Meta-Finite Model Expansion (Preliminary Report)

Eugenia Ternovska, David Mitchell, Brendan Guild

Simon Fraser University

In this Talk

We discuss a potential role of descriptive complexity and finite model theory in developing foundations of declarative programming for solving search problems

We argue that Model Expansion (MX) is the underlying task for this type of programming

Our goal is to incorporate arithmetic and other secondary structures in our MX-based framework and to ensure a capturing NP property

In this Talk

We discuss existing approaches with similar goals (embedded model theory and meta-finite model theory)

— what works and what does not

We mention some future directions

Declarative Programming

A program is “declarative”

if it describes what something is like,
rather than how to create it. (Meaning 1, Wikipedia)

(The second meaning calls declarative programming everything which is not imperative, but in fact, it is mixed)

We are interested in declarative programming of search problems

Declarative Languages for Search Problems

A big variety:

- some are based on non-classical logics (e.g. logic programming under Stable semantics),
- many have ad-hoc syntactic limitations “to optimize solver performance”, without theoretical foundation,
- some use languages under denotational semantics

most are designed for solving NP-Search problems

A Natural Approach

describe the problem in ∃SO, and

solve particular instances by (a uniform) reduction to SAT and application of a SAT solver.

The underlying task: Model Expansion (abbreviated MX)

Given: ϕ in logic $\mathcal{L}$ and structure $\mathcal{A}$ for a part of $vocab(\phi)$
Construct: an expansion of $\mathcal{A}$ to satisfy ϕ

Fagin's, Grädel's etc. results give "programming languages" for various complexity classes (e.g. for NP, ϕ is FO)

([MT05]: this is the right thing to do in practice)

Why has not been done a long time ago?

Conventional wisdom: FO logic (or its extensions) cannot work as the foundation for constraint programming:

- Cannot build practical efficient systems for such a general syntax;
- Languages that work in practice do not resemble logic (on the surface):
  - they have types and complex domains;
  - some constraints are expressed directly, and are directly dealt with by the underlying solvers e.g.
    - relation is a bijection
    - variables in a set have all distinct values

Our View

It is our belief that, contrary to the conventional wisdom, descriptive complexity is
the appropriate theoretical foundation
for the field of declarative constraint programming,

and that effective practical modelling languages will turn out to be syntactic variants of FO^{++}
– similar to the development in databases,

and that the underlying task will often turn out to be model expansion (the example of Essence and other)

What we Have Done so far

We have taken initial steps toward demonstrating practical feasibility

We produced an implementation that performs as well as much more mature systems using other approaches.

The language for this system is essentially FO extended with a limited use of inductive definitions

This removes one doubt about our approach (that language restriction is necessary)

To Really Make it Work, we Need

Arithmetic and other secondary structures (e.g. strings)

Could give $+, \times$ etc. as part of an instance, but people don't want to deal with arithmetic mod some number

How does one design a language that allows the user to use arithmetic as if it was unrestricted, and at the same time does not take us outside of NP?

Existing work:

- embedded model theory [Libkin and others]
- meta-finite model theory [Grädel, Gurevich]

Embedded Model Theory

study of finite structures whose domain is drawn from some infinite structure

$\exists x$: x ranges over U (“natural”)

$\exists x \in adom$: x ranges over A (“active domain”)

- Motivated by database problems:
  - real databases & query languages have numbers
  - constraint databases

Much of the work in the area involves finding conditions under which questions about embedded finite models can be reduced to questions about normal finite models.

Which of those are useful for us?

Embedded Finite Models Results Useful?

Results about generic queries are not very useful – our axiomatizations are rarely generic

Results about natural-active collapse and o-minimality will likely to be useful for expressiveness and grounding

- Don’t address immediate needs
  - Need a logical formalization of FO-MX with arithmetic
  - still need capturing NP!
  - *plus* a practical mechanism for constructing solutions

We often have 2 (or more) sorts, usually non-numeric and numeric, with functions from one to the other

$\Rightarrow$ meta-finite model theory seems more suitable

Meta-Finite Structires [Grädel and Gurevich]

two-sorted structure $\mathcal{D} = (\mathcal{A}, \mathcal{R}, \mathcal{W})$, where

$\mathcal{A}$ is a finite primary structure,

$\mathcal{R}$ is the secondary structure; and

$\mathcal{W}$ is a set of weight functions from A^k to R

Typically, $\mathcal{R}$ is fixed & infinite, e.g. $\mathbb{N}$ with arithmetic.

Quantification is over the primary domain only

For a capturing NP result:

Restrict structures to those with “small weights”

i.e., if $w(\bar{a}) = s$ then $|s| = poly(|\mathcal{A}|)$

Meta-Finite Structures [Grädel and Gurevich]

Arithmetic structures $\mathcal{R}$: contain at least 0, 1, +, $\times$, <, multiset operations max, min, Σ (sum), Π (product). All functions, relations, and multiset operations of $\mathcal{R}$ are such that they are evaluated in polytime.

Theorem [Grädel and Gurevich]: Let $\mathcal{K}$ be a class of structures with small weights which is closed under isomorphisms. The following are equivalent:

(i) $\mathcal{K}$ is in NP

(ii) $\mathcal{K}$ is a primary generalized spectrum
(i.e., only the primary part is expanded, not the weight functions)

Our Needs

- Need to provide the user with a natural way to axiomatize problems, at least as in the existing languages
  - quantification over $\mathcal{R}$ (secondary domain) is essential
  - need “mixed” predicates with arguments ranging over both primary and secondary domains
  - want arithmetic functions/relations in the expansion vocabulary — solutions may contain numbers!
- Need a capturing NP property
- Need to be able to solve by grounding (=propositionalization)

What is in the Expansion Vocabulary?

We'd like weight functions and mixed predicates . . .

Problem: On arithmetic structures, unrestricted metafinite spectra capture the r.e. sets [Grädel and Gurevich]

- Require all expansion predicates (∃SO variables) to have "upper guards", e.g.

$$\forall x_1 \forall x_1 \forall x_1 \; (E(x_1, x_2, x_3) \supset UG_1(x_1, x_2) \wedge UG_2(x_3))$$

UG_i either come from the instance structure or are polytime constructable

Quantification over Secondary Domain

- "No quantification over the secondary domain" is too restrictive

- Require all quantification over the secondary domain to be guarded in the sense of k-guarded fragment GF_k [Gottlob et al]

- All upper & lower guards must be either given by the instance structure or poly-time constructable
(we use stratifiable inductive definitions with some restrictions)

Guards and Grounding

We think upper and lower guards are a good logical formalization of type system in existing constraint modelling languages.

Need to understand the expressiveness of the corresponding logic

Guards are essential for efficient grounding [PLTG:ijcai'07]

This work: we developed a grounding algorithm with arithmetic, where guards play an important role

Fixpoint Logics for Declarative Progr.

FO(ID) is the extension of FO with inductive definitions [Denecker/T:04,07]

A rule-based syntax with arbitrary FO formulas in the bodies

under the 2-valued well-founded sematics (the third truth value makes the entire axiomatization a contradiction)

Why not FO(LFP), FO(PFP), FO(IFP)?

FO(LFP) is “safe”, but we need negations

FO(PFP) is too expressive (we are mostly interested in NP)

FO(ID) vs FO(IFP)

Well-founded semantics of FO(ID) correctly represents definitions over well-founded order, and iterative inductive definitions [Denecker/T:04, DT:07:TOCL]

(occur frequently in mathematical definitions, and in common-sense reasoning [DT:07:AIJ])

Inflationary semantics of FO(IFP) correctly represent some examples [Grädel, Kreutzer]

but fails to provide the expected meaning to simple definitions over WF order

$$\left\{ \begin{array}{l} \forall x\ (E(x) \leftarrow x = 0), \\ \forall x\ (E(s(x)) \leftarrow \neg E(x)) \end{array} \right\}$$

Future Work

Needed: a model theory of
constraint modelling languages and
declarative programming of search problems,
with a uniform logical formalization and techniques for analysis of these languages

Four important application areas:

1) Design and Expressiveness Analysis of Languages:

- New languages an active area – but ad-hoc
- Languages becoming more complex (feature-rich)
- Analysis, comparison, etc., hard
- Need built-in arithmetic, strings etc.

Future Work (Cont.)

2) Pre-Processing (optimization):

- Recognition of special cases/best solver to use

- "Re-writing" and syntax restrictions to improve solver performance:

  - add redundant axioms that help solver
  - add axioms eliminating symmetries
  - change constraints used etc.

  (Very active, practically important area)

3) Debuggers and Verifiers

Future Work (Cont.)

4) Grounding (propositionalization):

- Best current technology for many problems:
  Ground then run a ground solver (SAT, PB, SMT, etc).

- Need partial or complete quantifier elimination depending on the target solver

- Need more sophisticated techniques and re-writing
  e.g. in k-guarded form

- Different techniques for different secondary structures