

Modular Systems: Semantics, Complexity

Shahab Tasharrofi and Eugenia Ternovska

Abstract

Modular Systems, a formalism introduced by the authors a few years ago, is a framework that allows for combinations of modules written in multiple languages. Informally, a module represents a piece of knowledge. It can be given by a knowledge base, be an agent, an ASP, CP, ILP, CP program, etc. Information propagation happens through equal vocabulary symbols. Formally, a module is a class of structures over the same vocabulary that share a universe. Due to the model-theoretic approach, the formalism is language-independent, and combines modules specified in arbitrary languages. Modules are combined using a small number of algebraic operations.

In this paper, we further develop the framework of Modular Systems. We introduce structural operational semantics and prove that the new semantics is equivalent to the original model-theoretic semantics. We briefly explain a multi-language logic of modular systems. We study the complexity of the formalism, and the descriptive complexity of modular systems, as a function of expressiveness of individual modules.

1 Introduction

There are many high-level declarative languages for specifying search problems and very efficient solvers for solving problems in those languages. Examples include Constraint Modelling Languages (Essence, Zinc, ...) [Frisch et al., 2008; de la Banda et al., 2006] Answer Set Programming (ASP) systems, e.g. [Gebser, Kaufmann, and Schaub, 2012], IDP system [Wittocx, Marién, and Denecker, 2008], Enfragmo [Aavani et al., 2012], NP-Spec [Cadoli et al., 2000], etc. It is often better to use a particular language and solver for one part of the problem, and another logic for another.

Motivating Example: Logistics Service Provider

Example 1 *This modular system can be used by a company such as Oracle. It decides how to pack goods and deliver them. It solves two NP-complete tasks interactively, – Multiple Knapsack (module M_K) and Travelling Salesman Problem (module M_{TSP}). The system takes orders from customers*

(items $Items(i)$ to deliver, their profits $p(i)$, weights $w(i)$), and the capacity of trucks available $c(t)$, decides how to pack $Pack(i, t)$ items in trucks, and for each truck, solves a TSP problem. The feedback about solvability of TSP is sent back to M_K . Module M_{TSP} takes a candidate solution from M_K , together with the graph of cities and routes with distances, allowable distance limit and destinations for each product. The output of this module is the route, for each truck $Route(t, n, c)$, where t is a truck, n is the number in the sequence, and c is a city. The Knapsack problem is written, in, e.g. Integer Linear Programming (ILP), and TSP in Answer Set Programming (ASP). The modules M_K and M_{TSP} are composed in sequence, with a feedback going from an output of M_{TSP} to an input of M_K . A solution to the compound module, M_{LSP} , to be acceptable, must satisfy both sub-systems.

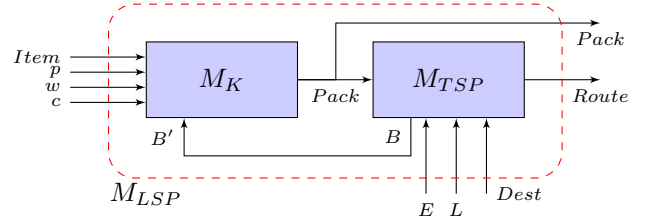


Figure 1: Logistics Service Provider ($M_K \triangleright M_{TSP}$)[$B = B'$].

Our goal is a specification language with a formal and uniform semantics for systems like this. In this paper, we continue the line of research we started in [Tasharrofi and Ternovska, 2011] where Modular Systems were introduced. They were further developed in [Tasharrofi, 2013; Wu, 2012; Tasharrofi, Wu, and Ternovska, 2012; Tasharrofi and Ternovska, 2014b; 2014a; Ensan and Ternovska, 2015]. *The main conceptual shift proposed in those works was to take model-theoretic approach seriously.* A module is a mathematical notion, a class of structures over the same all-inclusive universe. Informally, it represents a piece of knowledge. A module can be given by a knowledge base, be a specification of a robotic agent, be e.g. an ASP, CP, ILP, CP program. It can even be a human making decisions. Any decision procedure, of arbitrary complexity, would work.

Contributions After presenting model-theoretic semantics, we define a novel structural operational semantics (SOS) of MSs in the style of [Plotkin, 1981], and prove that the new se-

mantics is equivalent to the original model-theoretic semantics. We briefly explain a multi-language logic of modular systems. Our formalism gives foundations for the development of new combined logics, where each module can be axiomatized in any logic with model-theoretic semantics. We study the complexity of the formalism of MSs, and the descriptive complexity of modular systems, as a function of expressiveness of modules involved. Our results allow us to argue for *implementability* of a solver for modular systems.

Novelty In model-theoretic semantics, unlike previous work on MSs, we clearly separate syntax and semantics of the algebra (in Preliminaries), and provide more precision. We present a new view on MSs, *as a transition system specified with novel SOS*. To our knowledge, our logic of MSs is the only one that combines modules specified in *arbitrary* languages with model-theoretic semantics. The logic can be viewed as a *logic of information flow*. This feature is useful in e.g. business process modelling. The problem of model expansion is very common, but has been studied very little (see, e.g. [Kolokolova et al., 2010]) compared to the well-studied tasks of satisfiability and model checking. Our work is the only known to us study of complexity of this task for *combinations of formalisms*.

2 Preliminaries

A *vocabulary* (denoted, e.g. $\tau, \sigma, \varepsilon, \nu$) is a set of non-logical (predicate and function) symbols. A τ -structure, e.g. $\mathcal{A} = (A, R_1^A, \dots, R_n^A, f_1^A, \dots, f_m^A, c_1^A, \dots, c_l^A)$ is a domain A together with interpretations of relation, function and constants (0-ary actions) in τ . Below, we view functions as a particular kind of relations, and $\mathcal{B}|_\sigma$ mean structure $\mathcal{B}$ restricted to vocabulary σ .

Model Expansion task Given: a formula ϕ in logic $\mathcal{L}$ over vocabulary $\sigma \cup \varepsilon$, such that $\sigma \cap \varepsilon = \emptyset$ and σ -structure $\mathcal{A}$. Find: structure $\mathcal{B}$ such that $\mathcal{B}|_\sigma = \mathcal{A}$ and $\mathcal{B} \models \phi$. We call σ *instance (input)* and ε *expansion (output)* vocabularies. Such tasks are common in AI planning, scheduling, logistics, supply chain management, etc. Logic $\mathcal{L}$ corresponds to a specification/modelling language. The case $\sigma = \emptyset$ is called *model generation*. The case $\varepsilon = \emptyset$ corresponds to *model checking*, when full structure is given, and the task is to check if the formula is true in the structure.

A **Module** is a class of structures that share vocabulary and universe. The vocabulary of module M is denoted $\text{vocab}(M)$. We assume that all modules are over an all-inclusive universe U^1 . A module can be axiomatized by FO logic, specified in CSP, or even a Java program, as long as model-theoretic semantics can be provided². Input (σ_M), output (ε_M) vocabularies are assigned to each primitive module M , where $\sigma_M \cap \varepsilon_M = \emptyset$. As we mentioned, σ_M or ε_M can be empty. Note that the same axiomatization can be used in different ways. Say, in 3-Colouring, the edges of a graph can be given

on the input, or a colouring could be given, and one could be searching for possible edges.

Modules are combined using the following algebraic operations. Projection ($\pi_\nu(M)$) hides some vocabulary of a module. Composition ($M_1 \triangleright M_2$) connects outputs of M_1 to inputs of M_2 . Union ($M_1 \cup M_2$) models choice. Complementation ($\overline{M}$) does “the opposite” of what M does. Feedback ($M[R = S]$) connects output S of M to its input R and was inspired by feedbacks in logical circuits. Intuitively, the operations correspond to conjunction, disjunction, negation and existential quantifier. Feedback represents fixpoints (not necessarily minimal) of modules viewed as operators. One can introduce other operations, e.g. least fixpoints or combinations of the operations above.

We now define the syntax of the algebra of modular systems. Following [Järvisalo et al., 2009], we say modules M_1 and M_2 are *composable* if $\varepsilon_{M_1} \cap \varepsilon_{M_2} = \emptyset$ (no output interference). Module M_2 is *independent* from M_1 if $\sigma_{M_2} \cap \varepsilon_{M_1} = \emptyset$ (no cyclic dependencies). A module is *primitive* if the only sub-module (algebraic sub-formula) of it is itself. *Well-formed modular systems* $\text{MS}(\sigma, \varepsilon)$, with instance (σ) and expansion (ε) vocabularies, are defined recursively.

- If M is a primitive module with instance (input) vocabulary σ and expansion (output) vocabulary ε , then $M \in \text{MS}(\sigma, \varepsilon)$.
- If $M \in \text{MS}(\sigma, \varepsilon)$, $\tau \subseteq \sigma \cup \varepsilon$, then $\pi_\tau(M) \in \text{MS}(\sigma \cap \tau, \varepsilon \cap \tau)$.
- If $M \in \text{MS}(\sigma, \varepsilon)$, $M' \in \text{MS}(\sigma', \varepsilon')$, M is composable with and independent from M' , then $(M \triangleright M') \in \text{MS}(\sigma \cup (\sigma' \setminus \varepsilon), \varepsilon \cup \varepsilon')$.
- If $M \in \text{MS}(\sigma, \varepsilon)$, $M' \in \text{MS}(\sigma', \varepsilon')$, and they are independent, then $(M \cup M') \in \text{MS}(\sigma \cup \sigma', \varepsilon \cup \varepsilon')$.
- If $M \in \text{MS}(\sigma, \varepsilon)$, $R \in \sigma$, $S \in \varepsilon$, and R and S are of the same type and arity, then $M[R = S] \in \text{MS}(\sigma \setminus \{R\}, \varepsilon \cup \{R\})$.
- If $M \in \text{MS}(\sigma, \varepsilon)$ then $\overline{M} \in \text{MS}(\sigma, \varepsilon)$.

A modular system is given by an algebraic formula, with input-output vocabulary specified for each primitive module. *Subsystems* correspond to sub-formulas and are modules themselves.

Example 2 (Simple Modular System) Consider the following axiomatizations of modules³, each in the corresponding logic $\mathcal{L}_i$, as specified in Figure 2. $P_{M_1} := \{\mathcal{L}_{WF} : a \leftarrow b\}$, $P_{M_2} := \{\mathcal{L}_{WF} : a \leftarrow c\}$, $P_{M_3} := \{\mathcal{L}_{SM} : d \leftarrow \text{not } a\}$, $P_{M_4} := \{\mathcal{L}_P : b' \vee c' \equiv \neg d\}$.

We underlined instance symbols in each module.

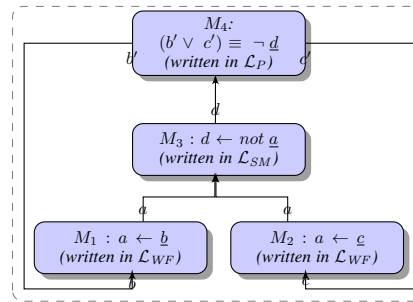


Figure 2: Modular system where modules are axiomatized in different languages: propositional logic $\mathcal{L}_P$, logic programs under well-founded and stable model semantics ($\mathcal{L}_{WF}$, $\mathcal{L}_{SM}$).

¹A multi-sorted version of the framework would be like it is done in classical FO logic.

²CSP has a natural model-theoretic semantics where mappings to domain elements are given by expansion functions. Java programs, if are of input-output type, can be viewed as model expansion tasks (and represent modules), regardless of what they do internally.

³In realistic examples, module axiomatizations are much more complex and contain multiple rules or axioms.

The modular system in Figure 2 is represented by $M := \pi_{\{a,b,c,d\}}(((M_1 \cup M_2) \triangleright M_3) \triangleright M_4)[c = c'][b = b']$. Module $M' := (((M_1 \cup M_2) \triangleright M_3) \triangleright M_4)$ has $\sigma_{M'} = \{b, c\}$, $\varepsilon_{M'} = \{a, b', c', d\}$. After adding feedbacks, we have $M'' := M'[c = c'][b = b']$, which turns instance symbols b and c into expansion symbols, so we have $\sigma_{M''} = \emptyset$ and $\varepsilon_{M''} = \{a, b, c, b', c', d\}$. Also the interpretations of c and c' , and b and b' must coincide. Finally, projection hides c' and b' .

Module M corresponds to the whole modular system within dotted border. Its input-output vocabularies are as follows: $\sigma_M = \emptyset$, $\varepsilon_M = \{a, b, c, d\}$, b' and c' are “hidden” from the outside. They are auxiliary expansion symbols.

Modules $(M_1 \cup M_2)$ and M_3 in this example are composable (no output interference) and independent (no cyclic dependencies), M_1 and M_2 are independent.

Model-theoretic semantics associates, with each modular system, a set of structures. Each such structure is called a *model* of the modular system. The semantics does not put any finiteness restriction on the domains. Thus, the framework works for modules with infinite structures. We assume that the domains of all structures are included in a (potentially infinite) universal domain U .

Definition 1 (Models of a Modular System) Let $M \in \text{MS}(\sigma, \varepsilon)$ be a modular system and $\mathcal{B}$ be a $(\sigma \cup \varepsilon)$ -structure. We construct the set M^{mt} of models of module M by structural induction on the structure of a module.

Primitive Module: $\mathcal{B}$ is a model of M if $\mathcal{B} \in M$.

Projection: $\mathcal{B}$ is a model of $M := \pi_{(\sigma \cup \varepsilon)}(M')$ (with $M' \in \text{MS}(\sigma', \varepsilon')$) if a $(\sigma' \cup \varepsilon')$ -structure $\mathcal{B}'$ exists such that $\mathcal{B}'$ is a model of M' and $\mathcal{B}'$ expands $\mathcal{B}$.

Composition: $\mathcal{B}$ is a model of $M := M_1 \triangleright M_2$ (with $M_1 \in \text{MS}(\sigma_1, \varepsilon_1)$ and $M_2 \in \text{MS}(\sigma_2, \varepsilon_2)$) if $\mathcal{B}|_{(\sigma_1 \cup \varepsilon_1)}$ is a model of M_1 and $\mathcal{B}|_{(\sigma_2 \cup \varepsilon_2)}$ is a model of M_2 .

Feedback: $\mathcal{B}$ is a model of $M := M'[R = S]$ (with $M' \in \text{MS}(\sigma', \varepsilon')$) if $R^{\mathcal{B}} = S^{\mathcal{B}}$ and $\mathcal{B}$ is model of M' .

To save space, we skip union and complementation.

3 Structural Operational Semantics (SOS)

The novel semantics we introduce is structural because, e.g., the meaning of $M_1 \triangleright M_2$ is defined through the meaning of M_1 and M_2 . We assume a potentially infinite vocabulary τ that subsumes the vocabularies of all modules considered.

Definition 1 (Modules as Operators) We say that a well-formed modular system M (non-deterministically) maps τ -structure $\mathcal{B}_1$ to τ -structure $\mathcal{B}_2$, notation $(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2$, if we can apply the rules of the structural operational semantics (below) starting from this expression and arriving to true. In that case, we say that transition $(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2$ is derivable.

Primitive modules M :

$$\frac{(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{\text{true}} \text{ if } \mathcal{B}_2|_{(\sigma \cup \varepsilon)} \in M \text{ and } \mathcal{B}_2|_{(\tau \setminus \varepsilon)} = \mathcal{B}_1|_{(\tau \setminus \varepsilon)}.$$

We proceed by induction on the structure of modular system.

Projection $\pi_\nu(M)$:

$$\frac{(\pi_\nu(M), \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2} \text{ if } \mathcal{B}'_1|_\nu = \mathcal{B}_1|_\nu \text{ and } \mathcal{B}'_2|_\nu = \mathcal{B}_2|_\nu.$$

Composition $M_1 \triangleright M_2$:

$$\frac{(M_1 \triangleright M_2, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{(M_1, \mathcal{B}_1) \longrightarrow \mathcal{B}' \text{ and } (M_2, \mathcal{B}') \longrightarrow \mathcal{B}_2}.$$

Union $M_1 \cup M_2$:

$$\frac{(M_1 \cup M_2, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{(M_1, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}, \quad \frac{(M_1 \cup M_2, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{(M_2, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}.$$

Feedback $M[R = S]$:

$$\frac{(M[R = S], \mathcal{B}_1) \longrightarrow \mathcal{B}_2}{(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2}, \text{ if } R^{\mathcal{B}_1} = S^{\mathcal{B}_2}.$$

Definition 2 (Operational Semantics) Let M be a well-formed modular system in $\text{MS}(\sigma, \varepsilon)$. The semantics of M is given by the following set of structures. (see Figure 3)

$$M^{op} := \{\mathcal{B} \mid (M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2 \text{ and } \mathcal{B}|_\sigma = \mathcal{B}_1|_\sigma, \mathcal{B}|_\varepsilon = \mathcal{B}_2|_\varepsilon\}.$$

Transitions and Inertia SOS allows one to view MSs as *transition systems* where states are τ -structures and transitions are expansions, see Figure 3. In each transition, the interpretation of ε changes. Interpretation of all other symbols, including those in σ , stays the same (by inertia). This is similar to frame axioms in the situation calculus. Similarly, SOS specifies inertia for *primitive* modules only. Suppose after M , we applied M' in sequence, $M \triangleright M'$. Then M' would be applied to each $\mathcal{B}_2$ extending Figure 3 to the right.

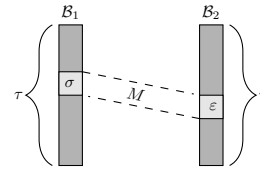


Figure 3: $(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2$. Each $\sigma \cup \varepsilon$ -structure in module M has interpretation of its σ part on the left, and of ε part on the right.

Projection as an Operator Let $\text{vocab}(M) = \sigma' \cup \varepsilon'$, let $\nu = \sigma \cup \varepsilon$, $\sigma \subseteq \sigma'$, $\varepsilon \subseteq \varepsilon'$. Module $\pi_\nu(M)$, viewed as an operator, is applied to τ -structure $\mathcal{B}_1$. It (a) expands σ -part of $\mathcal{B}_1$ to σ' by an arbitrary interpretation over the same domain, and then (b) applies M to the modified input, (c) projects the result of application of M onto ε , ignoring everything else, (d) the interpretations of $\tau \setminus \varepsilon$ are moved by M 's inertia.

Corollary 1 Every result of application of M is its fixpoint. That is, for any τ -states $\mathcal{B}_1, \mathcal{B}_2$, if $(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2$, then $(M, \mathcal{B}_2) \longrightarrow \mathcal{B}_2$.

Proof: (Outline) Since $(M, \mathcal{B}_1) \longrightarrow \mathcal{B}_2$, the interpretation of ε is already changed by M , nothing will be changed by another application of M , and $(M, \mathcal{B}_2) \longrightarrow \mathcal{B}_2$. ■

Theorem 1 (Operational = Model-theoretic Semantics) Let M be a well-formed modular system in $\text{MS}(\sigma, \varepsilon)$. Then, its model-theoretic and operational semantics coincide.

The proof is by induction on the structure of the system.

4 Multi-Language Logic of Modular Systems

Logic of Information Flow Just as relational algebra has a counterpart in relational calculus, our algebra has a counterpart in higher-order logic. The algebraic operations correspond to conjunction, disjunction, negation and existential

quantifier. This is, in fact, how the multi-logic logic of modular systems (the logic of information flow) is defined. The limited space does not allow us to introduce this logic in detail, but one can obtain some idea from the following multi-logic formula describing the system in our example in Figure 2. $\phi_M := \exists b' \exists c' (((\{\mathcal{L}_{WF} : a \leftarrow b\} \vee \{\mathcal{L}_{WF} : a \leftarrow c\}) \wedge \{\mathcal{L}_{SM} : d \leftarrow \text{not } a\}) \wedge \{\mathcal{L}_P : b' \vee c' \equiv \neg d\}) [b = b' \wedge c = c'])$. Local input symbols are underlined.

5 Complexity and Expressiveness

Whenever we talk about combining multiple languages, it is important to talk about the *complexity and/or expressiveness of the combination of these languages*. In this section, we are concerned with these questions in the framework of MSs.

In order to talk about these concepts, we first have to define a few basic notions about the complexity of a modular system. Since, according to their model-theoretic semantics, modular systems are sets of structures, we focus on the complexity of deciding/solving such a set of structures.

Hence, similar to the common practice in Descriptive complexity [Immerman, 1982], we also assume a standard encoding of structures as binary strings. Also, since we now talk about computational complexity, we need to assume that all domains are finite. For finite structure $\mathcal{A}$, we define $|\mathcal{A}|$ to be the size of $\mathcal{A}$'s domain. Note that, according to this definition, for a class $\mathcal{K}$ of finite structures over a vocabulary τ , the size of encoding of a structure $\mathcal{A} \in \mathcal{K}$ is bounded by a polynomial on $|\mathcal{A}|$. Furthermore, as is common in descriptive complexity, a *problem* is just a class $\mathcal{K}$ of finite $(\sigma \cup \varepsilon)$ -structures.

Since a modular system M can also be viewed as a class $\mathcal{K}$ of $(\sigma_M \cup \varepsilon_M)$ -structures, we say that modular system M *represents* a problem P if (1) $\sigma_M = \sigma$, (2) $\varepsilon_M = \varepsilon$, and $M = P$, i.e., M and P have the same input-output signatures and consist of the same class of finite structures. Using this notation, *complexity of modular system M* is exactly the complexity of the problem P represented by M and the *expressiveness of modular systems framework* (or a fragment of it) is defined in terms of the problems (classes of finite structures) expressible in that fragment of modular systems. We can thus abuse some notations from complexity theory to re-define polynomial hierarchy for modular systems as follows:

Definition 3 (Complexity of a Modular System) Let $M \in \text{MS}(\sigma, \varepsilon)$ be a modular system:

1. We say $M \in \Delta_{k+1}^P$ if, given structure $\mathcal{A}$, producing the set of all possible expansions of $\mathcal{A}$ in M (i.e., $\mathcal{A} \mapsto \{\mathcal{B} \mid \mathcal{B}|_\sigma = \mathcal{A}, \mathcal{B} \in M\}$) can be computed using a poly-time oracle Turing machine T with access to a Σ_k^P -complete problem.
2. We say $M \in \Sigma_{k+1}^P$ if, given structure $\mathcal{A}$, checking whether or not an expansion $\mathcal{B}$ of $\mathcal{A}$ belongs to M can be done in the complexity class Δ_k^P .

Note that, definitions of complexity classes Σ_k^P and Δ_k^P in Definition 3, is different from the usual definitions of Δ_k^P and Σ_k^P from complexity theory. In complexity theory, Δ_k^P and Σ_k^P are classes of decision problems while, for us, problems in these classes are allowed to have outputs (rather than just accepting or rejecting a structure). However, once $\varepsilon = \{\}$,

i.e., once there is no outputs, our definition coincides with the complexity-theoretical definition as desired. Also, our definitions allow a modular system $M \in \Delta_k^P$ to have polynomially many different expansions $\mathcal{B}$ for the same instance structure $\mathcal{A}$. We call those $\mathcal{B}$'s the *answers to $\mathcal{A}$* and denote them by $M(\mathcal{A})$, i.e., $M(\mathcal{A}) := \{\mathcal{B} \mid \mathcal{B}|_\sigma = \mathcal{A}, \mathcal{B} \in M\}$. By Definition 3, $|M(\mathcal{A})|$ is polynomially bounded by $|\mathcal{A}|$. That is, polynomial function $p(\cdot)$ exists such that $|M(\mathcal{A})| \leq p(|\mathcal{A}|)$ for all finite structures $\mathcal{A}$.

Definition 4 (Model Expansion for Modular Systems)

Let $M \in \text{MS}(\sigma, \varepsilon)$ be a modular system. The task of model expansion for M takes a σ -structure $\mathcal{A}$ and finds (or reports that none exists) a $(\sigma \cup \varepsilon)$ -structure $\mathcal{B}$ that expands $\mathcal{A}$ and is a model of the entire system M . Such a structure $\mathcal{B}$ is a solution of M for input $\mathcal{A}$. Also, such a structure $\mathcal{A}$ is said to be accepted if it has an answer and rejected otherwise.

Note that, using Definitions 3 and 4, model expansion task for a modular system $M \in \Delta_k^P$ is always a Δ_k^P -recognizable class of structures (in the descriptive complexity sense) and model expansion task for a modular system $M \in \Sigma_k^P$ is also always a Σ_k^P -recognizable class of structures (again, in the descriptive complexity sense). Hence, our definitions closely correspond to their counterparts in descriptive complexity.

Theorem 2 (Complexity of Operators on Δ_k^P Modules)

Let M_1 and M_2 be two modular systems in the complexity class Δ_k^P . Then, the following holds:

- (1) $M_1 \triangleright M_2 \in \Delta_k^P$,
- (2) If $\varepsilon_{M_1} = \varepsilon_{M_2}$ then $M_1 \cup M_2 \in \Delta_k^P$,
- (3) If $\sigma_{M_1} \subseteq \tau$ then $\pi_\tau(M_1) \in \Delta_k^P$,
- (4) If $\varepsilon_{M_1} \neq \varepsilon_{M_2}$ then $M_1 \cup M_2 \in \Sigma_{k+1}^P$,
- (5) If $\sigma_{M_1} \not\subseteq \tau$ then $\pi_\tau(M_1) \in \Sigma_{k+1}^P$, and,
- (6) $M_1[R = S] \in \Sigma_{k+1}^P$.

Proof: (1): Construct the Δ_k^P procedure for $M_1 \triangleright M_2$ by combining the procedures for M_1 and M_2 . Let $p_1(n)$ and $p_2(n)$ be two polynomials bounding the number of answers returned by M_1 and M_2 respectively. Also, given instance structure $\mathcal{A}$, let $S := M_1(\mathcal{A})$ be the set of possible answers to $\mathcal{A}$ according to M_1 . Then, the set of possible answers to $\mathcal{A}$ according to $M_1 \triangleright M_2$ is $(M_1 \triangleright M_2)(\mathcal{A}) := \bigcup_{\mathcal{B} \in S} M_2(\mathcal{B})$. Note that the number of possible answers in $(M_1 \triangleright M_2)(\mathcal{A})$ is bounded by $p_1(|\mathcal{A}|) \times p_2(|\mathcal{A}|)$.

(2): Similar to (1), construct the Δ_k^P procedure for $M_1 \cup M_2$ as follows: $(M_1 \cup M_2)(\mathcal{A}) := M_1(\mathcal{A}) \cup M_2(\mathcal{A})$. Also, note that the number of possible solutions to $(M_1 \cup M_2)(\mathcal{A})$ is bounded by $p_1(|\mathcal{A}|) + p_2(|\mathcal{A}|)$.

(3): Construct the Δ_k^P procedure for $\pi_\tau(M_1)$ so that $(\pi_\tau(M_1))(\mathcal{A}) := \{\mathcal{B}|_\tau \mid \mathcal{B} \in M_1(\mathcal{A})\}$. Note that this procedure returns at most $p_1(|\mathcal{A}|)$ answers.

(4): Given instance structure $\mathcal{A}$, guess an expansion $\mathcal{B}$ of $\mathcal{A}$ to vocabulary of $M_1 \cup M_2$. Now, accept $\mathcal{B}$ if and only if either $\mathcal{B}|_{\text{vocab}(M_1)} \in M_1(\mathcal{A})$ or $\mathcal{B}|_{\text{vocab}(M_2)} \in M_2(\mathcal{A})$.

(5): Given instance structure $\mathcal{A}$, guess an expansion $\mathcal{B}$ of $\mathcal{A}$ to M_1 's vocabulary and accept $\mathcal{B}$ iff $\mathcal{B} \in M_1(\mathcal{B}|_{\sigma_{M_1}})$.

(6): Given instance structure $\mathcal{A}$, use non-determinism to obtain expansion $\mathcal{A}'$ of $\mathcal{A}$ so that $\mathcal{A}'$ interprets R as well. Now, accept all structures $\mathcal{B} \in M_1(\mathcal{A}')$ where $S^{\mathcal{B}} = R^{\mathcal{B}}$. ■

Theorem 2 asserts that, once applied to modular systems $M_1, M_2 \in \Delta_k^P$, operators in our modular system framework can be syntactically divided into two sub-groups: (1) the ones that preserve the complexity of their operands, and, (2) the ones that may increase the complexity over that of their operands. In Theorem 2, the first three cases identify the former group, also called *power-preserving operators*, and the next three cases identify the latter group, also called *power-increasing operators*.

While Theorem 2 showed that, our operators can be divided based on their behavior over Δ_k^P modules, the following theorem shows that all our modular system operators act accordingly when applied to Σ_{k+1}^P modules:

Theorem 3 (Complexity of Operators on Σ_{k+1}^P modules)
Let M_1 and M_2 be two modular systems in the complexity class Σ_{k+1}^P . Then, all well-formed modular systems $(M_1 \triangleright M_2)$, $(M_1 \cup M_2)$, $\pi_\tau(M_1)$ and $M_1[R = S]$ also belong to the complexity class Σ_{k+1}^P .

Proof: Given an instance structure $\mathcal{A}$, in each case, we use non-determinism to guess all the non-input vocabulary symbols (i.e., either output vocabulary symbols or those vocabulary symbols that have been projected out). Then, we use the Δ_k^P procedures of M_1 and M_2 to check if our guess should be accepted. ■

Based on Theorem 3, whenever two Σ_{k+1}^P modules are combined, the result still remains to be a Σ_{k+1}^P module. In combination with Theorem 2, it can be informally said that, in our MS framework, once the complexity rises from Δ_k^P to Σ_{k+1}^P , it remains there (the complexity cannot rise anymore).

As described above, many complexity-theoretical concepts such as a problem's complexity or a language's expressiveness can be translated to the domain of modular systems by using the concept of a module representing a problem. One such important concept is the capturing property:

Definition 5 (Capturing) *Let F be a fragment of modular systems and C be a complexity class. Then we say that F -MX captures C if, for every finite σ -structure $\mathcal{K}$, we have $\mathcal{K} \in C$ if and only if a modular system $M \in F$ exists so that the task of model expansion for M accepts exactly the structures in $\mathcal{K}$.*

This property is important because it simultaneously provides us with a *complexity guarantee* and *universality guarantee*. That is, it says that every problem in fragment F can be solved in complexity class C (complexity guarantee) and that every problem in complexity class C can be represented in fragment F of MSs (universality guarantee). Thus, F can be thought of as a *universal framework* for problems in C .

We present three fragments $F_\cup^k$, F_π^k and F_π^k of MSs. The intuition behind these fragments is that they only contain primitive modules $M \in \Delta_k^P$ and are all closed under power-preserving operations, while each of them is also closed under one of the power-increasing operators. That is, $F_\cup^k$ is closed under union, F_π^k under feedback and so on. We want to show that, although the primitive modules in F_π^k (with $*$ being $\cup$, $=$ or π) consists of only Δ_k^P modules, each fragment F_π^k of MSs is capable of expressing all problems in Σ_{k+1}^P .

Define $F_\cup^k$, F_π^k and F_π^k (for $k \in \mathbb{N}$) as a fragment of well-formed modular systems such that:

1. For each problem $P \in \Delta_k^P$, we have a primitive module $M_P \in F_\pi^k$ (with $*$ being $\cup$, $=$ or π),
2. F_π^k (with $*$ being $\cup$, $=$ or π) is closed under all power-preserving operators,
3. F_π^k (resp. $F_\cup^k$ and F_π^k) is closed under the feedback operation (resp. under union and projection operations).

The following theorem states that each of these three fragments of modular systems captures complexity class Σ_{k+1}^P :

Theorem 4 *For all $k \in \mathbb{N}$, we have F_π^k -MX (with $*$ being $\cup$, $=$ or π) captures Σ_{k+1}^P .*

Proof: (Expressiveness, $F_\pi^k \supseteq \Sigma_{k+1}^P$) If $\mathcal{K} \in \Sigma_{k+1}^P$ then, by definition, a Δ_k^P -recognizable class $\mathcal{K}'$ of finite structures over vocabulary $\sigma \cup \varepsilon$ exists so that, for each σ -structure $\mathcal{A}$, we have $\mathcal{A} \in \mathcal{K}$ if and only if a $(\sigma \cup \varepsilon)$ -structure $\mathcal{B} \in \mathcal{K}'$ exists that expands $\mathcal{A}$. Let $M' \in \text{MS}(\sigma \cup \varepsilon, \{\})$ be a primitive module that contains exactly the structures in $\mathcal{K}'$. Clearly $M' \in \Delta_k^P$ (because $\mathcal{K}' \in \Delta_k^P$). We want to show that, using M' , we can build modular system $M_1 \in L_1^k$, $M_2 \in L_2^k$ and $M_3 \in L_3^k$ so that M_1 , M_2 and M_3 contain exactly the structures in $\mathcal{K}$.

For L_1^k , we define $M_1 := \pi_\sigma((N'[\varepsilon' = \varepsilon]) \triangleright M')$ where ε' is a set of new vocabulary symbols which are in one-to-one correspondence with ε and, $N' \in \text{MS}(\varepsilon', \varepsilon)$ is a module that accepts those structures $\mathcal{B}$ where $\varepsilon^{\mathcal{B}} = \varepsilon'^{\mathcal{B}}$.

Clearly, N' is poly-time computable and, thus $N' \in \Delta_k^P$. Therefore, $M_1 \in \Delta_k^P$ as well. Moreover, observe that $N'[\varepsilon' = \varepsilon]$ accepts all interpretations of ε . Hence, M_1 accepts exactly those structures in $\mathcal{K}$ (as required).

For L_2^k , we define $M_2 := \pi_\sigma([(N_1 \cup N_2) \triangleright N] \triangleright M')$ where $N_1 \in \text{MS}(\{\}, \varepsilon_1)$, $N_2 \in \text{MS}(\{\}, \varepsilon_2)$ and $N \in \text{MS}(\varepsilon_1 \cup \varepsilon_2, \varepsilon)$ are as follows:

1. ε_1 and ε_2 are new vocabulary symbols that are in one-to-one correspondence with vocabulary symbols in ε ,
2. N_1 (resp. N_2) contains exactly those structures $\mathcal{B}$ where $\varepsilon_1^{\mathcal{B}} = \{\}$ (resp. $\varepsilon_2^{\mathcal{B}} = \{\}$),
3. N contains exactly those structures $\mathcal{B}$ where either $\varepsilon_1^{\mathcal{B}} = \varepsilon^{\mathcal{B}}$ and $\varepsilon_2^{\mathcal{B}} = \{\}$ or $\varepsilon_2^{\mathcal{B}} = \varepsilon^{\mathcal{B}}$ and $\varepsilon_1^{\mathcal{B}} = \{\}$.

Note that, since N_1 and N_2 always output empty sets, they are poly-time computable and, therefore, $N_1, N_2 \in \Delta_k^P$. Similarly, N is also poly-time computable and hence in Δ_k^P . Thus, we have $M', N_1, N_2, N \in L_2^k$ and, so, $M_2 \in L_2^k$.

It is also easy to observe that, due to the union operation between N_1 and N_2 , the vocabulary symbols in ε can take all possible interpretations. Hence, M_2 contains exactly those structures in $\mathcal{K}$ (as required).

For L_3^k , we define $M_3 := \pi_\sigma(M')$. Then, by definition $M_3 \in L_3^k$ and M_3 contains exactly the structures in $\mathcal{K}$.

(Complexity, $F_\pi^k \subseteq \Sigma_{k+1}^P$) Follows by induction on the structure of M where Theorem 3 is applied in each induction step. ■

6 Related Work

Many combinations of logical formalisms have been proposed, notably logics for semantics web, with different types of integration, mostly by embedding of one (non-monotonic)

logic into another, within one theory, or combinations through a conjunction of two different parts. More details, reviews and references can be found in [Eiter et al., 2008; de Bruijn et al., 2007]. However, more complex combinations are needed, where logics are not embedded, but used ‘as is’.

To our knowledge, no other work proposed a framework and a research program of *programming from reusable components* in knowledge-intensive computing, and suggested model-theoretic and operational techniques. There are, however, important works that study related issues. Recently, [Lierler and Truszczyński, 2014] introduced a formalism with compositions (essentially, conjunctions) of modules given through solver-level inferences. The importance of that work is in a unifying mathematical view on the work of a variety of solvers and their combinations, which we believe is essential. In [Tasharofi and Ternovska, 2014b], we generalized inference-based modules of [Lierler and Truszczyński, 2014] as another representation of modules within our algebra. That generalization represents inference modules as sets of structures, as introduced in [Tasharofi and Ternovska, 2011]. The basic notions of [Lierler and Truszczyński, 2015], i.e., the notion of a module and information propagation through the same vocabulary symbols, are identical, in its mathematical essence, to those in [Tasharofi and Ternovska, 2011; 2014b]. Modules in [Lierler and Truszczyński, 2015] are axiomatized (represented by theories) in logics associated with modules, which is a particular way of giving a module (a class of structures) in our framework, where any decision procedure is possible. The only difference is that [Tasharofi and Ternovska, 2011] specify input-output vocabulary in addition. Compositions in [Lierler and Truszczyński, 2015], which we call Products, cannot achieve the same expressive power as our algebra since other basic algebraic operations are not expressible through Products. For example, they cannot express Projections, which are needed, for example, if two modules have the same vocabulary, $\{P, Q, R\}$ but P is needed to be “local” in each module, and only Q and R used for communication. Feedbacks and Unions are also not expressible through Products⁴. The largest part of [Lierler and Truszczyński, 2015] is devoted to establishing connections to Multi-Context Systems (MCSs) [Brewka and Eiter, 2007]. MCSs are perhaps the closest framework to ours, among earlier work. However, it has different goals. The semantics of MCSs is very abstract. Beliefs that interpret knowledge bases are arbitrary entities, e.g. formulas. This choice gives a great generality, but the price is that uniformity of analysis and algorithmic techniques is not possible. In addition, communication in MCSs happens through rules similar to rules of logic programs with negation as failure. These rules are representable in a variant of modular systems [Tasharofi and Ternovska, 2014a].

Another important direction is Satisfiability Modulo Theory (SMT) [Sebastiani, 2007]. It is not applicable to *arbi-*

⁴In fact, the five *basic* operations of our algebra are Union, Set Difference, Product, Projection and Selection. Sequential and parallel compositions are derived from those. Selection is used to link modules with different vocabularies, and even express grounding (instantiation using input structures). Details are in a forthcoming paper.

trary modules, potentially taken from the web and reusable. In contrast, we offer methods of combining arbitrary modules through a unifying semantics. The same applies to combined solvers such as [Gebser, Ostrowski, and Schaub, 2009].

[Järvisalo et al., 2009] developed a constraint-based formalism where modules were combined through projection and composition. The work was extended in [Tasharofi and Ternovska, 2011] with more operations and model-theoretic semantics.

7 Conclusion

We described a modular system framework, where combinations of modules are achieved by applying algebraic operations. We defined structural operational semantics, and proved that it is equivalent to model-theoretic semantics. We presented a multi-language logic, a syntactic counterpart of the algebra of modular systems. Finally, we analyzed computational complexity of modular systems.

We believe that the study is important for several reasons.

We believe that classic model theory is the right abstraction tool and a good common ground for combining formalisms from different communities. It is sufficiently general and provides a rich machinery developed by generations of researchers. The machinery includes, for example, deep connections between expressiveness and computational complexity. We believe that foundations in model theory can make the interaction between various solver communities on one hand and the KR community on the other much more easy and fruitful. Most importantly, *uniform grounds are necessary for constructing efficient solving algorithms* (work submitted for publication).

The MSs framework gives rise to a whole new family of KR formalisms by giving the semantics to complex combinations of modules. This is a significant extension of ASP, and can be viewed as a “*modular multi-language ASP*”. In the past, combining ASP programs was only possible, under some conditions, in sequence. Now, we can combine them in a loop, use projections to hide parts of the vocabularies, etc. (In fact, Java and C programs can be combined, if model-theoretic semantics is provided). Previously, in ASP, all modules and their combinations had to be interpreted under one semantics (e.g. stable model semantics). Now, any model-theoretic semantics of individual modules is allowed. (Some modules can be axiomatized in other logics for e.g. performance benefits). Our multi-language logic can be used for creation of new combined logics, such as those needed for semantic web.

The Operational semantics allows one to apply the extensive research on transition systems (and their verification) in the context of MSs.

Our complexity study allows us to develop formalisms with controlled expressiveness. Since [Mitchell and Ternovska, 2005], we have been arguing for importance of capturing complexity classes in KR formalisms. Such results show that all problems (and no more) in a complexity class are expressible. The “no more” part is responsible for implementability. In general, at most “NP-step” is added by the algebraic operations, compared to the highest power among

primitive modules. The fundamental importance of this fact is that *a solver, working in cooperation with solvers of individual modules, is implementable with the current solver technology that is able to handle NP*. A high-level algorithm that iteratively questions individual modules with partial 3-values structures is given in [Tasharrofi and Ternovska, 2011]. If a structure is accepted by a module, the module may suggest (through a formula) what else should be true. If a structure is rejected by a module, a reason (also a formula) may be given. The algorithm reasons with accumulated formulas, using 3-valued partial structures as a data structure, and may retract its earlier decisions. Information is added until all unknowns are filled. Instantiations of the algorithm were shown to capture the work of SMT (DPPL(T)), ASP-CP, IDP, ILP in [Tasharrofi, Wu, and Ternovska, 2011; Wu, 2012].

References

- [Aavani et al., 2012] Aavani, A.; Wu, X. N.; Tasharrofi, S.; Ternovska, E.; and Mitchell, D. G. 2012. Enfragmo: A system for modelling and solving search problems with logic. In *18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, 15–22.
- [Brewka and Eiter, 2007] Brewka, G., and Eiter, T. 2007. Equilibria in heterogeneous nonmonotonic multi-context systems. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI'07) - Volume 1*, 385–390. AAAI Press.
- [Cadoli et al., 2000] Cadoli, M.; Ianni, G.; Palopoli, L.; Schaerf, A.; and Vasile, D. 2000. NP-SPEC: An executable specification language for solving all problems in NP. *Computer Languages* 26:165–195.
- [de Bruijn et al., 2007] de Bruijn, J.; Eiter, T.; Polleres, A.; and Tompits, H. 2007. Embedding non-ground logic programs into autoepistemic logic for knowledge-base combination. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI2007)*. Hyderabad, India: AAAI Press.
- [de la Banda et al., 2006] de la Banda, M. J. G.; Marriott, K.; Rafeh, R.; and Wallace, M. 2006. The modelling language Zinc. *Principles and Practice of Constraint Programming (CP 2006)* 700–705.
- [Eiter et al., 2008] Eiter, T.; Ianni, G.; Lukasiewicz, T.; Schindlauer, R.; and Tompits, H. 2008. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence* 172:1495 – 1539.
- [Ensan and Ternovska, 2015] Ensan, A., and Ternovska, E. 2015. Modular systems with preferences. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI2015)*. Buenos-Aires, Argentina: AAAI Press.
- [Frisch et al., 2008] Frisch, A. M.; Harvey, W.; Jefferson, C.; Martínez-Hernández, B.; and Miguel, I. 2008. Essence: A constraint language for specifying combinatorial problems. *Constraints* 13:268–306.
- [Gebser, Kaufmann, and Schaub, 2012] Gebser, M.; Kaufmann, B.; and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence* 187-188:52–89.
- [Gebser, Ostrowski, and Schaub, 2009] Gebser, M.; Ostrowski, M.; and Schaub, T. 2009. Constraint answer set solving. In *Proceedings of the 25th International Conference on Logic Programming (ICLP'09)*, Lecture Notes in Computer Science (LNCS), 235–249. Springer-Verlag.
- [Immerman, 1982] Immerman, N. 1982. Relational queries computable in polynomial time. In *STOC '82: Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, 147–152.
- [Järvisalo et al., 2009] Järvisalo, M.; Oikarinen, E.; Janhunen, T.; and Niemelä, I. 2009. A module-based framework for multi-language constraint modeling. In *Proceedings of the 10th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Computer Science (LNCS)*, 155–168. Springer-Verlag.
- [Kolokolova et al., 2010] Kolokolova, A.; Liu, Y.; Mitchell, D.; and Ternovska, E. 2010. On the complexity of model expansion. In *Proc., 17th Int'l Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-17)*, 447–458. Springer. LNCS 6397.
- [Lierler and Truszczyński, 2014] Lierler, Y., and Truszczyński, M. 2014. Abstract modular inference systems and solvers. In *Proceedings of the 16th International Symposium on Practical Aspects of Declarative Languages (PADL'14)*.
- [Lierler and Truszczyński, 2015] Lierler, Y., and Truszczyński, M. 2015. An abstract view on modularity in knowledge representation. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*.
- [Mitchell and Ternovska, 2005] Mitchell, D. G., and Ternovska, E. 2005. A framework for representing and solving NP search problems. In *Proc. AAAI'05*, 430–435.
- [Plotkin, 1981] Plotkin, G. 1981. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University. Also published in: *Journal of Logic and Algebraic Programming*, 60-61:17-140, 2004.
- [Sebastiani, 2007] Sebastiani, R. 2007. Lazy satisfiability modulo theories. *Journal of Satisfiability, Boolean Modeling and Computation (JSAT)* 3:141–224.
- [Tasharrofi and Ternovska, 2011] Tasharrofi, S., and Ternovska, E. 2011. A semantic account for modularity in multi-language modelling of search problems. In *Proceedings of the 8th International Symposium on Frontiers of Combining Systems (FroCoS)*, 259–274.
- [Tasharrofi and Ternovska, 2014a] Tasharrofi, S., and Ternovska, E. 2014a. Generalized multi-context systems. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR2014)*.

- [Tasharrofi and Ternovska, 2014b] Tasharrofi, S., and Ternovska, E. 2014b. Three semantics for modular systems. In *Proceedings of NMR'2014*.
- [Tasharrofi, Wu, and Ternovska, 2011] Tasharrofi, S.; Wu, X. N.; and Ternovska, E. 2011. Solving modular model expansion tasks. In *Proceedings of the 25th International Workshop on Logic Programming (WLP'11)*, volume abs/1109.0583. Computing Research Repository (CoRR).
- [Tasharrofi, Wu, and Ternovska, 2012] Tasharrofi, S.; Wu, X. N.; and Ternovska, E. 2012. Solving modular model expansion: Case studies. In *Postproceedings of the 19th International Conference on Applications of Declarative Programming and Knowledge Management and 25th Workshop on Logic Programming*, 175–187. Lecture Notes in Artificial Intelligence (LNAI).
- [Tasharrofi, 2013] Tasharrofi, S. 2013. *Solving Model Expansion Tasks: System Design and Modularity*. Ph.D. Dissertation, Simon Fraser University, Burnaby, BC, Canada.
- [Wittocx, Marién, and Denecker, 2008] Wittocx, J.; Marién, M.; and Denecker, M. 2008. The IDP system: A model expansion system for an extension of classical logic. In *Proceedings of the 2nd Workshop on Logic and Search*, 153–165.
- [Wu, 2012] Wu, X. N. 2012. Solving model expansion tasks: System design and modularity. Master's thesis, Simon Fraser University, Burnaby, BC, Canada.