A Semantic Account for Modularity in Multi-language Modelling of Search Problems

S. Tasharrofi and E. Ternovska

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

Abstract. Motivated by the need to combine systems and logics, we develop a modular approach to the model expansion (MX) problem, a task which is common in applications such as planning, scheduling, computational biology, formal verification. We develop a modular framework where parts of a modular system can be written in different languages. We start our development from a previous work, [14], but modify and extend that framework significantly. In particular, we use a model-theoretic setting and introduce a feedback (loop) operator on modules. We study the expressive power of our framework and demonstrate that adding the feedback operator increases the expressive power considerably. We prove that, even with individual modules being polytime solvable, the framework is expressive enough to capture all of NP, a property which does not hold without loop. Moreover, we demonstrate that, using monotonicity and anti-monotonicity of modules, one can significantly reduce the search space of a solution to a modular system.

1 Introduction

Formulating AI tasks as model finding has recently become very promising due to the overwhelming success of SAT solvers and related technology such as SMT. In our research direction we focus on a particular kind of model finding which we call *model expansion*. The task of model expansion underlies all search problems where for an instance of a problem, which we represent as a logical structure, one needs to find a certificate (solution) satisfying certain specification. For example, given a graph, we are looking for its 3-colouring in a classic NP-search problem. Such search problems occur broadly in applications; they include planning, scheduling, problems in formal verification (where we are looking for a path to a bug), computational biology, and so on. In addition to being quite common, the task of model expansion is generally simpler than satisfiability from the computational point of view. Indeed, for a given logic $\mathcal{L}$, we have, in terms of computational complexity,

$$\text{MC}(\mathcal{L}) \leq \text{MX}(\mathcal{L}) \leq \text{Satisfiability}(\mathcal{L}),$$

where MC($\mathcal{L}$) stands for model checking (structure for the entire vocabulary of the formula in logic $\mathcal{L}$ is given), MX($\mathcal{L}$) stands for model expansion (structure interpreting a part of the vocabulary is given) and Satisfiability($\mathcal{L}$) stands for satisfiability task (where we are looking for a structure satisfying the formula). A comparison of the complexity

of the three tasks for several logics of practical interest is given in [15]. Satisfiability problem has been studied for many logics of practical interest, however model expansion problem has not been studied. In particular, issues related to combining specialized formalisms have been investigated, to a large degree, for satisfiability problem but not at all for model expansion. As we develop our framework, we aim at understanding the expressive power of the operations we add. Knowing the expressiveness of a framework is essential in particular to understanding the complexity of solution finding.

Our contributions are as follows:

- We develop a semantics-based formalism which abstractly represents combinations of modules. Our model-theoretic view allows us to study modular systems independently from the logical languages in which each module is axiomatized.
- In [14], the authors define a modular constraint-based framework where different modelling languages such as ASP, CP and SAT can be combined. We considerably extend their work mostly due to to the introduction of loops and results that follow.
- Unlike [14], we represent modules as sets of structures, each such set corresponding to a model expansion task solved by a module. This model-theoretic view is essential (1) to study the expressiveness of the framework itself, and its expressiveness as a function of the expressiveness of the languages of individual modules; (2) to connect to descriptive complexity (capturing complexity classes). In both cases, the constraint-based approach [14] is not suitable – one needs to talk about formulas being true in a structure, thus the model-theoretic view.
- We formulate an algebra on our modules (module expansion tasks). Several algebraic operations have already been used in [14], although in a constraint setting. An essential contribution here is the addition of a loop (or feedback) operator. Loops are present in all non-trivial computer programs and systems, including those consisting of multiple modules. In all the results in this paper, the loop operator is essential.
- We then investigate the expressive power of modular systems. We set apart and study the expressive power which is added purely by the algebraic operations. Among the operations, the loop operator is the most interesting. Adding loops gives a jump from P to NP: we prove that NP is captured even with all modules being polytime, due to the loop operator. In fact, adding it gives a jump in the polynomial time hierarchy. The operators introduced in [14] do not add additional expressive power, while the loop operator does.
- A crucial question is how to compute solution to the modular system under the assumption that we can compute solutions of individual modules. We begin our investigation of this question. We study some cases where solution to a modular system can be approximated in polynomial time by relying on the construction used in the well-founded semantics of logic programs.
- In many cases, we can view modules as operators. We consider monotonicity and antimonotonocity properties of modules viewed as operators. These are important properties because they allow us to derive some knowledge about solutions to the entire modular system.

2 Background: Model Expansion Task

In [17], the authors formalize combinatorial search problems as the task of *model expansion (MX)*, the logical task of expanding a given (mathematical) structure with new relations. Formally, the user axiomatizes their problem in some logic $\mathcal{L}$. This axiomatization relates an instance of the problem (a *finite structure*, i.e., a universe together with some relations and functions), and its solutions (certain *expansions* of that structure with new relations or functions). Logic $\mathcal{L}$ corresponds to a specification/modelling language. It could be an extension of first-order logic such as FO(ID), or an ASP language, or a modelling language from the CP community such as ESSENCE [12]. MX task underlies many practical approaches to declarative problem solving, which motivates us to investigate modularity in the context of the MX task.

Recall that a vocabulary is a set of non-logical (predicate and function) symbols. An interpretation for a vocabulary is provided by a *structure*, which consists of a set, called the domain or universe and denoted by $dom(.)$, together with a collection of relations and (total) functions over the universe. A structure can be viewed as an *assignment* to the elements of the vocabulary. An expansion of a structure $\mathcal{A}$ is a structure $\mathcal{B}$ with the same universe, and which has all the relations and functions of $\mathcal{A}$, plus some additional relations or functions. The task of model expansion for an arbitrary logic $\mathcal{L}$ (abbreviated $\mathcal{L}$-MX), is:

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

We call σ, the vocabulary of $\mathcal{A}$, the *instance* vocabulary, and $\varepsilon := vocab(\phi) \setminus \sigma$ the *expansion* vocabulary[1].

Example 1. The following formula ϕ of first order logic constitutes an MX specification for Graph 3-colouring:

$$\begin{gathered}\forall x\,[(R(x) \vee B(x) \vee G(x)) \\ \wedge\neg((R(x) \wedge B(x)) \vee (R(x) \wedge G(x)) \vee (B(x) \wedge G(x)))] \\ \wedge\, \forall x \forall y\,[E(x,y) \supset (\neg(R(x) \wedge R(y)) \\ \wedge\neg(B(x) \wedge B(y)) \wedge \neg(G(x) \wedge G(y)))].\end{gathered}$$

An instance is a structure for vocabulary $\sigma = \{E\}$, i.e., a graph $\mathcal{A} = \mathcal{G} = (V; E)$. The task is to find an interpretation for the symbols of the expansion vocabulary $\varepsilon = \{R, B, G\}$ such that the expansion of $\mathcal{A}$ with these is a model of ϕ:

$$\underbrace{(\overbrace{V; E^{\mathcal{A}}}^{\mathcal{A}},\ R^{\mathcal{B}}, B^{\mathcal{B}}, G^{\mathcal{B}})}_{\mathcal{B}} \models \phi.$$

The interpretations of ε, for structures $\mathcal{B}$ that satisfy ϕ, are exactly the proper 3-colourings of $\mathcal{G}$.

Given a specification, we can talk about a set (class) of $\sigma \cup \varepsilon$-structures which satisfy the specification. Alternatively, we can simply talk about a set (class) of $\sigma \cup \varepsilon$-structures as an MX-task, without mentioning a particular specification the structures satisfy.

[1] By ":=" we mean "is by definition" or "denotes".

3 Modular Systems

Definition 1 (Module). *A* module M *is an MX task, i.e., a set (class) of* $\sigma \cup \varepsilon$*-structures.*

Characterizing a module using a set of structures does not assume anything about how it is specified, which makes our study language-independent. A modular system is formally described as a set of primitive modules (individual MX tasks) combined using the operations of: (1) Projection($\pi_\tau(M)$) which restricts the vocabulary of a module, (2) Composition($M_1 \rhd M_2$) which connects outputs of M_1 to inputs of M_2, (3) Union($M_1 \cup M_2$) and (4) Feedback($M[R = S]$). Operations (1)-(3) were introduced in [14], in a constraint setting. Here, we use a model-theoretic setting. The feedback operation is new here, and it is essential since all non-trivial systems use loops. Moreover, adding this operation increases the expressive power of modular systems.

Definition 2 (Composable, Independent [14]). *Modules* M_1 *and* M_2 *are* composable *if* $\varepsilon_{M_1} \cap \varepsilon_{M_2} = \emptyset$ *(no output interference). Module* M_1 *is* independent *from* M_2 *if* $\sigma_{M_1} \cap \varepsilon_{M_2} = \emptyset$ *(no cyclic module dependencies).*

Definition 3 (Modular Systems). *Modular systems are built inductively from constraint modules using projection, composition, union and feedback operators:*
1. A module is a modular system.
2. For modular system M *and* $\tau \subseteq \sigma_M \cup \varepsilon_M$*, modular system* $\pi_\tau(M)$ *is defined such that (a)* $\sigma_{\pi_\tau(M)} = \sigma_M \cap \tau$*, (b)* $\varepsilon_{\pi_\tau(M)} = \varepsilon_M \cap \tau$*, and (c)* $\mathcal{B} \in \pi_\tau(M)$ *iff there is a structure* $\mathcal{B}' \in M$ *with* $\mathcal{B}'|_\tau = \mathcal{B}$*.*
3. For composable modular systems M *and* M' *(no output interference) with* M *independent from* M' *(no cyclic module dependencies),* $M \rhd M'$ *is a modular system such that (a)* $\sigma_{M \rhd M'} = \sigma_M \cup (\sigma_{M'} \setminus \varepsilon_M)$*, (b)* $\varepsilon_{M \rhd M'} = \varepsilon_M \cup \varepsilon_{M'}$*, and (c)* $\mathcal{B} \in (M \rhd M')$ *iff* $\mathcal{B}|_{vocab(M)} \in M$ *and* $\mathcal{B}|_{vocab(M')} \in M'$*.*
4. For modular systems M_1 *and* M_2 *with* $\sigma_{M_1} \cap \sigma_{M_2} = \sigma_{M_1} \cap \varepsilon_{M_2} = \varepsilon_{M_1} \cap \sigma_{M_2} = \emptyset$*, the expression* $M_1 \cup M_2$ *defines a modular system such that (a)* $\sigma_{M_1 \cup M_2} = \sigma_{M_1} \cup \sigma_{M_2}$*, (b)* $\varepsilon_{M_1 \cup M_2} = \varepsilon_{M_1} \cup \varepsilon_{M_2}$*, and (c)* $\mathcal{B} \in (M_1 \cup M_2)$ *iff* $\mathcal{B}|_{vocab(M_1)} \in M_1$ *or* $\mathcal{B}|_{vocab(M_2)} \in M_2$*.*
5. For modular system M *and* $R \in \sigma_M$ *and* $S \in \varepsilon_M$ *being two symbols of similar type (i.e., either both function symbols or both predicate symbols) and of the same arities; expression* $M[R = S]$ *is a modular system such that (a)* $\sigma_{M[R=S]} = \sigma_M \setminus \{R\}$*, (b)* $\varepsilon_{M[R=S]} = \varepsilon_M \cup \{R\}$*, and (c)* $\mathcal{B} \in M[R = S]$ *iff* $\mathcal{B} \in M$ *and* $R^{\mathcal{B}} = S^{\mathcal{B}}$*.*

Further operators for combining modules can be defined as combinations of basic operators above. For instance, [14] introduced $M_1 \blacktriangleright M_2$ (composition with projection operator) as $\pi_{\sigma_{M_1} \cup \varepsilon_{M_2}}(M_1 \rhd M_2)$. Also, $M_1 \cap M_2$ is defined to be equivalent to $M_1 \rhd M_2$ (or $M_2 \rhd M_1$) when $\sigma_{M_1} \cap \varepsilon_{M_2} = \sigma_{M_2} \cap \varepsilon_{M_1} = \varepsilon_{M_1} \cap \varepsilon_{M_2} = \emptyset$. Here is an example of a modular system M combined from modules M_1, M_2, M_3, M_4 and M_5:

$$M := \pi_{E,H_1}\left[([M_1 \rhd (M_2 \cap M_3)] \rhd M_4)\,[H_1 = H_5] \rhd M_5\right].$$

One can look at M as an algebraic formula where, for example, sub-formulas $M_1 \rhd (M_2 \cap M_3)$ and M_2 both represent modules that appear in M. We call modules M_1, M_2, M_3, M_4 and M_5 *primitive in* M because they do not contain any operations.

Proposition 1. *A modular system constructed using composition, projection, feedback and union is a module.*

Proposition 2 (Law of Substitution). *Let M_1 be a modular system, M' an arbitrary (not necessarily primitive) module that appears in M_1, and let M'' be a modular system such that $M' = M''$ (equality of two sets (classes) of structures). If we replace M' in M_1 by M'', then for the resulting compound system M_2, we have $M_1 = M_2$.*

Definition 4 (Models, Solutions). *For a modular system M, a σ_M-structure $\mathcal{A}$ and a $(\sigma_M \cup \varepsilon_M)$-structure $\mathcal{B}$, we say $\mathcal{B}$* is a model of *M if $\mathcal{B} \in M$. We also say $\mathcal{B}$* is a solution to *$\mathcal{A}$* in *M if $\mathcal{B} \in M$ and $\mathcal{B}$ expands $\mathcal{A}$.*

Comparison with [14] The framework [14] is based on a set of variables $\mathcal{X}$ each $x \in \mathcal{X}$ having a domain $D(x)$. An *assignment* over a subset of variables $X \subseteq \mathcal{X}$ is a function $\mathcal{B} : X \to \cup_{x \in X} D(x)$, which maps variables in X to values in their domains. A constraint C over a set of variables X is characterized by a set C of assignments over X, called the *satisfying assignments*. The variables of [14] are represented here by the elements of the combined $\sigma \cup \varepsilon$ vocabulary. Assignments are structures here, and we use symbols of various vocabularies instead of variables. It is essential to reformulate this notion using structures since we want to go back and forth between modules and logics in which modules are represented. In addition, we streamline most definitions. We eliminated input and output vocabularies, which can be defined, if needed, as subsets of instance and expansion vocabularies using projections. The main difference is the addition of loops, which are essential in all the results here.

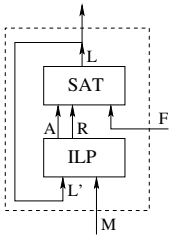

Fig. 1. Modular System Representing an SMT Solver for the Theory of Integer Linear Arithmetic

Example 2 (SMT Solvers). Consider Figure 1: It shows two boxes with solid lines which correspond to primitive MX modules and a box with dotted borders which depicts our module of interest. The vocabulary here consists of all symbols A, R, L, L', M and F where symbols A, R and L' are internal to the module, while others form the module's interface. Also, there is a line connecting L to L' which depicts a feedback.

Overall, this modular system describes a simple SMT solver for the theory of Integer Linear Arithmetic (T_{ILA}). Our two MX modules are SAT and ILP. They work on different parts of a specification. The ILP module takes a set L' of literals and a mapping M from atoms to linear arithmetic formulas. It returns two sets R and A. Semantically, R represents a set of subsets of L' so that $T_{ILA} \cup M|_r$ is *unsatisfiable* for all subsets $r \in R$. Set A represents a set of propagated literals together with their justifications, i.e., a set of pairs (l, Q) where l is an unassigned literal (i.e., neither $l \in L'$ nor $\neg l \in L'$) and Q is a set of assigned literals asserting $l \in L'$, i.e., $Q \subseteq L'$ and $T_{ILA} \cup M|_Q \models M|_l$

(the ILA formula $M|_l$ is a logical consequence of ILA formulas $M|_Q$). The SAT module takes R and A and a propositional formula F and returns set L of literals such that: (1) L makes F true, (2) L is not a superset of any $r \in R$ and, (3) L respects all propagations (l, Q) in A, i.e., if $Q \subseteq L$ then $l \in L$. Using these modules and our operators, module SMT is defined as below to represent our simple SMT solver:

$$SMT := \pi_{\{F,M,L\}}((ILP \rhd SAT)[L = L']). \tag{1}$$

The combined module SMT is correct because, semantically, L satisfies F and all models in it should have $R = \emptyset$, i.e., $T_{ILA} \cup M|_L$ is satisfiable. This is because ILP contains structures for which if $r \in R$, then $r \subseteq L' = L$. Also, for structures in SAT, if $r \in R$ then $r \not\subseteq L$. Thus, to satisfy both these conditions, R has to be empty. Also, one can easily see that all sets L which satisfy F and make $T_{ILA} \cup M|_L$ satisfiable are solutions to this modular system (set $A = R = \emptyset$ and $L' = L$). So, there is a one-to-one correspondence between models of the modular system above and SMT's solutions to the propositional part.

Example 3 (Hamiltonian Path). In this example, we describe the Hamiltonian path problem through a combination of basic modules M_1, M_2, M_3, M_4 and M_5. We start by an informal description of each of these modules.

Module M_1 takes binary relations E and H_1 and outputs their intersection H_2. Modules M_2 and M_3 take H_2 as their input and, respectively, output binary relations H_3 and H_4 so that (1) they both are subsets of H_2, (2) all tuples in H_3 are unique with respect to the element in their first positions, and (3) all tuples in H_4 are unique with respect to the element in their second positions. Next, relations H_3 and H_4 are passed to module M_4 which outputs their intersection as the binary relation H_5. Now, H_5 is fed back into module M_1 to create a loop. Finally, M_5 takes H_5 and accepts it iff undirected transitive closure of H^5 is V^2 (V being the domain). More formally, modular system M for Hamiltonian path problem is defined as:

$$M := \pi_{E,H_1}\left[([M_1 \rhd (M_2 \cap M_3)] \rhd M_4)\,[H_1 = H_5] \rhd M_5\right].$$

Using our definitions for operations on modules, we have that $\sigma_M = \{E\}$ and $\varepsilon_M = \{H_1\}$. We claim that model expansion task for M finds a Hamiltonian path in a graph: let $\mathcal{A} = (V; E^{\mathcal{A}})$ be a graph and $\mathcal{B} = (V; E^{\mathcal{A}}, H_1^{\mathcal{B}})$ be any expansion of $\mathcal{A}$ to $\sigma_M \cup \varepsilon_M$. We know that $\mathcal{B} \in M$ iff there is expansion $\mathcal{B}'$ of $\mathcal{B}$ to $\{E, H_1, H_2, H_3, H_4, H_5\}$ such that $\mathcal{B}'|_{\{E,H_1,H_2\}} \in M_1$, $\mathcal{B}'|_{\{H_2,H_3\}} \in M_2$, $\mathcal{B}'|_{\{H_2,H_4\}} \in M_3$, $\mathcal{B}'|_{\{H_3,H_4,H_5\}} \in M_4$, $\mathcal{B}'|_{\{H_5\}} \in M_5$ and $H_1^{\mathcal{B}'} = H_5^{\mathcal{B}'}$.

Module M describes the Hamiltonian path problem because, first, any model $\mathcal{B}'$ as above has to give common interpretations to all relations H_1 to H_5. This is because $H_5 \subseteq H_3 \cap H_4 \subseteq H_2 \subseteq H_1 \subseteq H_5$. So, the common interpretation $\mathcal{R}$ to these symbols should be (1) the graph of a partial function (by definition of M_2), (2) one-to-one (by definition of M_3), and, (3) a subset of the edges (by definition of M_1). So, $\mathcal{R}$ is a collection of vertex disjoint paths and cycles in the input graph $\mathcal{A}$. Thus, as M_5 asserts that all vertices should be reachable to each other via $\mathcal{R}$, then $\mathcal{R}$ is either a cycle or a simple path passing all vertices, i.e., either a Hamiltonian cycle or a Hamiltonian path.

4 Expressive Power

The authors of [17] emphasized the importance of *capturing NP* and other complexity classes. The capturing property, say for NP, is of fundamental importance as it shows that, for a given language:

(a) *we can express all of NP* – which gives the user an assurance of universality of the language for the given complexity class,

(b) *no more than NP can be expressed* – thus solving can be achieved by means of constructing a universal polytime reduction (called *grounding*) to an NP-complete problem such as SAT or CSP.

In the context of modular systems, we also want to investigate the expressive power of the defined language. This section defines model-theoretic properties that a module may satisfy such as totality, determinacy, polytime chability/solvability, monotonicity, anti-monotonicity. We then capture NP in a modular setting with modules satisfying some of those properties. While the focus of this result is on NP, by no means is the expressive power of the modular framework limited to NP.

Definition 5 (Extension). *For τ-structures $\mathcal{A}$ and $\mathcal{A}'$, we say $\mathcal{A}'$* extends *$\mathcal{A}$, and write $\mathcal{A} \sqsubseteq \mathcal{A}'$, if we have: (a) $dom(\mathcal{A}) = dom(\mathcal{A}')$, (b) for predicate symbol $R \in \tau$ we have $R^{\mathcal{A}} \subseteq R^{\mathcal{A}'}$.*

We sometimes abuse the notation and, for interpretations S_1 and S_2 of symbol S in two structures with the same domain, write (1) $S_1 \sqsubseteq S_2$ to say S_2 extends S_1, (2) $S_1 \sqcap S_2$ (resp. $S_1 \sqcup S_2$) to denote $S_1 \cap S_2$ (resp. $S_1 \cup S_2$) for predicate symbol S. We also use $\sqsubset$ and $\sqsupset$ to denote proper extension, i.e., similar to $\sqsubseteq$ and $\sqsupseteq$ but without equality.

Modular Systems as Operators

Definition 6 (Total Modular Systems). *For modular system M and vocabulary τ, we say M* is τ-total w.r.t. C *(C being a class of structures) if all τ-structures in C are τ-restrictions of some structure in M.*

Our definition of totality is conceptually similar to [14] but more general because, here, τ is not neccessarily a subset of σ (instance vocabulary). We might omit writing C in Definition 6 either if it is obvious from the context or if it is not important, i.e., discussion holds for all classes of structures.

Definition 7 (Deterministic Modular Systems). *For modular system M and sets of symbols τ and τ', we say M* is τ-τ'-deterministic *if for all structures $\mathcal{B}$ and $\mathcal{B}'$ in M, we have if $\mathcal{B}|_\tau = \mathcal{B}'|_\tau$ then $\mathcal{B}|_{\tau'} = \mathcal{B}'|_{\tau'}$.*

A module M that is both τ-τ'-deterministic and τ-total can be viewed as a *mapping* from τ-structures to τ'-structures, i.e., for all τ-structures $\mathcal{A}$, there is a unique τ'-structure $\mathcal{A}'$ so that for all structures $\mathcal{B} \in M$ if $\mathcal{B}|_\tau = \mathcal{A}$ then $\mathcal{B}|_{\tau'} = \mathcal{A}'$. Note that existence and uniqueness of $\mathcal{A}'$ are guaranteed by τ-totality and τ-τ'-determinacy of M. For such M, we write $M_{\tau,\tau'}(\mathcal{A})$ to denote the unique structure $\mathcal{A}'$ that M associates to $\mathcal{A}$. We might omit τ and τ' and just write $M(\mathcal{A})$ if they are clear from the context.

Proposition 3. *For τ-τ'-deterministic modular system M:*
1. *If $\tau'' \supseteq \tau$, then M is also τ''-τ'-deterministic.*
2. *If $\tau'' \subseteq \tau'$, then M is also τ-τ''-deterministic.*

Proposition 4. *If M is both τ_1-τ_2-deterministic and τ_1'-τ_2'-deterministic, M is also $(\tau_1 \cup \tau_1')$-$(\tau_2 \cup \tau_2')$-deterministic.*

Definition 8 (Monotonicity and Anti-Monotonicity). *For modular system M and sets of symbols τ_1, τ_2 and τ_3, we say M* is τ_1-τ_2-τ_3-monotone *(resp. τ_1-τ_2-τ_3-anti-monotone) if for all structures $\mathcal{B}$ and $\mathcal{B}'$ in M, we have:*

$$\begin{array}{c} \textit{if } \mathcal{B}|_{\tau_1} \sqsubseteq \mathcal{B}'|_{\tau_1} \textit{ and } \mathcal{B}|_{\tau_2} = \mathcal{B}'|_{\tau_2} \textit{ then} \\ \mathcal{B}|_{\tau_3} \sqsubseteq \mathcal{B}'|_{\tau_3} (\textit{resp. } \mathcal{B}'|_{\tau_3} \sqsubseteq \mathcal{B}|_{\tau_3}). \end{array}$$

Proposition 5. *Let M be a τ_1-τ_2-τ_3-monotone or a τ_1-τ_2-τ_3-anti-monotone module. Then M is $(\tau_1 \cup \tau_2)$-τ_3-deterministic.*

Proof. We prove this for the monotone case. The other case is similar. Let $\mathcal{B}, \mathcal{B}' \in M$ be such that $\mathcal{B}|_{\tau_1 \cup \tau_2} = \mathcal{B}'|_{\tau_1 \cup \tau_2}$. Then, (1) $\mathcal{B}|_{\tau_2} = \mathcal{B}'|_{\tau_2}$, (2) $\mathcal{B}'|_{\tau_1} \sqsubseteq \mathcal{B}|_{\tau_1}$, and, (3) $\mathcal{B}|_{\tau_1} \sqsubseteq \mathcal{B}'|_{\tau_1}$. Thus, by (1) and (2), we know $\mathcal{B}'|_{\tau_3} \sqsubseteq \mathcal{B}|_{\tau_3}$ and, by (1) and (3), we have $\mathcal{B}|_{\tau_3} \sqsubseteq \mathcal{B}'|_{\tau_3}$. Thus, $\mathcal{B}|_{\tau_3} = \mathcal{B}'|_{\tau_3}$.

Expressive Power We introduced several properties that a modular system may have, i.e., totality, determinacy, monotonicity and anti-monotonicity. We also proved that determinacy is a consequence of monotonicity or anti-monotonicity. Hence, it may look like the systems composed of only total (anti-)monotone modules are of very restricted computational power. However, as Theorem 1 shows, due to the presence of loops (feedbacks), the modular framework expresses all of NP although all individual modules are polytime solvable. One can extend Theorem 1 to prove that the feedback operator causes a jump from one level of the polynomial hierarchy to the next, i.e., with modules from Δ_k^P (level k of the polynomial hiearchy), and in the presence of feedbacks, modular framework expresses all of Σ_{k+1}^P.

Definition 9 (Polytime Checkability, Polytime Solvability). *Let M be a module with instance vocabulary σ and expansion vocabulary ε.* M is polytime checkable *if there is a polytime program V which, given a $(\sigma \cup \varepsilon)$-structure $\mathcal{B}$, accepts $\mathcal{B}$ if and only if $\mathcal{B} \in M$. Also,* M is polytime solvable *if there is a partial function F computable in polytime such that for all structures $\mathcal{A}$: (1) $F(\mathcal{A})$ is defined if and only if there is structure $\mathcal{B} \in M$ expanding $\mathcal{A}$, and (2) if $F(\mathcal{A})$ is defined then $F(\mathcal{A}) \in M$ and $F(\mathcal{A})$ is the only structure in M which expands $\mathcal{A}$.*

Note that polytime solvability implies determinism. In theoretical computing science, a problem is a subset of $\{0,1\}^*$. However, in descriptive complexity, the equivalent definition of a problem being a set of structures is adopted. The following theorem gives a capturing result for NP:

Theorem 1 (Capturing NP over Finite Structures). *Let $\mathcal{K}$ be a problem over the class of finite structures closed under isomorphism. Then, the following are equivalent:*

1. *$\mathcal{K}$ is in NP,*
2. *$\mathcal{K}$ is the models of a modular system where all primitive modules M are σ_M-ε_M-deterministic, σ_M-total, σ_M-$vocab(\mathcal{K})$-ε_M-anti-monotone, and polytime solvable,*
3. *$\mathcal{K}$ is the models of a modular system with polytime checkable primitive modules.*

Proof. (1) $\Rightarrow$ (2): To prove this direction, we give a modular system M' which contains only one primitive module M. Primitive module M given in the proof satisfies all conditions of totality, determinacy, anti-monotonicity and polytime solvability as required by the theorem statement. Module M' feeds M's output to part of its input and projects out some auxiliary vocabulary required by M.

The proof in this direction follows the fact that, when allowing auxiliary vocabulary, ASP programs can express first order sentences (via Lloyd-Topor transformation). Thus, as FO MX captures NP over the class of finite structures, so do ASP programs (modulo the auxiliary vocabulary).

Now, consider a problem $\mathcal{K}$ in NP with vocabulary σ, i.e., an isomorphism-closed set of finite σ-structures. By the above argument, there is an ASP program P with instance vocabulary σ and expansion vocabulary ε_P which (when restricted to σ) accepts exactly those structures in $\mathcal{K}$. We now introduce a module M with $\sigma_M := \sigma \cup \varepsilon_P$ and $\varepsilon_M := \varepsilon'_P$ (where ε'_P consists of new predicate symbols R' for each predicate symbol $R \in \varepsilon_P$). Given an instance structure $\mathcal{A}$, module M works by first computing the ground program P' of P w.r.t. $dom(\mathcal{A})$. Then, M computes the reduct of P' under $\mathcal{A}$, denoted as $P'^{\mathcal{A}}$. Finally, M takes the deductive closure of $P'^{\mathcal{A}}$ and gives it as output.

Obviously, M is σ_M-ε_M-deterministic, σ_M-total and polytime computable. Also, M is σ_M-σ-ε_M-anti-monotone because, for a fixed interpretation to σ_P, an increment in ε_P makes $P'^{\mathcal{A}}$, and thus the deductive closure, smaller. Now, we define module $M' := \pi_\sigma(M[\varepsilon_P = \varepsilon'_P])$. Observe that models of M' are exactly those accepted by P.

(2) $\Rightarrow$ (3): This direction is trivial because if a modular system uses only polytime solvable primitive modules then it also uses only polytime checkable primitive modules.

(3) $\Rightarrow$ (1): Let M be a modular system whose models coincide with $\mathcal{K}$ and whose primitive modules are polytime checkable. Then, $\mathcal{K}$ is in NP because one can nondeterministically guess all the interpretations of expansion symbols of M (the set of these symbols is equal to the union of the expansion vocabularies of all M's primitive modules) and then use polytime checkability of M's primitive modules to check if this is a good guess (according to the modules, and thus according to the system itself).

Theorem 1 demonstrates the additional power that the feedback operator has brought to us. Its proof assumes that modules are described in languages with the ability to manipulate input programs and sets of atoms, and to compute fixpoints. Examples of such languages are those that capture P in the presence of ordering relation over domain elements, or the like. However, note that, in our model-theoretic view, the language that modules are described in is not important at all.

Note that Theorem 1 shows that when basic modules are restricted to polytime checkable modules, the modular system's expressive power is limited to NP. Without this restriction, the modular framework can represent Turing-complete problems. As an example, one can encode Turing machines as finite structures and have modules that accept a finite structure if and only if it corresponds to a halting Turing machine.

Theorem 1 shows that the feedback operator causes a jump in expressive power from P to NP (or, more generally, from Δ_k^P to Σ_{k+1}^P). The proof uses a translation from ASP programs to deterministic, total, anti-monotone and polytime modules. The following running example elaborates more in this direction.

Example 4 (Stable Model Semantics). Let P be a normal logic program. We know S is a stable model for P iff $S = Dcl(P^S)$ where P^S is the reduct of P under set S of atoms (a positive program) and Dcl computes the deductive closure of a positive program, i.e., the smallest set of atoms satisfying it. Now, let $M_1(S, P, Q)$ be the module that given a set of atoms S and ASP program P computes the reduct Q of P under S. Observe that M_1 is $\{S\}$-total and $\{S\}$-$\{P\}$-$\{Q\}$-anti-monotone, and polytime solvable. Also, let $M_2(Q, S')$ be a module that, given a positive logic program Q, returns the smallest set of atoms S' satisfying Q. Again, M_2 is $\{Q\}$-total, $\{Q\}$-$\{\}$-$\{S'\}$-monotone and polytime solvable. However, $M := \pi_{\{P,S\}}((M_1 \rhd M_2)[S = S'])$ is a module which, given ground ASP program P, returns all and only the stable models of P. Therefore, the NP-complete problem of finding a stable model for a normal logic program is defined by combining total, deterministic, polytime solvable, and monotone or anti-monotone modules.

Example 4 shows that the computational power of stable models is included in the modular framework. As we will see later, this phenomenon is not accidental but is a consequence of anti-monotone loops (feedbacks). Moreover, we already know that the modular framework does not impose minimality constraint on the solution to its modules (while stable model semantics does). Thus, this framework can define sets of structures that cannot be defined in ASP.

5 Approximating Solutions

Until now, we introduced modular systems and talked about their expressive power. However, an important question associated with every modeling language is how one can find a solution to a specification in such a language. While we will address this question in a future work, here, we give some results on how to intelligently reduce the space we have to search in order to find a solution. We call this space the *candidate solution space*. To do so, we start with simple properties about extending monotonicity and anti-monotonicity to complex modules. We prove that, in the presence of loops and monotone or anti-monotone primitive modules, the combined systems satisfy many interesting properties such as existence of smallest solutions or minimality of solutions. We then develop methods for intelligently reducing the candidate solution space.

Proposition 6. *Let M be a τ_1-τ_2-τ_3-monotone (resp. anti-monotone) module. Then:*

1. *If $\tau' \subseteq \tau_1$ then M is also a τ'-$(\tau_2 \cup (\tau_1 \backslash \tau'))$-$\tau_3$-monotone (resp. anti-monotone) module.*
2. *For a set ν of symbols such that $\tau_3 \cap \nu = \emptyset$, we have M is also $(\tau_1 \cup \nu)$-τ_2-τ_3-monotone (resp. anti-monotone).*
3. *For a set ν of symbols, we have that M is also τ_1-$(\tau_2 \cup \nu)$-τ_3-monotone (resp. anti-monotone).*

4. *If $\tau' \subseteq \tau_3$ then M is also a τ_1-τ_2-τ'-monotone (resp. anti-monotone) module.*

Proposition 7. *Let M be a module that is both τ_1-τ_2-τ_3-monotone and τ'_1-τ'_2-τ'_3-monotone (resp. τ_1-τ_2-τ_3-anti-monotone and τ'_1-τ'_2-τ'_3-anti-monotone) such that $(\tau_1 \cup \tau'_1) \cap (\tau_3 \cup \tau'_3) = \emptyset$. Then, M is also $(\tau_1 \cup \tau'_1)$-$(\tau_2 \cup \tau'_2)$-$(\tau_3 \cup \tau'_3)$-monotone (resp. $(\tau_1 \cap \tau'_1)$-$(\tau_2 \cup \tau'_2)$-$(\tau_3 \cup \tau'_3)$-anti-monotone).*

Proposition 8 ((Anti-)Monotonicity Preservation). *For τ_1-τ_2-τ_3-monotone (resp. anti-monotone) modular system M and general modular system M', we have:*

1. *$M \rhd M'$ is τ_1-τ_2-τ_3-monotone (resp. anti-monotone).*
2. *$M' \rhd M$ is τ_1-τ_2-τ_3-monotone (resp. anti-monotone).*
3. *If M' is ν-τ_2-deterministic for some ν, then $M' \rhd M$ is τ_1-ν-τ_3-monotone (resp. anti-monotone).*
4. *If $\tau_1 \cup \tau_2 \subseteq \nu$ then $\Pi_\nu M$ is τ_1-τ_2-$(\nu \cap \tau_3)$-monotone (resp. anti-monotone).*
5. *$M[S_1 = S_2]$ is τ_1-τ_2-τ_3-monotone (resp. anti-monotone)*

Proposition 9 (Monotonicity under Composition). *For modular systems M and M' and vocabularies τ_1, τ'_1, τ_2, τ'_2, τ_3 and τ'_3 such that $\tau'_1 \subseteq \tau_3$:*

1. *If M is τ_1-τ_2-τ_3-monotone and M' is τ'_1-τ'_2-τ'_3-monotone, $M \rhd M'$ is τ_1-$(\tau_2 \cup \tau'_2)$-τ'_3-monotone.*
2. *If M is τ_1-τ_2-τ_3-anti-monotone and M' is τ'_1-τ'_2-τ'_3-monotone, $M \rhd M'$ is τ_1-$(\tau_2 \cup \tau'_2)$-τ'_3-anti-monotone.*
3. *If M is τ_1-τ_2-τ_3-monotone and M' is τ'_1-τ'_2-τ'_3-anti-monotone, $M \rhd M'$ is τ_1-$(\tau_2 \cup \tau'_2)$-τ'_3-anti-monotone.*
4. *If M is τ_1-τ_2-τ_3-anti-monotone and M' is τ'_1-τ'_2-τ'_3-anti-monotone, $M \rhd M'$ is τ_1-$(\tau_2 \cup \tau'_2)$-τ'_3-monotone.*

Proof. We prove the first case. The rest is similar. For $P := M \rhd M'$, let $\mathcal{B}, \mathcal{B}' \in P$ be such that $\mathcal{B}|_{\tau_2 \cup \tau'_2} = \mathcal{B}'|_{\tau_2 \cup \tau'_2}$ and $\mathcal{B}|_{\tau_1} \sqsubseteq \mathcal{B}'|_{\tau_1}$. By monotonicity of M, we have $\mathcal{B}|_{\tau_3} \sqsubseteq \mathcal{B}'|_{\tau_3}$. So, as $\tau'_1 \subseteq \tau_3$, we also have $\mathcal{B}|_{\tau'_1} \sqsubseteq \mathcal{B}'|_{\tau'_1}$. Hence, by monotonicity of M', we have $\mathcal{B}|_{\tau'_3} \sqsubseteq \mathcal{B}'|_{\tau'_3}$.

These properties give us ways of deriving that a complex modular system is monotone or anti-monotone by looking at similar properties of basic constraint modules. For instance, for our two previous examples, we have:

Example 5 (Composition in Hamiltonian Path). Modules M_1, M_2, M_3 and M_4 in Example 3 are respectively $\{H_1, E\}$-$\{\}$-$\{H_2\}$-monotone, $\{H_2\}$-$\{\}$-$\{H_3\}$-monotone, $\{H_2\}$-$\{\}$-$\{H_4\}$-monotone and $\{H_3, H_4\}$-$\{\}$-$\{H_5\}$-monotone. So, by Proposition 9, $M' := M_1 \rhd (M_2 \cap M_3)$ is both $\{H_1, E\}$-$\{\}$-$\{H_3\}$-monotone and $\{H_1, E\}$-$\{\}$-$\{H_4\}$-monotone. Thus, Proposition 7 asserts M' is also $\{H_1, E\}$-$\{\}$-$\{H_3, H_4\}$-monotone. Thus, $M'' := M' \rhd M_4$ has to be $\{H_1, E\}$-$\{\}$-$\{H_5\}$-monotone (by Proposition 9).

Example 6 (Composition in ASP Programs). Modules M_1 and M_2 in Example 4 are respectively $\{S\}$-$\{P\}$-$\{Q\}$-anti-monotone and $\{Q\}$-$\{\}$-$\{S'\}$-monotone. So, by Proposition 9, $M' := M_1 \rhd M_2$ is $\{S\}$-$\{P\}$-$\{S'\}$-anti-monotone.

The rest of this section considers the important case of monotone or anti-monotone loops, i.e., monotone or anti-monotone modules under the feedback operator. Note that,

although our theorems concern modules feeding their outputs back to their inputs, these modules are usually not primitive modules, but composite modules whose monotonicity or anti-monotonicity is derived by our previous propositions.

Theorem 2 (Smallest Solution). *Let M be a $(\tau \cup \{S\})$-total and $\{S\}$-τ-$\{R\}$-monotone modular system and $M' := M[S = R]$. Then, for a fixed interpretation to τ, M' has exactly one smallest solution with respect to predicate symbol R.*

Proof. Standard Tarski proof.

Theorem 2 relates smallest solutions of monotone loops in modular systems to least fixpoints of monotone operators. Therefore, many natural problems such as transitivity or connectivity are smallest solutions of some monotone modules under feedbacks. However, Theorem 2 only states that a smallest solution exists and is unique but it does not limit the models to it. The smallest solution is used to prune the candidate solution space by discarding all candidate solutions that do not extend the smallest solution.

Proposition 10 (Anti-Monotonicity and Minimality). *For $\{S\}$-τ-$\{R\}$-anti-monotone modular system M and for modular system $M' := M[S = R]$, we have that when interpretation to τ is fixed, all models of M' are minimal with respect to the interpretations of R.*

Proof. Let $\mathcal{B}_1, \mathcal{B}_2 \in M'$ be such that $B_1|_\tau = B_2|_\tau$ and $R^{\mathcal{B}_1} \sqsubseteq R^{\mathcal{B}_2}$. So, because, in M', R is fed back to S, we have $S^{\mathcal{B}_1} \sqsubseteq S^{\mathcal{B}_2}$. Hence, by $\{S\}$-τ-$\{R\}$-anti-monotonicity of M, we have that $R^{\mathcal{B}_1} \sqsupseteq R^{\mathcal{B}_2}$. Thus, $R^{\mathcal{B}_1} = R^{\mathcal{B}_2}$ and $B_1|_{\tau \cup \{R\}} = B_2|_{\tau \cup \{R\}}$, i.e., there does not exist any two structures in M' which agree on the interpretation to τ but, in one of them, interpretation of R properly extends R's interpretation in the other one.

The minimality of solutions to anti-monotone loops means that these loops may not have a smallest solution. Nevertheless, we are still able to prune the candidate solution space by finding lower and upper bounds for all the solutions to such a loop. Consider the following process for a $(\tau \cup \{S\})$-total and $\{S\}$-τ-$\{R\}$-anti-monotone modular system M where S and R are relational symbols of arity n:

$$L_0 = \emptyset, U_0 = [dom(\mathcal{A})]^n,$$
$$L_{i+1} = R^{M(\mathcal{A} \,||\, \mathcal{U}_i)}, U_{i+1} = R^{M(\mathcal{A} \,||\, \mathcal{L}_i)},$$

where $dom(\mathcal{L}_i) = dom(\mathcal{U}_i) = dom(\mathcal{A})$, $S^{\mathcal{L}_i} = L_i$, $S^{\mathcal{U}_i} = U_i$ and, for two structures $\mathcal{A}_1$ and $\mathcal{A}_2$ over the same domain but distinct vocabularies, $\mathcal{A}_1||\mathcal{A}_2$ is defined to be the structure over the same domain as $\mathcal{A}_1$ and $\mathcal{A}_2$ and with the same interpretation as them.

Theorem 3 (Bounds on Solutions to Anti-Monotone Loops). *For $(\tau \cup \{S\})$-total and $\{S\}$-τ-$\{R\}$-anti-monotone modular system M (where $S \in \sigma_M$ and $R \in \varepsilon_M$ are symbols of arity n), and for modular system $M' := M[S = R]$ and τ-structure $\mathcal{A}$, the approximation process above has a fixpoint $(L^*_{\mathcal{A}}, U^*_{\mathcal{A}})$ such that for all $\mathcal{B} \in M'$ with $\mathcal{B}|_\tau = \mathcal{A}$, we have $L^*_{\mathcal{A}} \sqsubseteq R^{\mathcal{B}}$ and $R^{\mathcal{B}} \sqsubseteq U^*_{\mathcal{A}}$.*

Proof. We prove this for relational symbols. Extending it to function symbols is straightforward. Given τ-structure $\mathcal{A}$, consider the set $S = \{\mathcal{B} \in M' \mid \mathcal{B}|_\tau = \mathcal{A}\}$. We first prove (by induction on i) that, for all i, we have: $L_i \sqsubseteq L_{i+1}$, $U_i \sqsupseteq U_{i+1}$, $L_i \sqsubseteq \bigsqcap_{\mathcal{B}\in S} R^{\mathcal{B}}$, and $U_i \sqsupseteq \bigsqcup_{\mathcal{B}\in S} R^{\mathcal{B}}$.

The base case is easy because L_0 is the empty set and U_0 contains all possible tuples. For the inductive case:

1. By induction hypothesis, $U_i \sqsupseteq U_{i+1}$. So, by anti-monotonicity of M, we have: $L_{i+1} = L^{M(\mathcal{A} \,||\, \mathcal{U}_i)} \sqsubseteq L^{M(\mathcal{A} \,||\, \mathcal{U}_{i+1})} = L_{i+2}$. Similarly, $U_{i+1} \sqsupseteq U_{i+2}$.
2. Again, by induction hypothesis, $U_i \sqsupseteq \bigsqcup_{\mathcal{B}\in S} R^{\mathcal{B}}$. So, for all structures $\mathcal{B} \in S$, we have: $U_i \sqsupseteq R^{\mathcal{B}}$. Therefore, $L_{i+1} = L^{M(\mathcal{A} \,||\, \mathcal{U}_i)} \sqsubseteq R^{\mathcal{B}}$. Thus, $L_{i+1} \sqsubseteq \bigsqcap_{\mathcal{B}\in S} R^{\mathcal{B}}$. Similarly, we also have $U_{i+1} \sqsupseteq \bigsqcup_{\mathcal{B}\in S} R^{\mathcal{B}}$.

So, as $\bigsqcap_{\mathcal{B}\in S} R^{\mathcal{B}} \sqsubseteq \bigsqcup_{\mathcal{B}\in S} R^{\mathcal{B}}$, we have that, for all i, $L_i \sqsubseteq U_i$. Thus, there exists ordinal α where (L_α, U_α) is the fixpoint of the sequence of pairs (L_i, U_i). Denote this pair by $(L^*_{\mathcal{A}}, U^*_{\mathcal{A}})$. Observe that, by above properties, $L^*_{\mathcal{A}} \sqsubseteq R^{\mathcal{B}}$ and $R^{\mathcal{B}} \sqsubseteq U^*_{\mathcal{A}}$ for all $\mathcal{B} \in S$ (as required).

Similar to Theorem 2, Theorem 3 also prunes the search space by limiting the candidate solutions to only those that are both supersets of the lower bound obtained by the process and subsets of the upper bound obtained by it.

Example 7 (Well-Founded Models). As discussed in Example 6, the module $M' := M_1 \rhd M_2$ is $\{S\}$-$\{P\}$-$\{S'\}$-anti-monotone. Thus, by Proposition 10, the module M defined in Example 4 can only have minimal solutions with respect to symbol S for a fixed input P. Moreover, by Proposition 3, we can find lower and upper bounds to all the solutions of module M for a fixed P. Unsurprisingly, these bounds coincide with the well-founded model of the logic program P.

6 Related Work

The work that has motivated our current paper is [14]. There, the authors define a framework in which different modelling languages such as ASP, CP and SAT can be combined on equal terms. We considerably extend their work mostly due to to the introduction of loops and results that follow. Detailed comparison is in Section 3.

An early work on adding modularity to logic programs is [7]. The authors derive a semantics for modular logic programs by viewing a logic program as a generalized quantifier. One generalization considers the concept of modules in declarative programming [18]. The authors introduce the concept of modular equivalence in normal logic programs under the stable model semantics. Their work is motivated by the fact that weak equivalence in logic programs fails to give a congruence relation and strong equivalence is not fully appropriate either. They define a weaker form of equivalence which gives rise to a congruence relation which they call modular equivalence. This work, in turn, is extended to define modularity in disjunctive programs in [13]. These two works focus on bringing the concept of modular programming into the context of logic programs and dealing with difficulties that arise there. On the other hand, our work focuses on the abstract notion of a module and what can be inferred about a modular system based on what is known about modules and how they are combined. There are several

other approaches to adding modularity to ASP languages and ID-Logic as those described in [3, 1, 6]. These works also put an emphasis on extending a specific language with the modularity concept. However, in our work, we are mostly concerned with mixing several knowledge representation languages. In addition, modular programming enables ASP languages to be extended by constraints or other external relations. This view is explored in [8, 9, 20, 4, 16]. While this view is advantageous in its own right, our work is different because we use a completely model-theoretic approach. Some practical modelling languages incorporate other modelling languages. For example, X-ASP [19] and ASP-PROLOG [10] extend prolog with ASP. Also ESRA [11], ESSENCE [12] and Zinc [2] are CP languages extended with features from other languages. However, these approaches give priority to the host language while our approach gives equal weight to all modelling languages that are involved. Yet another direction is the multi-context systems. In [5], the authors introduced non-monotonic bridge rules to the contextual reasoning and originated an interesting and active line of research followed by many others for solving or inconsistency explanation in non-monotonic multi-context systems. In this field, motivation comes from distributed knowledge (such as interacting agents) or partial knowledge (where trust or privacy issues are important).

7 Conclusion and Future Work

In this paper, we presented our first steps towards developing a modular approach to solving model expansion task, a task which is very common in applications, and is generally easier than satisfiability for the same logic. We described an algebra of modular systems, which includes a new operation of feedback (loop). We have shown that the loop operation adds a significant expressive power – even when all compound modules are polytime, one can can express all (and only) problems in NP. This property does not hold without the loop operation. We have also shown that the solution space of modular systems can be reduced under a natural condition on the individual modules.

In this paper, we talked about structures in general. However, in computing science, on one hand, we are interested in only the finitely representable structures and, on the other hand, most practical problems have numeric parts without any explicit bound on how big the numbers are. Therefore, for us, another interesting direction is to focus on finitely representable structures, and on structures embedded into a background structure with an infinite domain, and to understand how the framework should be modified in this setting.

Our semantics-based formalism is the first step towards developing a logic of modular systems. The logic will have a counterpart of our algebraic operations on the syntactic level. The main goal of the logic would be to address the issue of how a modular system can be "solved" – a formula would describe the system, and its models would be abstract representations of solutions to the entire system, as a function of solutions to individual modules.

Another direction is to model the task of searching for a solution to a modular system such as SMT and similar systems, while focusing on model expansion task. We plan to develop an algorithm that finds a solution through accumulating a set of constraints that a model has to satisfy. One of the interesting consequences of this work

would be to equip that algorithm with the results obtained here so that it can search the solution space more effectively. We believe that this direction may contribute to practical solvers design for declarative modelling languages.

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

References

1. Balduccini, M.: Modules and signature declarations for a-prolog: Progress report. In: SEA. pp. 41–55 (2007)
2. de la Banda, M., Marriott, K., Rafeh, R., Wallace, M.: The modelling language zinc. Principles and Practice of Constraint Programming-CP 2006 pp. 700–705
3. Baral, C., Dzifcak, J., Takahashi, H.: Macros, macro calls and use of ensembles in modular answer set programming. In: Proc. ICLP'06, LNCS. pp. 376–390
4. Baselice, S., Bonatti, P.A., Gelfond, M.: Towards an integration of answer set and constraint solving. In: Proc. ICLP'05, LNCS. pp. 52–66
5. Brewka, G., Eiter, T.: Equilibria in heterogeneous nonmonotonic multi-context systems. In: Proc. AAAI'07. pp. 385–390
6. Denecker, M., Ternovska, E.: A logic of non-monotone inductive definitions. ToCL 9(2), 1–51 (2008)
7. Eiter, T., Gottlob, G., Veith, H.: Modular logic programming and generalized quantifiers. In: Proc. LPNMR '97. pp. 290–309. Springer-Verlag (1997)
8. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In: Proc. IJCAI'05. pp. 90–96
9. Elkabani, I., Pontelli, E., Son, T.C.: Smodels a – a system for computing answer sets of logic programs with aggregates. pp. 427–431
10. Elkhatib, O., Pontelli, E., Son, T.: Asp- prolog: A system for reasoning about answer set programs in prolog. In: Proc. PADL'04. pp. 148–162
11. Flener, P., Pearson, J., Ågren, M.: Introducing esra, a relational language for modelling combinatorial problems. Logic Based Program Synthesis and Transformation pp. 214–232 (2004)
12. Frisch, A., Harvey, W., Jefferson, C., Martínez-Hernández, B., Miguel, I.: Essence: A constraint language for specifying combinatorial problems. Constraints 13, 268–306 (2008)
13. Janhunen, T., Oikarinen, E., Tompits, H., Woltran, S.: Modularity aspects of disjunctive stable models. In: Proc. LPNMR'07. pp. 175–187. LNAI
14. Järvisalo, M., Oikarinen, E., Janhunen, T., Niemelä, I.: A module-based framework for multi-language constraint modeling. In: Proc. LPNMR'09. LNCS, vol. 5753, pp. 155–168
15. Kolokolova, A., Liu, Y., Mitchell, D., Ternovska, E.: On the complexity of model expansion. In: Proc. of LPAR-17. LNCS, vol. 6397, pp. 447–458 (2010)
16. Mellarkod, V., Gelfond, M., Zhang, Y.: Integrating answer set programming and constraint logic programming. AMAI 53(1), 251–287 (2008)
17. Mitchell, D.G., Ternovska, E.: A framework for representing and solving NP search problems. In: Proc. AAAI'05
18. Oikarinen, E., Janhunen, T.: Modular equivalence for normal logic programs. In: Proc. NMR'06. pp. 10–18
19. Swift, T., Warren, D.S.: The XSB System (2009), `http://xsb.sourceforge.net/`
20. Tari, L., Baral, C., Anwar, S.: A language for modular answer set programming: Application to acc tournament scheduling. In: Proc. ASP'05. pp. 277–292. CEUR-WS