Declarative Programming of Search Problems with
Built-in Arithmetic

Eugenia Ternovska, David G. Mitchell, and Brendan Guild

Simon Fraser University
Burnaby, BC, V5A 1S6
{ter, mitchell, bguild}@cs.sfu.ca

Abstract. We address the problem of providing a logical formalization of arithmetic in declarative modelling languages for NP search problems. The challenge is to simultaneously allow quantification over an infinite domain such as the natural numbers, provide naturally modelling facilities, and limit expressive power of the language. To address the problem, we introduce an extension of the model expansion (MX) based framework to finite structures embedded in an infinite secondary structure, together with “double-guarded” logics for representing MX specifications for these structures. The logics also contain multi-set functions (aggregate operations). Our main result is that these logics, capture the complexity class NP on “small-cost” arithmetical structures. We also give a grounding algorithm for specifications in the logic.

1 Introduction

Several lines of work in “constraint modelling languages” or “declarative programming for search problems” aim to produce high-level declarative languages for representing combinatorial search problems, together with solvers for applying these languages in practice. Underlying much of this work is a common logical task, which we call model expansion. In our group, following up on our proposal of [MT05], we are pursuing a program of developing both theory and practical systems based explicitly on the formalization of search as model expansion. Defined for an arbitrary logic $\mathcal{L}$, the task is:

Model Expansion for logic $\mathcal{L}$ (abbreviated $\mathcal{L}$ MX)

Given: 1. An $\mathcal{L}$-formula $\phi$

2. A structure $\mathcal{A}$ for a part $\sigma$ of $\text{vocab}(\phi)$

Find: an expansion $\mathcal{B}$ of $\mathcal{A}$ that satisfies $\phi$.

We call the vocabulary of $\mathcal{A}$ the instance or input vocabulary, and $\varepsilon := \text{vocab}(\phi) \setminus \sigma$ the expansion vocabulary. In the “combined setting” an instance consists of a structure together with a formula; in the “parameterized setting”, which is our focus here, we fix a formula for each problem, which plays the role of a problem specification. Then an instance is a finite structure, and each expansion of this structure that satisfies the specification formula is a solution.
Example 1. We are given (1) the following formula $\phi$ of first order logic:

$$\forall x \left[ \left( R(x) \lor B(x) \lor G(x) \right) \land \neg \left( R(x) \land B(x) \right) \land \neg \left( R(x) \land G(x) \right) \land \neg \left( B(x) \land G(x) \right) \right] \land \forall x \forall y \left[ E(x, y) \supset \left( \neg \left( R(x) \land R(y) \right) \land \neg \left( B(x) \land B(y) \right) \land \neg \left( G(x) \land G(y) \right) \right) \right]$$

and (2) a graph $A = G = (V, E)$, i.e., a structure for vocabulary $\sigma = \{E\}$. The task is to find an expansion $B$ of $A$ that satisfies $\phi$ (if any). To illustrate:

$$\models A \left( (V; E^A, R^B, B^B, G^B) \right) \models \phi.$$ 

An interpretation for the expansion vocabulary $\varepsilon := \{R, B, G\}$ given by the structure $B$ is a candidate solution for the search problem specified. The interpretations of $\varepsilon$, for structures $B$ that satisfy $\phi$, correspond exactly to the proper 3-colourings of $G$.

One benefit of formalization in logic is that the area of descriptive complexity theory [Imm99] — the study of the relationship between computational complexity and expressiveness of logics — provides tools for studying and controlling the expressive power of modelling languages. For example, Fagin’s theorem [Fag74], which states that the classes of finite structures definable in $\exists\Sigma_1$ are exactly those in NP, can be rephrased as the statement that parameterized FO MX captures NP. Thus, FO MX provides a natural basis for languages for specifying NP search problems. Capturing results such as these provide valuable practical, as well as scientific, information. The fact that FO MX can express every problem in NP provides assurance of sufficient expressiveness to a user facing an NP search problem. The fact that it can express no problem beyond NP provides assurance that a solver can be built using grounding technology (i.e., automated reduction to SAT), in which grounding is always polytime.

At least two solvers have been constructed explicitly based on model expansion for (extensions of) FO: A prototype solver, MXG, built in our group [MTHM06, Moh07], and the IDP system produced in Marc Denecker’s group [WM08]. It is also not hard to see that model expansion underlies many other languages for modelling combinatorial problems, even those not explicitly based on logic. Under this view, a specification in the language is an axiomatization of a model expansion task in some logic. (See for example [MT08], where this view is applied in analyzing the expressiveness of the specification language ESSENCE.) A challenge to providing a purely logical account of such languages is that the logics in question may be, syntactically speaking, quite rich and perhaps unconventional. In some cases they have rich type systems that may require an enriched notion of finite structures to provide a natural formalization.

Formalizing Arithmetic An important feature of most practical languages is having constructs for expressing arithmetic properties. FO MX, and thus the language of MXG, can express every problem in NP, but when numerical properties are involved numbers...
must be encoded with collections of abstract domain elements, and arithmetic operations by defined relations on these encodings. Having “built-in” arithmetic is much more natural.

The work presented here represents steps toward a logical foundation for languages with built-in arithmetic, and methods for controlling their expressive power. Ultimately, we would like to produce fully formalized languages in which modellers may use built-in arithmetic in the most natural way possible, with minimal restrictions on syntax. Further, we would like to produce such languages with controlled expressive power, and in particular languages that can express exactly the NP search and optimization problems.

The difficulty is that arithmetic involves infinite domains, and to restrict the power of such languages requires limiting the range of quantified variables and expansion predicates. In doing so, there is a danger of altering the standard semantics of the arithmetic operators, and also of making analysis of expressiveness more difficult. We believe that beginning with a logical formalization clarifies some of the issues of semantics that arise when designing modelling languages with finite arithmetic (especially in the presence of a type system), although demonstrating this is not a purpose of the present paper. As pointed out in [MT08], adding built-in numbers can easily produce inadvertent increases in expressive power. Moreover, even if we restrict such a language to NP in a natural way, it becomes unclear if it can express all of NP.

Almost all declarative languages for modelling combinatorial problems have some built-in arithmetic, and employ some means of restricting ranges of variables. To our knowledge, there are none for which the arithmetic constructs have been formalized and expressiveness determined (by which we mean the class of specifiable problems characterized, not just the determination of complexity upper and lower bounds). Most of these languages also include constructs to express optimization problems, where the characterization of expressive power is significantly harder.

**Contributions** In this paper, we develop an extension of FO model expansion with arithmetic, in a way that captures NP on a restricted class of structures. In particular, we make the following contributions:

1. Develop a notion of “embedded model expansion”, with an infinite background structure, and in particular for arithmetic structures, which include the usual functions on the natural numbers (or integers), as well as aggregates such as those used in SQL and many constraint languages (Section 2);
2. Introduce a logic for producing embedded MX specifications. The logic is a variant of the \(k\)-guarded fragment of FO (or FO(ID)), in which upper and lower guards control access to the infinite background domain (Section 2);
3. Show that, on classes of structures where numbers are not allowed to be too large, this logic captures NP (Section 3);
4. Generalize our result by using an extension of classical logic with inductive definitions to allow poly-size “user-defined” guard relations (Section 4);
5. Present a grounding algorithm for embedded MX with arithmetic (Section 5).
We believe our notions of embedded structures and double-guar ded logics provide a natural way to meet the goals of producing languages with formalized built-in arithmetic and controlled expressiveness. We also believe these formalize much that is done in practice (but without formalization) in some existing modelling languages.

2 Embedded MX with Arithmetic

Embedded finite model theory (see [Lib04]) is the study of finite structures whose domain is drawn from some infinite structure, motivated by the need to study databases that contain numbers and constraint databases. Rather than think of a database as a finite structure, we take it to be a set of finite relations over an infinite domain.

**Definition 1.** A structure $A$ is embedded in an infinite background structure $M = (U, \bar{M})$ if it is a structure $A = (U; R)$ with a finite set $R$ of constants, finite relations and functions taking non-zero values on a finite set of elements, $\bar{M} \cap R = \emptyset$. The set of elements of $U$ that occur in some relation of $A$ is the active domain of $A$, denoted $\text{adom}_A$.

The concept of an embedded structure is used in logics for expressing database queries. Here, we use it in a logic for MX specifications (generalized queries). The vocabularies consist of (1) the vocabulary of $A$ which is our instance vocabulary $\sigma$; (2) the vocabulary $\nu$ of an infinite secondary structure $M = (U, \bar{M})$, such as the arithmetical structure defined below; and (3) an expansion vocabulary $\varepsilon$. A formula $\phi$ over $\sigma \cup \nu \cup \varepsilon$ represents an MX specification. The task of model expansion remains the same: an embedded $\sigma$-structure is expanded to satisfy $\phi$.

To control the expressive power of logics for MX with infinite secondary structures, we need to limit both range of quantified variables and the range of expansion predicates. For this purpose, we use an adaptation of the guarded fragment $GF_k$ of FO [GLS01]. In formulas of $GF_k$, a conjunction of up to $k$ atoms acts as a guard for each quantified variable.

**Definition 2.** The $k$-guarded fragment $GF_k$ of FO is the smallest set of formulas that:
1. contains all atomic formulas;
2. is closed under Boolean operations;
3. contains $\exists \bar{x}(G_1 \land \ldots \land G_m \land \phi)$, provided the $G_i$ are atomic formulas, $m \leq k$, $\phi \in GF_k$, and the free variables of $\phi$ appear in the $G_i$.

Here, $G_1 \land \ldots \land G_m$ is called the guard of $\bar{x}$. Since $GF_k$ is closed under negation, universal quantification can be treated as an abbreviation in the usual way, and universal quantifiers are guarded as in $\forall \bar{x}(G_1 \land \ldots \land G_m \supset \phi)$.

For now, we assume that all guards are instance predicates, thus restricting the range of quantified variables. We call these guards lower guards. Since $GF_k$ is closed under negation, universal quantification can be treated as an abbreviation in the usual way, and universal quantifiers are guarded as in $\forall \bar{x}(G_1 \land \ldots \land G_m \supset \phi)$.

In addition to lower guards on quantified variables, we require that for each expansion predicate $E \in \varepsilon$, we include the following axiom: $\forall \bar{x}(E(\bar{x}) \supset G(\bar{x}))$, where the extent of predicate $G$, which we call an upper guard. We may also guard $E$ with a conjunction
of upper guards jointly guarding the variables: $\forall \bar{x} (E(\bar{x}) \supset G_1(\bar{x}_1) \land \cdots \land G_k(\bar{x}_k))$, where the union of all variables in the $G$s is precisely $\bar{x}$. For each function in $\varepsilon$, we upper-guard the graph of that function. We require for now that all upper and lower guards are from the instance vocabulary. Later, we relax this restriction by adding a mechanism for “user-defined” guard relations that may contain elements not in the active domain. We assume that predicate $\text{adom}$ is always present in instance vocabulary $\sigma$, thus $\text{adom}(\bar{x})$ can be used as a guard (upper or lower).

**Definition 3.** The double-guarded fragment $\text{GGF}_k(\varepsilon)$ of FO, for a given vocabulary $\varepsilon$, is the set of formulas of the form $\phi \land \psi$, with $\varepsilon \subseteq \text{vocab}(\phi \land \psi)$, where $\phi$ is a formula of $\text{GF}_k$, and $\psi$ is a conjunction of upper guard axioms for each symbol of $\varepsilon$ that occurs in $\psi$. Each upper guard axiom involves at most $k$ guards, none of which are in $\varepsilon$.

Note that $\text{GGF}_k(\varepsilon)$ is just guarded $\exists \text{SO}$ where all FO and SO quantifiers are guarded.

Note also that in $\exists \text{SO}$, if we drop SO quantifiers, the remaining formula is FO. In our work, we mostly talk about MX for formulas which are not FO.

For writing MX specifications for embedded structures, we will extend the logic $\text{GGF}_k(\varepsilon)$ with vocabulary for a fixed background structure $\cal M$. We will talk about “$\text{GGF}_k(\varepsilon)$ MX specifications with background structure $\cal M$”. We need to define the logic for such specifications with respect to the particular background structure involved, as the terms allowed in formulas depend upon the structure. The background structure of interest here is the arithmetical structure. It will allow us to write MX specifications with built-in arithmetic and aggregate operations, as used in SQL and some constraint languages. To a large extent, we follow [GG98], although we use the embedded setting and the guarded fragment defined above. See Section 6 and Appendix A for details. For any set $R$, we use $\text{fm}(R)$ to denote the class of all finite multisets over $R$. Any function $f : U \rightarrow U$ defines a multiset $\text{mult}(f) = \{ f(a) : a \in U \}$ over $U$, the domain of $A$. As in the metafinite setting of [GG98], we consider our secondary structures with a collection of multiset operations $\Gamma : \text{fm}(U) \rightarrow U$, e.g. max, min, sum, product. We elaborate on their syntax and semantics.

**Definition 4.** An Arithmetical structure is a structure $\cal N$ containing at least $(\mathbb{N}; 0, 1, \chi, <$, $+ , \cdot, \text{min}, \text{max}, \sum, \prod)$, with domain $\mathbb{N}$, the natural numbers, and where $\text{min, max, } \sum, \text{ and } \prod$ are multi-set operations and $\chi(\bar{x})$ is the characteristic function. Other functions, predicates, and multi-set operations may be included, provided every function and relation of $\cal N$ is polytime computable.

We now define the terms allowed in $\text{GGF}_k(\varepsilon)$ MX specifications with secondary structure $\cal N$. As is common in logic textbooks, $\phi(\bar{x})$ denotes that $\bar{x}$ contains the free variables of $\phi$, and $\phi[\bar{a}]$ denotes $\phi$ together with the valuation of free variables $\bar{x}$ of $\phi$ as the domain elements $\bar{a}$.

**Definition 5 (well-formed terms).** Let $\tau$ be the vocabulary $\sigma \cup \nu \cup \varepsilon$ and $V$ a countable set of variables. The set of well-formed terms is the closure of the sets of variables $V$ and constants of $\tau$ under the following operations:

1. If $f$ is a $\tau$-function of arity $n$, other than a multiset operation or characteristic function, and $\bar{t}$ is a tuple of terms of length $n$ then $f(\bar{t})$ is a term.
2. If \( \Gamma \) is a multiset operation of \( \nu \), \( f(\bar{x}, \bar{y}) \) is a term, \( \phi(\bar{x}, \bar{y}) \) is a \( \tau \)-formula such that 
\[ \exists \bar{x} \phi(\bar{x}) \] is a \( k \)-guarded formula, then 
\[ \Gamma^\bar{x}(f(\bar{x}, \bar{y}) : \phi(\bar{x}, \bar{y})) \],

is a term with free variables \( \bar{y} \).

3. If \( \phi \) is a \( \tau \)-formula, then \( \chi[\phi] \), the characteristic function, is a term with the same 
free variables \( \bar{x} \) as \( \phi \). Again, we require \( \exists \bar{x} \phi(\bar{x}) \) to be a \( k \)-guarded formula.

In case 2, we do not require the free variables \( \bar{y} \) in \( \phi \) inside the term to be guarded 
within \( \phi \), but they will be guarded in the larger formula where the term appears. Note 
that the multiset operation acts much like a quantifier, binding the free variables \( \bar{x} \).

**Semantics of terms** The interpretation of the characteristic function \( \chi[\phi](\bar{x}) \) is 
\[ \chi[\phi]^D(\bar{a}) := \begin{cases} 1 & \text{if } D \models \phi[\bar{a}] \\ 0 & \text{otherwise.} \end{cases} \]

for any structure \( D \) and a tuple of domain elements \( \bar{a} \) of the same length as \( \bar{x} \). For the 
multiset operations, let \( G(\bar{y}) \) be the term \( \Gamma^\bar{x}(f(\bar{x}, \bar{y}) : \phi(\bar{x}, \bar{y})) \). The interpretation of 
\( G(\bar{y}) \) on a \( \tau \)-structure \( D \) with valuation \( \bar{b} \) for \( \bar{y} \) is 
\[ G^D(\bar{b}) := \Gamma(\Phi f^D(\bar{a}, \bar{b}) : \text{for all } \bar{a} \text{ such that } D \models \phi[\bar{a}, \bar{b}] ). \]

For readability, we omit true \( \phi \) and write \( \Gamma^\bar{x}(f(\bar{x}, \bar{y})). \) Sometimes we omit free variables 
and write \( \Gamma^\bar{x}(f : \phi). \) Some important multiset operations (e.g. \( \Sigma \) and \( \max \) on \( N \)) are 
invariants under adding arbitrary occurrences of 0 to the multiset: \( \Gamma(S) = \Gamma(S \cup \{0, \ldots, 0\}) \) for all \( S \in \text{fin}(R) \). Then we use \( \Gamma^\bar{x}(f \times \chi[\phi]) \) instead of \( \Gamma^\bar{x}(f : \phi). \)

**Definition 6.** An embedded GGF\( k \) MX specification with secondary structure \( N \) is 
a set of GGF\( k \) sentences over \( \sigma \cup \varepsilon \cup \nu \), with terms as in Definition 5, where the 
user selects an expansion \( \varepsilon \) and an instance \( \sigma \) vocabularies. The secondary \( \nu \)-structure 
is the arithmetical structure of Definition 4.

**Example 2.** Here is an embedded MX specification of the KNAPSACK problem (search 
version). Instance vocabulary \( \sigma := \{O, w, v, b, k\} \), where: \( O \) is the set of objects, \( w \) is 
the weight function, \( v \) is the value function, \( b \) is the weight bound, \( k \) is the value target.

Expansion vocabulary \( \varepsilon := \{O'\} \), where \( O' \) is the set of selected objects. Background 
structure: the arithmetical structure \( N \). Upper guard axiom: \( \forall x(O'(x) \supset O(x)) \).  
Axioms:
\[ \Sigma_x(w(x) : O(x) \land O'(x)) \leq b \]
\[ k \leq \Sigma_x(v(x) : O(x) \land O'(x)) , \]

where \( \leq \) is the standard abbreviation. The use of the guard \( O \) corresponds to introducing 
a type “all objects” in the language of the system, where the upper guards axiom is replaced 
by the corresponding type declaration. \( \Diamond \)

**Example 3.** MACHINE SCHEDULING PROBLEM [Hoo00] The task is to assign jobs 
to machines so that constraints on release and due date are satisfied. Machines are
single-task and a cost function is minimized. The input structure lists jobs, machines and possible start times. Release date, due date for each job are also given, as well as cost of running each job on each machine, and the duration each job takes on each machine. The instance vocabulary, $\sigma$, consists of: $\text{Job}(j)$ – the set of jobs to be scheduled; $\text{Machine}(m)$ – the set of machines to perform jobs; $\text{Time}(t)$ – all possible starting times; $\text{ReleaseDate}(j)$ – each job has a release date; $\text{DueDate}(j)$ – each job has a due date; $\text{Cost}(j, m)$ – cost of performing job $j$ on machine $m$; $\text{Duration}(j, m)$ – duration of executing $j$ on $m$.

Background structure: the arithmetical structure $\mathcal{N}$. The active domain consists of all time points, costs, due and release dates, and durations, as well as jobs and machines represented as numbers (having several separate sorts instead is also possible). Expansion vocabulary consists of two functions: $\text{Assignment}(j)$ maps jobs to machines; $\text{StartTime}(j)$ maps jobs to start times. In the search version of the problem we have axiom $\Sigma_j(\text{Cost}(j, \text{Assignment}(j)) : \text{Job}(j)) \leq k$, where $k$ is given by the instance. Optimization version (outside the scope of this paper) includes objective function: minimizing: $\Sigma_j(\text{Cost}(j, \text{Assignment}(j)) : \text{Job}(j))$. Upper guard axioms:

$\forall j \exists m \ (\text{Assignment}(j) = m \supset \text{Machine}(m) \land \text{Job}(j))$

$\forall j \forall t \ (\text{StartTime}(j) = t \supset \text{Time}(t) \land \text{Job}(j))$

Axioms:

$\forall j (\text{Job}(j) \supset \text{StartTime}(j) \geq \text{ReleaseDate}(j))$

$\forall j (\text{Job}(j) \supset \text{StartTime}(j) + \text{Duration}(j) \leq \text{DueDate}(j))$

At most one job at each machine at a given time:

$\forall t (\text{Time}(t) \supset (\forall m \ (\text{Machine}(m) \supset$

$\max_j(\text{count}_j(\psi(j, m, t))) = 1)),$

where $\psi$ defines the set of jobs being executed on machine $m$ at time $t$:

$\psi(j, m, t) :=$

$\text{Job}(j) \land \text{Assignment}(j) = m \land \text{Time}(\text{StartTime}(j))$

$\land \text{StartTime}(j) \leq t < \text{StartTime}(j) +$

$\text{Duration}(j, \text{Assignment}(j)),$

and $\text{count}_j(\psi(j, m, t))$ abbreviates $\Sigma_j(\chi[\psi(j, m, t)])$.

It is easy to see that all axioms are in $\text{GGF}_k(\varepsilon)$. ◊

**SQL Examples**

The following SQL query returns the maximum value in column $k$ among the tuples in table $T$ that satisfy the Boolean condition $C$: SELECT MAX($k$) FROM $T$ WHERE $C$. It is represented by the multiset operation:

$max_k \{ x_k : \exists x_1 \ldots \exists x_{k-1} \exists x_{k+1} \ldots \exists x_n T(\bar{x}) \land C(\bar{x}) \}$, where $\bar{x} := x_1 \ldots x_n$.

The following query returns the number of rows in $T$ that satisfy the Boolean condition $C$: SELECT COUNT(*) FROM $T$ WHERE $C$. Its representation is: $\sum (\chi[T(\bar{x}) \land C(\bar{x})])$. We have expressed all other SQL aggregates, but this is not included here.

**Remark 1.** Upper and lower guards are a good logical formalization of the type systems in some existing constraint modelling languages and in our system. The use of lower guards corresponds to declaring types of variables, and upper guard axioms correspond to declaring types of expansion predicates and functions. The authors of [MT08] used this approach to formalize part of the type system of ESSENCE [FGJ+07].
Remark 2. The presentation above is oversimplified – elements of the active domain are drawn from the arithmetical background structure, i.e., there is just one sort, and all elements are ordered. Having the elements ordered is not entirely satisfactory because we don’t normally think of arbitrary sets of elements as having an order, and most of model theory does not make this assumption. On the other hand, once we write input to a computer program down, an ordering materializes. Having one sort is by no means a requirement. None of the properties described in this paper changes if the background structure (and thus the vocabularies) is multi-sorted, with additional finite domains.

3 Capturing NP in the Presence of Arithmetic

Here, we present our main result. We can apply it to any language which is a syntactic variant of our language (e.g. a suitable fragment of ESSENCE). Here, we consider the decision problem associated with embedded MX, and look at the parameterized setting (data complexity), where the formula is fixed and instances are finite structures.

For an embedded arithmetical structure $\mathcal{A}$, define the cost of $\mathcal{A}$ to be $\lceil \log(l) \rceil$, where $l$ is the largest number in $\text{dom}(\mathcal{A})$, i.e., cost is the size of the binary encoding of that number. A class $\mathcal{K}$ of embedded arithmetical structures has small cost if there is some $k \in \mathbb{N}$ such that $\text{cost}(\mathcal{A}) < |\mathcal{A}|^k$, for every $\mathcal{A} \in \mathcal{K}$. This is a generalization of the notion of a metafinite structure with small weights of [GG98]. The restriction to small cost ensures that we have encodings of the structures which are of size polynomial in their domain size. Small cost structures have no numbers larger than $2^{\text{poly}(|\mathcal{A}|)}$.

A class $\mathcal{K}$ of $\tau$-structures is an embedded spectrum if there is a first-order sentence $\phi$ of a vocabulary $\tau' := \tau \cup \varepsilon$ of logic GGF$_k$($\varepsilon$) such that $D \in \mathcal{K}$ iff there exists an expansion $D'$ of $D$ with $D' \models \phi$.

Theorem 1. Let $\mathcal{K}$ be an isomorphism-closed class of small-cost arithmetical embedded structures over vocabulary $\sigma$. Then the following are equivalent: (1) $\mathcal{K} \in \text{NP}$, (2) $\mathcal{K}$ is an embedded spectrum.

The small cost condition comes from our proof technique, where we mimic numbers by tuples of domain elements and then apply Fagin’s theorem. With a reasonable amount of confidence, we can now report that the small cost condition can be replaced by one that is significantly weaker (under a small modification of the secondary structure), and that cannot be relaxed any further.

4 User-Defined Guard Relations

So far, the numbers that may occur in a solution for an instance are restricted to those that occur in instance because every expansion predicate has an upper guard composed of only instance predicates. There are many problems where this is too restrictive, an obvious example being integer factorization. (We can define many search problems with “new” numbers in solutions, but these must be encoded with elements of the instance structure, rather than appearing directly.) To relax this limitation, we introduce “user-defined guard relations”. We now consider specifications consisting of two formulas, $D$
and $\phi$. Formula $D$ is over vocabulary $\sigma \cup \delta$, where $\delta$ is a set of predicate symbols not in $\sigma$, and for each instance structure $A$, $D$ defines an expansion of $A'$ that includes the new user-defined guard relations. The active domain of $A'$ will be the union of $\text{dom}_A$ and any elements of the defined guard relations. Formula $\phi$ of $\text{GGF}_k(\varepsilon)$, over vocabulary $\sigma \cup \delta \cup \nu \cup \varepsilon$, defines an embedded model expansion task for each $A'$. That is, $\phi$ is such a specification with a larger instance vocabulary $\sigma \cup \delta$.

Informally, $D$ should be a device for defining sets of numbers beyond those in the instance, in the aid of letting $\phi$ be a more natural axiomatization of the problem than it could be without these extra guard relations. For $D$ to fulfill this role, the relations it defines should be unique (for each instance), and easy to compute. Formally, we require $D$ to fulfill the following property.

**Property 1 (Good $D$).**

1. $D$ defines a total function $f_D$ from embedded $\sigma$-structures to embedded $\sigma \cup \delta$-structures;
2. $f_D$ is polytime computable.

We may choose to use a syntax for $D$ that is distinct from that of $\phi$. Whatever the syntactic form we choose for $D$, we will require that it is in some logic with the property that, for every allowed $D$, it is decidable (preferably in polytime) if $D$ is good (satisfies property 1). Thus, we can effectively decide if a specification has appropriate user-defined guard relations.

**Capturing NP with User-Defined Guards** Let us denote by $\text{DGGF}_k(\varepsilon, \delta)$ a logic obtained by an extension of $\text{GGF}_k(\varepsilon)$ to allow specification of user-defined guards as just described. A $\text{DGGF}_k(\varepsilon, \delta)$ MX specification $\phi \land D$ with background structure $N'$ is defined as in Definition 6, except that it also includes the part $D$.

**Lemma 1.** Let $\phi \land D$ be a $\text{DGGF}_k(\varepsilon, \delta)$ MX specification with $N'$, where $D$ is good. Let $K$ be a small cost class of embedded $\sigma$-structures, and $K'$ be the class of $\sigma \cup \gamma$-structures obtained from $K$ by expanding each structure of $K$ with the relations defined by $D$. That is, $K' = \{f_D(A) : A \in K\}$. Then the following are equivalent:

1. $K$ is in NP
2. $K'$ is in NP,
3. $K$ is an embedded spectrum,
4. $K'$ is an embedded spectrum,

**Proof.** (Sketch) The equivalence of 1) and 3) is by Theorem 1, as is the equivalence of 2) and 4). To complete the proof, it is enough to show that 1) and 2) are equivalent. To show that a set is in NP, we need only show that there are polytime verifiable certificates for the set. If $K$ is an embedded spectrum, then expansions of $A \in K$ satisfying $\phi$ are such certificates. Now, suppose that $K \in \text{NP}$, and $B$ is a certificate that $A \in K$. We claim that $B$ is also a certificate that $A' = f_D(A) \in K'$. To verify $B$ as a certificate that $A' \in K'$, we compute $A$ from $A'$ and then use $B$ to verify that $A \in K$. For the other direction, assume $K' \in \text{NP}$, and $B$ is a certificate that $A' \in K$. Again, we claim that $B$ is a certificate also for $A \in K$: compute $A'$ from $A$, and then use $B$ to verify that $A' \in K'$. 
**A Good $D$ for $N$** Lemma 1 gives general conditions under which we may capture NP with user defined guard relations. In general, similar conditions will hold for many choices of secondary structure, and for each there will be many choices for the form of $D$ that will satisfy those conditions. We now present one choice for the form of $D$ that satisfies the conditions in the case of arithmetical structures. We wish to choose this form so that easily checkable syntactic conditions are sufficient to ensure a given $D$ is good.

Our guard relations will be defined by induction, using the syntax and semantics of FO(ID), the extension of FO with inductive definitions (see [DT08]). Inductive definitions are specified with a rule-based syntax, with arbitrary FO formulas in the bodies, under the 2-valued well-founded semantics (the third truth value makes the entire axiomatization a contradiction). We give an example, and refer to [DT08] for syntax and semantics.

**Example 4.** The following inductive definition defines the odd numbers on $N$.

$$\{\forall x \ (\text{Odd}(x) \leftarrow x = 0 \lor \exists y \ (\text{Odd}(y) \land x = y + 2))\}$$

The defined (intensional) symbols are those in the head, and open (extensional) are the rest. We assume that definitions contain no free variables, and every variable that occurs in the head also appears in the body. Each definition may simultaneously define several relations.

**Definition 7.** Say a formula $\psi$ of FO(ID) is in form DEF if it satisfies the following.

1. $\psi$ is a conjunction of definitions of FO(ID) having a well-founded partial pre-order on definitions such that all open (extensional) symbols are either from $\sigma$ (the instance vocabulary) or defined by a definition which is strictly smaller in the pre-order.
2. each definition is either:
   (a) Of the form $\{\forall x \ (G(x) \leftarrow x \leq \text{size}(\phi))\}$, where $\text{size}(\phi)$ is an abbreviation for $\sum_x(\chi(\text{adom}(x) \land x = x))$, or
   (b) either positive or stratifiable, and every rule is of the form
      $$\{ \forall \bar{x} \bar{y} \ (G(\bar{x}, \bar{y}) \leftarrow y_1 = t_1(\bar{x}) \land \ldots \land y_k = t_k(\bar{x}) \land \phi(\bar{x})) \},$$

   where $k \geq 0$ and $\exists \bar{x} \phi(\bar{x})$ is guarded, with all guards are either from the input structure, or are defined earlier in the pre-order.

**Lemma 2.** It is polytime decidable if a formula is in the form DEF.

**Lemma 3.** If $D$ is in form DEF, it is good.

**Example 5.** Suppose, for illustration, we have a search problem on weighted directed graphs, and are interested only in nodes within some particular distance of a given node. The same is achievable by combining all these definitions into one large stratifiable definition.
s. In our MX axiomatization, we may need a guard $P(x)$ which represents this set of nodes. We have a sequence of definitions $(\Delta_1, \Delta_2, \Delta_3)$:

$$\Delta_1 := \{ \forall x (\text{within bound}(x) \leftarrow x \leq \text{size}(\text{adom})) \} ,$$

where adom is the active domain of the instance. The guard defined in $\Delta_1$ restricts allowable distances.

$$\Delta_2 := \left\{ \begin{array}{l}
\forall x (\text{distance}(x, 0) \leftarrow x = s), \\
\forall x \forall y \forall d \forall d_1 \forall d_2
(\text{distance}(y, d) \leftarrow \text{within bound}(d) \land \\
\text{distance}(x, d_1) \land E(x, y, d_2) \land d = d_1 + d_2)
\end{array} \right\} ,$$

which defines the distances to all nodes reachable from s provided they are within the distance bound.

$$\Delta_3 := \{ \forall x (P(x) \leftarrow \exists d \text{distance}(x, d)) \} ,$$

which defines the set of all nodes within the pre-specified distance from the node s.

**Adding Inductive Definitions to DGGF$_k$** In [MT05, MTHM06], we proposed specifically to use FO(ID), rather than just FO, as the basic logic for MX specifications of NP search problems. Here, again, we find FO(ID) convenient. A generalization of GF$_k$ to the case with inductive definitions is given in [PLTG07]. The other logics used in the present paper also generalize to the case where inductive definitions are added, and the main results of the paper will also hold in this case.

**5 Grounding for Embedded MX**

Let $\phi$ be an embedded MX axiomatization of a search problem in logic $DGGF_k(\varepsilon)$, and $\mathcal{A}$ an instance structure for $\phi$. We describe a method for constructing satisfying expansions of $\mathcal{A}$ (i.e., solutions), by grounding.

Define $\phi_{ax}$ to be the set of regular axioms of $\phi$ (i.e., those formulas in DGF$_k$ with guards given by $\mathcal{A}$), and $\phi_{ug} = \phi \setminus \phi_{ax}$ to be the set of upper guard axioms of $\phi$. The method is in two stages. In the first, we construct a ground formula $\psi = \text{Gnd}(\phi, \mathcal{A})$, with the property that expansions of $\mathcal{A}$ satisfying $\psi$ are exactly those satisfying $\phi$. Since the upper guard axioms have not been taken into account, it is possible that some of these expansions are infinite, or that all of them violate the upper guards. In the second stage, we modify $\psi$ in accordance with the upper guards so that we can generate expansions of $\mathcal{A}$ satisfying $\phi$ by a call to a propositional solver, such as a SAT solver.

$^2$ Technically, given the power of $\exists$SO we have with FO model expansion, we could do without inductive definitions, however expressing many useful concepts including reachability or transitive closure in $\exists$SO is extremely complex and infeasible in practice.
**Grounding** $\phi_{ax}$ Grounding may be seen as a generalization of model checking, or Boolean query evaluation. In [PLTG07], an algorithm is presented for grounding of MX axiomatizations in the guarded fragment of FO based on a generalization of the relational algebra. Here, we extend this method to guarded formulas in the embedded case.

Let $A = (U, \vec{R})$ be an embedded $\sigma$-structure. The defined guards of $\phi$ may introduce new elements not in $\text{adom}(A)$, and which may be part of solutions for $A$. Let $\text{new}(A)$ denote those elements. For the remainder of this section, let $A = \text{adom}(A) \cup \text{new}(A)$, and let $\tilde{A}$ denote a set of new constant symbols, one for each element of $A$.

**Definition 8 (Grounding, Reduced Grounding).** Formula $\psi$ is a grounding of $\phi$ over embedded structure $A = (U; \vec{R})$ if

1. $\psi$ is a ground formula over the vocabulary $\sigma \cup \varepsilon \cup \tilde{A} \cup M$;
2. for every structure $B$ which expands $A$ to $\text{vocab}(\phi)$, it holds that $B \models \phi$ iff $(B, \tilde{A}^B) \models \psi$, where $\tilde{A}^B$ denotes the interpretation of the new constant symbols.

Formula $\psi$ is a reduced grounding if it is over the vocabulary $\varepsilon \cup \tilde{A}$, i.e., contains no symbols of the instance vocabulary or vocabulary of the background structure.

The algorithm uses the following data structure.

**Definition 9 (extended $X$-relation [PLTG07]).** Let $X$ be a tuple of variables. An extended $X$-relation $R$ over $A$ is a set of pairs $(\gamma, \psi)$ such that

1. $\psi$ is a ground formula over $\varepsilon \cup \tilde{A}$ and $\gamma : X \to A$;
2. for every $\gamma$, there is at most one $\psi$ such that $(\gamma, \psi) \in R$.

One can think of an $X$-relation $R$ as a representation of a (unique) mapping $\delta_R$ from instantiations of variables in $X$ to ground formulas:

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

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

The role of an extended $X$-relation is to represent the reduced groundings for all instantiations of the variables for a formula with free variables. We call such a representation an answer to the formula with respect to the structure.

**Definition 10 (Answer to $\phi$ wrt $A$).** Let $\phi$ be a formula with free variables $X$. We say extended $X$-relation $R$ is an answer to $\phi$ wrt $A$ if for any $\gamma : X \to A$, we have that $\delta_R(\gamma)$ is a reduced grounding of $\phi[\gamma]$ over $A$. Here, $\phi[\gamma]$ denotes the result of instantiating free variables in $\phi$ according to $\gamma$.

**The Grounding Algorithm** We compute an answer to a formula by computing answers to its subformulas and combining them according to the connectives. The operations to do this are natural generalizations of the operations of relational algebra to extended $X$-relations. The answer to a sentence is a reduced grounding for the sentence.

To adapt the grounding procedure from [PLTG07], we need three observations:
1. since quantification is guarded we need take into account only finite subsets of potentially infinite expansion relations;
2. any specification with an aggregate operator can be re-written as a specification without aggregate operators (albeit with additional expansion relations);
3. all vocabulary of the background structure can be evaluated out during grounding.

Thus, to ground \( \phi \) with respect to embedded structure \( \mathcal{A} \), we first we-write \( \phi \) to have no multi-function operators, and then execute the following algorithm.

**Procedure** \( \text{Gnd}(\mathcal{A}, \phi) \)

Suppose \( \phi \) is of the form \( \exists y (G_1 \land \cdots \land G_m \land \psi) \).

Let \( \mathcal{R} \) be \( G_1(\mathcal{A}) \land \cdots \land G_m(\mathcal{A}) \).

Let \( \psi' \) be \( \neg \psi \) with \( \neg \) pushed inward so it occurs only before an atom or an \( \exists \).

Then \( \text{Gnd}(\mathcal{A}, \phi) = \pi\text{Gnd}(\mathcal{A}, \mathcal{R}, \phi') \).

**Procedure** \( \text{Gnd}(\mathcal{A}, \mathcal{R}, \phi) \)

- If \( \phi \) is a positive literal of an instance predicate, then \( \text{Gnd}(\mathcal{A}, \mathcal{R}, \phi) = \mathcal{R} \rtimes \phi(\mathcal{A}) \);
- If \( \phi \) is \( \neg \phi' \), where \( \phi' \) is an atom of an instance predicate, then \( \text{Gnd}(\mathcal{A}, \mathcal{R}, \phi) = \mathcal{R} \rtimes^c \phi'(\mathcal{A}) \);
- If \( \phi \) is a literal of an expansion predicate, then \( \text{Gnd}(\mathcal{A}, \mathcal{R}, \phi) = \{(\gamma, \phi[\gamma]) \mid \gamma \in \mathcal{R}\} \);
- \( \text{Gnd}(\mathcal{A}, \mathcal{R}, (\phi \land \psi)) = \text{Gnd}(\mathcal{A}, \mathcal{R}, \phi) \cap \text{Gnd}(\mathcal{A}, \mathcal{R}, \psi) \);
- \( \text{Gnd}(\mathcal{A}, \mathcal{R}, (\phi \lor \psi)) = \text{Gnd}(\mathcal{A}, \mathcal{R}, \phi) \cup \text{Gnd}(\mathcal{A}, \mathcal{R}, \psi) \);
- \( \text{Gnd}(\mathcal{A}, \mathcal{R}, \exists y \phi) = \mathcal{R} \rtimes \text{Gnd}(\mathcal{A}, \exists y \phi) \);
- \( \text{Gnd}(\mathcal{A}, \mathcal{R}, \neg \exists y \phi) = \mathcal{R} \rtimes^c \text{Gnd}(\mathcal{A}, \exists y \phi) \).
- If \( \phi \) is \( B(\bar{t}) \) for a background structure predicate \( B \),
  then \( \text{Gnd}(\mathcal{A}, \mathcal{R}, \phi) = \{(\gamma, \psi) \mid (\gamma, \psi) \in \mathcal{R} \land \bar{t}[\gamma] \in B\} \);
- If \( \phi \) is \( \neg B(\bar{t}) \) for a background structure predicate \( B \),
  then \( \text{Gnd}(\mathcal{A}, \mathcal{R}, \phi) = \{(\gamma, \psi) \mid (\gamma, \psi) \in \mathcal{R} \land \bar{t}[\gamma] \notin B\} \);

The last two rules handle potentially infinite background structures. Relations on those structures (e.g. \( \preceq \)) do not have to be enumerated, they are tested as in \( \bar{t}[\gamma] \in B \). For simplicity of presentation, background structure functions are viewed as their graphs.

**Correctness and Complexity of Gnd()**

**Theorem 2.** If \( \mathcal{A} \) is an embedded structure and \( \phi \) a formula in \( \text{GF}_k \), where guards are given by \( \mathcal{A} \), \( \text{Gnd} \) returns an answer to \( \phi \) with respect to \( \mathcal{A} \). Hence, if \( \phi \) is a sentence, \( \text{Gnd} \) returns a reduced grounding of \( \phi \) over \( \mathcal{A} \).

The time complexity of the algorithm is \( O(l^2 n^k q) \), where \( l \) is the size of the formula, \( n \) is the size of the largest guard, and \( q \) is the complexity of checking a literal in \( \mathcal{M} \).

Provided guards are polytime computable, the algorithm gives a uniform polytime reduction to SAT. A generalization of this algorithm to handle inductive definitions can be obtained from the corresponding algorithm in the [PLTG07].
Computing Expansions from Gnd$(\phi_{ax},A)$ This subsection benefited from a discussion with, and points made by, Toni Mancini. Let $\psi = \text{Gnd}(\phi_{ax},A)$, a reduced grounding of $\phi_{ax}$. The desired expansions of $A$ are those models of $\psi$ that satisfy the upper guard axioms. To obtain these, we first modify $\psi$ as follows. For every atom of the form $P(\bar{a})$ that occurs in $\psi$, we check if $\bar{a} \in G^A$, the upper guard for $P$. (We may do this by explicitly computing the upper guard relation, although in some cases there may be more efficient ways.) If $\bar{a} \notin G^A$, then we conjoin $\neg P(\bar{a})$ with $\psi$. Call the resulting formula $\psi'$. Now consider $\psi'$ to be a propositional formula. If $\alpha$ is a satisfying assignment for $\psi'$, then denote by $\text{mod}(\alpha)$ the model of $\psi'$ obtained by (expanding $A$ with), for each expansion predicate $P$, the interpretation defined by

$$\{\bar{a} : \alpha(P(\bar{a})) = \text{true}\}.$$ 

**Theorem 3.** If $\alpha$ is a satisfying assignment for $\psi$, then $\text{mod}(\alpha)$ is a model of $\phi$. If $\alpha'$ is an extension of $\alpha$ that is consistent with the upper guards, $\text{mod}(\alpha')$ is a model of $\phi$.

The construction of $\psi'$ from $\phi$ and $A$ can be done polynomial time, because each stage is. Notice that in modifying $\psi$ to produce $\psi'$ we conjoin with it many atoms. These can all be efficiently eliminated, and the formula simplified, by applying unit propagation, also in polynomial time. We can also enumerate all solutions for $A$, if we want, as follows. Enumerate satisfying assignments for $\psi'$ (for example, using a SAT solver that can enumerate all satisfying assignments), and for each, enumerate the elements in the upper guards and produce a solution for each extension of each satisfying assignment that does not violate the upper guards. This can all be done in time polynomial in the number of solutions generated (plus the time for the SAT solver calls, of course).

6 Related work

Two existing lines of theoretical work aim to similarly extend finite model theory applications with infinite domains, with the goal of preserving applicability of finite model techniques. These are embedded model theory (see [Lib04]) from database theory, and metainfinite model theory [GG98,Gra07]. However, neither of these directly addresses the problems we face. (See Appendix A for detailed discussion.) Preliminary versions of this work were presented as workshop talk at LCC’07 and in a technical report [TM07]. The authors of [CM06] adopted the related viewpoint that $\exists SO$ in an appropriate mathematical abstraction of many practical constraint languages in which to carry out a general study of techniques for reasoning with and re-writing specifications. In [MC07,Man05], Mancini explains that $\exists SO$ has sufficient expressive power to axiomatize finite arithmetic (e.g. modulo domain size) using existential SO variables for (guessed) arithmetic functions and relations, although he does not suggest it as a practical approach. The MX-based IDP system [WM08] has a language which contains arithmetic and aggregates, and a grounder that handles these, but we are not aware of any expressiveness analysis for the language. Constraint specification languages such as ESSENCE [FGJ 07] allow the user to use quite generous form of arithmetic. In [MT07,MT08], we present various fragments of ESSENCE as model expansion (with
Built-in Arithmetic

metafinite structures) for syntactic variants of FO and SO logic, to carry out an analysis of expressiveness of the language. We discuss several related issues such as compact representation of an input domain and its impact on the complexity of the problems expressed. A version of ASP with infinite domains is described in [HNV06]. The authors study satisfiability problem, not model expansion, and decidability and complexity results for that problem are obtained for several variants of ASP (loosely guarded programs and generalized programs similar to Datalog LITE). While the variant of the guarded fragment GF_k used in our work naturally corresponds to type systems of constraint specification languages such as ESSENCE, we do not see a clear correspondence between types and the loosely guarded fragment used in [HNV06]. No work we are aware of has presented a framework for search problems with built-in arithmetic where the user is given an assurance of universality for a given complexity class.

References


A Discussion of Embedded Models and Metafinite Models

Embedded Model Theory  Much work in the area involves reducing questions about queries over embedded finite models to questions about queries over normal finite models. Many results are restricted to generic queries, but declarative programming axiomatizations involving numbers are rarely generic, so these are of limited use to us. Other results about the natural-active collapse and o-minimality could be useful to us elsewhere, but do not address the problems we deal with here. For example, a natural-domain-active-domain collapse for $\exists \forall O$ for finite models embedded in an infinite structure $M$ holds if $O$ quantification is over subsets of the active domain only, but we cannot have this restriction because solutions to search problems often involves numbers not contained in their instance. While some results of embedded model theory are likely to be useful, the framework does not provide conditions for capturing complexity classes.

Metafinite Structures [GG98] are sorted structures $D = (A, R, \mathcal{W})$, where $A$ is a finite primary structure, $R$ is the secondary structure, and $\mathcal{W}$ is a set of “weight functions” from $A^k$ to $R$. Typically, $R$ is a fixed infinite structure, such as the natural numbers $\mathbb{N}$ with standard arithmetic operations. Quantification is over the primary domain only. Metafinite structures are often considered with multiset operations such as max, min, sum, product. This is an important variant of metafinite structures since it allows one to easily formalize a wide range of aggregate operations such as those in SQL and several existing constraint languages. An Arithmetic structure $R$ contains at least $0, 1, +, \times, <$, multiset operations max, min, $\Sigma$ (sum), $\Pi$ (product). All functions, relations, and multiset operations of $R$ must be polytime.

Logics for Metafinite Structures are designed to allow application of methods of finite model theory. These are two-sorted logics, interpreted over combined two-sorted structures. In [GG98], the logics contain, in addition to the standard terms and formulas, a collection of weight terms which denote functions from the primary to the secondary part. New weight terms can be built by applying functions of the secondary structure to applications of other weight terms. Logics for metafinite structures always contain a characteristic function $\chi[\phi](\bar{x})$ which maps formula $\phi$ and an assignment of domain elements to free variables $\bar{x}$ to 0 or 1, which are elements of the secondary structure.

To obtain capturing of NP, the authors of [GG98] define a notion of metafinite spectrum, a counterpart to the generalized spectra in Fagin’s sense, and restrict attention to metafinite spectra of structures with “small weights” i.e., if $w$ is a weight function, and $w(\bar{a}) = s$ then $|s| = poly(|A|)$.

Theorem 4 (Grädel and Gurevich). Let $K$ be a class of small weight arithmetical structures that is closed under isomorphisms. The following are equivalent: (1) $K$ is in NP, (2) $K$ is a primary generalized spectrum (i.e., only the primary part is expanded, not the weight functions).

Discussion The work of Grädel and Gurevich to a large degree inspired our work here. However, the requirement of “no quantification over the secondary structure” was too restrictive for our purposes. In [GG98], access to the secondary structures is
through weight terms only. In natural MX specification, quantification over elements of background structures is essential. Instead of weight terms of [GG98], we introduced guarded quantification. Using lower guards was not sufficient – on arithmetical structures, unrestricted metafinite spectra capture the r.e. sets [GG98], which would imply the same property for our formalism. Thus, we needed upper guards as well. In our proof of capturing NP, we needed to develop a way to deal with numbers appearing as arguments of expansion predicates, which was not needed in [GG98]. Using guards (upper and lower) is especially natural because guards generalize types of practical systems. However, guards coming from the instance structure are not enough. We introduced a logic with user-definable guards and proved the property of capturing NP for it.

In our earlier work [TM07], we developed a two-sorted framework of metafinite model expansion much closer to [GG98], where we used mixed relations with arguments of primary and secondary universes. However, keeping track of which elements come from primary and which from secondary domains in definitions (e.g. that of structures of small cost) and theorems made exposition unnecessarily complex. Since an instance structure can be viewed as a database, embedded model theory seemed more appropriate. We see one more benefit of using the embedded setting. Having elements drawn from one universe simplifies development of a type system with subtypes – predicate symbols can be used to represent different types, and implications can be used to represent type taxonomies.

B Proof of Theorem 1

Proof. Suppose 2 holds. Upper guards ensure that we only need consider expansions with polynomially many tuples, and the small cost condition guarantees that the encodings of the expanded structures are of polysize, so we may guess an expansion in polytime. Recall that FO model checking over finite structures is polytime. To see that polytime model checking also holds in the current case, we need only observe that each function and relation of the background structure is polytime and that lower guards ensure every variable ranges over at most polynomially many values. Membership in NP follows.

For the other direction, if \( K \) is in NP, we may construct a formula \( \phi \) sufficient for showing 2), as follows. This is a generalization of an encoding used [GG98]. Consider a relation \( R(\overline{s}) \), where \( \overline{s} = s_1s_2 \ldots s_r \) denotes the tuple of arguments. For each such relation, we will introduce expansion relations \( I_{R,i} \) and \( S_{R,i}^1 \ldots S_{R,i}^r \). In each of these \( i \) is a \( k \)-tuple giving a base-\( n \) encoding (where \( n \) is the combined domain size, \( n = D \)), of a number in \( \{0, \ldots, n - 1\} \).

- \( I_{R,i}(\overline{t}) \) denotes that there is an \( \overline{t}^{th} \) tuple in \( R \).
- \( S_{R,i}^j(\overline{t}, \overline{J}) \) denotes that in the \( \overline{t}^{th} \) tuple (if there is such a tuple in \( R \)), the \( \overline{J}^{th} \) bit of the binary representation of \( s_i \) is 1.

For each expansion relation \( R \), we have one relation \( S_{R,i}^l \) for each argument to \( R \), where \( i \) is the index of the argument. The role of \( S_{R,i}^l \) is to encode the binary representation of each number occurring in \( R \). Argument \( \overline{t} \) indexes the possible tuples of \( R \) (the number of which is polynomial in \( n \)), while \( \overline{J} \) indexes the (polynomially many) possible
bits in a binary representation a number. The $S_R$s cannot distinguish absence of the $\vec{h}$
tuple from $R$ from the bits in the $\vec{h}$ tuple being all zero, so we use $I_R$ to indicate which
of the $n^k$ possible tuples for $R$ are actually present.

Now, we write a formula in our logic, over the vocabulary of $K$ plus the expansion
to the $I_R$ and $S_l^r, \ldots, S_r^r$, that says that these encode $R$ correctly. We begin by writing
a FO formula that does the job, and then describe how to transform it into a formula of
our logic $\text{GGF}_k(\varepsilon)$. For each expansion predicate symbol $R$, define:

$$\phi_R := \forall s_1 \ldots \forall s_r \left[ R(\bar{x}) \iff \exists \bar{I}_R(\bar{i}) \land \bigwedge_i (\Sigma_j(\chi[S_l^r(i, \bar{j})] \times \Pi_y(2 : y < \text{value}(\bar{j})) = s_l)) \right],$$

where $\text{value}(\bar{j})$ is $j_0n^0 + \ldots j_mn^m$, where $n$ is the size of the primary domain and $m$
is the length of the tuple $\bar{j}$, i.e., $\bar{j}$ is a base-$n$ representation of a number.

For each particular $l$ and $i$, the term $\Sigma_j(\chi[S_l^r(i, \bar{j})] \pi_y(2 : y < \text{value}(\bar{j}))$ computes the value of $s_l$ from its representation $S_l^r(i, \bar{j})$. Recall that the number $s_l$ is represented in $S_l^r(i, \bar{j})$, which is true (for a fixed $i$) on those tuples $\bar{j}$ which encode numbers of positions which are 1 in the binary representation of $s_l$. We have such a formula $\phi_R$ for each $R$.

Let $\mathcal{K}'$ be the class of structures $\mathcal{B}$ of the form $\mathcal{B} = ((A, \bar{R}, \bar{S}_R, N))$ for each
$((A, \bar{R}), N) \in \mathcal{K}$. Clearly, $\mathcal{K}$ is in NP if and only if $\mathcal{K}'$ is. Further $\mathcal{K}'$ is in NP if and
only if there is a first-order formula $\psi$ in the vocabulary of $\mathcal{B}$ such that a structure $\mathcal{B}$ is in $\mathcal{K}'$ if and only if there is an expansion $\mathcal{B}'$ of $\mathcal{B}$ to the vocabulary of $\psi$ so that $\mathcal{B}' \models \psi$.

Taking the conjunction of $\psi$ with all the $\phi_R$, we have a first-order formula $\phi$ such that
every structure $\mathcal{D}$ is in $\mathcal{K}$ if and only if there is an expansion $\mathcal{D}'$ of $\mathcal{D}$ to $\sigma'$ with $\mathcal{D}' \models \phi$.

It remains to show that we can re-write our FO formula into our logic $\text{GGF}_k$. Considering the $\phi_R$, we first re-write the bi-conditional into two material implications. In the $\Rightarrow$ direction, the upper guard for $R$ suffices as the guard. In the $\Leftarrow$ direction, we use the active domain quantifiers, which suffice as guards. $\Phi$ has quantification only over the primary domain, so is trivially guarded.