PBINT, a logic for modelling search problems involving arithmetic

Shahab Tasharrofi and Eugenia Ternovska

Simon Fraser University, Canada
{sta44,ter}@cs.sfu.ca

Abstract. Motivated by computer science challenges, Grädel and Gurevich [13] suggested to extend the approach and methods of finite model theory beyond finite structures, an approach they called Metafinite Model Theory. We develop this direction further, in application to constraint specification/modelling languages. Following [27], we use a framework based on embedded model theory, but with a different background structure, the structure of arithmetic which contains at least $(\mathbb{N}; 0, 1, +, \times, <, ||\ ||)$, where $||x||$ returns the size of the binary encoding of x. We prove that on these structures, we can *unconditionally* capture NP using a variant of a guarded logic. This improves the result of [27] (and thus indirectly [13]) by eliminating the small cost condition on input structures.

As a consequence, our logic (an idealized specification language) allows one to represent common arithmetical problems such as integer factorization or disjoint scheduling naturally, with built-in arithmetic, as opposed to using a binary encoding. Thus, this result gives a remedy to a problem with practical specification languages, namely that there are common arithmetical problems that can be decided in NP but cannot be axiomatized naturally in current modelling languages. We give some examples of such axiomatizations in PBINT and explain how our result applies to constraint specification/modelling languages.

1 Introduction

This paper shows an application of descriptive complexity [17, 12] to developing foundations of constraint specification/modelling languages, a sub-area of AI. Such languages are developed in several communities, have associated solvers, and are intended as universal languages for search problems in some complexity classes, usually NP (e.g. scheduling, planning, etc.). Examples include languages and systems of Answer Set Programming [25, 10], modelling languages from the CP community such as ESSENCE [9], or the language of the IDP system[1] [29]. These languages do not closely correspond to FO logic – they often contain inductive definitions and built-in arithmetic. Designers usually focus on the convenience of the language, and rarely pay attention to the expressiveness. For each language, several tasks can be studied – satisfiability and model checking are among them. Here, since we are interested in search problems, we focus on the task of *model expansion (MX)*, the logical task of expanding a given structure

[1] A language and system based on FO(ID), an extension of first-order logic with inductive definitions under well-founded semantics, see [7].

with new relations. The user axiomatizes their problem in some logic $\mathcal{L}$ (a specification/modelling language). The task of model expansion for $\mathcal{L}$ (abbreviated $\mathcal{L}$-MX), is:

Model Expansion for logic $\mathcal{L}$
Given: *1.* An $\mathcal{L}$-formula ϕ with vocabulary $\sigma \cup \varepsilon$
2. A structure $\mathcal{A}$ for σ
Find: an expansion of $\mathcal{A}$, to $\sigma \cup \varepsilon$, that satisfies ϕ.

Thus, we expand the structure $\mathcal{A}$ with relations and functions to interpret ε, obtaining a model $\mathcal{B}$ of ϕ. The complexity of this task obviously lies in-between that of model checking (the entire structure is given) and satisfiability (no part of a structure is given). In the *combined setting* an instance consists of a structure together with a formula. We focus here on *data complexity*, where the formula is fixed and the input consists of an instance structure only. We call σ, the vocabulary of $\mathcal{A}$, the *instance* vocabulary, and $\varepsilon := vocab(\phi) \setminus \sigma$ the *expansion* vocabulary.

Remark 1. Since FO MX can specify exactly the problems that ∃SO can, one might ask why we don't stick with the standard notion of ∃SO model checking. Primarily it is because we rarely use pure FO, and because for each language $\mathcal{L}$, MX is just one among several tasks of interest.

Descriptive complexity [17] and metafinite model theory [13, 12] can find applications in constraint modelling languages. The authors of [21] emphasized the importance of the property of *capturing NP* and other complexity classes for such languages. The capturing property is of fundamental importance as it shows that, for a given language:
(a) *we can express all of NP* – giving the user an assurance of universality of the language for a given complexity class,
(b) *no more than NP can be expressed* – thus solving can be achieved by means of constructing a universal polytime reduction to an NP complete problem such as SAT or CSP. This reduction is called propositionalization or *grounding*.
The authors proposed to take the capturing property as a *fundamental guarding principle* in the development and study of declarative programming for search problems in this complexity class, and started careful development of foundations of modelling languages of search problems based on extensions and fragments of first-order (FO) logic. While the current focus is on the complexity class NP, by no means, do we suggest that the expressive power of the languages for search problems should be *limited* to NP. Our goal is to design languages for non-specialist users who may have no knowledge of complexity classes. The users will be given a simple syntax within which they are *safe*, and they would be encouraged to express their problems in that syntax.

The classic Fagin's theorem [8], relating ∃SO and NP, states that parameterized (formula is fixed) FO MX captures NP. However, FO lacks many features needed for practical specification languages such as a built-in support for arithmetic. Fagin's theorem allows one to represent all problems in NP, however, there is no direct way to deal with numbers and operations on them since in logic we have abstract domain elements. Therefore, problems involving numbers have to encode their inputs and outputs using elements of the domain. A usable logic for these problems would use standard arithmetic, as in all realistic modelling languages.

A solution, inspired by a previous proposal [13], was given in [27]. There, MX ideas were extended to *embedded MX* to provide mathematical foundation for dealing with infinite arithmetic structure with $\times$, $+$, $<$, etc., and aggregate functions (min, sum etc.), operations that are considered "built-in". The authors needed a method for handling operations with outputs outside of the input domain, as is common in practical languages. They also desired universal quantification over integers since it is convenient and is used in practice. Access to the arithmetical structure through weight terms as in [13] was not sufficient. They defined two new logics, GGF$_k$ and DGGF$_k$. The former is an extension of the k-guarded fragment of FO (or FO(ID)), in which instance predicates are used as guards of quantifiers and expansion predicates (here, GG stands for double-guarded, not for [13]). DGGF$_k$, is an extension of GGF$_k$ in which definable guards are allowed, provided they are polysize in the domain size. The extension allows for quantifying over variables whose values fall outside of the input domain. It was proven that, under a small-cost condition, NP is captured for both fragments. That is, (a) for every problem in NP (represented by a class of logical structures) there is a specification in the logic such that an instance (a structure) is in the class iff there is an expansion of that structure that satisfies the specification; and (b) for every specification in the logic, the task of MX is in NP. The small cost condition says that values of the input numbers cannot be bigger than $2^{poly(n)}$, where $poly(n)$ is some polynomial in the size n of the domain.

The two fragments provide natural axiomatizations (that do not involve binary encodings) but have two limitations:

(1) Poly-size guards are too limiting. Suppose we want to output the total weight of all items in a $Knapsack$ (an expansion predicate). A natural axiom would be:
$\exists x\ (G(x) \wedge Output(x) \wedge x = \Sigma_y(\ Weight(y)\ :\ Knapsack(y))$. We cannot use a polysize guard G: there are up to 2^n distinct sums, where n is the size of the domain.

(2) Small cost condition cannot be satisfied in natural axiomatizations of some common problems in NP such as integer factorization or a quadratic programming problem: given m, a, c find x such that $x^2 = a\ (\mathbf{mod}\ m) \wedge x > c$.
The values of the given integers, m and a, are, in general, unlimited in the size of the input domain, which is 3.

As we show in [26], several practical modelling languages, including the system languages of ASP and IDP, meet the same challenges regarding the small cost condition. Both these languages capture NP over small cost arithmetical structures and, also, none of these languages can axiomatize factorization or the quadratic residue problem (two prominent non-small cost problems) using their built-in arithmetic. Although we did not discuss it in [26], the same analysis applies to some other system languages such as NP-SPEC [3] to show that their built-in arithmetic has limited expressibility.

Here, we introduce a new logic, PBINT, which is suitable for modelling problems that involve arithmetic as it eliminates the small cost condition and captures NP for all problems involving arithmetic. The logic can be viewed as an idealized specification/modelling language. PBINT uses a different background structure than that of [27], namely the structure containing at least $(\mathbb{N}; 0, 1, +, \times, <, ||\ ||)$, among other polytime relations. PBINT eliminates the two drawbacks above while retaining three important features:

– **Natural logic.** All problems with arithmetical operations can be axiomatized naturally (without e.g. binary encodings and with quantifiers over numbers).
– **Capturing NP**. The result is due to a new kind of existential and upper guards, and a slightly different set of allowable arithmetic operations.
– **Polytime Grounding**. The grounding time is polynomial in the size of the binary encoding of the input structure (not necessarily in the domain size).

2 Background: MX with Arithmetic

Throughout the paper, we use $:=$ for "denotes", $\Rightarrow$ for material implication, and $\exists\bar{x}$ for $\exists x_1 \ldots \exists x_n$, similarly for $\forall\bar{x}$.

Embedded MX Embedded finite model theory (see [19]), the study of finite structures whose domain is drawn from some infinite structure, was introduced to study databases that contain numbers and numerical constraints. 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 $\mathcal{A}$ is* embedded *in an infinite* background *(or* secondary*) structure $\mathcal{M} = (U; \bar{M})$ if it is a structure $\mathcal{A} = (U; \bar{R})$ with a finite set $\bar{R}$ of finite relations and functions, where $\bar{M} \cap \bar{R} = \emptyset$. The set of elements of U that occur in some relation of $\mathcal{A}$ is the* active domain of $\mathcal{A}$*, denoted $adom_{\mathcal{A}}$.*

Example 1. Consider a company database with a table containing employee numbers, salaries and pension plans. This database is a finite structure embedded in the infinite background structure of the natural numbers with the standard arithmetic operations. Queries over embedded databases may use the database relations and the arithmetical operations whose interpretation is provided by the infinite background structure. E.g. the following query (a FO formula with free variable x) returns people whose total salary and pension plan contribution is above \$100,000: $\exists s \exists p\ (empl(x, s, p) \wedge s + p \geq \$100,000)$.

In database research, embedded structures are used with logics for expressing queries. Here, we use them similarly, with logics for MX specifications. Throughout, we use the following conventions: σ denotes the vocabulary of the embedded structure $\mathcal{A} = (\mathcal{U}; \bar{\mathcal{R}})$, which is the instance structure; ν denotes the vocabulary of an infinite background structure $\mathcal{M} = (U; \bar{M})$; ε is an expansion vocabulary; $\bar{R}$ and $\bar{M}$ always denote the interpretations of σ and ν, respectively. We treat $\bar{R}$ and $\bar{M}$ as tuples or as sets, depending on the context. A formula ϕ over $\sigma \cup \nu \cup \varepsilon$ constitutes an MX specification. The model expansion task remains the same: expand a (now embedded) σ-structure to satisfy ϕ.

A Logic for Embedded MX: Double-guarded logic Just as [27], we use a guarded logic in an embedded setting, which allows us to quantify over elements of the background structure (unlike, e.g. [13]). Again, we use an adaptation of the guarded fragment GF_k of FO [11]. 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 (with respect to σ) is the smallest set of formulas that:*

1. *contains all atomic formulas;*
2. *is closed under Boolean operations;*
3. *contains $\exists \bar{x}\ (G_1 \wedge \ldots \wedge G_m \wedge \phi)$, provided the G_i are atomic formulas of σ, $m \leq k$, $\phi \in GF_k$, and each free variable of ϕ appears in some G_i.*
4. *contains $\forall \bar{x}\ (G_1 \wedge \ldots \wedge G_m \supset \phi)$ provided the G_i are atomic formulas of σ, $m \leq k$, $\phi \in GF_k$, and each free variable of ϕ appears in some G_i.*
   For a formula $\psi := \exists \bar{x}\ (G_1 \wedge \ldots \wedge G_m \wedge \phi)$, conjunction $G_1 \wedge \ldots \wedge G_m$ is called the existential guard *of the tuple of quantifiers $\exists \bar{x}$;* universal guard *is defined similarly.*

Example 2. Let ε be $\{E_1, E_2\}$. The following formula is not guarded: $\forall x \forall y\ (E_1(x, y) \supset E_2(x, y))$. It is guarded when E_1 is replaced by P which is not in ε. The following formula is the standard encoding of the temporal formula $Until(P_1, P_2)$: $\exists v_2\ (R(v_1, v_2) \wedge P_2(v_2) \wedge \forall v_3\ (R(v_1, v_3) \wedge R(v_3, v_2) \supset P_1(v_3)))$. The formula is 2-guarded, i.e., is in GF_2, but it is not 1-guarded.

The guards of GF_k are used to restrict the range of quantifiers. We also use "upper guard" axioms, which restrict the elements in expansion relations to those occurring in the interpretation of guard atoms. To formalize this, we introduce the following restriction of FO, denoted $\mathrm{GGF}_k(\varepsilon)$.

Definition 3. *The double-guarded fragment* $\mathrm{GGF}_k(\varepsilon)$ *of FO, for a given vocabulary ε, is the set of formulas of the form $\phi \wedge \psi$, with $\varepsilon \subset vocab(\phi \wedge \psi)$, where ϕ is a formula of GF_k, and ψ is a conjunction of* upper guard axioms*, one for each symbol of ε occurring in ψ, of the form $\forall \bar{x}\ (E(\bar{x}) \supset G_1(\bar{x}_1) \wedge \cdots \wedge G_m(\bar{x}_m))$, where $m \leq k$, and the union of free variables in the G_i is precisely $\bar{x}$.*

We call the guards of GF_k, that restrict the range of quantifiers, *lower guards*, and the guards from Def. 3 *upper guards*. Upper guards on expansion functions are discussed later. In GGF_k, all upper and lower guards are from the instance vocabulary σ, so ranges of quantifiers and expansion predicates are explicitly limited to $adom_{\mathcal{A}}$. In DGGF_k, this restriction is relaxed, adding a mechanism for "user-defined" guard relations that may contain elements outside $adom_{\mathcal{A}}$. The authors assume that the instance vocabulary always contains the predicate symbol $adom_{\mathcal{A}}$, which always denotes the active domain. Then $adom_{\mathcal{A}}(x)$ can be used as a guard atom (upper or lower)[2]. Guards provide a logical formalization of some aspects of the type systems of some existing constraint modelling languages [22]. Lower guards correspond to declaring the types of variables, and upper guards to declaring the types of expansion predicates.

So far, we explained how *formulas* are constructed. To finish definition of the logic, we need to define well-formed terms. This definition depends in the vocabulary of the background structure. The authors of [27] used *arithmetical structures*, same as [13].

[2] The relation which corresponds to the active domain is definable with respect to each instance structure, but the defining FO formula requires disjunctions, thus cannot be used as a guard and the predicate symbol $adom_{\mathcal{A}}(x)$ is necessary.

Arithmetical Structure In addition to standard arithmetical operators, it has a collection of *multiset operations*, including max, min, sum and product.

Definition 4. *An* Arithmetical structure *is a structure $\mathcal{N}$ containing at least $(\mathbb{N}; 0, 1, \chi, <, +, \cdot, min, max, \Sigma, \Pi)$, with domain $\mathbb{N}$, the natural numbers, and where min, max, Σ, and Π are multiset operations and $\chi[\phi](\bar{x})$ is the characteristic function. Other functions, predicates, and multiset operations may be included, provided every function and relation of $\mathcal{N}$ is polytime computable.*

Well-formed terms are defined over $\nu \cup \sigma \cup \varepsilon$ by induction, as usual. The details are not important here. The authors of [27] proved that, using operations mentioned above, the MX task for logics GGF_k and DGGF_k and for structures embedded in arithmetical structures captures NP under small cost condition.

3 Logic PBINT

The first step towards eliminating the limitations of the previous logics is to use a different background structure.

Definition 5. *A* Compact Arithmetical structure *is a structure $\mathcal{N}^c$ containing at least $(\mathbb{N}; 0, 1, +, \times, <, ||\ ||)$ with domain $\mathbb{N}$, the natural numbers, where 0, 1, $+$, $\times$ and $<$ have their usual meaning and $||x||$ returns the size of binary encoding of number x, i.e., $||x|| = 1 + \lfloor log_2(x+1) \rfloor$. Other functions, predicates, and multi-set operations (min, max etc.) may be included, provided every function and relation of $\mathcal{N}^c$ is polytime computable.*

Our capturing results in Section 4 remain valid even when the background structure's domain changes from $\mathbb{N}$ to $\mathbb{Z}$ (although a more detailed proof would be needed).

Requirements on σ As before, we consider embedded MX, but the embedding is into the compact arithmetical structure. We make some assumptions about the instance vocabulary σ. It contains predicate $adom_{\mathcal{A}}$ and a constant $SIZE$. The constant $SIZE$ is equal to $|adom_{\mathcal{A}}| \times S$ where $|adom_{\mathcal{A}}|$ is the number of elements in the active domain and S is the size of binary encoding of the maximum element of the active domain. In other words, $SIZE$ upper-bounds the number of bits needed to encode (in binary) the input structure $\mathcal{A}$ embedded in $\mathcal{N}^c$. We also need a constant $default$ denoting a particular default value needed in upper guards on functions. Its meaning is specified by the user.

Logic PBINT We introduce a new logic, PBINT, standing for Polynomially Bounded Integers. This logic is a variant of the double-guarded logic except we use compact arithmetical structures and allow functions in σ and ε, or new kinds of guards, with more freedom in existential and upper guards on the outputs of expansion functions. The three forms of guards in PBINT are as follows:

1. **Instance Guards** are instance predicates (including $adom_{\mathcal{A}}$) interpreted by the instance structure $\mathcal{A}$. Note that, although we do not require it to be so, all specifications can be rewritten to only use $adom_{\mathcal{A}}$ as a guard.

2. **Polynomial Range Guards** are relations of the form $p(SIZE) \leq x \leq p'(SIZE)$ with p and p' two polynomials.
3. **PBINT Guards** are relations of the form $||x|| \leq poly(SIZE)$ where $poly(SIZE)$ is a polynomial depending only on the constant $SIZE$.

Instance guards and polynomial range guards define ranges of size at most polynomial in the binary encoding size of structure. However, PBINT guards can define ranges with exponentially many different integers. For example, condition $||x|| \leq SIZE$ is equivalent to $x \leq 2^{SIZE-1} - 1$, exponential the in value of $SIZE$. Also note that guards definable by stratifiable inductive definitions with (1), (2) as the base cases can be added without changing our results.

Definition 6 (logic PBINT). *We define our logic as follows.*
Background Structure: *the compact arithmetical structure.*
Terms *are constructed as usual over* $\nu \cup \sigma \cup \varepsilon$.
Formulas:

(a) **Upper Guards**
- *i. Expansion* relations *are upper-guarded by instance or polynomial range guards.*
- *ii. An expansion* function f *has an upper guard axiom of the form* $\forall \bar{x} \forall y\ (f(\bar{x}) = y \Rightarrow (G(\bar{x}, y) \vee y = default))$ *where* $G(\bar{x}, y)$ *is a conjunction of guards jointly guarding variables* $\bar{x}$ *and* y *so that* $\bar{x}$ *is upper-guarded by instance or polynomial range guards and* y *is upper-guarded by any of the three types of guards.*

(b) **Lower Guards**
- *i. Existential guards: any of the three types of guards.*
- *ii. Universal guards: instance or polynomial range guards.*

The upper guards on expansion functions need the $y = default$ disjunct to keep the upper guard meaningful. Otherwise, any expansion function would have its input restricted by a guard $G(\bar{x}, y)$ which is in contradiction to the totality of functions (therefore making the specification outright false).

A similar problem happens when instance functions are allowed: the usual definition of active domain becomes meaningless because a function is total and thus defined on all the integers making the active domain equal to $\mathbb{N}$. While it is possible to disallow the instance functions, it is certainly not desirable for any nice practical logic. Therefore, we choose to allow instance functions, but to require them to have upper guards and the "default" value for inputs outside of the intended range, just as for expansion functions. Active domain now contains all elements of the universe contained in all instance relations, together with all elements in the ranges of the instance functions. This trick also enables us to ensure that both instance and expansion functions have finite (also polynomial size) representation.

Example 3 (Disjoint Scheduling). Given a set of Tasks, $t_1, \cdots, t_n$ and a set of constraints, find a schedule that satisfies all the constraints. Each task t_i has an earliest starting time $EST(t_i)$, a latest ending time $LET(t_i)$ and a length $L(t_i)$. There are also two predicates $P(t_i, t_j)$, that says task t_i should end before task t_j starts, and $D(t_i, t_j)$, which means that the two tasks t_i and t_j cannot overlap. We are asked to find two functions $start(t_i)$ and $end(t_i)$ satisfying the given conditions.

In PBINT, we axiomatize this problem as follows: Instance vocabulary σ consists of symbols EST, LET, L, $Task$, P and D. Expansion vocabulary consists of two functions $start$ and end which are upper-guarded as follows:

$$\begin{aligned}
&\forall t \forall s\ (start(t) = s \Rightarrow \\
&\quad (Task(t) \wedge ||s|| \leq SIZE) \vee s = default), \\
&\forall t \forall e\ (end(t) = e \Rightarrow \\
&\quad (Task(t) \wedge ||e|| \leq SIZE) \vee e = default).
\end{aligned}$$

The following sentences axiomatize the problem statement:

$$\begin{aligned}
&\forall t_i\ (Task(t_i) \Rightarrow start(t_i) \geq EST(t_i)), \\
&\forall t_i\ (Task(t_i) \Rightarrow end(t_i) \leq LET(t_i)), \\
&\forall t_i\ (Task(t_i) \Rightarrow start(t_i) + L(t_i) = end(t_i)), \\
&\forall t_i \forall t_j\ (P(t_i, t_j) \Rightarrow end(t_i) \leq start(t_j)), \\
&\forall t_i \forall t_j\ (D(t_i, t_j) \Rightarrow \\
&\quad end(t_i) \leq start(t_j) \vee end(t_j) \leq start(t_i)).
\end{aligned}$$

In a practical language, usually upper and lower guards are defined by types and need not be given explicitly. For example, here, the predicate $Task$ is a type and functions $start$ and end are functions from the type $Task$ to integer type. So, predicate $Task$ disappears from the above sentences.

Now, let us compare how this problem is axiomatized under small cost condition. Note that this class of structures dissatisfies the small cost condition because values in a structure, e.g. $LET(t_i)$, are not related to the domain size, i.e., the number of tasks. So, another class of structures is needed here. One general choice would be to encode all numbers in binary. For example, instead of function EST, there will be predicate $EST'(t_i, j)$ for which, $EST(t) = n$ iff $n = \Sigma_k(2^k : EST'(t, k))$. To simplify, let us also assume that there is a unary relation B defining the set of bit indices, e.g. all values appearing in the second position of a tuple in EST' are in B. Also, assume that 0 is the minimum value in B and that if $p > 0$ is in B, so is $p - 1$. Now that we are no longer working on "built-in" numbers, numerical operations have to be axiomatized. For example, formula $start(t_i) \leq EST(t_i)$ is replaced by: (all quantified variables below are lower-guarded by B)

$$\begin{aligned}
&E(0) \vee \exists k\ (\neg start'(t_i, k) \wedge EST'(t_i, k) \wedge E(k + 1)), \\
&E(k) := \forall k'\ (k' \geq k \Rightarrow (start'(t_i, k) \Leftrightarrow EST'(t_i, k))).
\end{aligned}$$

Other inequalities in the above axioms are replaced by similar formulas. The addition operator in $start(t_i) + L(t_i) = end(t_i)$ is axiomatized as follows: (all quantifiers are lower-guarded by B and operator $\oplus$ stands for logical xor)

$$\begin{aligned}
&\forall k\ (start(t_i, k) \oplus L(t_i, k) \oplus C(k) \Leftrightarrow end(t_i, k)), \\
&C(k) := \\
&\quad \exists k'\ (L(t_i, k') \wedge start(t_i, k') \wedge k' < k \wedge CF(k, k')), \\
&CF(k, k') := \\
&\quad \forall k''\ (k' < k < k'' \Rightarrow L(t_i, k'') \vee start(t_i, k'')).
\end{aligned}$$

The first axiomatization is incomparably more natural.

Example 4 (Factorization). You are given a number n and asked to find some nontrivial factorization for n. Here, σ only has constant n and ϵ only has constants p and q which are upper-guarded as follows $||p|| \leq SIZE$ and $||q|| \leq SIZE$ where $||c|| \leq SIZE$ abbreviates $\forall m\ (c = m \Rightarrow ||m|| \leq SIZE)$ in case of zero-ary (constant) expansion functions. Now, the axiomatization is:

$$p > 1 \wedge p < n \wedge q > 1 \wedge q < n \wedge p \times q = n.$$

Example 5 (Quadratic Residues). You are given numbers r, n and c and asked to find a number x such that $x^2 \equiv r\ (\mathbf{mod}\ n)$ and $x < c$. Here, instance vocabulary consists of constants n, r and c and expansion vocabulary only has constant x upper-guarded by sentence $||x|| \leq SIZE$. The axiomatization consists of two sentences $0 \leq x \wedge x \leq c \wedge x < n$ and $\exists q\ (||q|| \leq SIZE \wedge x \times x = q \times n + r)$.

Both Factorization and Quadratic Residue problems would have to be axiomatized in binary in the logics of [27].

4 Capturing NP

Theorem 1. *Let $\mathcal{K}$ be an isomorphism-closed class of compact arithmetical embedded structures over vocabulary σ. Then the following are equivalent:*

1. *$\mathcal{K} \in NP$,*
2. *there is a PBINT sentence ϕ of a vocabulary $\tau = \sigma \cup \nu \cup \varepsilon$, such that $\mathcal{A} \in \mathcal{K}$ iff there exists an expansion $\mathcal{B}$ of $\mathcal{A}$ with $\mathcal{B} \models \phi$.*

The proof for the two different directions of this theorem are given in separate subsections. But, first, we introduce a characterization for PTIME due to Bellantoni and Cook [1] which is needed for our proof of (1) $\Rightarrow$ (2).

4.1 Bellantoni-Cook Characterization of PTIME

We briefly describe a functional language introduced by Bellantoni and Cook [1] which captures polytime functions. It has originally been defined to work on strings in $\{0,1\}^*$. But, as such strings encode numbers, we have reformulated the operations in numerical terms.

Functions in Bellantoni-Cook form have two sets of parameters separated by a semicolon. Parameters to the left of semicolon are called "normal" inputs and those to its right are "safe" inputs. This separation disables the possibility of introducing recursions whose depth depend on the result of other recursions. This property is essential to prove that such functions are poly-time computable. Here are the constructs:

1. Zero: $Z(;) = 0$.
2. Projections $\pi_j^{n,m}(x_1, \cdots, x_n; x_{n+1}, \cdots, x_{n+m}) = x_j$.
3. Successors $S_0(;a) = 2 \times a,\ S_1(;a) = 2 \times a + 1$.
4. Modulo 2: $M(;a) = a\ \mathbf{mod}\ 2$.
5. Predecessor: $P(;a) = \lfloor \frac{a}{2} \rfloor$.
6. Conditional: $C(;a,b,c) = \mathbf{if}\ 2|a\ \mathbf{then}\ b\ \mathbf{else}\ c$.

7. Safe recursion that defines $n+1$-ary function f based on n-ary function g and the $n+2$-ary functions h_0 and h_1:

$$f(0,\overline{x};\overline{y}) = g(\overline{x};\overline{y}),$$
$$f(2a,\overline{x};\overline{y}) = h_0(a,\overline{x};\overline{y},f(a,\overline{x};\overline{y})),$$
$$f(2a+1,\overline{x};\overline{y}) = h_1(a,\overline{x};\overline{y},f(a,\overline{x};\overline{y})).$$

8. Safe composition that defines function f based on functions $r_0,\cdots,r_{k+k'}$:

$$f(\overline{x};\overline{y}) = \\ r_0(r_1(\overline{x};),...,r_k(\overline{x};);r_{k+1}(\overline{x};\overline{y}),...,r_{k+k'}(\overline{x};\overline{y})).$$

This language interests us as it is a purely syntactic characterization of PTIME which is based on numbers. Furthermore, the language is free of any unnatural functions for bounding growth of numbers.

Bellantoni-Cook's theorem says that any function defined in this form is PTIME and that for any PTIME computable function $f(\overline{a})$, there is a function $f'(w;\overline{a})$ such that $f(\overline{a}) = f'(w;\overline{a})$ for all a's and for all w's satisfying $||w|| \geq p_f(\overline{||a||})$ (where $||x||$ is the binary encoding size of x and p_f is a polynomial depending on f and constructible based on Bellantoni-Cook's proof).

4.2 NP $\subseteq$ PBINT MX

Proof. Let us first review our proof structure for (1) $\Rightarrow$ (2).

1. We know NP problems have PTIME verifiers.
2. By Bellantoni and Cook's theorem, every such polytime verifier can be given in their syntax.
3. So, it remains to show that, verifier V in Bellantoni-Cook form, can be turned into axiomatization ϕ in PBINT so that for σ-structure $\mathcal{A}$, ϕ is satisfiable by an expansion of $\mathcal{A}$ iff there is a polysize certificate for $\mathcal{A}$ accepted by V.

As this proof is so detailed, we only include the proof idea here. The full proof is in the Appendix.

The proof constructs PBINT specification ϕ based on verifier V. In ϕ, expansion vocabulary ε consists of:

1. $B : \mathbb{N} \times \mathbb{N}$ to map start times to functions, e.g., $B(5, c_f)$ means that, at time 5, function f has started running (c_f is a constant used to refer to function f).
2. $E : \mathbb{N} \times \mathbb{N}$ which, similarly, maps start times to end times, e.g., $E(5, 10)$ means that the function that had started running at time 5 ended running at time 10, and together with $B(5, c_f)$, it means that function f started at time 5 and ended at time 10.
3. $r : \mathbb{N} \to \mathbb{N}$ is used to store result of function executions. It is needed only when execution of a function terminates. For instance, continuing example above, having $r(10) = 2$ means that result of computing function f is 2 (because it was f that ended at time 10).
4. $arg : \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ is used for storing function arguments. Semantically, $arg(n, m) = k$ means that the m^{th} argument of function starting at time n is k. For example, having $arg(5, 1) = 7$ together with examples listed in items (1) to (3), means that

function f started running and at time 5 on argument 7, and it finished at time 10 and gave result 2, so $f(7) = 2$.

This expansion vocabulary enables us to simulate the behavior of a program in Bellantoni-Cook form. We will have axioms saying what each function does. For example, for base function Z we have that it finishes immediately and gives zero as result. This is axiomatized as below:

$$\forall n\ (T(n) \wedge B(n, c_Z) \Rightarrow E(n, n+1)),$$
$$\forall n\ (T(n) \wedge B(n, c_Z) \Rightarrow r(n+1) = 0).$$

where $T(n)$ is a lower guard which bounds the time needed for simulating verifier V (a polynomial in the size of binary encoding of structure).

All other base functions have similarly simple axiomatizations. Functions defined in terms of other functions (using safe recursion or safe composition) need more complex (but still straightforward) axiomatizations. All these details can be found in the full proof.

There are some more subtle issues that have to be addressed in order to give a correct proof, e.g. encoding structures as numbers. All these subtleties are dealt with in the full proof given in the Appendix (which did not fit here, please see the online version).

4.3 PBINT MX$\subseteq$ NP

Proof. Here, the proof goes as follows:

1. We first show that, given an MX specification ϕ in PBINT, you can find equivalent ψ in $\exists$SO by using binary encodings of numbers.
2. Next, we show that structure $\mathcal{A}$ for ϕ is convertible to structure $\mathcal{A}'$ for ψ (in polytime) so that satisfying expansion $\mathcal{B}$ of $\mathcal{A}$ exists iff satisfying expansion $\mathcal{B}'$ of $\mathcal{A}'$ exists.
3. Then, by Fagin's theorem, PBINT MX$\subseteq$NP.

To obtain ψ, we first create a specification in which all existentially quantified variables with PBINT guard G are replaced by skolemized PBINT functions upper-guarded by G. Then, this specification is converted to ψ by replacing PBINT functions with relations that encode value of the function in binary. For example, for PBINT function $f(x_1, \cdots, x_n)$, relation $Q_f(x_1, x_2, \cdots, x_n, k)$ is introduced with k being guarded by new relation R. The idea is that $Q_f(x_1, \cdots, x_n, k)$ holds iff the k-th bit of binary encoding of $f(x_1, \cdots, x_n)$ is one.

We know that all numbers in a PBINT specification are guarded. Hence, there is a polynomial $p(n)$ such that $2^{p(SIZE)}$ is greater than all numbers generated in ϕ. So, assuming that we have a relation R containing all numbers $0, \cdots, p(SIZE)$, describing operations of background structure is easy. For example, assuming that x, y and z are encoded by unary predicates Q_x, Q_y and Q_z, the relation $x = y + z$ can be axiomatized

as follows:

$$\begin{aligned}
&\forall k\ (R(k) \Rightarrow \\
&\quad (Carry(k) \Leftrightarrow (Q_x(k) \Leftrightarrow (Q_y(k) \Leftrightarrow \neg Q_z(k))))), \\
&Carry(k) := \\
&\quad \exists k'\ (R(k') \wedge k' < k \wedge Q_y(k') \wedge Q_z(k') \wedge CF(k, k')), \\
&CF(k, k') := \\
&\quad \forall k''\ (R(k'') \wedge k' < k'' < k \Rightarrow Q_y(k'') \vee Q_z(k'')).
\end{aligned}$$

Although cumbersome, all other background operations can be similarly axiomatized.

Now, structure $\mathcal{A}'$ is obtained from $\mathcal{A}$ by adding unary relation R to $\mathcal{A}$ and converting all numbers in $\mathcal{A}$ to their binary representation. These tasks can be done in polytime and so obtaining $\mathcal{A}'$ from $\mathcal{A}$ is polytime achievable.

So, as ψ is in $\exists$SO, the task of model expansion for ψ is in NP. Also, as $\mathcal{A}$ is polytime convertible to $\mathcal{A}'$ and existence of a satisfying expansion for $\mathcal{A}$ is equivalent to existence of a satisfying expansion for $\mathcal{A}'$, model expansion for ϕ will also be in NP.

5 Related Work

Research in databases over infinite structures can be traced back to the seminal paper by Chandra and Harel [4]. There are several follow-up papers with developments in several directions including [28, 24, 13], and more recent [12]. Topor [28] studies the relative expressive power of several query languages in the presence of arithmetical operations. He also investigates domain independence and genericity in such frameworks.

Another line of database-motivated work over infinite background structures is embedded model theory (See [19, 20]). Work in this area generally reduces questions on embedded finite models to questions on normal finite models. An important result in this area is the natural-domain-active-domain collapse for $\exists$SO for embedded finite models, as well as other deep expressiveness results. The work also describes a notion of safety (through e.g. range-restriction) to achieve safety with many background structures, and connections between safety and decidability. The active domain quantifiers are similar to our proposal of lower guards, however our goal was to reflect what is used in practical languages, namely the so-called domain predicates of Answer Set Programming and type information from other languages. We've done it through the use of upper and lower guards. In general, research in database theory is mostly focused around computability and the expressive power of query languages, while our interest, following [12] is in capturing complexity classes, but in connection with specification/modelling languages. We plan, however, to investigate the applicability of domain-independence, range-restrictedness and other notions from embedded model theory to practical modelling languages.

Grädel and Gurevich [13] studied logics over infinite background structures in a more general computer science context. They characterized NP for arithmetical structures under some small weight property, generalized to the small cost condition in [27] (see [27] for a more detailed discussion). While this condition corresponds to existing practical languages such as ASP and IDP system (see our paper [26] for more details),

our work here gives an unconditional result for capturing NP in the presence of arithmetical structures, and thus is a step forward in the development of such languages. Instead of controlling access to the background structure through the use of weight terms [13], we rely on guarded fragments, which is much closer to practical specification languages.

The work we mentioned so far is the closest to our proposal, and was the most inspirational. The research on descriptive complexity in the embedded setting also includes the work of Grädel and Meer [15], as well as Grädel and Kreutzer [14]. Another line (Cook, Kolokolova and others [6]) establishes connections between bounded arithmetic and finite model theory, in particular by relying on Grädel's characterization of PTIME. While this work is less relevant here, we still plan to provide more detail in the journal version of the paper.

Another direction on capturing complexity classes is bounded arithmetic, including [2, 23, 1]. However, the characterization of complexity classes there is in terms of *provability* in systems with a limited collection of non-logical symbols, and is not applicable here.

There are many different characterizations of PTIME such as Leivant's [18], Immerman's [16], Cobham's [5] and Bellantoni-Cook characterization [1]. Leivant's characterization says that PTIME functions are exactly those that are provable in a logic called $L_2(QF^+)$. Immerman's logic is a fix-point logic with least fix-point operator and $\leq$ which works on structures with abstract domain elements. The two other characterizations of Cobham's and Bellantoni-Cook have the property of characterizing PTIME as a set of functions working on numbers and so better suited for our purpose of characterizing search problems over arithmetical structures. Also, the safe recursion and safe composition operators in the Cook-Bellantoni characterization give us a more natural way of guaranteeing that the result of the simulation we need in our proof falls within some bounds. Therefore, we choose the Bellantoni-Cook characterization [1] over Cobham's as the basis of our proof.

Built-in arithmetic is implemented in many modelling languages, e.g. the MX-based IDP system [29] and LPARSE [25]. However, as we showed in a parallel work in [26], such languages have limited expressiveness in the presence of arithmetic constraints. For example, we showed in [26] that the two problems of integer factorization and quadratic residues are not expressible in ASP and IDP systems using their built-in arithmetic. Also, in many cases, allowing arithmetic constraints without careful restrictions provides the language with very high expressiveness, as is shown for ESSENCE [22].

6 Conclusion

In modelling languages, you are frequently faced with the problem of having a framework to support both a natural specification of problems, and reasoning about those problems. In this paper, we took our measure of naturality to be being able to use "built-in" arithmetic, and our measure of reasoning to be being in NP. We showed some examples of problems of practical importance and argued that our fragment of logic is able to represent them naturally. We proved that embedded (in $\mathcal{N}^c$) MX for PBINT captures exactly NP. This result guarantees universality of our logic for this complexity

class and also settles our reasoning abilities by showing that all PBINT axiomatizations can be efficiently (in polytime) grounded to any state of the art solver of NP problems. Our work is a significant step forward from the previous proposal since it overcomes a number of limitations.

The language we proposed is natural because it is essentially FO logic, where guards can be made "invisible" through "hiding" them in a type system. Solving can be achieved through grounding to SAT, a work which is being performed in our group, but falls outside the scope of this paper.

In summary, our work has shown a new application of descriptive complexity and metafinite model theory, and contributed to those areas by improving a previous result of capturing NP for arithmetical structures. Future directions include (a) analysis of existing languages in connection with our results here, similar to what was done for ESSENCE with respect to the previous proposal [22]; (b) design of logics with different background structures, (c) extending the framework to deal with combination of languages, interacting in a modular system, (d) continue with our implementation development.

Acknowledgement. This work is generously funded by NSERC, MITACS and D-Wave. We also express our gratitude towards the anonymous referees for their useful comments.

References

1. S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions (extended abstract). In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 283–293, 1992.
2. S. R. Buss. *Bounded arithmetic*. PhD thesis, Princeton University, 1985.
3. M. Cadoli, L. Palopoli, A. Schaerf, and D. Vasile. Np-spec: An executable specification language for solving all problems in np. In *PADL '99: Proceedings of the First International Workshop on Practical Aspects of Declarative Languages*, pages 16–30, London, UK, 1998. Springer-Verlag.
4. A. Chandra and D. Harel. Computable queries for relational databases. *Journal of Computer and System Sciences*, 21:156–178, 1980.
5. A. Cobham. The intrinsic computational difficulty of functions. In *Y. Bar-Hillel ed., Proc. of the 1964 International Congress for Logic, Methodology, and the Philosophy of Science*, pages 24–30, 1964.
6. S. Cook and A. Kolokolova. A second-order system for polytime reasoning based on grädel's theorem. In *Proceedings of Sixteenth Annual IEEE Symposium on Logic in Computer Science (LICS '01)*, pages 177–186, 2001.
7. M. Denecker and E. Ternovska. A logic of non-monotone inductive definitions. *TOCL*, 9(2):1–51, 2008.
8. R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation, SIAM-AMC proceedings*, 7:43–73, 1974.
9. A. M. Frisch, M. Grum, C. Jefferson, B. M. Hernandez, and I. Miguel. The essence of essence: A constraint language for specifying combinatorial problems. In *Proc. of the Fourth International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pages 73–88, 2005.

10. M. Gebser, T. Schaub, and S. Thiele. Gringo: A new grounder for answer set programming. In C. Baral, G. Brewka, and J. Schlipf, editors, *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *Lecture Notes in Artificial Intelligence*, pages 266–271. Springer-Verlag, 2007.
11. G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. In *PODS '01*, 2001.
12. E. Grädel. *Finite Model Theory and Descriptive Complexity*, pages 125–230. Springer, 2007.
13. E. Grädel and Y. Gurevich. Metafinite model theory. *Inf. Comput.*, 140(1):26–81, 1998.
14. E. Grädel and S. Kreutzer. Descriptive complexity theory for constraint databases. In *Proceedings of the Annual Conference of the European Association for Computer Science Logic, CSL '99, Madrid*, volume 1683 of *LNCS*, pages 67–81. Springer, 1999.
15. E. Grädel and K. Meer. Descriptive complexity theory over the real numbers. *Mathematics of Numerical Analysis: Real Number Algorithms*, 32:381–403, 1996.
16. N. Immerman. Relational queries computable in polynomial time. In *STOC '82: Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pages 147–152, 1982.
17. N. Immerman. *Descriptive complexity*. 1999.
18. D. Leivant. A foundational delineation of computational feasibility. In *LICS '91: Proceedings of the sixth Annual IEEE Symposium on Logic in Computer Science*, pages 2–11, 1991.
19. L. Libkin. *Elements of Finite Model Theory*. 2004.
20. L. Libkin. *Embedded Finite Models and Constraint Databases*, pages 257–338. Springer, 2007.
21. D. G. Mitchell and E. Ternovska. A framework for representing and solving NP search problems. In *Proc. AAAI'05*, 2005.
22. D. G. Mitchell and E. Ternovska. Expressiveness and abstraction in ESSENCE. *Constraints*, 13(2):343–384, 2008.
23. A. Skelley. *Theories and Proof Systems for PSPACE and the EXP-Time Hierarchy*. PhD thesis, University of Toronto, 2005.
24. D. Suciu. Domain-independent queries on databases with external functions. *Theor. Comput. Sci.*, 190(2):279–315, 1998.
25. T. Syrjänen. *Lparse 1.0 User's Manual*, 2000. http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz.
26. S. Tasharrofi and E. Ternovska. Built-in arithmetic in knowledge representation languages. In *Proc. of Logic and Search (LaSh) 2010*, 2010.
27. E. Ternovska and D. G. Mitchell. Declarative programming of search problems with built-in arithmetic. In *Proc. of 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 942–947, 2009.
28. R. Topor. Safe database queries with arithmetic relations. In *Proc. 14th Australian Computer Science Conf*, pages 1–13, 1991.
29. J. Wittocx and M. Marien. *The IDP System*. KU Leuven, www.cs.kuleuven.be/~dtai/krr/software/idpmanual.pdf, June 2008.