Constraint Programming with Unrestricted Quantification

David Mitchell and Eugenia Ternovska*

School of Computing Science,
Simon Fraser University,
Burnaby, BC, V5A 1S6 Canada
{mitchell, ter}@cs.sfu.ca

Abstract. Search problems occur widely in AI, and a number of general-purpose constraint-based methods for solving them have been developed. Convenient modelling of many problems is enabled by use of quantifiers of various sorts, but the most prominent approaches support only limited use of quantifiers. A recently proposed constraint programming framework, based on classical logic and the notion of expansion of a finite structure with new relations, supports unrestricted use of both first-order and second-order quantifiers. The framework can be parameterized to capture various complexity classes, including NP and Σ_k^p for any k. Second-order quantifiers can be used to concisely model search problems at any of these complexity levels. First-order quantifiers can be used freely for modelling convenience, without affecting the complexity level which is determined by the second-order quantifiers. We explain this framework, discuss the roles of quantifiers, and give some examples.

1 Introduction

Challenging applications which require solving NP-hard search problems abound in many fields. Many of the general-purpose approaches to solving such problems fall naturally under the rubric "constraint programming". Arguably, the main activities in this area are the development of good modelling languages and techniques, and the development of effective solvers for these languages. The most widely studied of these approaches include propositional satisfiability (SAT), finite domain constraint satisfaction (CSP), answer set programming (ASP), and constraint logic programming (CLP). ASP and CLP have a restricted form of (implicit) quantification. One direction for improving modelling power and convenience in SAT and CSP is extension of the languages with quantifiers of various sorts.

In the case of SAT, quantifiers may be added in two ways. One is with a limited kind of first-order quantification, so-called "lifted SAT" or "propositional schemata" (see, for example, [1, 2]), to more concisely describe sets of similar

* Both authors supported by the National Sciences and Engineering Research Council of Canada

constraints. Depending on details, this approach may or may not change the computational complexity of the language. The other is with quantification of the propositional atoms, as in QSAT (QBF). In this case, each alternation of quantifiers gives an increase in the complexity by an exponential (assuming that the polynomial heirarchy does not collapse). Notice that, if we view QSAT as a fragment of classical second-order logic (SO), the quantifiers in QSAT are second-order.

ASP and CLP are both based on the language of logic programming, and thus include an implicit form of quantification. Normal variables which occur in both the head and body of a rule are implicitly universally quantified, while others are implicitly existential. Thus, quantification is limited to a restricted use of first-order quantifiers, where the outer quantifier is $\forall$ and the inner quantifier is $\exists$. Extending these languages with more general use of quantifier, such as first-order quantification in a different order, or quantification over relational (SO) variables is challenging.

Not all modelling limitations may be addressed by quantification, of course. For example, all the approaches described above may benefit from extension of the languages with built-in cardinality operators. Recursion is another important case. There are many applications, such as planning and verification, which involve encoding sequences of events or otherwise reasoning about actions. For these, the natural way to model certain properties involves recursion. While QSAT and QCSP provide sufficient expressive power to model these problems concisely, they have no built-in mechanism to support natural expression of recursion or induction, or minimization of sets in general. ASP and CLP naturally provide a form of recursion, but the semantics which provide the associated minimization rely on non-classical negation, which makes encoding many properties less intuitive than in a classical setting.

The Model Expansion Framework In [3, 4] we presented a formal framework for constraint programming based on classical logic, which we believe addresses a number of weaknesses in existing approaches. A novel feature of this approach is the support of unrestricted use of quantifiers, both first-order and second-order.

An important feature of the framework introduced in [3, 4] is the explicit separation of problem description and instance. An instance is represented by a finite structure, such as a graph. A solution will typically be a function or relation over the universe of the instance, such as a set of edges or a mapping of vertices to colours. The problem description is a formula, for example of first-order logic, which describes the relationship between instances and their solutions, i.e., between a graph and its 3-colourings.

This formulation of search problems has significant implications for the use of quantification in modelling, since quantifiers used in describing the problem occur in a fixed formula, rather than as part of the instance description. On one hand, first-order quantifiers may be used freely, without changing the complexity level of the modelling language. This is in contrast to QSAT and QCSP, where each quantifier alternation "moves" the modeler one exponential up in the poly-

nomial heirarchy. On the other hand, second-order quantifiers can be introduced exactly for the purpose of modelling problems at higher complexity levels.

Other features of the approach of [3, 4] include:

• Since it is based on classical logic, results and techniques from classical logic, and in particular finite model theory, can be applied directly for identifying tractable fragments, proving correctness of axiomatizations, etc.
• The full range of expressive features of classical logic can be used, including arbitrary use of equality and function symbols and both first-order and second-order quantification. Moreover, it can be extended with pre-defined functions and constraint relations, such as those used in CSP, CLP and ASP practice.
• In the framework, classical logic is extended with inductive definitions, providing a natural way to express recursion, including recursion through negation. This is important for modelling problems involving transition systems, such as in planning and verification.
• Most approaches begin with restrictive syntax to ensure practicality. In contrast, we consider it important to make the formal foundation as general as possible, while capturing exactly the complexity classes of interest and remaining close to classical logic. For example, we see no reason to arbitrarily restrict quantifier alternations, to disallow function and equality symbols, or to restrict the form of the formulas to a syntax similar to that of logic programming. In practice, of course, modellers will exploit this generality only to a point.

The remainder of the paper is organized as follows. In Section 2 we define model expansion and describe the basic approach. In Section 3 we describe how to capture NP and other complexity classes with model expansion, and Section 4 describes the extension of classical logic with inductive definitions. In Section 5 we encode CSP and ASP as model expansion, and in Sections 6 and 7 with give two modelling examples: the minimum-open-stacks problem from the Constraint Modelling Challenge 2005, and the problem of finding a winning strategy for the first player in the generalized Connect-4 game. Finally, in Section 8, we discuss future work toward making our approach practical.

2 Model Expansion

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 $||A||$. 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 results from finite model theory which we use but do not prove, we refer the reader to [5].

Model Expansion We cast computational problems as the following logical task.

Definition 1. *The model expansion problem MX is: Given a formula ϕ with vocabulary $vocab(\phi)$, and a finite structure I for vocabulary $\sigma \subset vocab(\phi)$, is there a structure A which is an expansion of 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 expansion relations witness the property. As a decision problem, we are interested in the existence of an expansion of A that satisfies ϕ, while the search problem is that of finding such an expansion.

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

$$\forall x \forall y \; [(Clique(x) \wedge Clique(y)) \supset (x = y \vee E(x, y))]$$

Let A be a structure which is an expansion 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.

Relation symbols which are not interpreted by the instance structure behave as existentially quantified second-order variables, so in the case of first-order (FO) ϕ, we have the same power as existential second-order logic ($\exists$SO) over finite structures.

Example 2. **(Quasigroup Completion)** 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:

$$\begin{array}{l} \forall r \forall i \; \exists c \; Cell(r, c) = i \\ \wedge \; \forall c \forall i \; \exists r \; Cell(r, c) = i \\ \wedge \; \forall r \forall c \forall i (GivenCell(r, c, i) \supset Cell(r, c) = i) \end{array}$$

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

Observe that here the description of constraints does not change with the size of the instance. Only that which is inherently part of the instance – in this case the size and the pre-determined cells – changes with instance. This is in contrast to many other frameworks where the constraint descriptions themselves change size with instance size, so the constraint model can only be presented as a schema. For example, a typical CSP encoding has n^2 variables $x_{i,j}$ for $i, j \in [n] = \{1, \ldots, n\}$, and the following constraints:

$$\begin{array}{ll} x_{i,j} \in [n] & \text{for each } i, j \in [n] \\ \text{alldiff}\{x_{i,1}, \ldots, x_{i,n}\} & \text{for each } i \in [n] \\ \text{alldiff}\{x_{1,i}, \ldots, x_{n,i}\} & \text{for each } i \in [n] \\ x_{i,j} = k & \text{for each given cell} \end{array}$$

The structure of this schema closely corresponds to that of the model expansion version, but it represents an set of $2n$ all-different constraints, each of size n. In contrast, the corresponding formulas in the model expansion version are a direct expression of the property represented by this large set of constraints.

3 Capturing Complexity Classes

The complexity of model expansion lies between satisfiability and model checking. For example, FO model checking is PSPACE-complete, FO model expansion is NEXPTIME-complete, and FO satisfiability (even the case of satisfiability by a finite model) is undecidable. Model expansion is like satisfiability, in that it involves finding a model, but undecidability is avoided by specifying the finite universe as part of the instance. Note that the set inclusion in $\sigma \subset vocab(\phi)$ in the formalization of the model expansion problem must be proper. The case where $\sigma = vocab(\phi)$ is model checking.

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

The proof is a straightforward reduction from Bernays-Schoenfinkel satisfiability, as given in [3], or from combined complexity of ∃SO over finite structures.

A Parameterized Version The intended scheme for using model expansion in representing and solving search problems in NP, and at other Σ_k^p levels of the polynomial heirarchy, is to represent the problem with a fixed formula, and represent instances with finite structures. To formalize this, we define a parameterized version of model expansion.

Definition 2. *Fix a formula ϕ and a vocabulary $\sigma \subset vocab(\phi)$. The parameterized model expansion problem $MX(\sigma, \phi)$ is: Given a finite structure I for vocabulary σ, is there an expansion A of I to $vocab(\phi)$ such that $A \models \phi$.*

In the intended methodology the formula ϕ describes the problem and σ represents a particular instance. By taking ϕ to be from different logics, for example classical first-order (FO) or second-order (SO) logic, or fragments of these, we may capture different complexity classes.

Capturing NP It is easy to see that for some choices of ϕ and σ, MX(ϕ,σ) is NP-complete, as shown by the following two examples.

Example 3. **(3-Colourability)** 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 expansion of 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:

$$\begin{aligned} &\forall x[(R(x) \vee B(x) \vee G(x)) \wedge \neg(R(x) \wedge B(x)) \\ &\qquad \wedge\neg(R(x) \wedge G(x)) \wedge \neg(B(x) \wedge G(x))] \\ \bigwedge\ &\forall x \forall y[E(x,y) \supset (\neg(R(x) \wedge R(y)) \\ &\qquad \wedge\neg(B(x) \wedge B(y)) \wedge \neg(G(x) \wedge G(y)))] \end{aligned}$$

MX(ϕ,σ) is equivalent to graph 3-colourability: The expansions of 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.

Example 4. **(3-SAT)** For a given set of clauses $\Gamma = \{C_1, \ldots C_m\}$, the input structure I has universe $\{a, \neg a | a \in atoms(\Gamma)\}$ and two relations, $Complements^{A_I}$ and $Clause^{A_I}$. Let ϕ be

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

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

In fact, a much stronger property can be shown: Parameterized FO model expansion captures exactly the NP search problems.

Definition 3. *We say a class of finite σ-structures K is* expressed by *$MX(\sigma, \phi)$ iff for any σ-structure I, $A_I \in K$ iff there is an expansion A of I such that $A \models \phi$.*

Assume standard encodings of languages by classes of structures, and vice versa (see, e.g. [5]).

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 $MX(\sigma, \phi)$.*

This is just a slight re-casting of Fagin's Theorem, which states the existential second-order logic ($\exists SO$) over finite structures captures exactly NP. This is because the expansion relation symbols (those in $vocab(\phi) \setminus \sigma$) behave as existentially quantified SO relation variables. (See [3] for further details.)

Notice that, in modelling problems in NP, we may make completely unrestricted use of first-order quantifiers, as well as function symbols and equality.

Other Complexity Classes Some lower complexity classes can be captured similarly, applying results for various fragments of $\exists$SO [6, 7]. For example, FO universal Horn $MX(\sigma, \phi)$ expresses P over ordered structures.

By using second-order quantifiers, we may also capture higher complexity classes. In particular, for each $k > 0$, the complexity class Σ_k^p of is captured by Π_{k-1}^1 $MX(\sigma, \phi)$. (A Π_k^1 formula is a SO formula in which there are k alternations of second-order quantifiers, in which the first quantifier is universal).

To illustrate representing problems above NP in the polynomial heirarchy, we choose the simple example of QSAT_2, the problem of deciding the truth of a quantified boolean formula ψ of the form $\exists \boldsymbol{x} \forall \boldsymbol{y} \theta$, which is Σ_2^p-complete.

Example 5. **($\mathbf{QSAT}_2$)** Consider a QSAT formula of the form $\exists \boldsymbol{x} \forall \boldsymbol{y} \theta$, where θ is propositional. For simplicity, we assume that θ is in CNF. Since our interest is in search problems, we formally address the witnessing problem, which is to find a truth assignment for the existentially quantified variables which makes the formula true.

The instance structure I is a two-sorted structure with domains for clauses and literals. The instance vocabulary is $\sigma = \{\text{InClause}, \text{Complements}, \text{Existential}\}$, where:

- $InClause(l, c)$ means that literal l occurs in clause c of ψ,
- $Complements(l_1, l_2)$ means that l_1 and l_2 are complementary literals,
- $Existential(l)$ means that the propositional variable for literal l is existentially quantified in ψ.

The expansion vocabulary (i.e., $vocab(\psi) \backslash \sigma$) contains one relation symbol ε, which will denote truth assignments to the existential literals. The relation gives the set of literals which are mapped to true. We will also use the relation variable symbol τ, which will be universally quantified, and will be used to talk about all truth assignments to θ. The first-order variables l and l' will be taken to range over literals. The second-order variables τ, ε and α range over sets of literals.

First, we axiomatize the property that relations which encode truth assignments map complementary literals to complementary values:

$$\forall \alpha \ (Consistent(\alpha) \equiv \forall l \ \forall l' \ (Complements(l, l') \supset \alpha(l) \equiv \neg \alpha(l'))).$$

Then we declare that ε satisfies this property:

$$Consistent(\varepsilon).$$

We must also ensure that ε gives truth values to exactly the set of (literals associated with) existentially quantified variables:

$$\forall l \ (\varepsilon(l) \equiv Existential(l)).$$

Now we axiomatize the main property, which is that every truth assignment τ for θ that agrees with ε on the assignments to existentially quantified variables satisfies θ:

$$\begin{aligned} \forall \tau \ [&(Consistent(\tau) \wedge \forall l \ (Existential(l) \supset \tau(l) \equiv \varepsilon(l))) \\ &\supset \ [\forall c \ \exists \ (InClause(c, l) \wedge \tau(l))]) \end{aligned}$$

Note that model expansion does not naturally capture Π_k^p levels. This is a consequence of the fact that every model expansion problem has an implicit second-order existential quantifier, which in turn is a consequence of our explicit decision to model search problems. (Indeed, if there are some σ and ϕ so that $MX(\sigma, \phi)$ is Π_k^p-complete, then PH collapses to the k-th level. In particular, if there are σ and ϕ such that $MX(\sigma, \phi)$ is co-NP-complete, then NP=co-NP.)

4 Inductive Definitions

Formally, FO model expansion has the same expressive power as ∃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 expansion. 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 classical logic with inductive definitions. This 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 [8, 9]). Inductive definition are useful in commonsense reasoning, as well as mathematics. For instance, it was shown [10] 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 $\forall \boldsymbol{x}\ (X(\boldsymbol{t}) \leftarrow \varphi)$, where $\boldsymbol{x}$ is a tuple of variables, X is a relation symbol of some arity r, $\boldsymbol{t}$ 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 $\forall \boldsymbol{x}\ (X(\boldsymbol{t}) \leftarrow \varphi)$ in a definition does not correspond to the disjunction $\forall \boldsymbol{x}(X(\boldsymbol{t}) \vee \neg\varphi)$ 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 $\forall \boldsymbol{x}\ (X(\boldsymbol{t}) \leftarrow \varphi)$, $X(\boldsymbol{t})$ 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 and SO semantics with the well-founded semantics of logic programming [11–13]. For precise details see [9]. For an intuitive explanation of the well-founded semantics and why it formalizes different forms of inductive definitions see [13]. Effective modelling requires modularity, which comes naturally with classical logic, but is non-trivial when recursion through negation is present. Modularity conditions for ID-logic have been given in [9].

Example 6. **(Transitive Closure)** We represent the problem of finding the transitive closure of a graph as a model expansion problem. The input vocabulary σ consists of a single symbol E, which represents the binary edge relation.

The universe of the input structure I is the set of vertices 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 7. **(Minimum Horn Model)** 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 expansion. 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 expansion 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:

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

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 expansion of 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 expansion 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 $MX(\sigma,\phi)$.*

This might seem surprising at first. Extending FO with inductive definitions do not increase the complexity of model expansion because, once relations have been chosen for the open relation symbols in a definition, relations for the defined symbols can be computed in polynomial time.

5 CSP and ASP as Model expansion

In this section, we encode CSP and ASP as parameterized model expansion. The encodings of these NP-complete problems take advantage of the availability of first-order quantifiers, including quantifier alternation.

CSP as Model expansion 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 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. Formula ϕ is:

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

ASP as Model expansion 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 7, above. $Neg(r, a)$ denotes that the occurrence of atom a in the body of r is negated. The expansion 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.

6 The Min-Open-Stacks Problem

For more on this problem, see [14]. We encode a search version of the problem, namely: given a 0-1 matrix A and positive integer k, find a permutation of the columns of A with cost at most k. Cost is defined as follows. Suppose $A = (A^1, \ldots A^n)$ is an $m \times n$ matrix, and σ a permutation of $\{1 \ldots n\}$. Then the cost of σ is the maximum cost of any column of the array $A^\sigma = (A^{\sigma(1)} \ldots A^{\sigma(n)})$. The cost of column i is the total number of rows j such that either $a_{j,i} = 1$ or there are columns $i1$ and $i2$ such that $i1 < i < i2$ and $a_{j,i1} = a_{j,i2} = 1$, or in other words there is a cell with a 1 both preceding and following column i in row j.

The input structure will consist of the relation A, which specifies the matrix, and constant k. Relation A consists of all those pairs (i, j) such that $a_{i,j} = 1$. For convenience only, we also include two relations $Rows$ and $Cols$ which give the indices of the rows and columns of A.

In the formula, we also have a predicate symbol P, for the permutation σ, and for convenience only, a predicate symbol B which will be a matrix based on the permuted version of A. The formula ϕ will state that P is a permutation of the set of column indices, that B is like a permuted version of A but with extra 1's, so that the cost of a column in B is just the number of 1's in that column, and that the cost of each column of B is at most k. We use $\forall x \in Rows\ \psi$ to abbreviate $\forall x(Rows(x) \supset \psi)$ and $\exists x \in Rows\ \psi$ to abbreviate $\exists x(Rows(x) \wedge \psi)$. We also use the predicate symbol $AtMost(n, setOfFormulas)$, which is a cardinality constraint, and $\leq$. These are taken to have their natural interpretations, and can be expected to be built-in to any realistic solver. Formula ϕ is the conjunction of the following formulas,

$$\forall x \in Cols\ \exists y \in Cols\ P(x, y)$$

$$\forall x \in Cols\ \exists y \in Cols\ P(y, x)$$

$$\forall x \in Cols\ \forall y \in Cols\ \forall z \in Cols\ \ P(x, y) \wedge P(x, z) \supset y = z$$

$$\left\{ \begin{array}{l} \forall r \in Rows\ \forall c \in Cols\ (B(r, c) \\ \quad \leftarrow \exists x \in Cols\ P(x, c) \wedge A(r, c) \\ \quad\quad \vee \exists i \in Cols\ \exists j \in Cols\ (i \leq c \leq j \wedge B(r, i) \wedge B(r, j))) \end{array} \right\}$$

$$\forall x \in Cols\ AtMost(k, \{B(c, r) : r \in Cols\})$$

The solution will be the extent of the predicate P, which specifies a permutation with the required properties.

7 The Game of Connect-C

The game of connect-4 involves two players, Red and White, placing pieces on a six by seven grid. On a turn, a player must place a piece of his own colour in a column of his choice, where it falls to the bottom-most unoccupied cell in the column. Red goes first. The winner is the first player to get four pieces in a row

vertically, horizontally, or on either diagonal. The game is a draw if the board fills without a winner.

The game was proposed as a challenge problem for researchers studying QSAT by Toby Walsh [15]. The game is known to be a win for the first player, but encoding it and proving this to be the case automatically appears non-trivial. A QSAT encoding of the generalized game (connect-c, for a board of size m by n) was given in [16]. For the connect-4 case, the encoding generates QSAT instance with 18687 variables and 70946 clauses, and with 17 quantifier alternations. We claim our encoding is simpler.

Here, we encode the generalized game as model expansion, where the search problem is to construct a winning strategy for Red. The game can be modelled in several ways. The model we present here is in the style of situation calculus, although a bit different from a situation calculus axiomatization. The quantifier alternation depth of the encoding is 2.

Each move is represented by a natural number, which specifies the column where a piece is being placed. Since the order in which players move is fixed, we may record any sequence of moves by the sequence of natural numbers. Any such sequence also uniquely determines the resulting board layout. For example, sequence $(8, 3, 4)$, represents the result of putting a red piece to column 8, then white to column 3, and then red to column 4. These sequences are like situations (i.e., sequences of actions) in the situation calculus.

The initial (instance) structure I is a two-sorted structure. One domain, call it $\mathcal{N}$, is a set of natural numbers from 1 to $n \times m$, where n and m are the dimensions of the game grid. The second domain is the set of sequences of natural numbers between 1 and m (the number of columns in the grid) of lengths from 0 and $n \times m$. The elements of this domain correspond to positions of the game. We denote this second domain by $\mathcal{D}$. The vocabulary of the structure I is

$$\sigma = \{S_0,\ c,\ n,\ m,\ \mathit{last\text{-}element},\ \mathit{length},\ \sqsubseteq,\ \leq,\ +\}.$$

- Constant S_0 denotes the initial situation, interpreted as the empty sequence in $\mathcal{S}$.
- Constant c is the number of pieces in a row required to win. It is interpreted by the natural number c in the domain $\mathcal{N}$.
- Constants n and m represent the last row and the last column number of the game grid, respectively.
- Function *last-element* denotes a mapping $f : \mathcal{S} \to \mathcal{N}$ from a situation to the last action performed, i.e., from a sequence to the last element of that sequence.
- Unary function *length* denotes the length of a list.
- Relation $\sqsubseteq$ is a prefix relation defined on pairs of situations (i.e., pairs of sequences). Here, $s \sqsubseteq s'$ denotes that s' is reachable from (is an extension of) s.
- Function $+$ and relation $\leq$ on natural numbers have the standard meaning.

We do not axiomatize the functions and relations on sequences. Although it is straightforward to do so, we think it reasonable to expect that any serious

implemented system would support some kind of list data structure and have such functions and the relations built-in.

We use the following expansion predicates:

- $R(i, j, s)$ — cell (i, j) is filled by Red in situation s;
- $W(i, j, s)$ — cell (i, j) is filled by White in situation s;
- $B(i, j, s)$ — cell (i, j) is the 'bottom' cell (i.e., the next cell to be filled in) in column j;
- $R\text{-}wins(s)$ — Red wins in situation s;
- $W\text{-}wins(s)$ — White wins in situation s;
- $R\text{-}turn(s)$ — in situation s, it is Red's turn to move;
- $Tr(s, s')$ — there is a transition from situation s to situation s', i.e., s' is the result of appending one move to the end of s.

In addition, the expansion vocabulary contains a strategy function δ which maps situations to actions to be performed next. This function will be unrestricted in situations where it's White's turn to move, and will specify a winning strategy for Red in the other situations. The model expansion task is to find the extensions of these predicates, and the strategy function. We assume that the positions in the stacks are numbered top-to-bottom, that is the top position is 1, and the bottom (first to be filled) position is n.

Our problem description is the conjunction of the following formulas.
Initially, it is Red's turn to move:

$$R\text{-}turn(S_0)$$

There are no pieces on the board in the initial situation:

$$\forall i \forall j \ (\neg R(i, j, S_0) \wedge \neg W(i, j, S_0)).$$

Initially, the first positions to be filled are located at the bottom, i.e., at the row of the grid numbered n:

$$\forall j \ B(n, j, S_0).$$

The transition relation holds for any pairs of situations such that the second element of the pair is one action longer than the first:

$$\forall s \forall s' \ (Tr(s, s') \ \equiv \ length(s') = length(s) + 1).$$

The successor situations are described as follows:

$$\forall s \ (R\text{-}turn(s') \equiv Tr(s, s') \wedge \neg R\text{-}turn(s))$$

$$\forall i \forall j \forall s \ (B(i, j, s') \equiv Tr(s, s') \wedge B(i-1, j, s) \wedge last\text{-}element(s) = j \vee B(i-1, j, s) \wedge last\text{-}element(s) \neq j)$$

$$\forall i \forall j \forall s \ (R(i, j, s') \equiv Tr(s, s') \wedge B(i, j, s) \wedge last\text{-}element(s) = j \wedge R\text{-}turn(s) \vee R(s))$$

$$\forall i \forall j \forall s \ (W(i, j, s') \equiv Tr(s, s') \wedge B(i, j, s) \wedge last\text{-}element(s) = j \wedge \neg R\text{-}turn(s) \vee W(s))$$

Now we describe what it means for a player to win a game. First, we introduce the following abbreviation denoting the situation in which there are c red (or white) pieces in a row.

$$
\begin{array}{l}
\forall s\ (c\text{-}R\text{-}in\text{-}a\text{-}row(s) \equiv \\
\exists i \exists j\ \ 0 \leq i < n\ \wedge\ 0 \leq j \leq m - c\ \wedge\ \forall k\ [0 \leq k < c \supset R(i, j + k, s)] \\
\vee\ \exists i \exists j\ \ 0 \leq i \leq n - c\ \wedge\ 0 \leq j < m\ \wedge\ \forall k\ [0 \leq k < c \supset R(i + k, j, s)] \\
\vee\ \exists i \exists j\ \ 0 \leq i \leq n - c\ \wedge\ 0 \leq j \leq m - c\ \wedge\ \forall k\ [0 \leq k < c \supset R(i + k, j + k, s)] \\
\vee\ \exists i \exists j\ \ 0 \leq i \leq n - c\ \wedge\ c - 1 \leq j < m\ \wedge\ \forall k\ [0 \leq k < c \supset R(i + k, j - k, s)])
\end{array}
$$

Winning situations for each player are defined inductively, by mutual recursion:

$$
\left\{
\begin{array}{l}
R\text{-}wins(S_0) \leftarrow false, \\
W\text{-}wins(S_0) \leftarrow false, \\
\forall s \forall s'\ (R\text{-}wins(s') \\
\qquad \leftarrow Tr(s, s') \wedge R\text{-}turn(s) \wedge c\text{-}R\text{-}in\text{-}a\text{-}row(s') \wedge \neg W\text{-}wins(s) \\
\qquad \vee R\text{-}wins(s)), \\
\forall s \forall s'\ (W\text{-}wins(s') \\
\qquad \leftarrow Tr(s, s') \wedge \neg R\text{-}turn(s) \wedge c\text{-}W\text{-}in\text{-}a\text{-}row(s') \wedge \neg R\text{-}wins(s) \\
\qquad \vee W\text{-}wins(s))
\end{array}
\right\}
$$

8 Future Work

Many tasks need to be carried out to turn our approach into a practical tool. Primary among these are:

• Development of practical modelling languages, based on experiments and experience with solving a variety of application problems. Classical logic is likely not an acceptable modelling language for industrial users. In analogy 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, we expect to produce industrial languages which are syntactic variants of FO(ID) and SO(ID).
• Further work on solver design and implementation. A prototype solver based on reduction to SAT is described in [17]. Native solvers are now under development. These will, in the manner of ASP solvers, involve combining the instance and problem description to produce a ground problem, and then using propositional solvers based on SAT-solver technology to find models to the ground instance.
• Adding aggregates and interpreted function and 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.
• Work on tractable cases and on cases which admit efficient grounding.

References

1. Kautz, H., McAllester, D., B., S.: Encoding plans in propositional logic. In: Proc., of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR-96), Morgan Kaufmann (1996) 374–384
2. East, D., Truszczynski, M.: Predicate-calculus based logics for modeling and solving search problems. ACM TOCL (2004) To appear.
3. Mitchell, D., Ternovska, E.: A framework for representing and solving NP search problems. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05). (2005) 430–435
4. Mitchell, D.G., Ternovska, E.: Model extension as a framework for solving NP-Hard search problems, (2005) Presented at the Seventh International Workshop on Logic and Computational Complexity (LCC-05), (http://www.cis.syr.edu/~royer/lcc/LCC05/).
5. Libkin, L.: Elements of Finite Model Theory. Springer (2004)
6. Leivant, D.: Descriptive characterizations of computational complexity. In: Second Annual Conference on Structure in Complexity Theory, IEEE Computer Society (1987) 203–217
7. Graedel, E.: Capturing complexity classes by fragments of second order logic. Theoretical Computer Science **101** (1992) 35–57
8. Denecker, M.: Extending classical logic with inductive definitions. In: Proc. CL'2000. (2000)
9. Denecker, M., Ternovska, E.: A logic of non-monotone inductive definitions and its modularity properties. In: Proc., LPNMR-04. (2004)
10. Denecker, M., Ternovska, E.: Inductive situation calculus. In: Proc., KR-04. (2004)
11. Van Gelder, A.: An alternating fixpoint of logic programs with negation. Journal of computer and system sciences **47** (1993) 185–221
12. Fitting, M.: Fixpoint semantics for logic programming - a survey. Theoretical Computer Science (2003) To appear.
13. Denecker, M., Bruynooghe, M., Marek, V.: Logic programming revisited: Logic programs as inductive definitions. ACM Transactions on Computational Logic (TOCL) **4** (2001)
14. Smith, B., Gent, I.: Constraint modelling challenge 2005 (2005) http://www.dcs.st-and.ac.uk/ ipg/challenge/.
15. Walsh, T.: Challenges for SAT and QBF (2003) SAT-03 Invited Talk.
16. Gent, I., Rowley, A.: Encoding connect-4 using quantified boolean formulae. Technical Report APES-68-2003, APES Research Group (2003) Available from (http://www.dcs.st-and.ac.uk/ apes/apesreports.html).
17. Pelov, N., Ternovska, E.: Reducing ID-Logic to propositional satisfiability. In: Proceedings of the Twentyfirst International Conference on Logic Programming (ICLP 2005). (2005)