A Framework for Representing and Solving NP Search Problems

David Mitchell and **Eugenia Ternovska**
Simon Fraser University
{mitchell,ter}@cs.sfu.ca

Abstract

NP search and decision problems occur widely in AI, and a number of general-purpose methods for solving them have been developed. The dominant approaches include propositional satisfiability (SAT), constraint satisfaction problems (CSP), and answer set programming (ASP). Here, we propose a declarative constraint programming framework which we believe combines many strengths of these approaches, while addressing weaknesses in each of them. We formalize our approach as a model extension problem, which is based on the classical notion of extension of a structure by new relations. A parameterized version of this problem captures NP. We discuss properties of the formal framework intended to support effective modelling, and prospects for effective solver design.

Introduction

Applications which require solving NP-complete problems or their associated search problems abound in AI and other fields. Progress in these areas is often limited by the inability to solve sufficiently large instances within practical time bounds. Propositional satisfiability (SAT), finite domain constraint satisfaction (CSP), and answer set programming (ASP) are arguably the most prominent declarative programming approaches to solving such problems. These may be seen as part of a larger program to develop a collection of widely applicable – and widely applied – practical tools and techniques for solving NP search problems, supported with mature mathematical foundations and theoretical tools, much as mathematical programming has done for a range of optimization problems. While SAT, CSP and ASP have made valuable contributions, they each have many limitations. Here we present a formal framework for representing and solving problems which we believe addresses at least some of these limitations.

Effectiveness of the declarative approach has perhaps been demonstrated most clearly by SAT. Current SAT solvers exhibit impressive performance on many industrial instances and, for example, have become a widely used tool for hardware verification. This success has been facilitated by the fact that SAT has very simple syntax and semantics. Unfortunately, SAT provides a poor modelling language and substantial effort may be required to find good encodings for some problems.

CSP provides somewhat better modelling capabilities. The area has also provided a number of useful techniques. For example, nogood learning and backjumping methods developed in the CSP context are central to modern SAT solvers (Mitchell 2005). Solvers for CSP, however, have found success primarily as components in constraint logic programming (CLP) tools. These provide rich problem solving environments, but are really general purpose programming languages and thus neither purely declarative nor tailored to problems of moderate (in)tractability.

ASP is the only area of these three that took modelling methodology seriously from the outset. ASP input languages provide essentially all the modelling facilities CSP does, within a purely declarative framework. A built-in recursion mechanism provides additional modelling convenience, particularly useful when encoding problems involving sequences of events, such as in verification and planning problems. ASP solvers are quite effective, and in particular may out-perform pure SAT-based methods when recursion is used in models. However, the recursion mechanism is provided by the combination of logic programming syntax and stable-model semantics (Marek & Truszczynski 1999; Niemela 1999), which is a burden as well as a blessing. Adapting results and techniques from other areas of logic may be challenging or impossible, and extending the formal foundation to richer languages can be challenging. Moreover, the semantics and the resulting style of expressing some properties are not entirely intuitive, and the prospects for ASP being widely adopted in industry seem poor.

Modelling Languages

One goal for our framework is to provide a foundation upon which practical modelling languages can be built. Such a language should at the least combine all the strengths of SAT, CSP and ASP, and hopefully provide many additional benefits. Since this is a paper about the formal foundation, we will speak primarily about facilities we want in this foundation to support practical modelling.

First and foremost, we believe that declarative modelling languages should be based, to the greatest extent possible, on classical logic. This choice permits taking advantage of the properties of classical logic, and in particular finite model theory, for example in identifying tractable fragments, proving correctness of axiomatizations, etc. Moreover, the combination of rich expressive features and clean semantics should facilitate producing syntactic variants, for use in industry, which have these same properties.

Second, on our observations of, for example, logics for knowledge representation and formal verification, database query languages, and current directions in SAT constraint programming research, indicate that certain facilities are crucial in practice. These include:

- quantification,
- an easy way to express reachability, thus recursion, including recursion combined with negation,
- inclusion of a variety of pre-defined (interpreted) constraint symbols, such as cardinality and other aggregate constraints, as well as some arithmetic,
- clear separation of the descriptions of problems and instances.

In addition, effective modelling requires modularity, which comes naturally with classical logic, but is non-trivial when recursion through negation is present.

A further goal for our framework, beyond these criteria, was to provide the most general possible formal foundation, while capturing exactly the complexity class of interest and remaining close to classical logic. To satisfy these criteria and goals, we propose a framework based on classical first-order logic (FO), formulated in such a way as to capture exactly the problems in NP, even while permitting unrestricted use of quantifiers, function symbols, and equality. To easily express recursion, we use an extension of first-order logic with inductive definitions, FO(ID).

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). This would be analogous to database practice, where most users of the query language SQL are unaware that it is a syntactic variant of (a slight extension of) FO.

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 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. For relation symbol R of vocabulary τ, the relation corresponding to R in a τ-structure A is denoted R^A. The size of a structure A is the number of elements in its universe, denoted $||A||$. We will also have need to consider total size of an explicit representation of a structure A, which we denote $size(A)$. For a formula ϕ, we write $vocab(\phi)$ for the collection of exactly those function and relation symbols which occur in ϕ. We reserve the symbol σ for the vocabulary of instance descriptions. To simplify presentation, we give our definitions and proofs for the case without function symbols, but all given results hold in their presence as well. For precise definitions and other results from finite model theory which we use but do not prove, we refer the reader to (Libkin 2004)

Model Extension

In the framework, we cast computational problems as the logical task of *model extension, MX*.

Definition: The problem *MX* is: Given a formula ϕ with vocabulary $vocab(\phi)$, and a finite structure A_I for vocabulary $\sigma \subset vocab(\phi)$, is there a structure A which is an extension of A_I to $vocab(\phi)$, and such that $A \models \phi$.

The idea is that the finite structure is an object of interest, such as a graph, and the formula specifies a question about the object, such as whether it is Hamiltonian or not, in such a way that the extension relations witness the property.

Example 1. *Let the input structure be a graph, $G = \langle V; E \rangle$, (i.e., E is binary, symmetric and irreflexive), and ϕ be:*

$$\forall x \forall y\ [(Clique(x) \land Clique(y)) \supset (x = y \lor E(x, y))]$$

Let A be a structure which is an extension of G to the vocabulary of ϕ. Then $A \models \phi$ iff $Clique^A$ is a set of vertices which form a clique in G.

Example 2. *Consider the vocabulary $\{Cell\}$, which we will use to specify the elements in a square matrix a, where $Cell(r, c, i)$ will mean that element $a_{r,c}$ is i. Let ϕ be:*

$$\begin{aligned} &\forall r \forall i\ \exists c\ Cell(r, c, i)\ \land\ \forall c \forall i\ \exists r\ Cell(r, c, i) \\ &\land \forall r \forall c \forall i \forall j\ [\ (Cell(r, c, i) \land Cell(r, c, j)) \supset i = j\ ] \\ &\land \forall r \forall c \forall i (GivenCell(r, c, i) \supset Cell(r, c, i)) \end{aligned}$$

In any structure A satisfying ϕ, the extension of relation $Cell$ lists the entries in a Latin Square of size $||A|| \times ||A||$. If A_I is just a universe of size n, then the model extension problem is that of finding an $n \times n$ Latin Square. If A_I also specifies the relation $GivenCell$, then we have the NP-complete Quasigroup Completion Problem.

Relation symbols which are not interpreted by the instance structure behave as existentially quantified second order variables, so in the case of FO ϕ, we have the same power as existential second order logic ($\exists$SO) over finite structures. Note that the set inclusion in $\sigma \subset vocab(\phi)$ in the formalization of the model extension problem must be proper. If $\sigma = vocab(\phi)$, we have model checking, not model extension.

There is a very close connection between model extension and the spectrum problem. The spectrum of a sentence ϕ is the set $\{n \in \mathbb{N} \mid \phi \text{ has a finite model of size } n\}$. 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 $\exists$SO sentence $\exists R_1, \ldots, \exists R_m\ \phi$, by associating a universe of size n with n. Thus, if $\sigma = \emptyset$, model extension coincides with the spectrum problem.

The complexity of model extension lies between satisfiability and model checking. FO satisfiability is undecidable, FO model extension is NEXPTIME-complete, and FO model checking is PSPACE-complete. Model extension avoids undecidability of FO by specifying the finite universe as part of the instance.

Theorem 1. *The first-order model extension problem is NEXPTIME-complete.*

The proof is a straightforward reduction from Bernays-Schoenfinkel satisfiability or from combined complexity of $\exists$SO over finite structures, and is given in the Appendix. (The case of Theorem 1 with $\sigma = \emptyset$ is equivalent to the result from (Jones & Selman 1974) that a set $X \subset \mathbb{N}$ is a FO spectrum iff it is in NEXPTIME.)

Capturing NP

To capture NP, we define a parameterized version.

Definition: Fix an unrestricted FO formula ϕ and a vocabulary $\sigma \subset vocab(\phi)$. The problem *MX*$(\sigma, \phi)$ is: Given a finite structure A_I for vocabulary σ, is there an extension A of A_I to $vocab(\phi)$ such that $A \models \phi$.

The intuition, and intended methodology, is that ϕ represents the problem, and a structure A_I for vocabulary σ represents a particular instance. As a decision problem, we are interested in the existence of an extension of A that satisfies ϕ, while the search problem is that of finding such an extension. It is easy to see that for some choices of ϕ and σ, MX(ϕ,σ) is NP-complete.

Example 3. *Let σ be $\{E\}$, so the input structure is a graph $A_I = G = \langle V; E\rangle$, and the vocabulary of ϕ be $\{E, R, B, G\}$. An an extension of A_I to this vocabulary gives a 3-colouring of the graph, with colours R,B and G. To require the colouring be total and proper, let ϕ be:*

$$\begin{array}{l} \forall x[(R(x) \lor B(x) \lor G(x)) \land \neg(R(x) \land B(x)) \\ \qquad \land\neg(R(x) \land G(x)) \land \neg(B(x) \land G(x))] \\ \bigwedge \ \forall x \forall y[E(x, y) \supset (\neg(R(x) \land R(y)) \\ \qquad \land\neg(B(x) \land B(y)) \land \neg(G(x) \land G(y)))] \end{array}$$

MX(ϕ,σ) is equivalent to graph 3-colourability: The extensions to A_I that satisfy ϕ, if there are any, correspond exactly to the proper 3-colourings of G. A slightly more complicated formula can express K-colourability, with an additional input relation to specify K.

This example shows that every problem in NP can be reduced in polytime to a model extension problem MX(ϕ,σ). Next we show the much stronger property that the problems in NP are exactly those that are equivalent to a problem MX(ϕ,σ), for some choice of ϕ and σ.

Definition: We say a class of finite σ-structures K is *expressed by MX*(σ, ϕ) iff for any σ-structure A_I, $A_I \in K$ iff there is an extension A of A_I such that $A \models \phi$.

We assume standard encodings of languages by classes of structures, and vice versa (see, e.g. (Libkin 2004)).

Theorem 2. *Let σ be a vocabulary, K a class of finite σ-structures. Then K is in NP iff for some FO formula ϕ, K is expressed by MX(σ, ϕ).*

Proof. $\Leftarrow$) Suppose that K is expressed by $\langle\phi, \sigma\rangle$. Then $A_I \in K$ iff some extension A of A_I to $vocab(\phi)$ satisfies ϕ. A is a suitable certificate for A_I, since its size is polynomial in the size of A, and $A \models \phi$ can be checked in time polynomial in the size of A. To see this, first notice that the sum of all arities of relations in A is at most $|\phi|$, so $size(A)$ can be at most $|A|^{|\phi|} = |A_I|^{|\phi|} \leq size(A_I)^{\phi}$. Since ϕ is fixed, this is polynomial $size(A_I)$. There is an algorithm which checks if $A \models \phi$ in time $O(|\phi| + size(A)^{|\phi})$, which is also polynomial in $size(A_I)$.

$\Rightarrow$) Suppose $K \in NP$. We need to find ϕ and σ such that $\langle\phi, \sigma\rangle$ expresses K. By Fagin's theorem (Fagin 1974) there is an $\exists$SO ψ such that $A \in K$ iff $A \models \psi$, where ψ is of the form $\exists\vec{P}\phi$, and ϕ is FO. Let $\sigma = vocab(\phi) - \{\vec{P}\}$. We have that for any σ-structure A_I, $A_I \in K \Leftrightarrow A_I \models \psi \Leftrightarrow A_I \models \exists\vec{P}\phi \Leftrightarrow A \models \phi$, where A is the extension of A_I to $vocab(\phi)$ by the relations $\vec{P}^A$ witnessing the existential second order quantifier. Thus $\langle\phi, \sigma\rangle$ expresses K. $\square$

Some lower complexity classes can be captured similarly, applying results for various fragments of $\exists$SO (Graedel 1992). For example, FO universal Horn *MX*(σ, ϕ) expresses P over ordered structures. The Σ^p_k levels of the Polynomial Hierarchy PH are captured by Π^1_{k-1} *MX*(σ, ϕ). Note that model extension does not naturally capture Π^p_k levels, and this is not just happenstance: If there are some σ and ϕ so that *MX*(σ, ϕ) is Π^p_k-complete, then PH collapses to the k-th level. In particular, if there are σ and ϕ such that *MX*(σ, ϕ) is co-NP-complete, then NP=co-NP.

Inductive Definitions

Formally, FO model extension has the same expressive power as $\exists$SO, so expresses all problems in NP. However, some properties that are important for modelling applications are not easy to express in this logic. The reader who thinks otherwise is invited to express transitive closure as FO model extension. That is, write a FO formula ϕ with vocabulary $\{E, TC\}$, such that, given graph $G = \langle V; E\rangle$, in any structure A which extends G and satisfies ϕ, TC^A is the transitive closure of G. (The related task of expressing in a formula that one vertex is reachable from another is easy, but does not do the job.) Our solution to this problem is to extend FO with inductive definitions. We shall see that such an extension makes expressing properties like transitive closure natural and trivial.

Inductive definitions are common in mathematics. For example, in logic the set of well-formed formula and the satisfaction relation $\models$ are defined inductively. Inductive definitions can be monotone (i.e., formulas) or non-monotone (i.e., $\models$). Both monotone and non-monotone induction are formalized in a natural way in the logic for non-monotone inductive definitions (ID-logic), which is an extension of classical logic (see (Denecker 2000; Denecker & Ternovska 2004b)). Inductive definition are useful in common-sense reasoning, as well as mathematics. For instance, it was shown (Denecker & Ternovska 2004a) that the situation calculus can be formalized in a natural way as an (non-monotone) iterated inductive definition in the well-ordered set of situations. In general, inductive definitions are an important form of human knowledge, and ID-logic is a good candidate for a modelling language.

A *definition* Δ is a set of rules of the form $\forall\, \vec{x}\ (X(\vec{t}) \leftarrow \varphi)$, where $\vec{x}$ is a tuple of variables, X is a relation symbol of some arity r, $\vec{t}$ is a tuple of terms of length r and φ is an arbitrary first-order formula. The connective $\leftarrow$ is called the *definitional implication*, and is distinct from material implication, for which we use $\supset$. A rule $\forall \vec{x}\ (X(\vec{t}) \leftarrow \varphi)$ in a definition does not correspond to the disjunction $\forall \vec{x}(X(\vec{t}) \vee \neg\varphi)$ although it implies it. Intuitively, definitional implication should be understood as the "if" found in rules in (informal) inductive definitions, such as "$\neg\phi$ is a formula if ϕ is". In the rule $\forall \vec{x}\ (X(\vec{t}) \leftarrow \varphi)$, $X(\vec{t})$ is called the *head* and φ is the *body*. A *defined symbol* of Δ is a relation symbol that occurs in the head of a rule of Δ; other relation symbols are called *open*. FO(ID) formulas are defined to be boolean combinations of definitions and FO formulas. The semantics of ID-logic extends the classical FO semantics with the well-founded semantics of logic programming (Van Gelder 1993; Fitting 2003; Denecker, Bruynooghe, & Marek 2001). For precise details see (Denecker & Ternovska 2004b). For an intuitive explanation of the well-founded semantics and why it formalises different forms of inductive definitions see (Denecker, Bruynooghe, & Marek 2001). Modularity conditions for ID-logic have been given in (Denecker & Ternovska 2004b).

Example 4. *We represent the problem of finding the transitive closure of a graph as a model extension problem. The input vocabulary σ consists of a single symbol E, which represents the binary edge relation. The universe of the input structure A_I is the set of verticies V. The formula consists of a definition with two rules, defining the relation TC.*

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

The rules state that the transitive closure of the set E of edges is the least relation containing all edges and closed under reachability.

Example 5. *The model of a definite logic program (i.e., one without negation) is the minimal model of the corresponding set of Horn clauses. In this example, we represent the task of computing the least model of a definite program as a task of model extension. Our input structure will represent the rules of the program using two relations, one which identifies the head atoms of rules and one which identifies body atoms. In the vocabulary for this structure, we have:*

- $H(r,h)$ *denotes that h is the head atom of rule r,*
- $B(r,a)$ *denotes that atom a occurs in the body of r,*

The extension vocabulary is the symbol M. The formula ϕ, represented by an inductive definition below, states the relationship between the rules of the program and the set of atoms M; in essence, it gives a declarative specification of the semantics of definite programs:

$$\left\{ \begin{array}{l} \forall a\ [M(a) \leftarrow\ \exists r\ (H(r,a) \\ \qquad \wedge\ (\neg\exists b\ B(r,b) \vee \forall b (B(r,b) \supset M(b))))] \end{array} \right\}$$

The formula says that an atom a is in the model if there is a rule with a in the head, and where the body is either empty or consists of atoms already in the model. In any extension of A_I that satisfies ϕ, M will list the atoms of the program in the unique minimal model of the program.

Extending FO with inductive definitions in this way makes many properties easier to express, but the expressive power of model extension is unchanged.

Theorem 3. *Let σ be a vocabulary, K a class of finite σ-structures. Then K is in NP iff for some FO(ID) formula ϕ, K is expressed by MX(σ, ϕ).*

Proof. Clearly, any problem in NP can be expressed as FO(ID) *MX*(σ, ϕ). Membership is shown by replacing FO with FO(ID) in the proof of Theorem 2, and adding the observation that model checking for FO(ID) is of the same complexity as for FO. ☐

This might seem surprising at first. The reason no power is added is that, once relations have been chosen for the open relation symbols in a definition, relations for the defined symbols can be computed in polynomial time.

ASP, SAT and CSP as Model Extension

In this section, we encode SAT, ASP and CSP as parameterized model extension *MX*(σ, ϕ).

ASP as Model Extension An answer set program P is a set of function-free ground clauses in the syntax of logic programming. The input structure will represent these rules using three relations. $H(r,h)$ and $B(r,a)$ are as in example 5, above. $Neg(r,a)$ denotes that the occurrence of atom a in the body of r is negated. The extension vocabulary is $\{SM\}$, and we will write our formula ϕ so that if $A \models \phi$, then SM^A consists of the atoms which are in a stable model of P. The formula ϕ, which gives a declarative specification of the stable model semantics, is:

$$\forall r\ [\neg R(r) \leftrightarrow \exists a\ (B(r,a) \wedge Neg(r,a) \wedge SM(a))]$$
$$\wedge \left\{ \begin{array}{l} \forall a\ [SM(a) \leftarrow\ \exists r\ (R(r) \wedge\ H(r,a) \\ \qquad \wedge\ \neg\exists b\ B(r,b))], \\ \forall a\ [SM(a) \leftarrow\ \exists r\ (R(r) \wedge\ H(r,a) \\ \qquad \wedge\ \forall b\ (B(r,b) \supset Neg(r,b) \vee SM(b))] \end{array} \right\}$$

The first conjunct is a formula which specifies the conditions under which a rule is in the reduct of the program P with respect to the model SM. The second conjunct is an inductive definition which says the model SM must be the least model of that reduct. The first rule in the definition handles the case of rules with empty bodies. In the second rule, which handles the general case, the disjunct $Neg(r,b)$ says that we ignore negated atoms, as they do not appear in the reduct of P.

3-SAT as Model Extension For a given set of clauses $\Gamma = \{C_1, \ldots C_m\}$, the input structure A_I has universe $\{a, \neg a | a \in atoms(\Gamma)\}$ and relations $Complements^{A_I}$ and $Clause^{A_I}$. Let ϕ be

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

A solution is the extension of the structure A_I by the relation $True^{A_I}$, which specifies which literals are mapped to true by a satisfying assignment.

It is interesting to observe what happens when we ground such instances. Assume a 3-CNF formula γ is represented as just described. Applying the most straightforward grounding procedure for ϕ given A_I produces a propositional formula ψ, which contains a number of pairs of clauses of the form $(True(a) \vee True(\neg a))$ and $(\neg True(a) \vee \neg True(\neg a))$. If these are deleted, and all remaining occurrences of literals of the form $True(\neg a)$ replaced by literals of the form $\neg True(a)$ (a process which can be carried out efficiently as a restricted application of resolution), the resulting formula is isomorphic to the original formula Γ.

CSP as Model Extension A CSP Instance is usually defined to be a tuple $\langle X, D, C \rangle$, where X is a set of variables, each of which ranges over the domain $D(x)$, and C is a set $C = \{C_1, \ldots, C_m\}$ of constraints. Each constraint is a pair $C_i = \langle S_i, R_i \rangle$, where $S_i = \langle x_{i,1}, \ldots x_{i,k} \rangle$ is a tuple of variables, called the scope, and $R_i \subseteq D(x_{i,1}) \times \ldots \times D(x_{i,k})$ is a relation of arity k, called the constraint relation. A solution, if there is one, is function α such that for every variable x, $\alpha(x) \in D(x)$, and for each constraint C_i, $\langle \alpha(x_{i,1}), \ldots \alpha(x_{i,k}) \rangle \in R_i$.

For brevity, we will assume that the domain of each variable x is exactly the set of values which at least one constraint permits x to take. Our instance vocabulary σ will have two relation symbols, S for constraint scopes, and R for constraint relations. $S(c, k, x)$ will denote that the k^{th} variable in the scope of constraint c is x. $R(c, t, k, a)$ will denote that the k^{th} element of the t^{th} tuple in the constraint relation of constraint c is the value a. Given a σ-structure A_I, we want to find a mapping of variables to values that satisfies the constraints. The vocabulary for ϕ is $\{S, R, C, V\}$, where C, which is for convenience only, will be the set of constraint names and V the value assignment. ϕ is:

$$\begin{aligned}&\forall c\, (C(c) \equiv \exists y \exists z S(c, y, z))\\ &\quad\wedge \forall c\, [C(c) \supset\\ &\qquad \exists t \forall k \forall x \forall a (S(c, k, x) \wedge V(x, a) \supset R(c, t, k, a))]\end{aligned}$$

Discussion

Our framework is similar in several respects to ASP, in that the underlying task in both cases is the extension of a partial structure to a model. The most obvious difference is that we use classical logic, which we consider to provide many advantages. Another important difference is that in our framework problem instances are given as a finite structure, whereas in ASP the instance is specified as a set of ground atoms. In ASP, separation of problem and instance descriptions are considered important (Marek & Truszczynski 1999), but maintained only at the level of convention. Since formally the ground atoms describing the instance are logic programming rules, the actual structure must be inferred from them. This requires restricting interest to Herbrand models, which in turn requires the language to be essentially function-free to avoid undecidability. (Actual ASP modelling languages do allow use of function symbols, but this use must be carefully restricted.) In model extension, the instance is defined to be a structure, and thus no such mechanics are required to invoke closure on the universe.

The formal distinction between problem description and instance description that we make has additional consequences. One is that both modeler and solver can, when desired, reason separately about the two descriptions. Another is that the modelling language for instances can, if desired, be different than the modelling language for problems. This makes sense because the instance description must specify a single finite structure, whereas the problem description must specify an infinite set of structures. Moreover, in many applications the problem description may be formulated just once, with great care, whereas many instance descriptions may be formulated, perhaps by many users.

The idea of providing programming environments for specifying exactly the problems in NP (or their search variants), is certainly not new. Some recent examples include the proposal of Cadoli and Mancini (Cadoli & Mancini 2002), who propose a constraint language based on $\exists SO$, but syntactically composed of extensions to SQL. ESRA (Flener, Pearson, & Agren 2003) is similarly motivated, and conveniently represents many problems, but the authors explicitly choose not to use recursion and negation, which we consider essential. There are several modelling languages for optimization problems, but these address a different need. One example that is also often used for search problems is OPL (Hentenryck 1999), but it aims at modelling generality, rather than targeting just the problems in NP.

The proposal that is most like ours is that of East and Truszczynski (East & Truszczynski 2004), which is an ASP-style system based on classical logic. The input is a pair encoding the problem as a formula and the instance as a set of ground atoms. The semantics is based on the set of Herbrand Models which satisfy the closed world assumption. The syntax is a restricted family of function-free FO formulas, but extended with definite Horn clauses to provide inductive definitions. Thus, in several respects it is more restricted than our framework.

Solver Construction

There remains the crucial question of whether or not the approach can be made to work well in practice. To a large extent, this depends on being able to produce effective solvers. An easy way to obtain a solver is to construct a translation to another language for which solvers already exist. For example, a translation of a restricted family of ID-Logic formulas to ASP is given in (Marien, Gilis, & Denecker 2004). Another approach, developed in our lab, is to construct a reduction to SAT and use SAT solvers. Such a reduction is described in (Pelov & Ternovska 2005). A prototype solver has also been constructed, for formulas specified in the language of the ASP input processor LPARSE, and demonstrates feasibility of the approach.

Native solvers for FO(ID) are now being developed in more than one lab. The best reason to believe such solvers can be effective is that they may be based on the same approach as ASP solvers, namely smart grounding followed by application of an engine for the propositional case. These propositional solvers may be based on the same technology that makes the best current SAT solvers effective.

Future Work

• Adding aggregates and interpreted functions relation symbols to the formal foundation. Interpreted functions should include some arithmetic. We believe this can be done based largely on existing work in classical logic and database theory. Some care is required, as adding arbitrary aggregates or arithmetic would change the complexity.

• Development of practical modelling languages, based on experiments and experience with solving a variety of application problems.

• Theoretical work on tractable cases and on cases which admit efficient grounding, especially cases which amount to elimination of sub-problems during grounding.

• Further work on solver design and implementation.

Appendix

Proof (Theorem 1): Let $n = |\phi| + size(A_I)$ be the total size of the input. To show membership in NEXPTIME, we must show we can guess an extension structure A, and then check that $A \models \phi$ in TIME(2^{n^c}) for $c \in \mathbb{N}$. The extension structure A can have total size at most $|A_I|^{|\phi|} \leq n^n = O(2^{nlogn})$, so we can guess this structure as required. There is an algorithm that checks if $A \models \phi$ in time $O(|\phi| + Size(A)^k)$, where k is the width of ϕ. k is certainly bounded by $|\phi|$, so we can check this in time $O(|\phi| + Size(A)^{|\phi|}) = O(n+(n^n)^n) = O((2^{nlogn})^n) = O(2^{n^2 logn})$. To show completeness, we reduce the NEXPTIME-complete problem of Bernays-Schoenfinkel (B-S) satisfiability to FO model extension. Formula ψ is a B-S formula if it is of the form $\exists x_1 \ldots \exists x_n \forall y_1 \ldots \forall y_m \psi'$, where ψ' is function-free, quantifier-free FO. For the reduction, given B-S formula ψ, we construct in polynomial time a FO formula ϕ and a structure A_I so that ψ has a model iff some extension A of A_I is a model of ϕ. It is known that if ψ has a model, then it has a model with universe size at most n. (Intuitively, we only need one element in the universe to witness each existential quantifier.) We take our structure A_I to have a universe of size n, in particular $\{1, \ldots n\}$, and no relations. Unfortunately, we don't know the exact size of any model of ψ. For example, ψ may say "all my models are of size k", for some $k < n$. So, we will make our formula do two jobs. The first is to select a suitable sized subset of $\{1, \ldots n\}$ to simulate the universe of a model of ψ. The second is to verify that, treating this subset as if it were the universe, the formula ψ is satisfied. Taking a disjunction over all possible universe sizes from 1 to n, our formula ϕ is:

$$\bigvee_{1 \leq i \leq n} [\, \exists \vec{x} \; (B_i^n \vec{x} \wedge \forall \vec{y} \; (B_i^m \vec{y} \supset \psi'))\,]$$

where the formula $B_i^k \vec{z}$ says that each of the variables in $\vec{z}$ have a value from the set $\{1, \ldots k\}$. It is defined to be

$$\bigwedge_{1 \leq j \leq k} \bigvee_{1 \leq l \leq i} z_j = l$$

The size of ϕ is $O(|\psi|^3)$. ∎

References

Cadoli, M., and Mancini, T. 2002. Combining Relational Algebra, SQL, and constraint programming. In *Proc., FroCos-02*, 147–161.

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

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

Denecker, M.; Bruynooghe, M.; and Marek, V. 2001. Logic programming revisited: Logic programs as inductive definitions. *ACM Transactions on Computational Logic (TOCL)* 4(2).

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

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

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

Fitting, M. 2003. Fixpoint semantics for logic programming - a survey. *Theoretical Computer Science*. To appear.

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

Graedel, E. 1992. Capturing complexity classes by fragments of second order logic. *Theoretical Computer Science* 101:35–57.

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

Jones, N., and Selman, A. 1974. Turing machines and the spectra of first-order formulas. *Journal of Symbolic Logic* 39:139–150.

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

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.

Marien, M.; Gilis, D.; and Denecker, M. 2004. On the relation between ID-Logic and Answer Set Programming. In *Proc., 9th European Conference on Logics in Artificial Intelligence*, 108–120. Springer. LNCS Volume 3229.

Mitchell, D. 2005. A SAT solver primer. *EATCS Bulletin* 85:112–133. Columns: Logic in Computer Science.

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.

Pelov, N., and Ternovska, E. 2005. Reducing ID-Logic to propositional satisfiability. Submitted.

Van Gelder, A. 1993. An alternating fixpoint of logic programs with negation. *Journal of computer and system sciences* 47:185–221.