Grounding for Model Expansion in k-Guarded Formulas with Inductive Definitions

Murray Patterson, Yongmei Liu, Eugenia Ternovska, Arvind Gupta
School of Computing Science
Simon Fraser University, Canada
{murrayp,y_liu,ter,arvind}@cs.sfu.ca

Abstract

Mitchell and Ternovska [2005] proposed a constraint programming framework based on classical logic extended with inductive definitions. They formulate a search problem as the problem of model expansion (MX), which is the problem of expanding a given structure with new relations so that it satisfies a given formula. Their long-term goal is to produce practical tools to solve combinatorial search problems, especially those in **NP**. In this framework, a problem is encoded in a logic, an instance of the problem is represented by a finite structure, and a solver generates solutions to the problem. This approach relies on propositionalisation of high-level specifications, and on the efficiency of modern SAT solvers. Here, we propose an efficient algorithm which combines grounding with partial evaluation. Since the MX framework is based on classical logic, we are able to take advantage of known results for the so-called guarded fragments. In the case of k-guarded formulas with inductive definitions under a natural restriction, the algorithm performs much better than naive grounding by relying on connections between k-guarded formulas and tree decompositions.

1 Introduction

NP search and decision problems occur widely in AI; modelling these problems as SAT, CSP and ASP and then using a corresponding solver are perhaps the most successful declarative approaches (in practice) to solving these problems. While each approach has made valuable contributions, they also have limitations. Modelling in propositional logic and then using a SAT solver is the oldest and the most developed of these approaches; the simplicity of the semantics has been the key driver in its success. However, propositional logic provides a poor modelling language in which there are no quantifiers, nor recursion. Moreover, in modelling a problem as SAT, there is no clear separation of instance and problem description. CSP provides somewhat better modelling capabilities, and CSP solvers rely on a number of solution techniques such as no-good learning and backjumping. CSP solvers, however, have found success primarily as components of general purpose programming languages, and are thus not purely declarative. ASP emerged from logic programming, and is based on the stable model semantics [Gelfond and Lifschitz, 1991]. The main advantage of ASP is that its language allows an implicit use of quantifiers and has a built-in recursion mechanism. However, some concepts that are natural in classical logic are difficult to express in ASP.

Mitchell and Ternovska [2005] proposed a declarative constraint programming framework based on classical logic extended with inductive definitions. They cast search problems as the classical problem of *model expansion* (MX), which is the problem of expanding a given structure with new relations so that it satisfies a given formula. Their long-term goal is to develop tools for solving hard combinatorial search problems, especially those in **NP**. In this framework, a problem is encoded in a logic such as first-order logic (FO) or an extension of FO, a problem instance is represented by a finite structure, and a solver generates extensions of initially unspecified predicates, which represent solutions to the problem.

The MX framework combines many strengths of these three mentioned approaches, as well as addresses their limitations. Its features include: a high-level modelling language that supports quantification and recursion, and a clear separation of the instance (a finite structure) from the problem description (a formula). Most importantly, the framework is based on classical logic, making it possible to exploit the many existing results from finite model theory and descriptive complexity [Kolokolova *et al.*, 2006]. In addition, axiomatizations are easier to understand, and we need not rely on stable model semantics to imitate the 'classical' part of the program, a common practice in ASP.

A key component in building practical solvers for the MX framework is *propositionalisation* (or *grounding*) of high-level specifications and the use of efficient SAT solvers. The idea is to first obtain a ground formula that contains only initially unspecified predicates and is satisfiable iff the original MX problem has a solution, and then call a propositional satisfiability solver. Two prototype solvers for propositional formulas with inductive definitions have been developed [Pelov and Ternovska, 2005; Mariën *et al.*, 2005].

The goal of this paper is to develop an efficient grounding algorithm for the MX framework. Since the framework is based on classical logic, we take advantage of results known for the guarded fragments [Andréka *et al.*, 1998;

Gottlob *et al.*, 2001]. We present a polytime algorithm for grounding k-guarded FO sentences with inductive definitions provided that all predicates in the guards are initially specified (a fragment which we call IGF$_k$). As explained later, this restriction is necessary for polytime grounding, and IGF$_k$ contains (in terms of expressive power) sentences with treewidth at most k. In practice, most sentences have small treewidth and thus can be put into IGF$_k$ with small k. Our algorithm runs in $O(\ell^2 n^k)$ time, where ℓ is the size of the sentence, and n is the size of the structure.

2 Grounding for Model Expansion (MX)

We formulate search problems as the problem of *model expansion* (MX).

Definition 2.1 (MX) *Given a sentence ϕ and a finite structure $\mathcal{A}$ with vocabulary $\sigma \subseteq vocab(\phi)$, find a structure $\mathcal{B}$ that is an expansion of $\mathcal{A}$ to $vocab(\phi)$ such that $\mathcal{B} \models \phi$.*

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

$$\begin{array}{l} \forall x[(R(x) \vee B(x) \vee G(x)) \wedge \neg(R(x) \wedge B(x)) \\ \quad \wedge\neg(R(x) \wedge G(x)) \wedge \neg(B(x) \wedge G(x))] \\ \bigwedge \forall x \forall y[E(x,y) \supset (\neg(R(x) \wedge R(y)) \\ \quad \wedge\neg(B(x) \wedge B(y)) \wedge \neg(G(x) \wedge G(y)))] \end{array}$$

Let $\mathcal{B}$ be an expansion of $\mathcal{A}$ to $vocab(\phi)$. Then $\mathcal{B} \models \phi$ iff it corresponds to a 3-coloring of G.

In the MX framework, there is a clear separation between problem instance (finite structure) and problem description (a formula). In grounding, we bring domain elements into the syntax by expanding the vocabulary and associating a new constant symbol with each element of the domain. For domain A, we denote the set of such constants by $\tilde{A}$. As notational conventions, we use σ to denote the instance vocabulary and ε the expansion vocabulary.

Definition 2.2 (Reduced Grounding for MX) *A formula ψ is a* reduced grounding *of a formula ϕ over a σ-structure $\mathcal{A} = (A, \sigma^{\mathcal{A}})$ if (1) ψ is a ground formula over $\varepsilon \cup \tilde{A}$; and (2) for every expansion structure $\mathcal{B} = (A, \sigma^{\mathcal{A}}, \varepsilon^{\mathcal{B}})$ over $vocab(\phi)$, $\mathcal{B} \models \phi$ iff $(\mathcal{B}, \tilde{A}^{\mathcal{B}}) \models \psi$, where $\tilde{A}^{\mathcal{B}}$ denotes the interpretation of the new constants $\tilde{A}$.*

During grounding, the symbols of the instance vocabulary are "evaluated out" and a reduced grounding is obtained.

Obviously, by the above definition, we have

Proposition 2.3 *Let ψ be a reduced grounding of a formula ϕ over a σ-structure $\mathcal{A}$. Then $\mathcal{A}$ can be expanded to a model of ϕ iff ψ is satisfiable.*

Thus, through grounding, the model expansion problem is reduced to the satisfiability problem.

In our grounding algorithm for a sentence, we must deal with subformulas ϕ of the original sentence (ϕ may contain free variables). For a σ-structure $\mathcal{A}$, we compute a formula ψ for each instantiation of the free variables of ϕ, such that each ψ is a reduced grounding of ϕ under the corresponding instantiation. We call such a representation *an answer to ϕ wrt $\mathcal{A}$*. We compute an answer to a formula, by computing answers to its subformulas and combining them according to the connectives. To represent such answers, we introduce the concept of an extended relation (extending the notion of a relation from relational database theory).

Definition 2.4 (extended X-relation) *Let $\mathcal{A} = (A; \sigma^{\mathcal{A}})$, and X be the set of free variables of formula ϕ. An* extended X-relation *$\mathcal{R}$ over A is a set of pairs (γ, ψ) s.t. (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 s.t. $(\gamma, \psi) \in \mathcal{R}$.*

An extended X-relation $\mathcal{R}$ is intended to represent a unique mapping (denoted by δ_R) from all possible instantiations of variables in X to ground formulas. For γ's not appearing in $\mathcal{R}$, the associated formula is $false$. We write $\gamma \in \mathcal{R}$ to mean that there exists ψ such that $(\gamma, \psi) \in \mathcal{R}$.

Definition 2.5 (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.*

As an example, let $\phi = \exists x \exists y \exists z \phi'$ where $\phi' = P(x,y,z) \wedge E(x,y) \wedge E(y,z)$, $\sigma = \{P\}$, and $\varepsilon = \{E\}$. Let $\mathcal{A}$ be a σ-structure such that $P^{\mathcal{A}} = \{(1,2,3),(3,4,5)\}$. Then this extended relation $\mathcal{R}$

x	y	z	ψ
1	2	3	$E(1,2) \wedge E(2,3)$
3	4	5	$E(3,4) \wedge E(4,5)$

is an answer to ϕ' wrt $\mathcal{A}$. It is easy to see, for example, that $\delta_{\mathcal{R}}(1,2,3) = E(1,2) \wedge E(2,3)$ is a reduced grounding of $\phi'[(1,2,3)] = P(1,2,3) \wedge E(1,2) \wedge E(2,3)$, and $\delta_{\mathcal{R}}(1,1,1) = false$ is a reduced grounding of $\phi'[(1,1,1)]$. The following extended relation is an answer to $\phi'' = \exists z \phi'$

x	y	ψ
1	2	$E(1,2) \wedge E(2,3)$
3	4	$E(3,4) \wedge E(4,5)$

Here, for example, $E(1,2) \wedge E(2,3)$ is a reduced grounding of $\phi''[(1,2)]$. Finally, the following represents an answer to ϕ, where the single formula is a reduced grounding of ϕ.

ψ
$[E(1,2) \wedge E(2,3)] \vee [E(3,4) \wedge E(4,5)]$

Just as relations in databases have an algebra, *i.e.*, a set of operations whose semantics correspond to the connectives of FO logic, we define an algebra for extended relations. Our algebra consists of join, join with complement, projection, intersection and union.

Definition 2.6 (join, join with complement) *Let $\mathcal{R}$ be an extended X-relation and $\mathcal{S}$ an extended Y-relation, both over domain A. Then*

1. *the* join *of $\mathcal{R}$ and $\mathcal{S}$ is the extended $X \cup Y$-relation $\mathcal{R} \bowtie \mathcal{S} = \{(\gamma, \psi) \mid \gamma : X \cup Y \to A, \gamma|_X \in \mathcal{R}, \gamma|_Y \in \mathcal{S}$, and $\psi = \delta_{\mathcal{R}}(\gamma|_X) \wedge \delta_{\mathcal{S}}(\gamma|_Y)\}$;*
2. *the* join *of $\mathcal{R}$* with the complement *of $\mathcal{S}$ is the extended $X \cup Y$-relation $\mathcal{R} \bowtie^c \mathcal{S} = \{(\gamma, \psi) \mid \gamma : X \cup Y \to A, \gamma|_X \in \mathcal{R}$, and $\psi = \delta_{\mathcal{R}}(\gamma|_X) \wedge \neg\delta_{\mathcal{S}}(\gamma|_Y)\}$.*

Proposition 2.7 *Suppose that $\mathcal{R}$ is an answer to ϕ_1 and $\mathcal{S}$ is an answer to ϕ_2, both wrt structure $\mathcal{A}$. Then (1) $\mathcal{R} \bowtie \mathcal{S}$ is an answer to $\phi_1 \wedge \phi_2$ wrt $\mathcal{A}$; (2) $\mathcal{R} \bowtie^c \mathcal{S}$ is an answer to $\phi_1 \wedge \neg\phi_2$ wrt $\mathcal{A}$.*

Definition 2.8 (**Y-projection**) *Let $\mathcal{R}$ be an extended X-relation and $Y \subseteq X$. The* Y-projection *of $\mathcal{R}$, denoted by $\pi_Y(\mathcal{R})$, is the extended Y-relation $\{(\gamma', \psi) \mid \gamma' = \gamma|_Y$ for some $\gamma \in \mathcal{R}$ and $\psi = \bigvee_{\{\gamma \in \mathcal{R} \mid \gamma|_Y = \gamma'\}} \delta_{\mathcal{R}}(\gamma)\}$.*

Proposition 2.9 *Suppose that $\mathcal{R}$ is an answer to ϕ wrt $\mathcal{A}$, and Y is the set of free variables of $\exists \bar{z}\phi$. Then $\pi_Y(\mathcal{R})$ is an answer to $\exists \bar{z}\phi$ wrt $\mathcal{A}$.*

Definition 2.10 (intersection, union) *Let $\mathcal{R}$ and $\mathcal{S}$ be extended X-relations. Then*

1. *the* intersection *of $\mathcal{R}$ and $\mathcal{S}$ is the extended X-relation $\mathcal{R} \cap \mathcal{S} = \{(\gamma, \psi) \mid \gamma \in \mathcal{R}$ and $\gamma \in \mathcal{S}$, and $\psi = \delta_{\mathcal{R}}(\gamma) \wedge \delta_{\mathcal{S}}(\gamma)\}$;*
2. *the* union *of $\mathcal{R}$ and $\mathcal{S}$ is the extended X-relation $\mathcal{R} \cup \mathcal{S} = \{(\gamma, \psi) \mid \gamma \in \mathcal{R}$ or $\gamma \in \mathcal{S}$, and $\psi = \delta_{\mathcal{R}}(\gamma) \vee \delta_{\mathcal{S}}(\gamma)\}$.*

Proposition 2.11 *Let ϕ_1 and ϕ_2 be formulas with the same set of free variables. Suppose that $\mathcal{R}$ is an answer to ϕ_1 and $\mathcal{S}$ is an answer to ϕ_2, both wrt $\mathcal{A}$. Then (1) $\mathcal{R} \cap \mathcal{S}$ is an answer to $\phi_1 \wedge \phi_2$ wrt $\mathcal{A}$; (2) $\mathcal{R} \cup \mathcal{S}$ is an answer to $\phi_1 \vee \phi_2$ wrt $\mathcal{A}$.*

In traditional relational algebra, projection can be done in linear time, and join can be done in time linear in the sum of the size of the input and output relations. Detailed algorithms and proofs for this can be found in the appendix of [Flum *et al.*, 2002]. The essential idea is this: first encode domain elements appearing in the input relations by natural numbers; then sort the encoded input relations using the bucket sort algorithm; next, compute the projection or join of the encoded input relations; finally, decode the resulting output relation. In our implementation of the operations on extended relations, we use the same idea except that we also need to copy formulas from the input relations to the output relation.

Let $\mathcal{R}$ be an extended X-relation. The size of $\mathcal{R}$, denoted by $\|\mathcal{R}\|$, is the sum of the size of each pair $(\gamma, \delta_{\mathcal{R}}(\gamma))$ in $\mathcal{R}$. Thus $\|\mathcal{R}\| = |X||\mathcal{R}| + \sum_{\gamma \in \mathcal{R}} \|\delta_{\mathcal{R}}(\gamma)\|$, where $|\mathcal{R}|$ denotes the cardinality of $\mathcal{R}$ as a set, and $\|\delta_{\mathcal{R}}(\gamma)\|$ is the size of the formula associated with γ. It is easy to see that projection takes $O(\|\mathcal{R}\|)$ time, and intersection and union take time $O(\|\mathcal{R}\| + \|\mathcal{S}\|)$. In general, join and join with complement take time $O(\|\mathcal{R}\| \cdot \|\mathcal{S}\|)$. However, in our grounding algorithm in the next section, whenever we perform an $\mathcal{R} \bowtie \mathcal{S}$ or $\mathcal{R} \bowtie^c \mathcal{S}$ operation, it is always the case that the set of variables of $\mathcal{S}$ is a subset of that of $\mathcal{R}$, every formula in $\mathcal{R}$ is $true$, and every formula in $\mathcal{S}$ is of size $O(\ell)$, where ℓ is the size of the formula to be grounded. So the relation part (the set of tuples) of the resulting extended relation is a subset of that of $\mathcal{R}$. Thus both operations can be done in time $O(\ell n)$, where n is the size of the relation part of $\mathcal{R}$.

3 An Algorithm for Grounding FO Formulas

The guarded fragment GF of FO was introduced by Andréka *et al.* [1998]. Here, any quantified subformula ϕ must be relativized by a guard, *i.e.*, an atomic formula over all free variables of ϕ. 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, and proved that k-guarded sentences can be evaluated in time $O(\ell^2 n^k)$ where ℓ is the size of the formula, and n is the size of the structure. The proof is by transforming GF$_k$-sentences into k-guarded Non-Recursive Stratified Datalog (NRSD) programs.

A fragment related to GF$_k$ is the bounded variable fragment of FO – FO^k, which denotes FO formulas that use at most k distinct variables. Kolaitis and Vardi [1998] showed that conjunctive queries in FO^k have the same expressive power as conjunctive queries of treewidth at most k. Gottlob *et al.* [2001] obtained a similar logical characterization for the notion of hypertreewidth, a generalization of treewidth. They noted that hypertreewidth is bounded above by treewidth and showed that GF$_k$ has the same expressive power as FO sentences of hypertreewidth at most k.

Definition 3.1 (GF$_k$) *The* k-guarded fragment *GF$_k$ of FO is the smallest set of formulas s.t. (1) GF$_k$ contains atomic formulas; (2) GF$_k$ is closed under Boolean operations; (3) GF$_k$ contains $\exists \bar{x}(G_1 \wedge \ldots \wedge G_m \wedge \phi)$, if the G_i are atomic 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.*

Note that this definition also includes universal quantification, since by (2), GF$_k$ is closed under negation. Also 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, by the definition, $\neg R(x, y, z)$ is a 1-guarded formula, but it cannot be evaluated in $O(n)$ time. However, the same complexity result applies to strictly k-guarded formulas, defined as follows:

Definition 3.2 (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 $m = 0$).

Liu and Levesque [2003] presented a polytime algorithm for evaluating strictly k-guarded formulas: Given a structure $\mathcal{A}$ and a formula $\phi \in$ SGF$_k$, the algorithm computes *the answer to ϕ wrt $\mathcal{A}$*, that is, $\{\gamma \mid \mathcal{A} \models \phi[\gamma]\}$, in time $O(\ell n^k)$. A reason that they considered strictly k-guarded formulas instead of just k-guarded sentences is that their algorithm is recursively defined. In this section, we adapt their algorithm for grounding formulas in the following fragment:

Definition 3.3 (RGF$_k$) *RGF$_k$ denotes the set of strictly k-guarded formulas such that no expansion predicate appears in any guard.*

Note that the restriction that "no expansion predicate appears in any guard" is necessary for polytime grounding. Indeed, there is no polytime grounding algorithm for 1-guarded sentences; otherwise, we would have a polytime reduction to SAT from MX for 1-guarded sentences, and hence the combined complexity of this problem would be in **NP**. However, it is **NEXP**-complete since MX for FO can be reduced to MX for 1-guarded sentences [Kolokolova *et al.*, 2006], and the combined complexity of MX for FO is **NEXP**-complete.

It is easy to see that any FO^k formula can be rewritten in linear time into an equivalent one in RGF$_k$, by using atoms of the form $x = x$ as parts of the guards when necessary. For example, the formula $\exists x \exists y[R(x) \wedge E(x,y)]$ can be rewritten into $\exists x \exists y[R(x) \wedge y = y \wedge E(x,y)]$, where R is an instance predicate, and E is an expansion predicate. Extending the result of [Kolaitis and Vardi, 1998], Flum *et al.* [2002] showed that FO^k has the same expressive power as FO formulas of treewidth at most k. Thus any FO formula with treewidth at most k can be put into an equivalent one in RGF$_k$. In practice, most formulas have small treewidth and thus can be put into RGF$_k$ with small k.

Our grounding algorithm uses extended relations and operations on them. 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 Gnd$(\mathcal{A}, \phi)$
Input: A structure $\mathcal{A}$ and a formula $\phi \in$ 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 $\pi_{\bar{x}}$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.

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

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

Unfortunately, the running time of Gnd is not $O(\ell n^k)$ anymore. The reason is that Gnd may generate intermediate extended relations whose size is not $O(n^k)$. To see why, let ϕ be the 1-guarded formula $\exists x[R(x) \wedge \exists y(S(y) \wedge E(y))]$, where E is an expansion predicate, and let $\mathcal{A}$ be a structure such that $R^{\mathcal{A}} = \{2i + 1 \mid 0 \leq i \leq m\}$, and $S^{\mathcal{A}} = \{2i \mid 0 \leq i \leq m\}$. Obviously, the formula $\bigvee_{i=0}^{m} E(2i)$ is a reduced grounding of ϕ over $\mathcal{A}$, and it has size $O(n)$. However, when we use Gnd, the intermediate extended relation corresponding to the subformula $R(x) \wedge \exists y(S(y) \wedge E(y))$ has size $O(n^2)$, as shown in the following table (on the left):

x	ψ
1	$\bigvee_{i=0}^{m} E(2i)$
3	$\bigvee_{i=0}^{m} E(2i)$
$\vdots$	$\vdots$
$2m+1$	$\bigvee_{i=0}^{m} E(2i)$

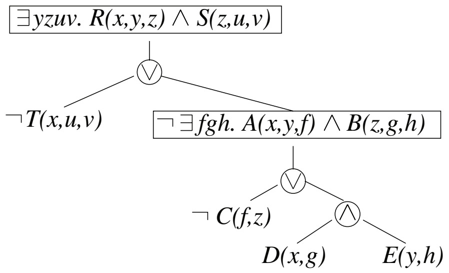
x	ψ
1	p
3	p
$\vdots$	$\vdots$
$2m+1$	p

To solve this problem, we let IGnd be the algorithm which is the same as Gnd except for the following: After each projection operation, we replace each formula in the resulting extended relation by a new propositional symbol. We also save the definitions of these new symbols. The output of the algorithm is the final extended relation together with the definitions of all the new symbols. For instance, in the above example, we will introduce a new propositional symbol p and save the definition $p \equiv \bigvee_{i=0}^{m} E(2i)$. Then the intermediate extended relation corresponding to the subformula $R(x) \wedge \exists y(S(y) \wedge E(y))$ will be as in the table on the right, and it has size $O(n)$.

To illustrate the algorithm, let ϕ be the formula

$$\exists yzuv.\underline{R(x,y,z) \wedge S(z,u,v)} \wedge \neg\{T(x,u,v) \wedge$$
$$\exists fgh.\underline{A(x,y,f) \wedge B(z,g,h)} \wedge [\neg C(f,z) \vee D(x,g) \wedge E(y,h)]\},$$

where T and E are expansion predicates, and let $\mathcal{A}$ be
$R^{\mathcal{A}} = \{(1,2,1), (1,2,2), (7,0,2), (7,0,3)\}$,
$S^{\mathcal{A}} = \{(2,2,5), (2,3,9), (4,0,2), (6,7,8)\}$,
$A^{\mathcal{A}} = \{(1,2,6), (7,0,2), (7,0,5)\}$, $B^{\mathcal{A}} = \{(2,3,4), (3,7,5)\}$,
$C^{\mathcal{A}} = \{(1,2), (2,2), (5,2), (6,2), (6,3), (9,6)\}$,
$D^{\mathcal{A}} = \{(1,2), (1,3), (5,7), (7,3), (8,8)\}$.
Then $\phi \in$ RGF$_2$, where the underlined parts are guards. First, we push negation symbols inward until they are in front of atoms or existentials. The following figure represents the resulting formula by a tree, where rectangular internal nodes stand for guarded existentials, circular internal nodes for disjunctions or conjunctions, and leaves for literals.



Now we process existentials from bottom to top. To process the bottom existential, we first evaluate the guard by a join operation. Then we process leaves C, D and E wrt this guard relation using rules 2, 1 and 3 of IGnd respectively. The result of the guard relation joined with the complement of C is

x	y	f	z	g	h	ψ
7	0	2	3	7	5	$true$
7	0	5	3	7	5	$true$

Next, we perform an intersection for conjunction, a union for disjunction, and a projection for existential, resulting in

x	y	z	ψ
1	2	2	p_1
7	0	2	p_2
7	0	3	p_3

$p_1 \equiv \quad E(2,4)$
$p_2 \equiv \quad E(0,4)$
$p_3 \equiv \quad true$

Now that we have an answer to this bottom existential, we process the top existential in a similar way for the final answer

x	ψ
1	q_1
7	q_2

$q_1 \equiv (\neg T(1,2,5) \vee \neg p_1) \vee (\neg T(1,3,9) \vee \neg p_1)$
$q_2 \equiv (\neg T(7,2,5) \vee \neg p_2) \vee (\neg T(7,3,9) \vee \neg p_2)$

Theorem 3.4 *Given a structure $\mathcal{A}$ and a formula $\phi \in RGF_k$, IGnd 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, IGnd returns a reduced grounding of ϕ over $\mathcal{A}$.)*

Proof Sketch: Rewrite ϕ into an equivalent formula ϕ' by pushing negations inwards. Correctness then follows by induction on the formula according to the algebra for extended relations. As to complexity, for each intermediate extended relation, there are $O(n^k)$ tuples (each guard is composed of at most k atoms), and each formula has size $O(\ell)$, due to the introduction of new propositional symbols. By the complexity analysis from Section 2, each of the $O(\ell)$ operations on extended relations can be done in time $O(\ell n^k)$. ∎

4 Grounding Inductive Definitions

While FO MX is often sufficient, modelling often requires recursion and recursion through negation. The solution is to use FO(ID) logic, which is FO extended with inductive definitions [Denecker, 2000; Denecker and Ternovska, 2004]. Fortunately, adding inductive definitions does not violate the main complexity-theoretic results [Mitchell and Ternovska, 2005; Kolokolova *et al.*, 2006].

The syntax of FO(ID) is that of FO extended with a rule saying that an inductive definition (ID) is a formula. An ID Δ is a set of *rules* of the form $\forall \bar{x}(X(\bar{t}) \leftarrow \phi)$, where X is a predicate symbol, $\bar{t}$ is a tuple of terms, and ϕ is an arbitrary FO formula. The connective $\leftarrow$ is called the definitional implication. In the rule $\forall \bar{x}(X(\bar{t}) \leftarrow \phi)$, $X(\bar{t})$ is called the head and ϕ the body. A defined predicate of an FO(ID) formula is a predicate symbol that occurs in the head of a rule in an ID. The semantics of FO(ID) is that of FO extended with one additional rule saying that a structure $\mathcal{A}$ satisfies an ID Δ if it is the 2-valued well-founded model of Δ, as defined in the context of logic programming [van Gelder *et al.*, 1993].

For example, the problem of finding the transitive closure of a graph can be conveniently represented as an MX problem for FO(ID). The formula consists of a definition with two rules, defining the predicate T. The instance vocabulary has a single predicate E, representing the binary edge relation.

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

This states that the transitive closure of edge set E is the least relation containing all edges and closed under reachability.

When it comes to MX for FO(ID), the expansion vocabulary usually includes all the defined predicates. All the definitions from Section 2, namely definitions of MX, reduced grounding, extended relation, and answer, directly apply to FO(ID). Also, through grounding, the MX problem for FO(ID) is reduced to the satisfiability problem of PC(ID), which stands for propositional calculus with inductive definitions. Currently, two prototypes of such solvers have been developed: one [Pelov and Ternovska, 2005] reduces satisfiability of PC(ID) to SAT, and the other [Mariën *et al.*, 2005] is a direct implementation that incorporates SAT techniques. While, both solvers deal with a restricted PC(ID) syntax, a solver for the general syntax is under construction in our lab.

Nonetheless, some important problems are represented as MX for FO(ID) where some defined predicates are in the instance vocabulary. The interpretation of these defined predicates is used to apply restrictions on the possible interpretations of the expansion predicates. In such a case, we will first do grounding treating all defined predicates as expansion predicates. Let ψ be the resulting ground formula. Then we add to ψ an extra constraint that encodes the interpretation of the defined predicates. Suppose G is the set of ground atoms of interpreted defined predicates that appear in ψ. Let φ be the conjunction of literals of G that are true according to the interpretation. Then $\psi \wedge \varphi$ is the final ground formula to be passed to the satisfiability solver. Thus we can restrict our attention to grounding where all defined predicates are expansion predicates.

In this section, we extend the algorithm from Section 3 to ground FO(ID) formulas in the following fragment:

Definition 4.1 (IGF$_k$) *IGF$_k$ is the extension of RGF$_k$ with inductive definitions such that for each rule, the body is in RGF$_k$ and all free variables of the body appear in the head.*

As with RGF$_k$, any FO(ID) formula with at most k distinct variables can be rewritten in linear time into an equivalent one in IGF$_k$. Here we show how to rewrite each rule so that it satisfies the restriction of the above definition. First, for each x that appears in the head but not the body, add $x = x$ to the body. Now since the body still uses at most k distinct variables, it can be rewritten into RGF$_k$.

We first present a procedure for grounding IDs:

Procedure GndID($\mathcal{A}, \Delta$)
Input: A structure $\mathcal{A}$ and an ID $\Delta \in$ IGF$_k$
Output: An answer to Δ wrt $\mathcal{A}$

For each rule r of Δ, suppose r is $\forall \bar{x}(X(\bar{t}) \leftarrow \phi)$, let Δ_r be $\{X(\bar{t})[\gamma] \leftarrow \psi \mid (\gamma, \psi) \in \text{IGnd}(\mathcal{A}, \phi)\}$. Return $\bigcup_{r \in \Delta} \Delta_r$.

To illustrate the procedure, let Δ be the ID

$$\left\{ \begin{array}{c} \forall x \forall y[T(x,y) \leftarrow \underline{E(x,y)}] \\ \forall x \forall y[T(x,y) \leftarrow \exists z(\underline{C(y) \wedge E(x,z)} \wedge T(z,y))] \end{array} \right\}$$

where T is an expansion predicate. Then $\Delta \in$ IGF$_2$. Note that for the purpose of illustration, we use $C(y)$ instead of $y = y$ as a part of the second guard so that we can produce a small grounding. Let $\mathcal{A}$ be $E^{\mathcal{A}} = \{(1,2),(2,3)\}$, and $C^{\mathcal{A}} = \{(3)\}$. GndID($\mathcal{A}, \Delta$) would proceed as follows:

1. Call IGnd($\mathcal{A}, E(x,y)$) to get $\mathcal{R}$
2. Call IGnd($\mathcal{A}, \exists z(C(y) \wedge E(x,z) \wedge T(z,y))$) to get $\mathcal{S}$

x	y	ψ
1	3	$T(2,3)$
2	3	$T(3,3)$

3. Replace rule 1 with $\{T(x,y)[\gamma] \leftarrow \psi \mid (\gamma, \psi) \in \mathcal{R}\}$
4. Replace rule 2 with $\{T(x,y)[\gamma] \leftarrow \psi \mid (\gamma, \psi) \in \mathcal{S}\}$

The resulting ground ID is

$$\left\{ \begin{array}{c} T(1,2) \leftarrow true \\ T(2,3) \leftarrow true \\ T(1,3) \leftarrow T(2,3) \\ T(2,3) \leftarrow T(3,3) \end{array} \right\}$$

We now extend the grounding algorithm IGnd from Section 3 by adding the following clause. We call this enhanced algorithm for grounding IGF$_k$ formulas IGnd$^+$.

0. If ϕ is an ID Δ or its negation $\neg\Delta$, then IGnd($\mathcal{A}, \mathcal{R}, \phi$) $= \{(\gamma, p) \mid \gamma \in \mathcal{R}\}$ or $\{(\gamma, \neg p) \mid \gamma \in \mathcal{R}\}$, where p is a new propositional symbol with the definition $p \equiv$ GndID($\mathcal{A}, \Delta$).

The following is the main theorem of this paper:

Theorem 4.2 *Given a structure $\mathcal{A}$ and a formula $\phi \in$ IGF$_k$, IGnd$^+$ 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 $\mathcal{A}$.*

5 Related Work

Our grounding algorithm is inspired by the grounding part of the model-checking algorithm of Datalog LITE [Gottlob *et al.*, 2002], which is essentially 1-guarded Datalog and admits model checking in time linear in both the size of the structure and the size of the formula. However, our grounding is aimed for the more general problem of model expansion. Our algorithm also has similarity with the grounding techniques of the ASP systems `Smodels` [Syrjänen and Niemelä, 2001], `Cmodels-2` [Lierler and Maratea, 2004], and `DLV` [Leone *et al.*, 2006], and the constraint programming system `NP-SPEC` [Cadoli and Schaerf, 2001]. These systems produce efficient groundings in practice, however, they use logic programming syntax, which is much more restrictive. The precise complexity of grounding for these systems has not been determined, while we give an apriori time bound. Finally, Ramachandran and Amir [2005] proposed polytime algorithms to propositionalise certain classes of FO theories so that satisfiability is maintained. Their goal is to solve satisfiability problem, not MX.

6 Conclusions

Classical logic is perhaps the most natural language for axiomatizing and solving computational problems. It has a long history, and sophisticated techniques have been developed. Classical logic has intuitive and well-understood semantics, convenient syntax, and it is widely used by both theoreticians and practitioners. We therefore believe that tools based on classical logic to the highest degree possible will have a strong appeal. The MX framework of Mitchell and Ternovska [2005] is a constraint programming framework based on classical logic extended with inductive definitions. This paper represents an important progress in making this framework practical by developing an efficient grounding algorithm. The algorithm performs well on k-guarded FO sentences with inductive definitions under the restriction that all predicates in the guards are initially specified; it runs in time $O(\ell^2 n^k)$, where ℓ is the size of the formula, and n is the size of the structure. To this end, we have proposed the concept of extended relations and defined an algebra on them. The essence of our work is to exploit the structure of FO formulas: Indeed, k-guarded formulas have the same expressive power as formulas with hypertreewidth at most k. A grounder based on the data structure of extended relations has been implemented in our group. It performs comparably on similar inputs to systems mentioned in related work.

References

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

[Cadoli and Schaerf, 2001] M. Cadoli and A. Schaerf. Compiling problem specifications into SAT. In *the European Symp. On Programming (ESOP)*, pages 387–401, 2001.

[Denecker and Ternovska, 2004] M. Denecker and E. Ternovska. A logic of non-monotone inductive definitions and its modularity properties. In *LPNMR*, pages 47–60, 2004.

[Denecker, 2000] M. Denecker. Extending classical logic with inductive definitions. In *Comput. Logic*, pages 703–717, 2000.

[Flum *et al.*, 2002] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

[Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[Gottlob *et al.*, 2001] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. In *Proc. of the 20th ACM Symp. on Principles of Database Systems (PODS)*, pages 195–206, 2001.

[Gottlob *et al.*, 2002] G. Gottlob, E. Grädel, and H. Veith. Datalog LITE: a deductive query language with linear time model checking. *ACM Trans. Comput. Logic*, 3(1):42–79, 2002.

[Kolaitis and Vardi, 1998] P. Kolaitis and M. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proc. of the 17th ACM Symp. on Principles of Database Systems (PODS)*, pages 205–213, 1998.

[Kolokolova *et al.*, 2006] A. Kolokolova, Y. Liu, D. Mitchell, and E. Ternovska. Complexity of expanding a finite structure and related tasks, 2006. The 8th Int. Workshop on Logic and Comput. Complexity (LCC).

[Leone *et al.*, 2006] N. Leone, G. Pfeifer, W.Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7(3):499–562, 2006.

[Lierler and Maratea, 2004] Y. Lierler and M. Maratea. Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In *LPNMR*, pages 346–350, 2004.

[Liu and Levesque, 2003] Y. Liu and H. Levesque. A tractability result for reasoning with incomplete first-order knowledge bases. In *IJCAI*, pages 83–88, 2003.

[Mariën *et al.*, 2005] M. Mariën, R. Mitra, M. Denecker, and M. Bruynooghe. Satisfiability checking for PC(ID). In *LPAR*, pages 565–579, 2005.

[Mitchell and Ternovska, 2005] D. Mitchell and E. Ternovska. A framework for representing and solving NP search problems. In *AAAI*, pages 430–435, 2005.

[Pelov and Ternovska, 2005] N. Pelov and E. Ternovska. Reducing inductive definitions to propositional satisfiability. In *ICLP*, pages 221–234, 2005.

[Ramachandran and Amir, 2005] D. Ramachandran and E. Amir. Compact propositional encodings of first-order theories. In *AAAI*, pages 340–345, 2005.

[Syrjänen and Niemelä, 2001] T. Syrjänen and I. Niemelä. The smodels system. In *LPNMR*, pages 434–438, 2001.

[van Gelder *et al.*, 1993] A. van Gelder, K. A. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1993.