACM transactions on computational logic*. To Appear.

East, D., & Truszczynski, M. (2006). Predicate-calculus based logics for modeling and solving search problems. *ACM Transactions on Computational Logic*, *7*(1), 38–83.

Ebbinghaus, H.-D., & Flum, J. (1995). *Finite model theory*. Springer Verlag.

Enderton, H. B. (2001). *A Mathematical Introduction to Logic*. Harcourt/Academic Press, New York.

Fagin, R. (1974). Generalized first-order spectra and polynomial-time recognizable sets. In Karp, R. (Ed.), *In: Complexity and Computation, SIAM-AMS Proc., 7*, pp. 43–73.

Flener, P., Pearson, J., & Agren, M. (2004). Introducing ESRA, a relational language for modelling combinatorial problems. In *In: M. Bruynooghe (ed), Logic Based Program Synthesis and Transformation: 13th International Symposium (LOPSTR '03), Revised Selected Papers, Lecture Notes in Computer Science, volume 3018.*, pp. 214–232. Springer.

Gottlob, G., Leone, N., & Scarcello, F. (2001). Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. In *PODS '01*, pp. 195–206.

Grädel, E. (1992). Capturing Complexity Classes by Fragments of Second Order Logic. *Theoretical Computer Science*, *101*, 35–57.

Grädel, E. (1999a). The decidability of guarded fixed point logic. In Gerbrandy, J., Marx, M., de Rijke, M., & Venema, Y. (Eds.), *JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*. Amsterdam University Press.

Grädel, E. (1999b). On the restraining power of guards. *Journal of Symbolic Logic*, *64*, 1719–1742.

Grädel, E. (2002). Guarded fixed point logic and the monadic theory of trees. *Theoretical Computer Science*, *288*, 129–152.

Grädel, E., & Otto, M. (1999). On logics with two variables. *Theoretical Computer Science*, *224*, 73–113.

Grädel, E., & Walukiewicz, I. (1999). Guarded fixed point logic. In *LICS'99*, pp. 45–55.

Immerman, N. (1982). Relational queries computable in polytime. In *14th ACM Symp.on Theory of Computing, Springer Verlag*, pp. 147–152.

Immerman, N. (1999). *Descriptive complexity*. Springer Verlag, New York.

Kolokolova, A. (2004). *Systems of bounded arithmetic from descriptive complexity*. Ph.D. thesis, University of Toronto.

Kolokolova, A., Liu, Y., Mitchell, D., & Ternovska, E. (2006). Complexity of expanding a finite structure and related tasks. In *Logic and Computational Complexity, workshop associated with LICS'06*.

Libkin, L. (2004). *Elements of Finite Model Theory*. Springer.

Liu, Y., & Levesque, H. J. (2003). A tractability result for reasoning with incomplete first-order knowledge bases. In *Proc. of the 18th Int. Joint Conf. on Artif. Intell. (IJCAI)*, pp. 83–88.

Livchak, A. (1982). Languages for polynomial-time queries. In *Computer-based modeling and optimization of heat-power and electrochemical objects*, p. 41.

Mariën, M., Mitra, R., Denecker, M., & Bruynooghe, M. (2005). Satisfiability checking for PC(ID). In *Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2005, Proceedings*, Vol. 3835 of *Lecture Notes in Computer Science*, pp. 565–579. Springer.

Mitchell, D. G., & Ternovska, E. (2005). A framework for representing and solving NP search problems. In *Proc. AAAI'05*, pp. 430–435.

Mitchell, D., Ternovska, E., Hach, F., & Mohebali, R. (2006). Model expansion as a framework for modelling and solving search problems. *SFU technical report*, *TR 2006-24*.

Patterson, M., Liu, Y., Ternovska, E., & Gupta, A. (2007). Grounding for model expansion in k-guarded formulas with inductive definitions. *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*.

Pelov, N., & Ternovska, E. (2005). Reducing inductive definitions to propositional satisfiability. In *Proc., ICLP-05*, pp. 221–234.

Stockmeyer, L. (1974). *The Complexity of Decision Problems in Automata Theory*. Ph.D. thesis, MIT.

Trahtenbrot, B. (1950). The impossibility of an algorithm for the decision problem for finite domains. *Doklady Academii Nauk SSSR*, *70*, 569–572. In Russian.

Vardi, M. Y. (1982). The complexity of relational query languages. In *14th ACM Symp.on Theory of Computing, Springer-Verlag*.

Vardi, M. Y. (1995). On the complexity of bounded-variable queries. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California*, pp. 266–276. ACM Press.