Model Expansion as a Framework for Modelling and Solving Search Problems

SFU Computing Science Technical Report TR 2006-24

DAVID MITCHELL, EUGENIA TERNOVSKA, FARAZ HACH, RAHELEH MOHEBALI
Simon Fraser University, Canada

We propose a framework for modelling and solving search problems using logic, and describe a project whose goal is to produce practically effective, general purpose tools for representing and solving search problems based on this framework. The mathematical foundation lies in the areas of finite model theory and descriptive complexity, which provide us with many classical results, as well as powerful techniques, not available to many other approaches with similar goals. We describe the mathematical foundations; explain an extension to classical logic with inductive definitions that we consider central; give a summary of complexity and expressiveness properties; describe an approach to implementing solvers based on grounding; present grounding algorithms based on an extension of the relational algebra; describe an implementation of our framework which includes use of inductive definitions, sorts and order; and give experimental results comparing the performance of our implementation with ASP solvers and another solver based on the same framework.

1. INTRODUCTION

We propose a framework for representing and solving search problems with logic, and describe work to date toward making this framework the basis for practical tools. A goal of the work is to produce general purpose tools that are effective in practice for a wide range of applications. The mathematical foundation is provided by finite model theory, which offers a body of results and techniques not available to many other approaches with similar goals. The framework can be applied to a variety of complexity classes, and problem modelling consists primarily of axiomatizing the problem in an appropriate logic. The framework was proposed in [Mitchell and Ternovska 2005b; 2005c; 2005a] and certain aspects were further developed in [Kolokolova et al. 2006; Patterson et al. 2006; 2007]. Here, we report on the progress to date. We explain the framework and investigate a number of properties. We also report on an implementation which is primarily directed at NP-search problems. There, modelling is in an extended first-order logic, and solving is based on reduction to the satisfiability problem for propositional logic (SAT).

Cook's theorem [Cook 1971] that every problem in the complexity class NP can be reduced in polynomial time to SAT suggested a scheme for solving problems in NP: (1) For each problem Π, implement a polytime reduction to SAT; (2) Given an

Authors' address: School of Computing Science, Simon Fraser University, 8888 University Drive, Burnaby, BC, Canada. Fax: +(604) 291-3045. email: {mitchell,ter,fhach,rmohebal}@cs.sfu.ca

instance of Π, apply the reduction to produce a propositional formula, then run the best SAT solver available to find a satisfying assignment (thus solution) if there is one. This was almost universally considered an idea of purely theoretical interest, but recent progress in building and applying SAT solvers shows that reduction to SAT can be an effective practical technique.

Fagin's theorem [Fagin 1974] showed that the first step, designing and implementing a reduction, can be replaced with declarative modelling. The theorem says that NP consists exactly of those problems that can be axiomatized in existential second order logic ($\exists$SO). The solving scheme becomes: (1) Axiomatize problem Π with $\exists$SO formula ϕ; (2) Given instance π of Π, check if $\pi \models \phi$. A possible implementation of (2) is by reduction to SAT. Results analogous to Fagin's followed for other complexity classes, most notably for P [Immerman 1982; Vardi 1982; Livchak 1983]. This study of the relationship between logical definability and complexity is called *descriptive complexity* [Immerman 1999]. The results show that logics can be seen as modelling languages for problems in corresponding complexity classes — another idea that has widely been considered of theoretical, but not practical, interest.

The main idea of our approach to search derives from these results, and is very simple. Our method involves two components, a finite structure $\mathcal{A}$, and a formula ϕ over a larger vocabulary. Vocabulary symbols of the formula which are not interpreted by $\mathcal{A}$ are second-order free variables. Search is formulated as the task of expanding the given structure $\mathcal{A}$ to the vocabulary of the formula, producing a model of the formula. We call the task *model expansion*, abbreviated MX.

A natural way to model a search problem is to view ϕ as a problem description. In this view, instances are finite structures. The formula, which is fixed, specifies the relationship between an instance and its solutions. We construct structures $\mathcal{B}$ which expand $\mathcal{A}$ to the vocabulary of ϕ, and satisfy ϕ. The solutions for an instance $\mathcal{A}$ are given by the interpretations in each expansion structure $\mathcal{B}$ of vocabulary symbols which are not interpreted by $\mathcal{A}$. This view clearly separates problem descriptions (formulas in some logic), from instances (finite structures), and matches with the descriptive complexity results mentioned: Fagin's theorem tells us that a search problem can be expressed this way with first-order logic (FO) if, and only if, its associated decision problem is in NP. Combined with a reduction from FO model expansion to SAT, we have a *universal method* for modelling and solving NP-search problems. Figure 1 illustrates the architecture of a solver based on this scheme. Other descriptive complexity results provide similar universal methods for a number of other complexity classes.

The formulation just described has many desirable properties. Yet, there are applications where it seems more convenient to describe certain properties of the instance with a formula, rather than in a structure. In this case, we may consider a *combined MX* setting, in which an instance consists of both a formula and a structure. In this setting, the formula may be a conjunction of too parts, of which one is fixed, and the one varies with the instance. We do not consider this setting at length here, but we do describe it's expressiveness and complexity and give an example in Section 2.

The foundation of our approach in descriptive complexity gives us an important advantage over several similar approaches which are based on logics with non-

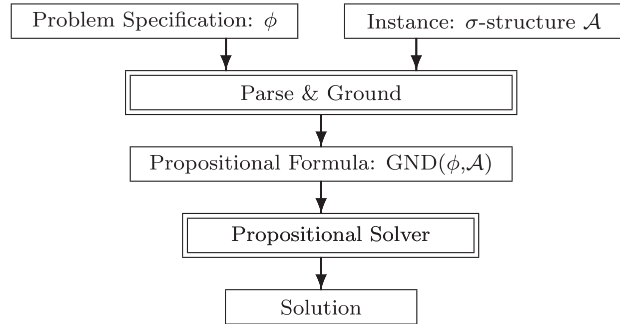

Fig. 1. Solving-by-Grounding Scheme for NP-Search

classical semantics. All similar systems we know of restrict the user's syntax, often quite severely, to ensure efficient implementation. Limits apply, for example, to quantifier alternations, function symbols, formula depth, negation, recursion, etc. In our view, all of these are useful syntactic devices. We prefer to restrict the language available as little as possible, leave the question of how liberally the syntax may be used in practice to practical experience. Allowing very general syntax makes implementation more work, but we are not very concerned that such features will be "over-used", since people, as well as programs, do not deal very well with complex axioms.

We do not suggest language restrictions are never useful, but when we apply them we want specific, measurable benefits. In our framework, descriptive complexity results allow us to fine-tune our languages for desired complexity classes, without imposing ad-hoc restrictions on syntax. For example, we will discuss the benefit to grounding-based solvers of writing axioms in the k-guarded fragment of FO, GF_k [Patterson et al. 2006; 2007].

An interesting property of our approach is that, under certain conditions, MX representations can exactly capture a user's intentions regarding what constitutes a solution. That is, if the user chooses a certain vocabulary for describing instances and their solutions, an MX axiomatization can always be provided using only this vocabulary. While a user may find use of additional "auxiliary" vocabulary symbols handy, their use is not necessary. The property is of practical interest because a solver may have to construct interpretations for all expansion symbols, so use of auxiliary symbols appears to imply more work for the solver. The property is also interesting because it establishes a notion of *capturing search* that is slightly different from the usual notion of capturing a complexity class, which applies to decision problems. In particular, FO model expansion does not capture NP-search in this sense, but FO(LFP) model expansion, in the presence of order, does. It is interesting that adding induction does increase expressive power when when we consider decision problems, but it does when we consider search.

1.1 Solving Search Problems with Logic: Main Ideas

Here, we present a (rather myopic) history of ideas in logic-based approaches to search. We restrict our attention to approaches that are intended as general-purpose methods for representing and solving search problems. We compare a number of implemented systems with ours in Section 8.

1.1.1 *Search as Validity Checking.* Our approach is natural in light of descriptive complexity results, but is not typical. Historically problem solving in logic was often cast as validity checking, as in the influential work of Green [Green 1969]. There, the user axiomatizes the claim that a solution exists, and a resolution theorem prover produces, as a side-effect of proving the claim, a term describing the solution. This idea was the basis of much later work on search in artificial intelligence[1], and on general-purpose planning systems in particular. Such approaches found many obstacles, leading to frequent rejection of predicate logic as a suitable basis for practical problem solving. Standard tasks such as validity and satisfiability are undecidable for first-order logic (FO), although it is not expressive enough even to describe such basic properties as transitive closure or reachability. Attempts to build practical tools often met with serious performance problems.

1.1.2 *Solving by Reduction: Propositional Model Finding.* In contrast to the experience in building general-purpose planning systems based on Green's scheme, it was shown in the 1990's that that reduction to SAT and application of a general-purpose SAT solver was more effective than special-purpose planning programs of the day [Kautz and Selman 1992]. SAT-based techniques are currently the best, or among the best, in several categories of the annual International Planning Competition [Gerevini et al. 2006]. Reduction to SAT has also been shown to be an effective method for model checking (perhaps better called bug-finding) in hardware verification [Biere et al. 1999]. SAT-based "bounded model checking" (BMC) is now standard in the electronic design industry, and of growing importance in software verification. Implementations of other logic-based tools routinely either use a SAT solver as the core engine, or implement variants of SAT solver methods. An attraction of basing systems on SAT solvers is that performance of standard SAT solvers improves regularly, and one can generally "plug in" the latest and best. SAT solvers are currently effective in a number of domains, and recent work suggests much more potential.

A number of related approaches to general problem solving are based on more expressive languages than propositional CNF formulas. These include finite-domain constraint satisfaction (CSP), satisfiability modulo theories (SMT), conjunctions of Pseudo-Boolean constraints, etc. These primarily address the problem that many constraints are not easily or concisely expressed as propositional CNF, but generally follow the same reduction-based paradigm.

1.1.3 *Separation of Instance from Problem Description.* When solving a problem by reduction to SAT, normal practice is to describe the propositional formulas produced by the reduction using informal schemata, and then implement the reduction, typically in C or C++. For problems that are very simple to state, this may

[1] It is also the basis of the Prolog programming language.

be an adequate methodology. However, real applications tend to involve messy or less intuitive features, and producing a correct reduction and implementation may require some effort. Worse, in practice solver efficiency may be very different under different reductions. Choosing among the many basic ideas for a reduction, and the many variants of each, must be done experimentally. Thus, to achieve good performance on challenging instance families, one may need to produce and implement many reductions. Since there are not standardized or formal tools to describe and implement reductions, this is a time-consuming and error-prone process. One suspects that better performance would be obtained over a wider range of applications if such tools were available.

For this and other reasons, researches in several areas have argued for methods which provide modelling languages, in which one describes the problem *independently of particular instances.* A solver takes as input a pair consisting of a problem specification and a distinct instance description. NP-SPEC [Cadoli et al. 2000] was among the first proposals for a logic-based system which makes this distinction. In Answer Set Programming (ASP), some authors have argued for the importance of this separation [Marek and Truszczynski 1999]. Formally, though, both parts are formulas (logic programs), and the distinction is made only as a convention, and it is not fully followed. In CSP, there a number of proposals, such as Essence [Frisch et al. 2007], although there is no widely-used standard to date.

1.1.4 *Expressive Logics as Languages for Modelling Search.* The logics proposed for modelling languages have been primarily non-classical, presumably because of the practical need for recursion. For example, the language of NP-SPEC [Cadoli et al. 2000] is based on Datalog, and ASP [Niemela 1999; Marek and Truszczynski 1999] is based on the language of logic programming with stable model semantics [Gelfond and Lifschitz 1991]. (It is curious that non-logic-based languages, such as ESRA and Essence, typically have no construct to express recursion.)

We believe that classical logic should be used for modelling to the greatest degree possible, and that recursion is essential for natural modelling. ID-logic is a logic which augments classical logic with a non-classical construction to represent induction, developed in [Denecker 2000; Denecker and Ternovska 2004b; 2007b]. The construct allows for natural modelling 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 and Ternovska 2004a; 2007a], and as such is a good knowledge representation language. FO(ID) is the fragment of ID-logic in which the classical part is FO. We proposed this as the logic of choice for modelling NP-search problems as model expansion in [Mitchell and Ternovska 2005b; 2005c]. East and Truszczynski's system ASPPS [East and Truszczynski 2004] may be viewed as a precursor of our work in the sense that it is the first implemented system for solving NP-search problems whose modelling language may be seen as a restriction of FO(ID) to a language of propositional schemata with rule-based notation and positive induction.

1.1.5 *Herbrand versus non-Herbrand Model Finding.* All general-purpose logic-based systems for search we know of are based on Herbrand model finding. This

approach has its roots in automated theorem proving, and became prevalent in logic programming. In the presence of function symbols, many computational problems, including model expansion, become undecidable if formulated in terms of Herbrand models. This is one reason we did not adopt Herbrand reasoning in our proposal.

1.1.6 *Capturing Complexity Classes.* Almost all known to us logic-based systems which express exactly the search problems in NP are based on non-classical logic. Perhaps the first such proposal is due to [Cadoli et al. 2000] called NP-SPEC, which used an extension of Datalog as a modelling language. For results on expressiveness and complexity of ASP we refer the reader to [Marek and Truszczynski 1999; Marek and Remmel 2003; Leone et al. 2006]. We have recently learned about the diploma project by Alban Rrustemi, under the supervision of Anuj Dawar, which explores a solving scheme very similar to ours, ∃SO as the representation language, and a reduction to SAT [Rrustemi 2004].

1.2 MX Solver Implementation

To demonstrate feasibility of our approach, we have implemented a solver for FO(ID)-MX, which — when applied in the parameterized setting — solves precisely the NP-search problems. There are many possible ways of building a solver for FO(ID) MX. One could, for instance, directly implement a search for desired structures. Alternately, one may use a generic reduction to SAT, or some other simple NP-complete problem, and then use a program for solving this problem. Even having made this choice, there are several options for the "target problem", and for how to carry out the reduction. For example, FO(ID)-MX could be reduced to SAT, but a reduction that captures the semantics of non-monotone inductive definitions is non-trivial (See [Pelov and Ternovska 2005]). Alternately, one could reduce to an extension of propositional logic with inductive definitions, but building a propositional solver for this language is complex (See [Mariën et al. 2005; Mariën et al. 2006]).

In our current implementation, called MXG, we perform a reduction to SAT, and side-step the complexity of dealing with arbitrary inductive definitions by handling only a "well-behaved" fragment, in which each definition can either be used to compute the defined predicate during grounding, or replaced with its completion [Clark 1978], which is a classical formula. The features of our implementation include multiple sorts, ordered structures, bounded quantifiers, etc.

1.3 Paper Outline

In Section 2 we give formal preliminaries and key definitions, and examine some basic properties of model expansion. We review FO(ID), the extension of FO with inductive definitions and describe the main features of our approach which include the use of multi-sorted logic and ordered structures. In the Section 3, we defined and explain our notion of *capturing search.* Section 4 defines and discusses grounding for MX, describes a grounding algorithm based on an extension of the relational algebra, and explains the result of [Patterson et al. 2007] on grounding for the k-guarded fragment. Section 5 presents what we know about the complexity and expressiveness of MX for various fragments and extensions of FO, along with these same properties for the related tasks of finite satisfiability and model checking as

described in [Kolokolova et al. 2006]. Section 6 describes our current implementation. Section 7 provides a comparison of the performance of our implementation with ASP systems and with MidL [Mariën et al. 2005; Mariën et al. 2006], another solver for FO(ID) model expansion. Section 8 compares our system with a number of related systems, and Section 9 offers some concluding remarks and discusses future work.

2. OUR PROPOSAL: THE MODEL EXPANSION FRAMEWORK

2.1 Formal Preliminaries

A vocabulary is a set τ of relation and function symbols, each with an associated arity. Constant symbols are zero-ary function symbols. A structure $\mathcal{A}$ for vocabulary τ (or, τ-structure) is a tuple containing a universe A, and a relation (function) for each relation (function) symbol of τ. If R is a relation symbol of vocabulary τ, the relation corresponding to R in a τ-structure $\mathcal{A}$ is denoted $R^{\mathcal{A}}$. For example, we write

$$\mathcal{A} := (A;\ R_1^{\mathcal{A}}, \dots R_n^{\mathcal{A}},\ c_1^{\mathcal{A}}, \dots c_k^{\mathcal{A}},\ f_1^{\mathcal{A}}, \dots f_m^{\mathcal{A}}),$$

where the R_i are relation symbols, f_i function symbols, and constant symbols c_i are 0-ary function symbols. The size of a structure $\mathcal{A}$ is the number of elements in its universe, denoted $|A|$. For a formula ϕ, we write $vocab(\phi)$ for the collection of exactly those function and relation symbols which occur in ϕ. For simplicity, we restrict our presentation to the case with no function symbols. However, all stated properties and results hold in their presence as well, with the possible exception of those in Section 4.2 on grounding for GF_k, the k-guarded fragment of FO. If $\mathcal{A} = (A; \vec{R}^{\mathcal{A}})$ and $\mathcal{B} = (A; \vec{R}^{\mathcal{A}}, \vec{S}^{\mathcal{B}})$, then $\mathcal{B}$ is an expansion of $\mathcal{A}$ to vocabulary $\vec{R} \cup \vec{S}$. For further terms and results from finite model theory, we refer the reader to [Immerman 1999; Libkin 2004; Ebbinghaus and Flum 1995]. A detailed discussion of fixpoint logics such as FO(LFP) and FO(IFP) can be found in e.g. [Dawar and Gurevich 2002] and in [Ebbinghaus and Flum 1995].

2.2 Main Definition: Model Expansion (MX)

For any logic $\mathcal{L}$, (e.g., FO, FO(ID), FO(LFP), SO, etc.,) the $\mathcal{L}$ model expansion problem, $\mathcal{L}$-MX, is:

Instance: A pair $\langle \phi, \mathcal{A} \rangle$, where
1) ϕ is an $\mathcal{L}$ formula, and
2) $\mathcal{A}$ is a finite σ-structure, for vocabulary $\sigma \subseteq vocab(\phi)$.

Problem: Find a structure $\mathcal{B}$ such that
1) $\mathcal{B}$ is an expansion of $\mathcal{A}$ to $vocab(\phi)$, and
2) $\mathcal{B} \models \phi$.

The vocabulary $\varepsilon := vocab(\phi) \setminus \sigma$, of symbols not interpreted by the instance structure, is called the *expansion vocabulary.* Symbols of the expansion vocabulary are second-order free variables.

Model expansion for logic $\mathcal{L}$ is equivalent to the task of witnessing the first block of existential quantifiers in $\exists$SO($\mathcal{L}$) model checking, where the $\exists$SO($\mathcal{L}$) is the logic

obtained from $\mathcal{L}$ by allowing existential quantification over relation and function symbols.

Sometimes, such as when we discuss capturing complexity classes, we will consider the related decision problem, where one is interested only in the existence of an expansion but does not need to construct it. In our framework, we always assume a multi-sorted logic[2] with equality. For example, we use two sorts below in Example 2.1. For further examples of multi-sorted MX axiomatizations see Section 7 and [Mitchell and Ternovska 2005a].

2.3 Parameterized and Combined Model Expansion

2.3.1 *Parameterised MX setting.* In this setting, the formula and the instance vocabulary σ are fixed, and an instance consists of a finite σ-structure $\mathcal{A}$. The setting corresponds to the notion of *data complexity* as defined in [Vardi 1982].

Example 2.1 K-Col as parameterized FO MX. *The problem is: Given a graph and a set K of colours, find a proper K-colouring of the graph. We will have two sorts, vertices V and colours K. The instance vocabulary is $\sigma := \{E\}$; The expansion vocabulary $\varepsilon := \{colour\}$, where colour is a function from verticies to colours. The formula is:*

$$\phi \ := \ \forall x \forall y \ (E(x,y) \supset (colour(x) \neq colour(y)))$$

where colour is a free second-order variable. An instance is a structure $\mathcal{A} = (V \cup K; E^{\mathcal{A}})$, and a solution is an interpretation for colour such that

$$\underbrace{(\overbrace{V \cup K; E^{\mathcal{A}}}^{\mathcal{A}}, colour^{\mathcal{B}})}_{\mathcal{B}} \models \phi.$$

Assume a standard encoding of languages as classes of structures as done in descriptive complexity (See, e.g., [Libkin 2004]). The connection of FO model expansion and $\exists$SO logic immediately implies the following theorem.

Theorem 2.1 [Fagin 1974]. *The first-order model expansion problem in the parameterised setting captures* NP.

The property means that FO model expansion is in NP, and moreover that *every (search) problem in NP* can be represented as parameterized FO MX. Thus, FO MX is a *universal framework* for representing NP-search problems. We have shown that adding inductive definitions preserves these properties, while providing modelling convenience not available in FO. This lead us to propose of FO(ID) as the language of choice for our framework [Mitchell and Ternovska 2005b; 2005c].

The following properties for MX in the parameterized setting are immediate from the definition of MX and results in [Grädel 1992] and [Stockmeyer 1977].

- On ordered structures, FO universal Horn MX captures P,
- On ordered structures, FO universal Krom MX captures NL,

[2] For theoretical considerations, this assumption can always be removed by the standard translation.

◦ Π^1_{k-1} MX captures the Σ^P_k level of the Polynomial Hierarchy

Thus, for each of these logics, MX provides a *universal representation scheme for problems in the corresponding complexity classes.*

Notice the requirement of ordered structures above. Ordered structures are those with a total ordering on domain elements. For precise details, we refer the reader to [Ebbinghaus and Flum 1995]. Intuitively, the order is necessary to mimic the computation of a Turing machine on a tape. While for NP the order can be represented by an existentially quantified second-order variable, it is conjectured that there is no logic capturing the class P on arbitrary structures. The conjecture is due to Gurevich, and the positive resolution of this conjecture (i.e., a proof that no such logic exists) would imply P≠ NP. For further discussion and references see [Libkin 2004].

The formal distinction between problem specification (formula ϕ) and instance description (finite structure $\mathcal{A}$) in the parameterised setting has some important consequences. One is that these an be reasoned about separately, by the modeller, and also by a solver which may apply pre-processing methods specific to each. We may also want the languages for describing instances and problems to be different. This is natural, as they specify different sorts of objects, but also practical, as in many applications the problem specification will be carefully refined then used for many instances.

2.3.2 *Combined MX setting.* In this setting, both formula ϕ and the structure $\mathcal{A}$ are part of the instance. The setting corresponds to the notion of *combined complexity* [Vardi 1982]. Here, we may still have a formula which is fixed for all instances, but an instance will consist of a formula together with a finite structure (which may be just the universe). The expansions must satisfy the conjunction of the two formulas.

Example 2.2 Planning as combined FO-MX. *Consider a blocks world planning problem in which we have stacks of blocks on a table and an operation which can move a block to a new location. An instance consists of a description of the initial situation, a description of the goal situation, and bound k on the length of the plan. Both initial and goal situations can easily be given by an instance structure if they are simple. We consider the case where the goal is disjunctive, for example we require that block a is either on the table or on top of block b. Here, it is convenient to describe this with a formula.*

Domains of the instance structure are: objects, *consisting of blocks and a table, and* steps *from* 0 *to the bound k on plan length. Instance vocabulary is: constant* 0*, unary function succ of sort "steps" (both with their standard meaning); table, and binary predicate $InitOn(x, y)$, giving the initial situation. Expansion vocabulary is: ternary predicate $Put(x, y, i)$, which will be the solution (a plan, consisting of one put action for each step), and auxiliary predicates $On(x, y, i)$ and $GoalOn(x, y)$. In Put and On, the first two arguments are of sort objects, and the third is of sort step; both arguments of GoalOn are objects.*

The formula consists of two parts, one fixed and one varying with the instance. The part that varies with the instance specifies the goal conditions, for example that

either block1 or block2 is on the table:

$$(GoalOn(block1, table) \vee GoalOn(block2, table))$$

The fixed part of the formula, or problem axiomatization, consists of the following formulas.
1) The initial situation is as specified,

$$\forall x \forall y \; (InitOn(x, y) \equiv On(x, y, 0)).$$

2) Block x can be moved to y only if nothing is on x or y,

$$\forall x \forall y \forall i \; (Put(x, y, i) \supset \neg \exists z \; On(z, x, i) \wedge \neg \exists w \; On(w, y, i) \wedge On(x, y, succ(i))$$

3) We have inertia: nothing moves except by a Put operation,

$$\forall x \forall y \forall i \; (On(x, y, i) \wedge \neg \exists z \; (Put(x, z, i)) \supset On(x, y, succ(i))).$$

4) The final state satisfies the goal formula:

$$\forall x \forall y (GoalOn(x, y) \equiv On(x, y, k))$$

We also have axioms saying that the table cannot be on anything, and that no more than one block is allowed directly on top of another.

We have the following.

THEOREM 2.2. *The first-order model expansion problem in the combined setting is* NEXPTIME-*complete.*

This is equivalent to a result in [Vardi 1982], and can be shown by an easy reduction from Bernays-Schönfinkel satisfiability or combined complexity of ∃SO over finite structures, as described in [Mitchell and Ternovska 2005b]. With a few exceptions, the remainder of paper focuses on the parameterized setting and NP.

2.4 Model Expansion in the Context of Related Tasks

Model expansion is closely related to the more studied tasks of model checking (MC) and finite satisfiability. In model checking, the entire structure is given and we ask if the structure satisfies the formula; in model expansion part of a structure is given and we ask for an expansion satisfying the formula; in finite satisfiability we ask is any finite structure satisfies the formula. Thus, for any logic $\mathcal{L}$, the complexity of model expansion — in both parameterized and combined cases — lies between that of the other two tasks:

$$\text{MC}(\mathcal{L}) \leq \text{MX}(\mathcal{L}) \leq \text{Satisfiability}(\mathcal{L})$$

In the case of FO, we have the following. In the *parameterized* (data complexity) setting FO MC is in AC_0: capturing AC_0 requires extending the logic, for example to FO(<,BIT) or FO(+,×) (see [Immerman 1999]). FO MX captures NP. In the *combined* setting (combined complexity), FO MC is PSPACE-complete [Stockmeyer 1974], and FO MX is NEXPTIME-complete. FO satisfiability in the finite is undecidable [Trakhtenbrot 1950]. Model expansion avoids undecidability of FO by specifying the finite universe as part of the instance. We discuss the complexity of MC, MX and satisfiability for some fragments and extensions of FO in Section 5.

The spectrum of a sentence ϕ is the set $\{n \in \mathbb{N} \mid \phi$ has a model of size $n\}$. If $\sigma = \emptyset$ and $vocab(\phi) = \{R_1, \ldots, R_m\}$, the spectrum of ϕ can be alternatively viewed as finite models (of the empty vocabulary) of the ∃SO sentence $\exists R_1, \ldots, \exists R_m\ \phi$, by associating a universe of size n with n. Thus, if $\sigma = \emptyset$, model expansion coincides with the spectrum problem. The case of Theorem 2.2 with $\sigma = \emptyset$ is equivalent to the result from [Jones and Selman 1974] that a set $X \subset \mathbb{N}$ is a FO spectrum iff it is in NEXPTIME. In [Fagin 1974] the class of models of an ∃SO formula were called a *generalized spectrum*.

There is a simple relationship between MX and database query answering. A database can be viewed as a finite structure $\mathcal{A}$, and a query as a formula $\psi(\bar{x})$ with free variables $\bar{x}$. Answering this query is equivalent to solving the model expansion problem for $\mathcal{A}$ and $\forall \bar{x}\ (P(\bar{x}) \equiv \psi(\bar{x}))$, where P is the only expansion symbol.

2.5 Sorts and Order

Formally, FO is sufficient to axiomatize every search problem in NP. However, for reasons both practical and theoretical, it is convenient to consider extensions of FO, as well as to focus on structures with certain properties. We always assume a multi-sorted logic, and implicitly extend our vocabulary with a unary predicate D for every sort D. This is convenient for writing understandable axiomatizations, and also helps with efficient grounding. The assumption could be removed with standard re-writing of formulas, using the unary sort predicates. In this section, we discuss two other fundamental variations that are central: restriction to ordered structures, and extension of FO with inductive definitions. These both provide considerable convenience in axiomatizing problems, and also contribute to interesting theoretical properties.

We take all structures to be ordered. More precisely, our vocabulary is always assumed to include the symbols $\{<, MAX, MIN, SUCC\}$. The domain of every sort has a total order $<$, with MAX and MIN denoting the extremal elements of the order, and $SUCC$ being the successor relation. These are parameterized by sort: elements of different sorts are incomparable.

We assume leq,$\neq$, etc. as abbreviations. We also introduce bounded quantifiers, $\forall x \circ y$, and $\exists x \circ y$, where x, y are of the same sort, and $\circ$ is one of the relational operators above. These are also abbreviations, but turn out to be very useful in efficient grounding.

As mentioned in section 2.3.1, using ordered structures is necessary (as far as we know) for capturing lower complexity classes. Similarly, as we explain below, capturing NP-search — a notion slightly different than the standard notion of capturing NP — requires ordered structures. The assumption of order is also extremely useful in practice, both as a convenience in axiomatizing problems and as a property of domains that can be taken advantage of for efficient grounding.

2.6 Inductive Definitions

Many applications require expression of recursive properties, such as reachability, temporal properties of dynamic systems, or minimum sets satisfying an inductive property. Such properties are not easily expressed as FO MX. ID-logic [Denecker 2000; Denecker and Ternovska 2004b; 2007b] is an excellent candidate to address this problem. This logic provides a convenient way to represent inductive

constructions which occur in mathematics and common-sense reasoning, including monotone definitions, (non-monotone) definitions over a well-founded set, and (non-monotone) iterative definitions. We proposed a fragment of this logic, FO(ID), as the language for solving NP-search problems with model expansion, in [Mitchell and Ternovska 2005b; 2005c; 2005a].

2.6.1 *Syntax of FO(ID).* The formulas of FO(ID) are constructed from atoms $P(\bar{t})$, and definitions Δ, closed under the standard use of $\wedge, \vee, \neg, \forall, \exists$. A definition Δ is a set of rules, which we illustrate by example.

Example 2.3 Transitive Closure of an Edge Relation.

$$\left\{ \begin{array}{l} \forall x \forall y\ (TC(x,y) \leftarrow E(x,y)), \\ \forall x \forall y\ (TC(x,y) \leftarrow \exists\ z\ (TC(x,z) \wedge E(z,y)) \end{array} \right\}$$

The syntax is quite general: the body of the rules may be unrestricted FO formulas. Disjunctions of definitions, multiple definitions for the same predicate, etc., are allowed.

2.6.2 *Semantics of ID-logic.* The $\leftarrow$ operator in rules of definitions is the *definitional implication*, with semantics provided by the well-founded semantics from logic programming [Van Gelder et al. 1991]. Satisfaction is defined by the following induction:

$$\begin{array}{lll} A \models P(\bar{t}) & \text{if} & \bar{t}^A \in P^A \\ A \models \phi \circ \psi \text{ (or } \circ \psi) & \text{if} & \circ \in \{\wedge, \vee, \neg, \forall, \exists\}, \text{ by the standard induction} \\ A \models \Delta & \text{if} & A \text{ is the 2-valued well-founded model of } \Delta \end{array}$$

In the context of MX, a definition can be used in multiple ways. For example, the definition of Example 2.3 can be used to compute the transitive closure of an edge relation given in the instance, or to construct an edge relation who's transitive closure has been given in the instance.

3. CAPTURING SEARCH

Consider all pairs (instance, certificate), for a particular problem in NP. The search problem would (trivially) be, given an instance and a problem description, find a certificate. The question is that whether the user can always write an MX axiomatization of the problem so that the expansion predicates are *precisely* the certificates the user was thinking of, and nothing else. This property is important from the perspective of practical problem solving. The smaller the expansion vocabulary is, the less work the grounder and the propositional solver have to do. The question is also interesting from a philosophical perspective — is there a precise sense in which user's intentions can be captured? Fagin's theorem, which equates ∃SO with NP, does not answer this question.

It turns out that if we consider FO(LFP)-MX instead of FO-MX as in Fagin's theorem, then we can *always* guarantee (in the presence of ordering on instance structures) that quantification over the relations representing the original certificates is enough! So, even though fixpoints do not give any extra power when we consider NP-decision problems, they come very handy when we consider search! Here, we formalize the intuitive notion of "capturing intended solutions".

Recall the definition of NP. We say that a binary relation R is *balanced* if $|y| \leq |x|^{O(1)}$ for all x and y satisfying $R(x, y)$. We say that $R(x, y)$ is a *checking relation* if R is balanced and

$$x \in L \quad \text{iff} \quad \exists y \; R(x, y).$$

Then

$$\text{NP} = \{L \in \Sigma^* \mid L \text{ has a polytime checking relation}\}.$$

For a particular NP-problem represented by language L, for all strings x, y such as $R(x, y)$ holds, x is an instance of the problem, and y is a certificate.

Definition 3.1 NP-search. NP-search *is the class of function problems of the following form: Given an input x and a polytime checking relation $R(x, y)$, if there is y satisfying $R(x, y)$, than output any such y, otherwise output 'no'.*

The same class of problem is often referred to "Functional NP, or FNP" in the literature. It is easy to see that each polytime checking relation $R(x, y)$ represents a particular NP-search problem.

In the descriptive complexity setting, consider a particular problem in NP. Instead of the checking relation on strings, consider the corresponding relation on structures $R(\mathcal{A}, \mathcal{C})$, where $\mathcal{A}$ is an instance σ-structure, and $\mathcal{C}$ is a certificate ε-structure, possibly (but not necessarily) over a different universe, and $|\mathcal{C}| \leq |\mathcal{A}|^{O(1)}$. The vocabularies of $\mathcal{A}$ and $\mathcal{C}$ represent user's intentions regarding the instances of the problem and it's solution, respectively. An important question is whether we can write MX axiomatizations which respect these intentions, i.e., the expansion vocabulary is exactly that of $\mathcal{C}$. The answer is essentially yes, although the arity of the symbols in the vocabulary of $\mathcal{C}$ may increase to overcome the difference in the universes. A simple modification in the vocabulary of $\mathcal{C}$ allows us to assume that structures $\mathcal{A}$, $\mathcal{C}$ are over the same universe. To see this, replace the checking relation R above with a new relation R' constructed as follows: replace each pair of structures $(\mathcal{A}, \mathcal{C}) \in R$ with a pair $(\mathcal{A}, \mathcal{C}')$, where $\mathcal{C}'$ is over the same universe as $\mathcal{A}$. To construct $\mathcal{C}'$, mimic the elements of $dom(\mathcal{C})$ with tuples of elements of $dom(\mathcal{A})$ and increase the arity of the relations in $\mathcal{C}'$ when needed.

Lemma 3.2. *If R is balanced, then so is R', i.e.,*
If for all $(\mathcal{A}, \mathcal{C}) \in R$, $|\mathcal{C}| \leq |\mathcal{A}|^{O(1)}$, then for all $(\mathcal{A}, \mathcal{C}') \in R'$, $|\mathcal{C}'| \leq |\mathcal{A}|^{O(1)}$.

Lemma 3.3. *If R is a polytime checking relation, then so is R'.*

So, given a checking relation R for an NP-language L, each pair $(\mathcal{A}, \mathcal{C}) \in R$ can be viewed as a $\sigma \cup \varepsilon$-structure $\mathcal{B}$, which is an expansion of a σ-structure $\mathcal{A}$.

Suppose a checking relation $R(x, y)$ for a problem in NP is given and is represented by a class of $\sigma \cup \varepsilon$-structures $\mathcal{K}$. We would like to know whether we can write an MX axiomatization over $\sigma \cup \varepsilon$ only (without introducing any superfluous relations or functions). This leads us to the notion of capturing NP-search.

Definition 3.4 capturing NP-search. *Let $\mathcal{L}$ be a logic. We say that $\mathcal{L}$* captures NP-search on ordered structures *if, for every class $\mathcal{K}$ of ordered $\sigma \cup \varepsilon$-structures, $\mathcal{K}|_\sigma \in NP$ iff there is $\phi \in \mathcal{L}$ with $vocab(\phi) = \sigma \cup \varepsilon$ such that $\mathcal{K} = Mod(\phi)$.*

Thus, if a logic $\mathcal{L}$ captures NP-search (on ordered structures, see section 2.3.1 and [Ebbinghaus and Flum 1995]), then, for a given search problem in NP represented by a $\sigma \cup \epsilon$-relation R (i.e., a class of ordered $\sigma \cup \epsilon$-structures), there is a formula $\phi \in \mathcal{L}$ such that for every ordered $\sigma \cup \epsilon$-structure $\mathcal{B}$,

$$\mathcal{B} \in R \ \text{ iff } \ \mathcal{B} \ \text{ satisfies } \phi.$$

Moreover, for every $\phi \in \mathcal{L}$ over vocabulary $\sigma \cup \epsilon$, every model $\mathcal{B}$ of ϕ represents a pair (instance, solution) and its membership in the corresponding checking relation can be verified in polytime. Since $\mathcal{B}$ above is a structure expanding a σ-structure $\mathcal{A}$, whenever logic $\mathcal{L}$ captures NP-search, there is an parameterized MX axiomatization ϕ in $\mathcal{L}$ which uses the vocabulary $\sigma \cup \epsilon$ only.

If we observe that every polytime class of relations can be viewed as an NP checking relation (or several, depending upon choice of σ and ε), then we see that the following holds.

THEOREM 3.5. *$\mathcal{L}$ captures NP-search iff $\mathcal{L}$ captures P.*

Thus, we have the following property.

COROLLARY 3.6. *FO(LFP) and FO(IFP) capture NP-search over ordered structures; FO does not capture NP-search.*

The statement follows from the definition of NP, the two lemmas above, and the fact that FO(IFP) captures P [Immerman 1982; Vardi 1986], while FO does not [Ajtai 1983; Furst et al. 1984].

COROLLARY 3.7. *Given an NP-search problem represented by a $\sigma \cup \varepsilon$ checking relation R, there is a parameterized model expansion axiomatization $\phi \in FO(LFP)$ such that the expansion vocabulary is precisely ε and for every σ-structure $\mathcal{A}$,*

$\mathcal{C}$ is a solution to $\mathcal{A}$ iff $\mathcal{B} \in Mod(\phi)$, where $\mathcal{B}$ is an expansion of $\mathcal{A}$ by $\mathcal{C}$,

and all structures are ordered.

There is no such axiomatization in FO over $\sigma \cup \varepsilon$ (since FO does not capture NP-search), but there is one over a larger vocabulary which includes $\sigma \cup \varepsilon$. The latter statement is a direct consequence of Fagin's theorem and the fact that P $\subseteq$ NP. Note that having ordered structures in Corollary 3.7 is essential. Given Gurevich's conjecture mentioned in section 2.3.1, it's unlikely that the corollary holds for arbitrary (unordered) structures.

While FO(LFP) and FO(ID) are related, FO(ID) requires additional vocabulary symbols to refer to each inductively defined relation. The corollary above does not hold for FO(ID), however in this logic, one additional relational symbol is enough. This is by a strait forward rewriting of an LFP expression by a positive inductive definition and by taking into account that by a normal form theorem for FO(LFP), one fixpoint expression is always enough, see e.g. [Ebbinghaus and Flum 1995].

3.0.3 *Eliminating Superfluous Vocabulary.* While Corollary 3.7 says that a user can always write parameterized MX axiomatizations in FO(LFP) without using any additional vocabulary besides $\sigma \cup \varepsilon$, it is unrealistic to expect that no auxiliary symbols will be used. It is therefore desirable to develop automated methods for

eliminating such symbols from axiomatizations. Suppose we start from a FO axiomatization with superfluous symbols. In principle, those symbols can always be replaced by fixpoint expressions which use the symbols of the desired vocabulary only. However, it is not clear how to automate that process, and the benefit of such a rewriting is unclear — the grounder (and possibly the solver) will have to be able to handle such expressions.

An interesting question is under what conditions superfluous vocabulary can be eliminated from FO axiomatizations, without appealing to fixpoint logics. Fully understanding this would answer an open question of Fagin, regarding the number and arities of second order variables required for ∃SO to capture NP [Fagin 1993]. The question is closely related to the problems of recognizing those ∃SO formulas which define NP-complete or polytime classes of structures (see [Medina and Immerman 1994]). This is an interesting topic for further research.

4. SOLVING BY GROUNDING

One approach to solving problems cast as MX is through propositionalization, or grounding, in which the question of existence of an expansion is reduced to propositional satisfiability, or some related propositional problem. Our proposal in [Mitchell and Ternovska 2005b] was, in fact, the result of looking for a way to combine the power of current SAT solvers with a very expressive modelling language.

In this section, we define the notions of grounding and reduced grounding, describe a generic grounding algorithm based on an extended form of the relational algebra, and outline an efficient algorithm for RGF_k, those MX axiomatizations in the guarded fragment GF_k of FO for which all guards are instance relations.

Informally, a grounding is a formula that results from eliminating first order variables by replacing them with constant symbols. To do this, we introduce a new constant symbol for each element of the domain. For domain A, we denote the set (or tuple, depending on context) of such constants by $\tilde{A}$. These are always interpreted by the corresponding domain elements.

In practice, we would like to produce groundings which are as simple as possible. In particular, since the instance structure gives interpretations for the instance vocabulary, all symbols from the instance vocabulary may be "evaluated out", leaving a "reduced grounding".

Definition 4.1 Grounding, Reduced Grounding for MX.
Formula $Gr(\phi, \mathcal{A})$ *is a* grounding *of formula* ϕ *over a* σ*-structure* $\mathcal{A} = (\mathcal{A}, \sigma^{\mathcal{A}})$ *if*

(1) $Gr(\phi, \mathcal{A})$ *is a ground formula over* $\sigma \cup \varepsilon \cup \tilde{A}$*, i.e., it contains no variables; and*
(2) *for every expansion structure* $\mathcal{B} = (\mathcal{A}, \sigma^{\mathcal{A}}, \varepsilon^{\mathcal{B}})$ *over vocab*(ϕ),

$$\mathcal{B} \models \phi \quad \textit{iff} \quad (\mathcal{B}, \tilde{A}^{\mathcal{B}}) \models Gr(\phi, \mathcal{A}),$$

where $\tilde{A}^{\mathcal{B}}$ *denotes the interpretation of the new constants* $\tilde{A}$.

A formula $Gr(\phi, \mathcal{A})$ *is a* reduced grounding *if it is over vocabulary* $\varepsilon \cup \tilde{A}$*, i.e., it does not contain any symbols of the instance vocabulary* σ.

Note that $Gr(\phi, \mathcal{A})$ is still a first-order formula. Also, note that our notion of grounding is distinct from that used in ASP and in logic programming in general.

Whereas we add constants to reflect an explicitly given universe, in ASP the universe is taken to be the set of constants appearing in the formulas, and ground instances of the formulas are produced over the Herbrand universe.

The following naive algorithm constructs a reduced grounding of a first-order sentence ϕ over a structure $\mathcal{A}$ in time $O(n^a)$, where a is the width of the formula (i.e., the maximum number of free variables in any sub-formula). For simplicity, first re-write ϕ in prenex normal form, so that $\phi = Q_1 x_1 \ldots Q_n x_n\ \psi$ where each Q_i is either $\forall$ or $\exists$, and ψ is quantifier free. Now, recursively construct a ground formula $Gnd(\phi, \mathcal{A})$ as follows:

$$Gnd(\phi,\mathcal{A}) = \begin{cases} \phi & \text{if } \phi \text{ is quantifier free,} \\ \bigvee_{a\in\tilde{A}} Gnd(\psi(a/x),\mathcal{A}) & \text{if } \phi \text{ is } \exists x\psi \\ \bigwedge_{a\in\tilde{A}} Gnd(\psi(a/x),\mathcal{A}) & \text{if } \phi \text{ is } \forall x\psi \end{cases}$$

Finally, change the grounding $Gnd(\phi, \mathcal{A})$ into a reduced grounding as follows. For each k-ary instance relation symbol R and each k-tuple $\vec{t}$ of elements of from $\tilde{U}$, if $\vec{t} \in R^{\mathcal{A}}$, (resp. $\vec{(t)} \notin R^{\mathcal{A}}$), replace each occurrence of $R(\vec{t}$ in ϕ with *true* (resp. *false*). Then recursively eliminate all occurrences of *true* and *false* in the obvious way, e.g., replacing $(\psi \wedge \mathit{true})$ with (ψ), etc.

In the parameterized setting ϕ is fixed, so this construction is poly-time. This does not mean it will be efficient enough in practice, and we are interested in ways to improve upon it. However, the following proposition suggests that we cannot simultaneously capture NP and have a grounding algorithm which is asymptotically much better.

PROPOSITION 4.1. *There is no logic $\mathcal{L}$, such that the following two properties hold simultaneously:*

(1) *$\mathcal{L}$-$MX(\sigma, \phi)$ captures NP, and*

(2) *For some natural number k, $\mathcal{L}$-$MX(\sigma, \phi)$ can be reduced to SAT in time $O(n^k)$, where n is the size of the instance structure (i.e., does not depend on ϕ).*

Supposing otherwise, we would have for every class $\mathcal{K}$ of structures in NP, an $O(n^k)$ time reduction to SAT, so membership in $\mathcal{K}$ can be decided in NTIME(n^k). But by the hierarchy theorem [Cook 1973], for every k, there are infinitely many problems in NP which are not in NTIME(n^k).

One *can* obtain better complexity results for particular classes of formulas, as illustrated later in this section, and a number of techniques can be used to produce better performance in practice than the naive algorithm.

4.1 Grounding with Extended Relational Algebra

An approach to obtaining faster grounding algorithm that we have been exploring is to view grounding as a generalization of database query answering. To that end, we consider grounding algorithms based on a generalization of the relational algebra. The algebra is over *extended X-relations*, a generalization of the database notion of X-relations introduced in [Patterson et al. 2007].

In this section, we describe a general grounding algorithm based on this algebra. In Section 6, we describe our implemented grounder, MXG, which uses a variant of this algorithm. In Section 4.2, we describe the algorithm from [Patterson et al.

2007], which has an improved upper bound for formulas in k-guarded form based on the algebra.

Definition 4.2 [Patterson et al. 2007] extended X-relation. *Let $\mathcal{A} = (A; \sigma^{\mathcal{A}})$, and X be a tuple of variables. An* extended X-relation *$\mathcal{R}$ over A is a set of pairs (γ, ψ) such that*

(1) ψ is a ground formula over $\varepsilon \cup \tilde{A}$ and $\gamma : X \to A$;

(2) for every γ, there is at most one ψ such that $(\gamma, \psi) \in \mathcal{R}$.

One can think of an X-relation $\mathcal{R}$ as a representation of the unique mapping $\delta_{\mathcal{R}}$ from instantiations of variables in X to ground formulas:

$$\delta_{\mathcal{R}}(\gamma) := \begin{cases} \psi & \text{if } (\gamma, \psi) \in \mathcal{R} \\ false & \text{if } \gamma \notin \mathcal{R} \end{cases}$$

where ψ is a ground formula over $\varepsilon \cup \tilde{A}$ and $\gamma : X \to A$.

Given a formula ϕ, and structure $\mathcal{A}$, to produce the grounding of ϕ wrt $\mathcal{A}$, we will consider subformulas χ of ϕ. For each subformula χ, we compute a formula ψ for each instantiation of the free variables of χ, such that each ψ is a reduced grounding of χ under the corresponding instantiation. We call such a representation *an answer to ϕ wrt $\mathcal{A}$.*

Definition 4.3 [Patterson et al. 2007] answer to ϕ wrt $\mathcal{A}$. *Let ϕ be a formula in $\sigma \cup \varepsilon$ with free variables X, $\mathcal{A}$ a σ-structure with domain A, and $\mathcal{R}$ an extended X-relation over $\mathcal{A}$. We say $\mathcal{R}$ is* an answer to ϕ wrt $\mathcal{A}$ *if for any $\gamma : X \to A$, we have that $\delta_{\mathcal{R}}(\gamma)$ is a reduced grounding of $\phi[\gamma]$ over $\mathcal{A}$. Here, $\phi[\gamma]$ denotes the result of instantiating free variables in ϕ according to γ.*

We compute an answer to a formula by computing answers to its subformulas and combining them according to the connective. For example, the answer to formula $D(x, y, z) \wedge E(y, z)$ with respect to $\mathcal{A} = (\{1, 2, 3\}; D^{\mathcal{A}}, E^{\mathcal{A}})$, where $D^{\mathcal{A}} = \{(1, 2, 3), (2, 3, 3)\}$ and E is an expansion predicate is the extended $\{x, y, z\}$-relation:

x	y	z	ψ
1	2	3	$E(2, 3)$
2	3	3	$E(3, 3)$

For the above example $\delta(\gamma) = E(2, 3)$ for the instantiation $\gamma = (1, 2, 3)$ and $\delta(\gamma) = E(3, 3)$ for $\gamma = (1, 2, 3)$. Otherwise $\delta(\gamma) = false$.

4.1.1 *Extended Relational Operators.* We generalize the standard relational operators – join, project, union,tional operators – join, project, union, division – to extended X-relations *wrt* a structure $\mathcal{A}$. The definitions are given below. There is a natural correspondence between the logical connectives and quantifiers and these operations. To compute a reduced grounding of a FO sentence ϕ over structure $\mathcal{A}$, our algorithm will recursively apply the extended relational algebra operators to these extended relations.

In the following, let $\mathcal{R}$ be an extended X-relation, $\mathcal{S}$ an extended Y-relation, $Z \subset X$. Then

$$\begin{array}{lll}
\mathcal{R} \bowtie S & = & \{(\gamma,\psi) |\ \gamma : (X \cup Y) \to A, \gamma|_X \in \mathcal{R}, \gamma|_Y \in S, \psi = \delta_{\mathcal{R}}(\gamma|_X) \wedge \delta_S(\gamma|_Y)\}. \\
\overline{\mathcal{R}} & = & \{(\gamma,\psi) |\ \gamma \in \mathcal{R}, \psi = \neg\delta_{\mathcal{R}}(\gamma)\} \cup \{(\gamma, true) | \gamma \notin \mathcal{R}\}. \\
\mathcal{R} \cup S & = & \{(\gamma,\psi) |\ \gamma : (X \cup Y) \to A, \gamma|_X \in \mathcal{R} \text{ or } \gamma|_Y \in S, \psi = \delta_{\mathcal{R}}(\gamma|_X) \vee \delta_S(\gamma|_Y)\}. \\
\pi_Z(\mathcal{R}) & = & \{(\gamma',\psi) |\ \gamma' : Z \to A, \gamma' = \gamma|_Z \text{ for some } \gamma \in \mathcal{R} \text{ and } \psi = \bigvee_{\{\gamma|_Z = \gamma'\}} \delta_{\mathcal{R}}(\gamma)\}. \\
\mathcal{R} / Z & = & \{(\gamma',\psi) |\ \gamma' : (X - Z) \to A, \text{ for all } \gamma'' : Z \to A \text{ there is } \gamma \in \mathcal{R} \text{ that} \\
& & \quad \gamma|_Z = \gamma'' \wedge \gamma|_{X-Z} = \gamma' \text{ and } \psi = \bigwedge_{\{\gamma|_{X-Z} = \gamma'\}} \delta_{\mathcal{R}}(\gamma)\}.
\end{array}$$

The following property slightly generalizes one given in [Patterson et al. 2007].

Proposition 4.2. *If extended relations* $\mathcal{R}, S$ *are the answers to* ϕ_1, ϕ_2 *over structure* $\mathcal{A}$*, then: (1)* $\mathcal{R} \bowtie S$ *is the answer to* $\phi_1 \wedge \phi_2$*; (2)* $\overline{\mathcal{R}}$ *is the answer to* $\neg\phi_1$*; (3)* $\mathcal{R} \cup S$ *is the answer to* $\phi_1 \vee \phi_2$*; (4)* $\pi_{X-\{x\}}\mathcal{R}$ *is the answer to* $\exists x \phi_1$*; (5)* $\mathcal{R}/\{x\}$ *is the answer to* $\forall x \phi_1$.

This immediately suggests the following algorithm, which recursively applies operators in correspondence with the formula structure.

$$\begin{array}{lll}
\mathrm{Gnd}(\mathcal{A}, P(\overline{x})) & = & \{(\gamma, true) |\ \gamma \in P^{\mathcal{A}}\} \text{ if } P \in \sigma \\
\mathrm{Gnd}(\mathcal{A}, P(\overline{x})) & = & \underline{\{(\gamma, P(\gamma))|}\ \gamma : X \to A\} \text{ if } P \in \varepsilon \\
\mathrm{Gnd}(\mathcal{A}, \neg\psi) & = & \overline{\mathrm{Gnd}(\mathcal{A}, \psi)} \\
\mathrm{Gnd}(\mathcal{A}, \phi \wedge \psi) & = & \mathrm{Gnd}(\mathcal{A}, \phi) \bowtie \mathrm{Gnd}(\mathcal{A}, \psi) \\
\mathrm{Gnd}(\mathcal{A}, \phi \vee \psi) & = & \mathrm{Gnd}(\mathcal{A}, \phi) \cup \mathrm{Gnd}(\mathcal{A}, \psi) \\
\mathrm{Gnd}(\mathcal{A}, \exists \overline{y} \phi) & = & \pi_{\{\overline{x}\} - \{\overline{y}\}} \mathrm{Gnd}(\mathcal{A}, \phi) \\
& & \quad \text{where } \{\overline{x}\} \text{ is the set of variables in the answer of } \mathrm{Gnd}(\mathcal{A}, \phi) \\
\mathrm{Gnd}(\mathcal{A}, \forall \overline{y} \phi) & = & \mathrm{Gnd}(\mathcal{A}, \phi) / \{\overline{y}\}
\end{array}$$

4.2 Grounding k-Guarded Formulas

In this section we sketch the result from [Patterson et al. 2007], giving an efficient algorithm to produce reduced groundings $\mathrm{Gnd}(\phi, \mathcal{A})$ under certain conditions. The conditions are that ϕ is in the k-guarded fragment GF_k [Gottlob et al. 2001] and that the relation symbols that act as guards in ϕ are not expansion symbols. The algorithm runs in $O(\ell^2 n^k)$ time, where ℓ is the size of the sentence, and n is the size of the largest relation in the structure.

The guarded fragment GF of FO was introduced by Andréka *et al.* [Andréka et al. 1998], and has recently received considerable attention. Here any existentially quantified subformula ϕ must be conjoined with a guard, i.e., an atomic formula over all free variables of ϕ. 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. Formally, GF_k is defined as follows:

Definition 4.4. *The* k-guarded fragment GF_k *of FO is the smallest set of formulas that:*

(*1*) *contains all atomic formulas;*

(*2*) *is closed under Boolean operations;*

(*3*) *contains* $\exists \overline{x}(G_1 \wedge \ldots \wedge G_m \wedge \phi)$*, provided the* G_i *are atomic formulas,* $m \leq k$*,* $\phi \in GF_k$*, and the free variables of* ϕ *appear in the* G_i*.*

Here, $G_1 \wedge \ldots \wedge G_m$ is called the guard *of ϕ.*

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

Note that the above mentioned complexity result $O(\ell^2 n^k)$ only applies to k-guarded sentences, not k-guarded formulas in general. To see why notice that $\neg\mathcal{R}(x, y, z)$ is a 1-guarded formula, but cannot be evaluated in $O(n)$ time. However, the same complexity result applies to strictly k-guarded formulas, defined as follows:

Definition 4.5 SGF_k. *The* strictly k-guarded fragment *SGF_k is the fragment of GF_k with formulas of the form $\exists\bar{x}(G_1 \wedge \ldots \wedge G_m \wedge \phi)$.*

Here, we are including the degenerate cases where $\bar{x}$ is empty (no leading existential quantifier), $m = 0$ (no free variables and therefore no guards), or ϕ is $true$. Thus any k-guarded sentence is strictly k-guarded (take $\bar{x}$ as empty and $m = 0$).

Definition 4.6 RGF_k. *RGF_k denotes the set of FO-MX axiomatizations ϕ which are k-guarded formulas for which no expansion predicate appears in any guard.*

The restriction that "no expansion predicate appears in any guard" is necessary for polynomial-time grounding. Indeed, there is no polytime grounding algorithm for 1-guarded sentences. Otherwise, we would have a poly-time reduction to SAT from FO-MX for 1-guarded sentences, and hence the combined complexity would be in NP. However, it is NEXP-complete [Vardi 1982].

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 guards when necessary. For example, the formula $\exists x \exists y E(x, y)$ can be rewritten as $\exists x \exists y[x = x \wedge y = y \wedge E(x, y)]$; and the formula $\exists x \exists y[R(x) \wedge E(x, y)]$ can be rewritten as $\exists x \exists y[R(x) \wedge y = y \wedge E(x, y)]$, where R is an instance predicate, and E is an expansion predicate.

Extending the result of [Kolaitis and Vardi 1998], Flum *et al.* [Flum et al. 2002] showed that FO^k, the set of FO sentences with at most k distinct variables, has the same expressive power as FO formulas of treewidth bounded by k. The fragment RGF_k contains FO^k, and thus contains (in terms of expressiveness) FO sentences with treewidth less than k, according to the logical characterization of treewidth [Kolaitis and Vardi 1998; Flum et al. 2002]. Thus any FO formula with treewidth less than k is equivalent to one in RGF_k. In practice, most sentences have small treewidth — people prefer to write axioms which are easy to understand. Database query evaluation for guarded fragments can be done more efficiently (in polytime) than for unrestricted FO (e.g. [Flum et al. 2002]), and similarly, the grounding algorithm of [Patterson et al. 2007] performs better in case of guarded formulas.

The efficiency of the algorithm comes from this: Whenever we perform an $\mathcal{R} \bowtie \mathcal{S}$ or $\mathcal{R} \bowtie^c \mathcal{S}$ operation, the set of variables of $\mathcal{S}$ is a subset of that of $\mathcal{R}$. Thus, the set of tuples of the resulting extended relation is a subset of that of $\mathcal{R}$. Both operations can be done in time $O(nm)$, where n is the size of the relation part of $\mathcal{R}$, and m is the maximum size of a formula in $\mathcal{R}$ or $\mathcal{S}$. The algorithm also ensures, by introducing new propositional symbols, that for any join $\mathcal{R} \bowtie \mathcal{S}$ or $\mathcal{R} \bowtie^c \mathcal{S}$, every formula in $\mathcal{R}$ or $\mathcal{S}$ is of size $O(l)$, where l is the size of the formula to be grounded. So, each join or join with complement is done in linear time.

An Algorithm for Grounding k-Guarded Formulas. If ϕ is an atomic formula $R(\bar{t}\,)$, we use $\phi(\mathcal{A})$ to denote the extended relation $\{(\gamma, true) \mid \bar{t}\,[\gamma] \in R^{\mathcal{A}}\}$.

Procedure $\mathrm{Gnd}(\mathcal{A}, \phi)$
Input: A structure $\mathcal{A}$ and a formula $\phi \in \mathrm{RGF}_k$
Output: An answer to ϕ wrt $\mathcal{A}$

Suppose $\phi(\bar{x}) = \exists\bar{y}(G_1 \wedge \ldots \wedge G_m \wedge \psi)$. Return $\pi_{\bar{x}}\mathrm{Gnd}(\mathcal{A}, \mathcal{R}, \psi')$, where $\mathcal{R}$ is $G_1(\mathcal{A}) \bowtie \ldots \bowtie G_m(\mathcal{A})$, and ψ' is the result of pushing $\neg$'s in ψ inward so that they are in front of atoms or existentials.

Given a structure $\mathcal{A}$ and a formula $\phi \in \mathrm{RGF}_k$, suppose $\phi(\bar{x}) = \exists\bar{y}(G_1 \wedge \ldots \wedge G_m \wedge \psi)$. Let $\mathcal{R}$ be $G_1(\mathcal{A}) \bowtie \ldots \bowtie G_m(\mathcal{A})$, and let ψ' be the result of pushing $\neg$'s in ψ inward so that they are in front of atoms or existentials. Return $\pi_{\bar{x}}\mathrm{Gnd}(\mathcal{A}, \mathcal{R}, \psi')$ with each formula replaced by a new propositional symbol. We also save the definitions of these new symbols.

Procedure $\mathrm{Gnd}(\mathcal{A}, \mathcal{R}, \phi)$ is defined recursively by:

(1) If ϕ is a positive literal of an instance predicate, then $\mathrm{Gnd}(\mathcal{A}, \mathcal{R}, \phi) = \mathcal{R} \bowtie \phi(\mathcal{A})$;
(2) If ϕ is $\neg\phi'$, where ϕ' is an atom of an instance predicate, then $\mathrm{Gnd}(\mathcal{A}, \mathcal{R}, \phi) = \mathcal{R} \bowtie^c \phi'(\mathcal{A})$;
(3) If ϕ is a literal of an expansion predicate, then $\mathrm{Gnd}(\mathcal{A}, \mathcal{R}, \phi) = \{(\gamma, \phi[\gamma]) \mid \gamma \in \mathcal{R}\}$.
(4) $\mathrm{Gnd}(\mathcal{A}, \mathcal{R}, (\phi \wedge \psi)) = \mathrm{Gnd}(\mathcal{A}, \mathcal{R}, \phi) \cap \mathrm{Gnd}(\mathcal{A}, \mathcal{R}, \psi)$;
(5) $\mathrm{Gnd}(\mathcal{A}, \mathcal{R}, (\phi \vee \psi)) = \mathrm{Gnd}(\mathcal{A}, \mathcal{R}, \phi) \cup \mathrm{Gnd}(\mathcal{A}, \mathcal{R}, \psi)$.
(6) $\mathrm{Gnd}(\mathcal{A}, \mathcal{R}, \exists\bar{y}\phi) = \mathcal{R} \bowtie \mathrm{Gnd}(\mathcal{A}, \exists\bar{y}\phi)$;
(7) $\mathrm{Gnd}(\mathcal{A}, \mathcal{R}, \neg\exists\bar{y}\phi) = \mathcal{R} \bowtie^c \mathrm{Gnd}(\mathcal{A}, \exists\bar{y}\phi)$.

THEOREM 4.7 [PATTERSON ET AL. 2007]. *Given a structure $\mathcal{A}$ and a formula $\phi \in RGF_k$, Gnd returns an answer to ϕ wrt $\mathcal{A}$ in $O(\ell^2 n^k)$ time, where ℓ is the size of ϕ and n is the size of $\mathcal{A}$. (Hence, if ϕ is a sentence, Gnd returns a reduced grounding of ϕ over $\mathcal{A}$.)*

The paper [Patterson et al. 2007] also describes a grounding procedure for a guarded fragment of ID-logic, which we will not discuss here.

5. EXPRESSIVENESS & COMPLEXITY OF MX AND RELATED TASKS

In this section we present a summary of most of what we know about the complexity and expressiveness of model expansion for FO and a variety of extensions and restrictions. For context, we also included the corresponding properties of the related, and more studied, tasks of finite satisfiability and model checking. The properties are shown in Table I. We restrict our discussion here to an explanation of the table and a few other remarks, and refer the reader to [Kolokolova et al. 2006] for proofs and further discussion.

For a given logic $\mathcal{L}$, we consider complexity of three problems.

(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 $\mathcal{A} \models \phi$?

Table I. Table of Complexity and Expressiveness Properties

Logic	Model checking		Model expansion		Satisfiability
	Combined	Data	Comb.	Data	
FO	PSPACE-c [S74]	$\equiv_{BIT}$ AC^0 [BIS90]	NEXP-c	$\equiv$NP [Fag74]	Und [Tra50]
FO(LFP)	EXP-c [Var82]	$\equiv_s$P [I82,V82,L82]	NEXP-c	$\equiv$ NP	Und.
FO(ID)	EXP-c [KLMT06]	$\equiv_s$P [KLMT06]	NEXP-c	$\equiv$ NP	Und.
FO^k	P-c	AC^0	NP-c [Var95]	NP-c,$\not\equiv$ NP	$Und_{(k>2)}$ $NEXP\text{-}c_{(k=2)}$ $EXP\text{-}c_{(k=1)}$
GF_k	P-c [GO99,GLS01]	AC^0	NEXP-c	$\equiv$ $NP_{(k>1)}$, $NP\text{-}c_{(k=1)}$	$Und_{(k\geq 2)}$ $2EXP\text{-}c_{(k=1)}$ [Gra99]
RGF_k	*n.a.*	*n.a.*	NP-c	NP-c, $\not\equiv$ NP	*n.a.*
μGF	UP$\cap$ co-UP	P	NEXP-c	NP-c	2EXP-c [GW99]
GF_k(ID)	EXP	P	NEXP-c	$\equiv NP_{(k>1)}$	$Und_{(k>1)}$

(2) *Model Expansion* (MX): given $(\mathcal{A}, \phi)$, where ϕ is a sentence in $\mathcal{L}$, $\mathcal{A}$ is a finite σ-structure and $\sigma \subseteq vocab(\phi)$, is there $\mathcal{B}$ for $vocab(\phi)$ which expands $\mathcal{A}$ and $\mathcal{B} \models \phi$?

(3) *Finite Satisfiability*: given a sentence ϕ in $\mathcal{L}$, is there a finite $\mathcal{A}$ for $vocab(\phi)$ such that $\mathcal{A} \models \phi$?

For model checking and model expansion we consider two notions of complexity, introduced by [Vardi 1982]. (Here we follow the presentation of [Libkin 2004].) Let $enc()$ denote some standard encoding of structures and formulas by binary strings.

DEFINITION 5.1. *Let K be a complexity class and $\mathcal{L}$ a logic.*

The data complexity *of $\mathcal{L}$ is K if for every sentence ϕ of $\mathcal{L}$ the language $\{enc(\mathcal{A}) | \mathcal{A} \models \phi\}$ belongs to K. The* combined complexity *of $\mathcal{L}$ is K if the language $\{(enc(\mathcal{A}), enc(\phi)) | \mathcal{A} \models \phi\}$ belongs to K.*

Let C be a class of finite structures. $\mathcal{L}$ captures *K on C ($\mathcal{L} \equiv_{\mathcal{C}} \mathcal{K}$) if data complexity of $\mathcal{L}$ is K and for every property of P of structures from C that can be tested with complexity K there is a sentence ϕ_P of $\mathcal{L}$ such that $\mathcal{A} \models \phi_{\mathcal{P}}$ iff $\mathcal{A}$ has property P, for every $\mathcal{A} \in \mathcal{C}$. We say that MX for a logic $\mathcal{L}$ captures a complexity class K if K is captured by $\exists SO(\mathcal{L})$.*

Data complexity of MX corresponds to the parameterized MX setting. The complexity of MX trivially falls between complexities of MC and satisfiability, since part of the input structure, which must include the universe, is given.

Table I has columns corresponding to the five tasks of interest, and rows corresponding to different logics. FO(ID) and FO(LFP) are FO extended with inductive definitions and least fixpoints, respectively. FO_k is the k-variable fragment of FO; GF_k is the k-guarded fragment of FO, which is strictly more expressive than FO_k. RGF_k is GF_k applied in MX, under the restriction that guards predicates may not

be expansion predicates. μGF is GF_1, extended with greatest and least fixpoint operators, and GF_k(ID) is GF_k extended with inductive definitions.

The notation for entries in the table is as follows. Consider the entry for logic $\mathcal{L}$ and task T. For complexity class K, the entry K means that task T for $\mathcal{L}$ is in K; the entry K_c means that for some set of $\mathcal{L}$-formulas T is K-complete; the entry $\equiv$K means that $\mathcal{L}$ captures K. A subscript on K, $\text{K}_{(P)}$, indicates the property holds under the condition P. The subscripted equivalences, $\equiv_s$ and $\equiv_{BIT}$ denote capturing provided that $\mathcal{L}$ is extended with a successor operator, or the BIT operator, respectively. FO(BIT) is equivalent in expressive power to FP($+\times$). [Immerman 1999].

5.1 Expressiveness of MX for Guarded Fragments

It is easy to see that there are NP-complete problems which are expressible as RGF_1-MX, as the following example shows.

Example 5.1 3-SAT as RGF_1-MX. *For a given set of clauses* $\Gamma = \{C_1, \dots C_m\}$, *the instance structure* $\mathcal{A}$ *has universe* $\{a, \neg a | a \in atoms(\Gamma)\}$ *and the relations* $Complements^{\mathcal{A}}$ *and* $Clause^{\mathcal{A}}$. *Let* ϕ *be*

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

A solution is an expansion of $\mathcal{A}$ *by the relation* $True^{\mathcal{A}}$, *which specifies which literals are mapped to true by a satisfying assignment.*

However, GF_1-MX does not capture NP. For example, the class of structures consisting of just a universe and the k-ary universal relation is not expressible with GF_1-MX. In [Kolokolova et al. 2006], it is shown that parameterized GF_k-MX, for $k \geq 2$, does capture NP. The proof relies on the fact that for every FO formula, there is logically equivalent formula in GF_k (over a larger vocabulary). The positive result on grounding of RGF_k described in Section 4.2, does not apply to GF_k, so the question arises: Does RGF_k, for some k, provide a universal language for NP? The answer is no.

Proposition 5.1 [Kolokolova et al. 2006]. *Parameterized* RGF_k*-MX does not capture NP, for any* k.

Since SAT can be decided in nondeterministic $O(n^2)$ time, by Theorem 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.

6. MXG: A GROUNDER/SOLVER FOR FO(ID) MODEL EXPANSION

In this section we describe an implemented solver, called MXG, for FO(ID) model expansion. The solver language allows axiomatizations in multi-sorted FO(ID) with arbitrary function-free FO formulas and a limited form of inductive defini-

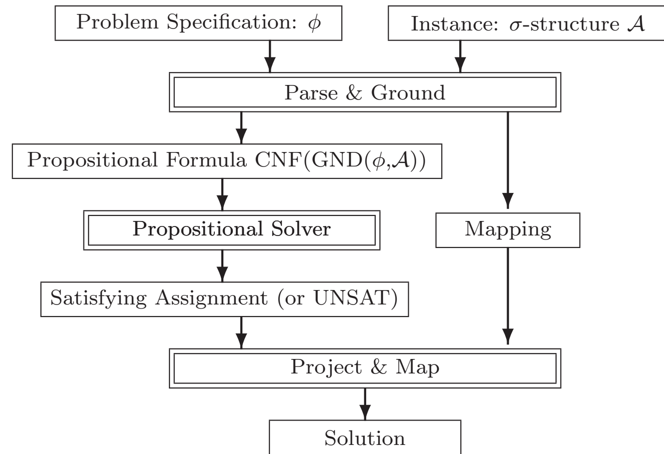

Fig. 2. MXG architecture. Double-lined boxes represent software components; single-lined boxes represent files.

tion. Linux executables for the current version of MXG, and examples, may be downloaded from http://www.cs.sfu.ca/research/groups/mxp/mxg/.

The architecture of MXG is illustrated in Figure 2. It takes as input a pair of files containing a *problem specification* and an *instance description*. The problem specification gives the vocabularies, type (sort) declarations, and axiomatization. The instance description gives the domains (i.e., a concrete set for each type or sort), and interpretations of the instance predicates. The grounding phase produces two files, one containing a propositional CNF formula in DIMACS form, and one specifying a mapping between variables in the CNF formula and ground atoms of the FO language. In principle any standard SAT solver can be used to search for satisfying assignments of the CNF formula, although at time of writing changing SAT solvers requires modifying the MXG source code. If a satisfying assignment is found, the mapping is used to produce a solution, in the form of a list of positive ground atoms describing the extension of each solution relation.

The MXG grounding algorithm is based on the extended relational algebra described in Section 4.1, but it is extended to directly produce a propositional CNF formula in DIMACS form. Several implementation-level devices and heuristics are used which reduce running time in comparison with a literal implementation of the algorithm of Section 4.1.

Our general syntax makes efficient implementation a substantial amount of work. A number of features we consider important for a real application system have not been implemented yet, and our grounding is currently slower than ASP grounders such as LPARSE. Nonetheless, the solver clearly demonstrates the viability of the MX approach. As shown in Section 7, the performance of our solver, which is in development for much less than a year, compares favourably with ASP systems that are more mature. Clearly, being able to take advantage of off-the-shelf SAT solvers plays a role here. We also find that our richer syntax is frequently convenient in writing and experimenting with axiomatizations. We believe that with additional

```
/* K-Colouring Problem Specification */
Given:
  type Clr Vtx;
  Edge(Vtx, Vtx)
Find:
  Colour (Vtx,Clr)
Satisfying:
  !m n c: Colour(n, c) & Colour(m, c) => ~Edge(n,m).
  !m c1 c2<c1: ~(Colour(m,c1) & Colour(m,c2)).
  !m: ? c: Colour(m,c).


/* K-Colouring Sample Instance Description */
Clr = [Red, Blue, Green]
Vtx = [A..E]
Edge = { A,B; A,D; B,C; D,C; E,D }
```

Fig. 3. K-Coloring Problem Specification and Instance Description

work our grounder can be made as fast as ASP grounders, at least for similarly limited syntax. However, even now grounding time is a small fraction of total solving time for instance families that are computationally non-trivial.

6.1 The MXG Solver Language

Our solver language syntax is, in part, a co-operative effort involving ourselves, the developers of the solver MidL [Mariën et al. 2006] and others. Differences between the languages of MXG and MidL primarily reflect differences in what is implemented in the respective systems. Appendix A gives the grammar for the MXG language. MXG Version 0.16 implements the language described by this grammar, except that inductive definitions are not handles in generality, as described below.

The solver input consists of a problem specification and an instance description, which have distinct syntax. The problem specification structure reflects nature of search problems: *Given* and instance, we wish to *find* a solution object *satisfying* certain properties. The *given* section specifies the types (sorts), and the vocabulary of the instance structure. The *find* part specifies the solution vocabulary. The *satisfying* part gives the problem axiomatization, including declaration of any auxiliary vocabulary used there. An instance description gives the concrete domains for each type, and instance relations in extensional form.

For illustration, Figure 3 gives a specification for graph K-Colouring, and an example of an instance for that specification. (Further specification examples can be found in Section 7.) The first line of each is a comment. The Given part of the specification declares the types Clr (the set of colours) and Vtx (the set of vertices), which will be given by the instance. The line "Edge(Vtx,Vtx)" declares Edge to be an instance relation symbol, which must be interpreted by a binary relation over Vtx^2. In the Find part, Colour(Vertex,Colour) declares the solution relation Colour to be a binary relation over Vtx $\times$ Clr. The Satisfying part consists of ASCII representations of three FO formulas. The mapping from logical symbols to ASCII symbols is given in Table II.

The language has a very simple type system, corresponding to standard multi-

Table II. ASCII Equivalents for Logical Symbols

Logical Symbol	$\forall$	$\exists$	$\wedge$	$\vee$	$\neg$	$\supset$
ASCII Representation	!	?	&	|	~	=>

sorted logic, except that we declare types (sorts) of arguments to predicate symbols, not of variables. Types of variables are inferred from their roles as terms, which must be consistent. Infix binary predicates `<=`, `=`, `>`, `>=`, `~=`, are all built-in. Formally, these are abbreviations. Only variables of the same type may be compared, and all order operators reflect the same order as `<`. The language also includes *bounded quantifiers*, `x<y` and `x>y`, which are also formally abbreviations. For example, `!x y<x` ϕ is the ASCII representation of $\forall x\ \forall y < x\ \phi$, which abbreviates $\forall x\ \forall y(y < x \supset \phi)$. Implementing these directly in the grounder seems to be more efficient than replacing them with their definitions.

6.1.1 *Instance Descriptions.* Instance descriptions give the domains of types and the extensions of instance relations. The syntax is illustrated in the K-colouring example of Figure 3. Domains may be given as an enumerated list of constant symbols (e.g., `[a,b,c]`); a range of integers (e.g., `[10..99]`); or a range of characters (e.g., `[a..q]`). For a enumerated list the order is as given; for a range of integers it is numerical; and for a range of characters it is alphabetic. Extensions of relations are given as a set of semi-colon separated tuples, with elements of tuples being comma-separated.

6.1.2 *Inductive Definitions.* In the solver syntax for inductive definitions, the universal quantifiers for variables occurring in heads of rules are implicit. For example, the definition of the transitive closure, TC, of an edge relation E is:

```
{ TC(u,v) <- E(u,v).
  TC(u,v) <- TC(u,w) & E(w,v).}
```

Another example is the relation Diff in the Blocked Queens axiomatization of Section 7.4.4.

MXG Version 0.16 requires only one defined relation symbol per definition, and bodies of rules are quantifier-free formulas in which free variables that do not occur in the head of the rule are taken to be existentially quantified. Moreover, heads of all rules must have the same tuple of variables. Such definition are handled correctly under the condition that each definition is either equivalent to its completion, or can be "directly computed", in the following sense. The grounder processes definitions in the order they appear in the specification file. When processing a definition, it "directly computes" the defined relation if the following hold of the definition: (1) the body of each rule is a conjunction of literals, and (2) there is no negative occurrence of the defined relation symbol, and (3) except for the relation being defined, every relation symbol is for an instance relation or a relation that has already been "directly computed". Otherwise, the grounder produces and then grounds the completion of the definition. If definition D be a set of rules $r_1, ..r_k$ of the form $r_i = P(\overline{x}) \leftarrow \phi_i(\overline{x}, \overline{y})$, then the completion of D is the formula:

$$\forall \overline{x}\ P(\overline{x})\ \Leftrightarrow \bigvee_{1 \le i \le k} \exists \overline{y}\ \phi_i(\overline{x}, \overline{y})$$

Definitions are equivalent to their completion when the induction is over a well-founded order (see [Denecker and Ternovska 2007b]).

7. EMPIRICAL PERFORMANCE RESULTS

Here we present an empirical comparison of the performance of MXG with MidL, an FO(ID) model expansion solver produced at the Katholieke Universitat Lueven (KU Leuven), and with several ASP solvers. We compare with ASP solvers for several reasons, including (1) ASP is the most widely known and most developed framework which is comparable in goals and techniques to ours; (2) The Asparagus repository of ASP solvers, axiomatizations, and benchmark instances[3] provides a useful resource for efficiently carrying out such a comparison. Our intention here is to show that our approach is feasible, and that it can provide at least comparable performance to ASP, which may be seen as the "nearest neighbor" technology. Comparison with a much wider range of approaches and systems, and over a wider range of problems and instances, is needed in the longer term. The following solvers were compared:

MXG 0.15 with RSat 1.3 for the SAT solver (Denoted MXG+RSat);
MidL 1.0.0, an "native" FO(ID) model expansion solver;
Cmodels 3.64, a SAT-based ASP solver[4];
smodels 2.32, a "native" ASP solver – the de facto standard for comparison ;
clasp rc3, a "native" ASP solver with clause learning;
DLV 2006-7-14, an ASP solver for disjunctive logic programs ;

Grounding for Cmodels, smodels and clasp was done using LPARSE version 1.0.17. MidL uses a variant of LPARSE which comes with the MidL download, and DLV does its own grounding. MXG 0.16, which we describe here and make available for download, differs from 0.15 in some aspects of file formats, minor syntactic points, as well as lifting the restriction to formulas in prenex form. However, performance on the specifications given here is identical.

Our choice of benchmark problems reflects two goals; to use a variety of problems with non-trivial instance collections, and to use problems for which ASP axiomatizations have been provided (presumably by people with some ASP expertise). We report results for:

(1) Graph K-colouring
(2) Hamiltonian Cycle
(3) Bounded Spanning Tree
(4) Blocked Queens
(5) Latin Square Completion

[3]http://asparagus.cs.uni-potsdam.de/
[4]Unfortunately, the version of Cmodels available at the time we ran our tests turned out to have bugs, which caused it to produce a significant number of incorrect results, as well as to crash on some inputs. Thus, the curves for Cmodels must be considered speculative. Nevertheless, Cmodels has a track record as a good ASP solver, and we expect that with the bugs repaired the performance profile will not be very different than shown in our plots.

(6) Social Golfer

We have three quite different sorts of problems on graphs, the first two being "classic" well-studied NP-hard graph problems. We also have three "typical" examples of finding combinatorial structures. Latin squares and Blocked Queens seem to have a similar flavour, but have very different solver performance profiles. We define the problems, specify the instances used, and give our MXG axiomatizations below, in Sections 7.4.1 through 7.4.6.

7.1 Overall Solver Performance

When we consider total solving time (time for grounding plus time for the ground solver), MXG+RSat had the best performance of the six solvers tested on two of the six problems studied (K-Colouring and Blocked Queens), was fourth best on Hamiltonian Cycle, and was second or third on the remaining three. Thus, overall, MXG+RSat performance is competitive with other current solvers.

The ASP solver clasp, with LPARSE doing the grounding, had the best overall performance on four problems, and was the only solver that did not come in last on some problem.

LPARSE and DLV grounding time is usually small in comparison with ground solving time. MXG is significantly slower than LPARSE at grounding some problems, and for these the MXG+RSat time is dominated by the grounding time. For example, the time for RSat to solve the ground formulas produced by MXG for Latin Square Completion and Bounded Spanning Tree instances is very small in comparison with MXG grounding time, and also in comparison with the ground solving time for other solvers. Separate grounding and solving times for MXG+RSat and LPARSE+clasp, for the problem of Latin Square Completion, are presented in Section 7.4.3 to illustrate.

Currently, MXG takes about three times as long as LPARSE on some problems. It is an interesting question whether a grounder for MXG syntax can be made as fast as the ASP grounders. If so, the resulting MX solver would dominate on the majority of problems tested here.

7.2 Cumulative Performance Plots

The performances plots (Figures 4 through 6) have the following format. The X-axis is time in seconds; the Y-axis is the cumulative number of instances solved *within a given time bound*. For example, a point at (5,10) indicates that among the instances tested, 5 were solved in 10 seconds or less *each*, and the remaining all required more than 10 seconds each. We plot a point for each instance solved, so if the ith instance solved required n seconds, there is a point at (i,n). We connect the points for each solver with a curve, as a visual aid.

Presenting solver times for non-trivial sets of instances of NP-hard problems is not always simple. Solvers of essentially the same quality often succeed on different instances even within a collection of very similar instances. (Even a tiny change to a solver can affect *which* instances are solved, while not affecting the number solved.) Thus, tables of run-times are hard to interpret, especially for large collections of instances. Also, we are typically interested in *scaling* – how performance with problem size – for a *general category* of instances, more than with performance on

particular small instances. Often the solvers with the best scaling performance are not the fastest on small instances. Further, run-times typically exhibit very high variance, so the mean run time may be dominated by a few extreme instances and not reflect typical performance. The cumulative plot format reduces emphasis on particular run-times, and gives a good overall picture of relative performance and scaling of solvers on a collection of instances. (Looking at run-times for particular instances often *is* very interesting, it is just not the best way to see the overall trend.)

Observe that the x-axis of all plots is logarithmic, and thus some care is required in interpreting the curves. The curve for a sequence of instances with linear growth in run-time will show up as a curve with large and rapidly increasing slope. A straight line suggests an exponential increase in run-times, and the smaller the slope the higher the exponent.

We ran all tests with a time cut-off of 30 minutes (1800 seconds), so the right-hand edge of each plot is at 1800 seconds. The upper edge of each plot is at a value a bit larger than the number of instances in the relevant collection, so we can see the points when all instances were solved within the cut-off time. If (x,y) is the extreme upper-right point of the curve for solver A, then A solved y instances in x seconds or less, and failed to solve any of the remaining instances within the 30-minute cut-off. In all cases, we report the total time for both grounding and solving. For instances which are not computationally trivial, LPARSE grounding is almost always a small fraction of solving time. MXG is significantly slower at grounding, and for some problems run-time is dominated by the grounding time.

7.3 Test Platform

All tests were run on Sun Fire VZ20 Dual Opteron computers with two 2.4 GHz AMD Opteron 250 processors having 1MB cache and 2GB of RAM per processor. The machines were running Suse Enterprise Linux 2.6.11. The executables for DLV, clasp, MidL and RSat were downloaded from the respective solver sites.[5] Executables for smodels, Cmodels and MXG were compiled with gcc version 3.3.4, using the default settings of the makefiles provided with the solver sources.

7.4 Problem Specifications and Performance Plots

In this section, we present the plots showing the relative performance of the various systems on the chosen benchmark problems, and also the MXG axiomatizations we used.[6] These axiomatizations are not the simplest possible, but also are not highly optimized. They are all produced by: (1) writing the simplest or most natural axiomatization we though of; (2) re-writing axioms with a syntactic form we know to be handled poorly by our current grounder implementation; (3) adding some redundant axioms which appear to improve performance. Step 2 only makes sense in the context of a general syntax, and we expect future grounder versions to

[5] http://www.dbai.tuwien.ac.at/proj/dlv/, http://www.cs.uni-potsdam.de/clasp/, http://www.cs.kuleuven.be/ maartenm/research/midl.html, http://reasoning.cs.ucla.edu/rsat/

[6] All MXG axioms here are in prenex form, because at the time we wrote them this was an MXG restriction. Version 0.16 does not have this restriction, but is essentially equivalent except for this and some other small syntactic differences.

significantly reduce the effects of this. Adding redundant constraints is a standard practice in most areas of constraint solving.

For the ASP solvers, we used axiomatizations downloaded from Asparagus. For MidL, we used axiomatizations for K-Colouring, Hamiltonian Cycle and N-Queens included in the MidL download package. To the latter, we added an axiom enforcing the blocked cells to produce a Blocked Queens axiomatization. For the remaining problems we used a direct translation of our MXG axiomatizations into the MidL language, which primarily amounted to replacing bounded quantifiers with their definitions.

We will say that a solver "solved" an instance if it produced a solution or correctly reported it unsatisfiable, and "failed to solve" the instance if it did not halt within the 30 minute time cut-off. We verified every solution produced during our tests with both MXG and smodels. For each problem, we estimate an order of preference of solvers based on performance. Since we prefer to reward good scaling over fast solving of small instances, our ordering is as follows: we prefer those solvers that solved the most instances within the cut-off time, and among those we prefer the one that minimized the maximum time for solving an instance. Solver order could change with an increased cut-off time, but this will be the case with any preference scheme that rewards good scaling.

7.4.1 *Graph K-Colouring.* Graph colouring is a classic and well-studied NP-hard search problem. An instance consists of a graph and a number K, and a solution is a proper K-colouring of the graph. That is, we want a function mapping vertices to a set of K colours, so that no two adjacent vertices are mapped to the same colour. Our test set consisted of 17 instances, 5 3-colouring instances from Asparagus and 12 instances from the LEI category of the graph colouring benchmark collection at http://mat.gsia.cmu.edu/COLOR/instances. These latter are are challenging instances on graphs of 450 vertices, variously with 5, 15 and 25 colours. All 17 instances are colourable with the alloted number of colours.

The performance of the solvers is shown in Figure 4. The figure clearly shows the following order of solvers, from best to worst: MXG+RSat, clasp, MidL, DLV, smodels. Notice that no solver was successful on all instances: MXG+RSat and clasp solved 13 of the 17 instances, while others solved 10 or fewer. The four instances that were not solved by MXG went unsolved by all solvers tested.

Reduction to SAT is known to be quite effective for graph colouring. Since the propositional formulas produced by MXG are same as the usual reduction from K-Col to SAT, it is not very surprising that MXG+RSat does well on this problem.

Because the current version of MXG does not support arithmetic, we take our instance to consist of the graph plus the set of colours, so we have two sorts: vertices and colours. The axiomatization simply says there is a binary relation Colour which must be a proper colouring of the vertices:

Given :
  type Clr Vtx;
  Edge(Vtx, Vtx)
Find :
  Colour(Vtx, Clr)
Satisfying :
  $\forall$ m n c Colour(n, c) $\land$ Colour(m, c) $\supset$ $\neg$Edge(n, m)

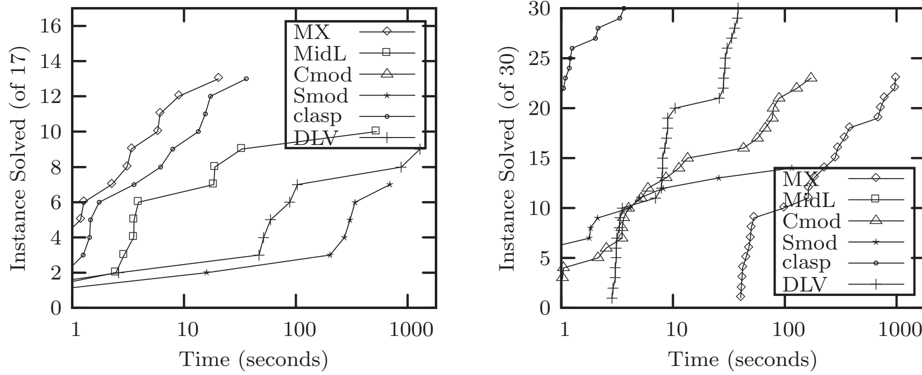

Fig. 4. Performance on K-Colouring (left) and Hamiltonian Cycle (right)

∀ m c1 c2<c1 ¬(Colour(m, c1) ∧ Colour(m, c2))
∀ m ∃ c Colour(m, c)

7.4.2 *Hamiltonian Cycle.* Hamiltonian Cycle is another "classic" NP-hard search problem: Given a graph, find a cycle in the graph which which visits every vertex exactly once. Existence is NP-complete. The test set consists of 30 instances from Asparagus, with sizes of 100 vertices and 800 edges; 150 vertices and 1200 edges; and 200 vertices and 1800 edges. All instances have Hamiltonian Cycles. Performance is illustrated in Figure 4. Here, MXG does not fair as well. The ordering appears to be: clasp, DLV, Cmodels, MXG, smodels, MidL. Only clasp and DLV solved all instances. Notice that smodels solved a number of instances faster than MXG, but that it only solved 14 instances while MXG solved 23. MXG is slower than smodels on easier instances, but appears to scale better. MidL does not appear in the plot because it failed to solve any instance. Interestingly, it seems to preform quite well on the Asparagus instances of Hamiltonian Path. MXG also performs poorly on those.

Here, not having general inductive definitions may be hurting MXG: Expressing that every vertex is on a single cycle is straightforward classically, but expressing that the cycle is unique is not. ASP and MidL axiomatizations can do this by directly expressing reachability. In our MXG axiomatization we use the built-in ordering $<$ on vertices which, together with the pair (MAX, MIN), produces a cycle that visits every vertex once. Then we require that Map() is a permutation on the vertices that maps to this cycle. The solver must also construct the actual set of edges, Hc, in agreement with this permutation. In the axiomatization, the predicate P provides a weak upper bound on Map which can be computed at grounding time, and provides a small performance improvement:

```
Given :
        type Vtx;
        Edge(Vtx, Vtx)
Find :
        Hc(Vtx, Vtx)
Satisfying :
```

Table III. Grounding and Solving Times for MXG+RSat and LPARSE+clasp, on Latin Square Completion. Times are mean times in seconds, with standard deviation in parentheses.

Grounder + Solver	*Grounder*	*GroundSolver*	*TotalTime*
MXG+RSat	24.72 (0.11)	0.64 (0.06)	25.40
LPARSE+clasp	0.54 (0.02)	12.49 (1.51)	13.03
Difference	24.18	−11.85	12.37

```
Map(Vtx, Vtx)
P(Vtx, Vtx)
{ P(x, n) ← x = MIN ∧ n = MIN
  P(x, n) ← P(y, n2) ∧ SUCC(n2, n) ∧ Edge(y, x) }
Map(MIN, MIN)
∀ x n Map(n, x) ⊃ P(x, n)
∀ n ∃ f Map(n, f)
∀ f ∃ n Map(n, f)
∀ n1 n2<n1 f ¬(Map(n1, f) ∧ Map(n2, f))
∀ n f1 f2<f1 ¬(Map(n, f1) ∧ Map(n, f2))
∀ n1 n2 f1 ∃ f2 [Map(n1, f1) ∧ (SUCC(n1, n2)∨(n1 = MAX ∧ n2 = MIN))] ⊃
                Map(n2, f2) ∧ (Edge(f1, f2) ∨ Edge(f2, f1))
∀ n1 n2 v u [(Edge(u, v) ∨ Edge(v, u)) ∧ Map(n1, v) ∧ Map(n2, u) ∧
                (SUCC(n1, n2) ∨ (n1 = MAX ∧ n2 = MIN))] ⊃ Hc(v, u)
∀ v u ∃ n1 n2 Hc(v, u) ⊃
                Map(n1, v) ∧ Map(n2, u) ∧ (Edge(v, u) ∨ Edge(u, v)) ∧
                (SUCC(n1, n2) ∨ (n1 = MAX ∧ n2 = MIN))
```

7.4.3 *Latin Square Completion.* A Latin Square (or Quasigroup) is an n by n matrix with elements in $\{1, \ldots n\}$, where every row and every column has every possible element. Constructing Latin Squares is easy. In the Latin Square Completion problem, an instance is the set $\{1, \ldots n\}$ plus prescribed values for certain elements. The task is to construct a Latin square consistent with the prescribed elements. Existence of such a completion is NP-complete. The test set consists of 100 instances from Asparagus. All are of size 30-by-30, and all have solutions.

The apparent ordering is: clasp, MXG, MidL, Cmodels, smodels, DLV. (It is tempting to read the plot as showing Cmodels better than MidL, but it is not: MidL solved all instanes within about a minute, whereas Cmodels left two unsolved after 30 minutes.) The performance plot is shown in Figure 5. Two observations are in order. First, in contrast to K-Colouring and Hamiltonian Cycle, there is a qualitative difference in performance of the solvers. At one extreme, DLV and smodels see the instances as all of distinct hardness. At the other extreme, MXG sees them as essentially identical. Second, the lead clasp holds over MXG+RSat is more than accounted for by the difference in grounding time. clasp solves the ground instances in about 12 seconds each, whereas RSat requires about 0.64 seconds each. This, as well as the remarkable uniformity of run-times for MXG+RSat, can be seen in Table III, which shows the breakdown of grounding and ground solving time for MXG+RSat and clasp.

The MXG axiomatization is the obvious one, plus one redundant constraint. Cell denotes matrix elements; we require that each row and column contains a bijection on $\{1, \ldots n\}$; the redundant constraint is that each cell has exactly one element:

Given :

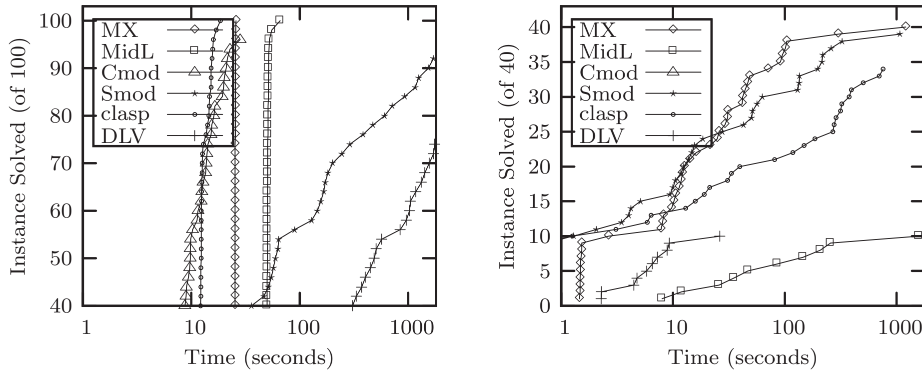

Fig. 5. Performance on Latin Square Completion (left) and Blocked Queens (right)

```
        type Num;
        Preassigned(Num, Num, Num)
Find :
        Cell(Num, Num, Num)
Satisfying :
        ∀ x y z Preassigned(x, y, z) ⊃ Cell(x, y, z)
        ∀ x z ∃ y Cell(x, y, z)
        ∀ x z y1 y2<y1 ¬(Cell(x, y1, z) ∧ Cell(x, y2, z))
        ∀ y z ∃ x Cell(x, y, z)
        ∀ y z x1 x2<x1 ¬(Cell(x1, y, z) ∧ Cell(x2, y, z))
        ∀ x y ∃ z Cell(x, y, z)
        ∀ x y z1 z2<z1 ¬(Cell(x, y, z1) ∧ Cell(x, y, z2))
```

7.4.4 *Blocked Queens.* The n-Queens problem is to place n Queens on an n-by-n chessboard, so that no two attack each other. Although once a favourite benchmark problem in constraint programming, finding solutions with a modern solver is trivial – not to mention that analytic solutions are well known. In the blocked queens problem, one has certain cells pro-scribed to not contain a queen. Existence of such a completion is NP-complete. The test set consists of 40 instances from Asparagus, of sizes from 28-by-28 to 56-by-56, 20 of which have solutions.

While having a similar flavour to Latin Square completion, the performance profile (at least on the Asparagus instances) is different, in that all solvers found the instances to vary considerably in difficulty. Moreover, the order has changed considerably, to: MXG, smodels, DLV, MidL, clasp.

The MXG axiomatization required some thought because the current version has no arithmetic, without which it is not very convenient to describe the diagonals. The solution is the inductively-defined subtraction relation diff(), which is computed by MXG during grounding. This is slower than built-in subtraction, but fast enough that MXG+RSat performs best overall. The remaining axioms are standard:

```
Given:
     type Num;
Find :
     Queen(Num, Num)
Satisfying :
```

```
Diff(Num, Num, Num)
Block(Num, Num)
{  Diff(i, x2, y2) ← (x2 = MIN ∧ y2 = i)
   Diff(i, x2, y2) ← (Diff(i, x1, y1) ∧ SUCC(x1, x2) ∧ SUCC(y1, y2)) }
∀ x y Queen(x, y) ⊃ ¬Block(x, y)
∀ i y2 x2 y1<y2 x1<x2 ¬(Queen(x1, y1) ∧ Queen(x2, y2) ∧
      Diff(i, x1, x2) ∧ Diff(i, y1, y2)))
∀ i y1 x2 y2<y1 x1<x2 ¬(Queen(x1, y1) ∧ Queen(x2, y2) ∧
      Diff(i, x1, x2) ∧ Diff(i, y2, y1)))
∀ x y1 y2<y1 Queen(x, y1) ⊃ ¬Queen(x, y2)
∀ x1 x2<x1 y Queen(x1, y) ⊃ ¬Queen(x2, y)
∀ x ∃ y Queen(x, y)
∀ y ∃ x Queen(x, y)
```

7.4.5 *Social Golfer*. The goal is to schedule $g \times s$ golfers into g groups of s players over w rounds (weeks), such that no two golfers play in the same group more than once. The test set consists of 174 instances from Asparagus, spanning (but not covering) the parameter range: number of weeks from 2 to 8; group size from 2 to 6; number of groups from 2 to 8. We know 72 instances to have solutions and 64 to have no solution, leaving 38 of unknown status. The order of solvers is: clasp, MXG, smodels, DLV, Cmodels, MidL.

```
Given :
        type Players Groups Groupsize Weeks;
Find :
        Plays(Players, Weeks, Groups)
Satisfying :
        M(Weeks, Players, Groupsize)
        Soc(Weeks, Players, Players)
        ∀ p w ∃ g Plays(p, w, g)
        ∀ p w g1 g2<g1 ¬(Plays(p, w, g1) ∧ Plays(p, w, g2))
        ∀ w p1 ∃ p2 (Soc(w, p1, p2)∨Soc(w, p2, p1))
        ∀ p1 p2>p1 w1 w2<w1 ¬(Soc(w1, p1, p2) ∧ Soc(w2, p1, p2))
        ∀ p1 p2>p1 w ∃ g Soc(w, p1, p2) ⇔ Plays(p1, w, g) ∧ Plays(p2, w, g)
        ∀ w g p1 p2<p1 l ¬(Plays(p1, w, g) ∧ Plays(p2, w, g) ∧ M(w, p2, l) ∧ M(w, p1, l))
        ∀ w p ∃ l M (w, p, l)
```

7.4.6 *Bounded Spanning Tree.* A spanning tree of a graph is a sub-graph that is a tree and visits every vertex. A spanning tree can be found in linear time. In this version of the problem, an instance consists of a directed graph and a bound K, and we require a directed spanning tree in which no vertex has out-degree larger than K. Existence is NP-complete. The test set is 30 instances from Asparagus, some with 35 and some with 45 vertices. All have solutions. All instances of the same size look identical to MXG, with 35 vertex instances taking about 2 seconds and 45 vertex instances taking about 8 seconds. The solver order is: clasp, MXG, Cmodels, MidL, smodels, DLV. As with Latin Square Completion, the difference between clasp and MXG+RSat is entirely accounted for by grounding time. If MXG grounding was as fast as LPARSE, MXG+RSat would perform better than LPARSE+clasp on these instances.

```
Given:
        type Vtx Bound;
        Edge(Vtx, Vtx)
Find :
```

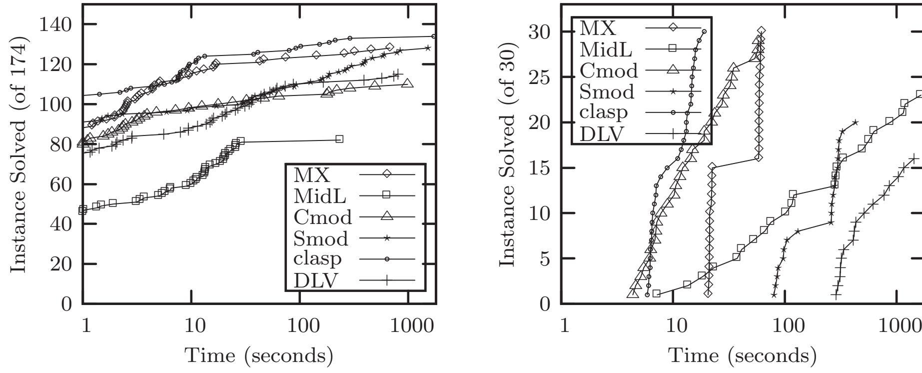

Fig. 6. Performance on Social Golpher (left) and Bounded Spanning Tree (right)

```
        Bstedge(Vtx, Vtx)
Satisfying :
        Map(Vtx, Vtx)
        M(Vtx, Bound)
        ∀ v ∃ u Map(v, u)
        ∀ u ∃ v Map(v, u)
        ∀ n f1 f2>f1 ¬(Map(n, f1) ∧ Map(n, f2))
        ∀ n1 n2>n1 f ¬(Map(n1, f) ∧ Map(n2, f))
        ∀ v u Map(MIN, v) ⊃ ¬Bstedge(u, v)
        ∀ u v x y≤x ¬(Map(x, u) ∧ Map(y, v) ∧ Bstedge(u, v))
        ∀ v u Bstedge(u, v) ⊃ Edge(u, v)
        ∀ v ∃ u ¬Map(MIN, v) ⊃ Bstedge(u, v)
        ∀ x u v<u ¬(Bstedge(u, x) ∧ Bstedge(v, x))
        ∀ u v w<v k ¬(Bstedge(u, v) ∧ Bstedge(u, w) ∧ M(v, k) ∧ M(w, k))
        ∀ u ∃ k M(u, k)
```

8. RELATED SYSTEMS

A broad survey of related work is infeasible here, and we focus our attention on implemented systems which roughly parallel MXG: general-purpose, logic-based systems for representation and solving of search problems, with a primary focus on NP-search.

8.1 Representation in ∃SO

Many researchers have entertained the idea of using ∃SO as a representation language for problems in NP.[7] We have heard anecdotal reports of projects to test the idea of using $\exists SO$ and reduction to SAT, in one or another form, to solve NP-complete problems. One such project is reported in [Rrustemi 2004]. We are

[7]Indeed, the first author made such a proposal to Hector Levesque as an undergraduate student in 1989, and certainly many had thought of it before then. The proposal elicited an explanation of two basic obstacles which have since been at least partially overcome: the need to extend classical logic with recursion, and the absence of a computational mechanism, such as an effective SAT solver, likely to work in practice.

not aware of any serious attempt to develop the idea further and to address issues which must be handled to produce practical tools.

Mancini [Mancini 2004] has explored various ways in which $\exists$SO axiomatizations can be manipulated to support effective search. Mancini views $\exists$SO as a natural abstraction of concrete modelling languages proposed in the constraint satisfaction and constraint programming communities. His work studies a variety of automatable ways to manipulate such axiomatizations to improve solver performance. A number of these have promise, and could naturally be applied to MX specifications.

In contrast to Mancini, we take $\exists$SO, with some extensions, as the actual modelling language to be implemented. We envision adding syntactic layers on top of the foundation of $\exists$SO, rather than developing such languages independently and viewing $\exists$SO as an abstraction of them.

8.2 Other FO(ID)-MX Solvers

An earlier solver produced at SFU used a slightly modified version of LPARSE, the standard ASP grounder, together with a translation from propositional logic with inductive definitions to SAT [Pelov and Ternovska 2005]. The formula ϕ was in the input syntax of LPARSE, but with FO(ID) semantics. The output of LPARSE was transformed to SAT using a level-mapping reduction for inductive definitions. While performance was acceptable on benchmark problems considered, we did not pursue the approach because using the level-mapping reductions did not appear promising as a general scheme.

MidL is a solver for FO(ID) model expansion developed by Marc Denecker's group at KU Leuven [Mariën et al. 2006]. The MidL input language is similar to that of MXG. Some differences, at time of writing, are that the MidL language does not have ordered domains, but does have a form of arithmetic. The MidL grounder is based on LPARSE, and produces ground formulas in propositional logic with inductive definitions, called PC(ID). The MidL solver engine handles this logic natively, rather than using reduction to SAT, as described in [Mariën et al. 2005].

8.3 ASPPS (ASP–Propositional Schemata)

East and Truszczynski's ASPPS [East and Truszczynski 2004] is the first system we know of based on the idea of extending classical logic with a recursion mechanism. The modelling language is classical formulas, in an ASP-like syntax, augmented by arithmetic, weight constraints, and Horn rules with minimum-model semantics. It may be seen as FO(ID) restricted to a formulas and rules of a simple syntactic form, and extended with weight constraints and arithmetic. ASPPS is based on Herbrand models, but otherwise could be viewed as a pre-cursor to FO(ID)-MX. The system supports grounding to SAT, and also to propositional CNF extended with cardinality constraints.

8.4 NP-SPEC and Spec2SAT

The earliest proposal we are aware of for a declarative language to represent arbitrary NP search problems exactly, is NP-SPEC [Cadoli et al. 2000]. The modelling language is stratified Datalog, extended by a set-based notation for describing finite domains, several aggregate operators, and notation for declaring particular relations to have certain properties, such as being a bijection, permutation, etc. An instance

it is a finite database, i.e., as set of ground atoms representing a multi-sorted finite structure. The initial implementation was provided by reduction to Prolog, with apparently poor performance. Later an implementation, called Spec2SAT, was provided using reduction to SAT [Cadoli and Schaerf 2005], but this handled only the fragment of the specification language without aggregates, recursion, or negative occurrences of defined predicates in rule bodies.

8.5 ESRA

ESRA [Flener et al. 2003] is The ESRA modelling language [Flener et al. 2003] is syntactically FO with extensions, provided with a denotational semantics. Extensions include domain declarations (sorts), arithmetic over finite sets of integers, set operations, a summation operator and a counting quantifier. Instances take the form of a collection of finite "domains", some of which may be sets of tuples from others. Solutions are instantiations of "decision variables", which may be relations or functions. The language is provides an interesting example because it appears formally to be equivalent to model expansion for an extension of FO, although not formalized in this way.

The ESRA authors regard it as a constraint programming language based on relations as an abstract data type – thus, one presumes, the choice of a denotational rather than model-theoretic semantics. ESRA is implemented in the form of compilation to OPL, a related modelling language with an effective solver implementation [Hentenryck 1999]. The ESRA authors aim to provide a simple, high-level language "capture common modelling idioms", while expressing dis-interest in any formal notion of completeness, such as capturing a class of problems. (Clearly ESRA can express every problem in NP. It seems unlikely is captures NP-search, in the sense of Section 3, as is has no recursion, although it is also possible that other features allow it to express problems beyond NP.)

8.6 ASP (Answer Set Programming)

Answer Set Programming [Marek and Truszczynski 1999; Niemela 1999] is based on the language of logic programming, with the stable model semantics [Gelfond and Lifschitz 1991]. There are many ground solvers, most of which take as input the ground programs produced by the LPARSE grounder. This language extends normal logic programs with weight constraints (a generalization of cardinality constraints), and arithmetic. A number of groups are working on the area, and new solvers appear regularly.

Instances are provided in the form of a set of ground atoms, which formally are part of the logic program. Separation of problem and instance descriptions is considered important [Marek and Truszczynski 1999] but maintained only as a convention which is not always followed in practice. The semantics is based on Herbrand models, which entails significant restrictions on the use of function symbols. In model expansion, no such mechanism is required to invoke closure on the universe.

The built-in recursion in ASP is often convenient, particularly for axiomatizing problems involving sequences of events, such as verification and planning problems. One problem, in our view, is that the entire formula (logic program) is involved in the recursion. Many properties which can be easily and naturally expressed

classically require a less natural expression within this recursion.

9. CONCLUSIONS AND FUTURE WORK

NP-hard search problems arise frequently in application areas ranging from software verification to bio-informatics, and often the inability to solve sufficiently large instances within practical time bounds is a significant obstacle to practical work. Practitioners in these areas need effective tools, but producing good implementations requires considerable effort; moreover, there are relatively few techniques which work well in practice, and these are implemented over and over in different domains. There appears to be a role for well-implemented general-purpose tools for modelling and solving search problems. Such tools could greatly facilitate the work of many practitioners who now must get by with poorly adapted tools and techniques, or invest a great deal of energy implementing domain-specific search algorithms. We believe that the model expansion framework has a potential as a basis for such technology.

9.1 Our Framework for Search

Our framework for representing and solving search problems with logic is based on a simple idea: we have a finite structure and are looking for an expansion of this structure to satisfy a formula. The framework has two settings, parameterized and combined. To date, our attention has been largely directed to the parameterized setting. Here, the formula is fixed and provides a problem specification. Some properties of the setting include:

(1) This setting separates problem description (a formula) from an instance of the problem (a finite structure).

(2) It provides a *universal framework* for search in a number of complexity classes through the capturing results of descriptive complexity.

(3) Since the formula is fixed, pre-processing of the formula to improve solver performance is possible.

(4) In this setting, user's intentions regarding search can be captured in a precise sense.

We have implemented a solver, MXG, for parameterized FO(ID) model expansion. The performance of the solver clearly shows feasibility of the approach, providing performance comparable to more mature ASP systems.

Properties of the combined setting include:

(1) The setting provides flexibility, in allowing an instance to be in part represented by a formula (cf. the block-world planning example in section 2.3.2).

(2) It allows one to express problems of higher complexity than NP (up to NEXPTIME) with FO or with FO(ID).

One natural direction of future work is to develop notions of capturing various complexity classes in this setting.

9.2 Future and Current Work: Foundations

One of the main remaining tasks is adding arithmetic and other structured domains, such as strings. For these, some "built-in" domains, with certain functions and predicates, will be added. Care is required, as these additions can easily change the task complexity. For example, in adding arithmetic is that we have an infinite domain, such as the natural numbers or reals, and in this setting model expansion is no longer decidable. In our current work, we formalize the notion of a "meta-finite" model expansion, where a secondary, potentially infinite structure is present, and investigate the conditions under which model expansion is decidable or under which we can preserve complexity and expressiveness properties of interest.

Other extensions that are current or future work include: aggregates; new sort declarations; "upper guards"; methods for "compiling out" auxiliary vocabulary symbols; a game-theoretic formalization of model expansion.

9.3 Future Work: Solvers

Within our current solver architecture, we plan to study several approaches to improving performance, such as various techniques for processing the problem specification; use of propositional languages richer than simple CNF, handling a broader range of inductive definitions; and recognition of special cases which allow for efficient grounding and solving; techniques for "on-the-fly" grounding.

We would like to investigate the option of having "normal forms" of MX axiomatizations, with appropriate algorithms to translate to those forms, for efficient grounding and solving. We also plan to explore alternate schemes for solver construction, such as the local search approach proposed in [Agren et al. 2005], and application of general finite-model finding tools, such as used in the theorem proving community.

9.4 Future Work: Modelling Languages

For many industrial practitioners, classical logic may not be an acceptable modelling language, so languages to be used in industry will likely be syntactic variants of FO(ID). Since SQL is essentially a syntactic variant of FO, and is widely used in industry, it seems natural to explore the idea of basing a modelling language on SQL. Another natural approach is to extend FO(ID) with constructs such as those in ESRA and other constraint languages, which provide considerable convenience and reduce the number of axioms needed to model a problem.

A. SYNTAX FOR MXG VERSION 0.16

A.1 Problem Specification Grammar

```
<TheoryFile>        ::= <GivenPart> <FindPart> <SatisfyingPart>
<GivenPart>         ::= Given : <TypeDeclaration> <SymbDeclaration>
<TypeDeclaraion>    ::= type <TypeNames> ;
<TypeNames>         ::= <TypeName> | <TypeNames> <TypeName>
<SymbDeclaration>   ::= | <SymbDeclaration> <a_SymbDCL>
<a_SymbDCL>         ::= <PredDCL> | <ConstDCL>
<PredDCL>           ::= <PredName> ( <PredTypes> )
<PredTypes>         ::= <TypeName> | <PredTypes> , <TypeName>
<ConstDCL>          ::= <ConstName> : <TypeName>
<FindPart>          ::= Find : <SymbDCL>
```

```
<SatisfyingPart>   ::= Satisfying : <SatisfyingRules>
<SatisfyingRules>  ::= <a_SatisfyingRule> | <SatisfyingRules> <a_SatisfyingRule>
<a_SatisfyingRule> ::= <an_Axiom> | <a_SymbDCL>
<an_Axiom>         ::= <FO_Formula> | { <IDD_Rules> }
<FO_Formula>       ::= <Unitary_Formula> |
                       <FO_Formula> <BinaryOperator> <Unitary_Formula>
<unitary_formula>  ::= ( <FO_Formula> ) | <QuantPart> : <Unitary_Formula> |
                       ~<Unitary_Formula> | <atomic_formula>
<atomic_formaula>  ::= <PredName> ( <Args> ) |
                       SUCC ( <Args> ) |
                       <Ord_Relation>
<Ord_Relation>     ::= <VarName> <OrdOperator> <VarName> |
                       <VarName> <OrdOperator> <ConstName> |
                       <VarName> <OrdOperator> MIN |
                       <VarName> <OrdOperator> MAX |
                       <ConstName> <OrdOperator> <VarName> |
                       MIN <OrdOperator> <VarName> |
                       MAX <OrdOperator> <VarName>
<QuantPart>        ::= <Quantifier> <Variables>
<Quantifier>       ::= ? | !
<Variables>        ::= <a_Var> | <Variables> <a_Var>
<a_Var>            ::= <VarName> | <Ord_Relation>
<Args>             ::= <an_Arg> | <Args> , <an_Arg>
<an_Arg>           ::= <VarName> | MIN | MAX | <ConstName>
<OrdOperator>      ::= < | <= | > | >= | =
<BinaryOperator>   ::= & | <vline> | => | <=>
<vline>            ::= |
<IDD_Rules>        ::= <IDD_Rule> | <IDD_Rules> <IDD_Rule>
<IDD_Rule>         ::= <PredName> ( <Args> ) <- <QF_FO_Formula>
<QF_FO_Formula>    ::= <atomic_formula> | ( <QF_FO_Formula> ) |
                       <QF_FO_Formula> <BinaryOperator> <atomic_formula> |
                       ~( <QF_FO_Formula> ) | ~<atomic_Formula>
```

A.2 Instance Description Grammar

```
<InstanceFile>    ::= <an_InstancePart> | <InstanceFile> <an_InstancePart>
<an_InstancePart> ::=  <InstnaceType> | <InstnacePred> | <InstanceConst>
<InstnaceType>    ::= <TypeName> = [ <TypeElements> ]
<TypeElements>    ::= <Character>..<Character> |
                      <Number>..<Number> | <TypeElementSet>
<TypeElementSet>  ::= <a_TypeElement> |
                      <TypeElementSet> ; <a_TypeElement>
<a_TypeElement>   ::= <ElementName> | <Number> | Character
<InstnacePred>    ::= <PredName> = { <Tuples> }
<Tuples>          ::= <a_Tuple> | <Tuples> ; <a_Tuple>
<a_Tuple>         ::= <a_TypeElement> | <a_Tuple> , <a_TypeElement>
<InstanceConst>   ::= <ConstName> = <a_TypeElement>
```

ACKNOWLEDGMENTS

Some ideas presented here were developed while visiting the Newton Institute for Mathematical Sciences in Cambridge during the program on Logic And Algorithms. We wish to thank the Institute and Moshe Vardi and Anuj Dawar for inviting us to take part in the program. The paper has benefited from discussions with Neil Immerman, Steven Cook, Lucas Bordeaux, Marc Denecker, Maarten Mariën, Johan

Wittocx, as well as other members of our research group, including Brendan Guild, Yongmei Liu, Antonina Kolokolova, Nhan Nguyen, Stella Chui, Murray Patterson. We are greatful to Nhan Nguyen for his implementation of a minimum-Horn model module for MXG; to Antonina Kolokolova and Leonid Libkin for referring us to some results in descriptive complexity at an early stage of the project; and to Martin Otto for pointing out that what we called "model extension" in [Mitchell and Ternovska 2005b; 2005c] should correctly be called "model expansion".

REFERENCES

Agren, M., Flener, P., and Pearson, J. 2005. Incremental algorithms for local search from existential second-order logic. In *Proc., CP 2005*. 47–61.

Ajtai, M. 1983. σ_1^1 formulae on finite structures. *Annals of Pure and Applied Logic* 24, 1–48.

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

Biere, A., Cimatti, A., Clarke, E., and Zhu, Y. 1999. Symbolic model checking without BDDs. In *Proc., Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*. LNCS Volume 1579.

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

Cadoli, M. and Schaerf, A. 2005. Compiling problem specifications into SAT. *Artificial Intelligence 162*, 89–120.

Clark, K. 1978. Negation as failure. In *Logic and Databases*, H. Gallaire and J. Minker, Eds. Plenum Press, 293–322.

Cook, S. 1971. The complexity of theorem proving procedures. In *Proc. 3rd Ann. ACM Symp. on Theory of Computing*. Association for Computing Machinery, New York, 151–158.

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

Dawar, A. and Gurevich, Y. 2002. Fixed point logics. *The Bulletin of Symbolic Logic 8,* 1, 65–88.

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

Denecker, M. and Ternovska, E. 2004a. Inductive situation calculus. In *Proc., KR-04*.

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

Denecker, M. and Ternovska, E. 2007a. Inductive situation calculus. *Artificial Intelligence*.

Denecker, M. and Ternovska, E. 2007b. A logic of non-monotone inductive definitions. *ACM transactions on computational logic (TOCL)*.

East, D. and Truszczynski, M. 2004. Predicate-calculus based logics for modeling and solving search problems. *ACM TOCL*. To appear.

Ebbinghaus, H.-D. and Flum, J. 1995. *Finite model theory*. Springer Verlag.

Fagin, R. 1974. Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity of Comput.* 43–73.

Fagin, R. 1993. Finite-model theory – a personal perspective. *Theoretical Computer Science 116*, 3–31.

Flener, P., Pearson, J., and Agren, M. 2003. Introducing ESRA, a relational language for modelling combinatorial problems. In *Proc., LOPSTR'03*.

Flum, J., Frick, M., and Grohe, M. 2002. Query evaluation via tree-decompositions. *J. ACM 49,* 6, 716–752.

Frisch, A., Grum, M., Jefferson, C., Martinez-Hernandez, B., and Miguel, I. 2007. The design of essence: A constraint language for specifying combinatorial problems. In *Proc., Twentieth International Joint Conference on Artificial Intelligence(IJCAI), January 9-13, 2007, Hyderabad, India*.

FURST, M., SAXE, J., AND SIPSER, M. 1984. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory* 17, 13–27.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

GEREVINI, A., DIMOPOULOS, Y., HASLUM, P., AND SAETTI, A. 2006. 5th international planning competition (ipc-5) web page: http://zeus.ing.unibs.it/ipc-5/.

GOTTLOB, G., LEONE, N., AND SCARCELLO, F. 2001. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 195–206.

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

GREEN, C. 1969. *Theorem Proving by Resolution as a Basis for Question-Answering Systems.* Edinburgh University Press, 183–205.

HENTENRYCK, P. V. 1999. *The OPL Optimization Programming Language*. MIT Press.

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

IMMERMAN, N. 1999. *Descriptive complexity*. Springer Verlag, New York.

JONES, N. AND SELMAN, A. 1974. Turing machines and the spectra of first-order formulas. *Journal of Symbolic Logic 39*, 139–150.

KAUTZ, H. AND SELMAN, B. 1992. Planning as satisfiability. In *Proc. ECAI-92*.

KOLAITIS, P. AND VARDI, M. 1998. Conjunctive-query containment and constraint satisfaction. In *Proc. of the 17th ACM Symp. on Principles of Database Systems (PODS)*. 205–213.

KOLOKOLOVA, A., LIU, Y., MITCHELL, D., AND TERNOVSKA, E. 2006. Complexity of expanding a finite structure and related tasks. In *Logic and Computational Complexity, workshop associated with LICS'06*.

LEONE, N., PFEIFER, G., FABER, W., EITER, T., GOTTLOB, G., PERRI, S., AND SCARCELLO, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL) 7,* 3.

LIBKIN, L. 2004. *Elements of Finite Model Theory*. Springer.

LIVCHAK, A. 1983. The relational model for process control. (in russian). *Automated Documentation and Mathematical Linguistics* 4, 27–29.

MANCINI, T. 2004. Declarative constraint modelling and specification-level reasoning. Ph.D. thesis.

MAREK, V. W. AND REMMEL, J. B. 2003. On the expressibility of stable logic programming. *TCLP 3*, 551.

MAREK, V. W. AND TRUSZCZYNSKI, M. 1999. *Stable logic programming - an alternative logic programming paradigm*. Springer-Verlag. In: The Logic Programming Paradigm: A 25-Year Perspective, K.R. Apt, V.W. Marek, M. Truszczynski, D.S. Warren, Eds.

MARIËN, M., MITRA, R., DENECKER, M., AND BRUYNOOGHE, M. 2005. Satisfiability checking for PC(ID). In *Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2005, Proceedings*. Lecture Notes in Computer Science, vol. 3835. Springer, 565–579.

MARIËN, M., WITTOCX, J., AND DENECKER, M. 2006. The IDP framework for declarative problem solving. In *Search and Logic: Answer Set Programming and SAT*. 19–34.

MEDINA, J. A. AND IMMERMAN, N. 1994. A syntactic characterization of np-completeness. In *Proc., LICS 1994*. 241–250.

MITCHELL, D. AND TERNOVSKA, E. 2005a. Constraint programming with unrestricted quantification. In *Proc. First International Workshop on Quantification in Constraint Programming (CP-2005 Workshop)*.

MITCHELL, D. AND TERNOVSKA, E. 2005b. A framework for representing and solving NP search problems. In *Proc. of the 20th National Conf. on Artif. Intell. (AAAI)*. 430–435.

MITCHELL, D. AND TERNOVSKA, E. 2005c. Model extension as a framework for representing and solving NP-hard search problems. In *Logic and Computational Complexity, workshop associated with LICS'05*.

Niemela, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3,4, 241–273.

Patterson, M., Liu, Y., Ternovska, E., and Gupta, A. 2006. Grounding for model expansion in k-guarded formulas. In *Search and Logic: Answer Set Programming and SAT*. 65–75.

Patterson, M., Liu, Y., Ternovska, E., and Gupta, A. 2007. Grounding for model expansion in k-guarded formulas with inductive definitions. In *Proc., Twentieth International Joint Conference on Artificial Intelligence (IJCAI), January 9-13, 2007, Hyderabad, India*.

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

Rrustemi, A. 2004. The application of sat-solvers in combinatorial problems. A dissertation submitted for the degree Diploma in Computer Science.

Stockmeyer, L. 1974. The complexity of decision problems in automata theory. Ph.D. thesis, MIT.

Stockmeyer, L. J. 1977. The polynomial-time hierarchy. *Theoretical Computer Science 3*, 1–22.

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

Van Gelder, A., Ross, K., and Schlipf, J. 1991. The well-founded semantics for general logic programs. *Journal of Assoc. Comput. Mach. 38,* 3, 620–650.

Vardi, M. 1986. Complexity of relational query languages. *Information and Control 68*, 137 –146.

Vardi, M. Y. 1982. The complexity of relational query languages. In *Proc. 14th ACM Symp. on Theory of Computing*. Springer Verlag, 137–146.