Three Semantics for Modular Systems

Shahab Tasharrofi and Eugenia Ternovska
Simon Fraser University, email: \{ter, sta44\}@cs.sfu.ca

Abstract

In this paper, we further develop the framework of Modular Systems that lays model-theoretic foundations for combining different declarative languages, agents and solvers. We introduce a multi-language logic of modular systems. We define two novel semantics, a structural operational semantics, and an inference-based semantics. We prove the new semantics are equivalent to the original model-theoretic semantics and describe future research directions.

Introduction

Modular Systems (MS) (Tasharrofi and Ternovska 2011) is a language-independent formalism representing and solving complex problems specified declaratively. There are several motivations for introducing the MS formalism:

- the need to be able to split a large problem into subproblems, and to use the most suitable formalism for each part,
- the need to model distributed combinations of programs, knowledge bases, languages, agents, etc.,
- the need to model collaborative solving of complex tasks, such as in satisfiability-based solvers.

The MS formalism gave a unifying view, through a semantic approach, to formal and declarative modelling of modular systems. In that initial work, individual modules were considered from both model-theoretic and operational view. Under the model-theoretic view, a module is a set (or class) of structures, and under the operational view it is an operator, mapping a subset of the vocabulary to another subset. An abstract algebra on modules was given. It is similar to Codd’s relational algebra and allows one to combine modules on abstract model-theoretical level, independently from what languages are used for describing them. An important operation in the algebra is the loop (or feedback) operation, since iteration underlies many solving methods. We showed that the power of the loop operator is such that the combined modular system can capture all of the complexity class NP even when each module is deterministic and polytime. Moreover, in general, adding loops gives a jump in the polynomial time hierarchy, one step from the highest complexity of the components. It is also shown that each module can be viewed as an operator, and when each module is (anti-) monotone, the number of the potential solutions can be reduced by using ideas from the logic programming community.

Inspired by practical combined solvers, the authors of (Tasharrofi, Wu, and Ternovska 2011; Tasharrofi, Wu, and Ternovska 2012) introduced an algorithm to solve model expansion tasks for modular systems. The evolution processes of different modules are jointly considered. The algorithm incrementally constructs structures for the expanded vocabulary by communicating with oracles associated with each module, who provide additional information in the form of reasons and advice to navigate the search. It was shown that the algorithm closely corresponds to what is done in practice in different areas such as Satisfiability Modulo Theories (SMT), Integer Linear Programming (ILP), Answer Set Programming (ASP).

Background: Model Expansion

In (Mitchell and Ternovska 2005), 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 the 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 (Frisch et al. 2008). The MX framework was later extended to infinite structures to formalise built-in arithmetic in specification languages (Tasharrofi and Mitchell 2009; Tasharrofi and Ternovska 2010a).

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 \( \text{dom}(\_\_\_\)\), 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.

Formally, the task of model expansion for an arbitrary logic \( \mathcal{L} \) is: Given an \( \mathcal{L} \)-formula \( \varphi \) with vocabulary \( \sigma \cup \varepsilon \) and a structure \( \mathcal{A} \) for \( \sigma \) find an expansion of \( \mathcal{A} \), to \( \sigma \cup \varepsilon \), that satisfies \( \varphi \). Thus, we expand the structure \( \mathcal{A} \) with rela-
fions and functions to interpret \( \varepsilon \), obtaining a model \( B \) of \( \phi \). We call \( \sigma \), the vocabulary of \( A \), the instance vocabulary, and \( \varepsilon := vocab(\phi) \setminus \sigma \) the expansion vocabulary. If \( \sigma = \emptyset \), we talk about model generation, a particular type of model expansion that is often studied.

Given a specification, we can talk about a set of \( \sigma \cup \varepsilon \)-structures which satisfy the specification. Alternatively, we can simply talk about a given set of \( \sigma \cup \varepsilon \)-structures as an MX-task, without mentioning a particular specification the structures satisfy. These sets of structures will be called modules later in the paper. This abstract view makes our study of modularity language-independent.

**Example 1** The following logic program \( \phi \) constitutes an MX specification for Graph 3-colouring:

\[
1 \{ R(x), B(x), G(x) \} I \leftarrow V(x).
\]
\[
\downarrow I \leftarrow R(x), R(y), E(x, y).
\]
\[
\downarrow I \leftarrow B(x), B(y), E(x, y).
\]
\[
\downarrow I \leftarrow G(x), G(y), E(x, y).
\]

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

\[
A = (V; E^A, R^B, B^B, G^B) \models \phi.
\]

The interpretations of \( \varepsilon \), for structures \( B \) that satisfy \( \phi \), are exactly the proper 3-colourings of \( G \).

The model expansion task is very common in declarative programming. – given an input, we want to generate a solution to a problem specified declaratively. This is usually done through grounding, i.e., combining instance structure \( A \) to a problem description \( \phi \) thus obtaining a reduction to a low-level solver language such as SAT, ASP, SMT, etc. Model Expansion framework was introduced for systematic study of declarative languages. In particular, it connects KR with descriptive complexity (Immerman 1982). It focuses on problems, not on problem instances, it separates instances from problem descriptions. Using the MX framework, one can produce expressiveness and capturing results for specification languages to guarantee:

• universality of a language for a class of problems,
• feasibility of a language by bounding resources needed to solve problems in that language.

In terms of complexity, MX lies in-between model checking (MC) (a full structure is given) and satisfiability (SAT) (we are looking for a structure). Model generation (\( \sigma = \emptyset \)) has the same complexity as MX. The authors of (Kolokolova et al. 2010) studied the complexity of the three tasks, MC, MX and SAT, for several logics. Despite the importance of MX task in several research areas, the task has not yet been studied sufficiently, unlike the two related tasks of MC and SAT.

By “:=” we mean “is by definition” or “denotes”. By \( vocab(\phi) \) we understand the vocabulary of \( \phi \).

---

**General Research Goal: Adding Modularity** Given the importance of combining different languages and solvers to achieve ease of axiomatization and the best performance, our goal is to extend the MX framework to combine modules specified in different languages. The following example illustrates what we are aiming for.

**Example 2 (Factory as Model Expansion)** In Figure 1 a part of a simple factory is represented as a modular system. Both the office and the workshop modules can be viewed as model expansion tasks. The instance vocabulary of the workshop is \( \sigma = \{ \text{RawMaterials} \} \) and expansion vocabulary \( \varepsilon = \{ R \} \). The bigger box with dashed borders is an MX task with instance vocabulary \( \sigma' = \{ \text{Orders, RawMaterials} \} \) and expansion vocabulary \( \varepsilon' = \{ \text{Plan} \} \) (the “internal” expansion symbols \( O \) and \( R \) are hidden from the outside). This task is a compound MX task whose result depends on the internal work of the office and the workshop, both of which can also have an internal structure and be represented as modular systems themselves.

**Contributions of this paper** In this paper, we further develop the framework of Modular Systems. In this framework, primitive modules represent individual knowledge bases, agents, companies, etc. They can be axiomatized in a logic, be legacy systems, or be represented by a human who makes decisions. Unlike the previous work, we precisely define the notion of a well-formed modular system, and clearly separate the syntax of the algebraic language and the semantics of the algebra of modular systems. The syntax of the algebra uses a few operations, each of them (except feedback) is a counterpart of an operation in Codd’s relational algebra, but over sets of structures rather than tables, and with directionality taken into account. The semantics of both primitive and compound modules is simply a set (class) of structures (an MX task). By relying on the semantics of the algebra, we then introduce its natural counterpart in logic. The logic for modular systems allows for multiple logics axiomatizing individual modules in the same formula. We expect that multi-language formalisms such as ID-logic (Denecker and Ternovska 2008) will be shown to be particular instances of this logic, and other combinations of languages will be similarly developed.

After giving the model-theoretic semantics of the algebra of modular systems, we define what it means, for a primitive module, to act as a non-deterministic operator on states of
the notion of a structure is important in KR as it abstractly expresses expressiveness and computational complexity. In addition, machinery includes, for example, deep connections between rich machinery developed by generations of researchers. The communities. It is sufficiently general and provides a ground for combining formalisms developed in different model theory is the right abstraction tool and a good ground for solving modular systems. The inference semantics is the third semantics of modular systems. Previously, in ASP, all modules had to be in sequence. Now, we can combine them in a loop, use programs satisfy our conditions for sequential compositions of modules. This is can be viewed, for example, using abstract inference rules they introduced. We believe it is an important work. In this paper, we show how inference system can be lifted and integrated with our Modular Systems framework. The advantage of this integrations is that, with the help of the inference semantics, we can now go into much greater level of details of propagation processes in our abstract algorithm for solving modular systems. The inference semantics is the third semantics of modular systems mentioned in the title.

The importance of abstract study of modularity We now would like to discuss the potential implications of abstract study of modularity for KR and declarative programming.

A family of multi-language KR formalisms The Modular Systems framework gives rise to a new family of KR formalisms by giving the semantics to the combination of modules. This is can be viewed, for example, as a significant extension of answer set programming (ASP). In the past, combining ASP programs that were created separately from each other 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. The previous results remain applicable. We expect, for example, that splitable programs under stable model semantics and stratifiable programs satisfy our conditions for sequential compositions of modules. Previously, in ASP, all modules had to be interpreted under one semantics (e.g. stable model semantics). Now, any model-theoretic semantics of individual modules is allowed. For example, some of the modules can be axiomatized, say, in first-order logic. That is, in particular, our proposal amounts to a “modular multi-language ASP”.

Foundations in model theory We believe that classic model theory is the right abstraction tool and a good common ground for combining formalisms developed in 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. In addition, the notion of a structure is important in KR as it abstractly represents our understanding of the world.

We believe that, despite common goals, the interaction between the CP community and various solver communities on one hand and the KR community is insufficient, and that foundations in model theory can make the interaction much more easy and fruitful.

Analyzing other KR systems Just as in the case of single-module system where we can use the purely semantical framework of model expansion, we can use the framework of Modular Systems to analyze multi-language KR formalisms and to study the expressive power of modular systems.

The modular framework generalizes naturally to the case where we need to study languages (logics) with “built-in” operations. In that case, embedded model expansion has to be considered, where the embedding is into an infinite structure interpreting, e.g., built-in arithmetical operations (Ternovska and Mitchell 2009; Tasharrofi and Ternovska 2010a).

Operational View Due to structural operational semantics, a new type of behaviour equivalence (bisimulation) can be defined on complex modules (e.g. represented by ASP programs). The operational view enables us to obtain results about our modular systems such as approximability of a sub-class of modular systems. While this operational view is novel and we have not developed it very much, we believe that this view allows one to apply the extensive research on properties of transition systems and the techniques developed in the situation calculus to prove useful facts about transition systems. We can do e.g. verification of correct behaviour, static or dynamic, particularly in the presence of arithmetic. The mathematical abstraction we proposed allows one to approach solving the problem of synthesis of modular systems abstractly, similarly to (Giacomo, Patrizi, and Sardina 2013). Just as a Golog program can be synthesized from a library of available programs, a modular system can be synthesized from a library of available solutions to MX tasks.

Related Work Our work on modularity was initially inspired by (Jarvisalo et al. 2009) who developed a constraint-based modularity formalism, where modules were represented by constraints and combined through operations of sequential composition and projection. A detailed comparison with their work was given in (Tasharrofi and Ternovska 2011).

The connections with the related formalism of Multi-Context Systems (MCSs), see (Brewka and Eiter 2007) and consequent papers, has been formally studied in (Tasharrofi 2013) and (Tasharrofi and Ternovska 2014). We only mention here that while the contexts are very general, and may have any semantics, not necessarily model-theoretic, the communication between knowledge bases happens through rules of a specific kind, that are essentially rules of logic programs with negation as failure. We, on the other hand, have chosen to represent communication simply through equality of vocabulary symbols, and to develop a model-theoretic algebra of modular systems.

Splitting results in logic programming (ASP) give conditions for separating a program into modules (Turner 1996).
The results rely on a specific semantics, but can be used for separating programs into modules to represent in our formalism. The same applies to modularity of inductive definitions [Denecker and Ternovska 2008] [Vennekens, Gilis, and Denecker 2006] [Denecker and Ter-novska 2004].

The Generate-Define-Test parts of Answer Set Programs, as discussed in [Denecker et al. 2012], are naturally representable as a sequential composition of the corresponding modules.

A recent work is [Lierler and Truszcynski 2014], where the authors introduce an abstract approach to modular inference systems and solvers was already mentioned, and is used in this paper.

**The Algebra of Modular Systems**

Each modular system abstractly represents an MX task, i.e., a set (or class) of structures over some instance (input) and expansion (output) vocabulary. Intuitively, a modular system is described as a set of primitive modules (individual MX tasks) combined using the operations of:

1. **Projection** $(\pi_{\nu}(M))$ which restricts the vocabulary of a module. Intuitively, the projection operator on $M$ defines a modular system that acts as $M$ internally but where some vocabulary symbols are hidden from the outside.
2. **Composition** $(M_1 \circ M_2)$ which connects outputs of $M_1$ to inputs of $M_2$. As its name suggests, the composition operator is intended to take two modular systems and defines a multi-step operation by serially composing $M_1$ and $M_2$.
3. **Union** $(M_1 \cup M_2)$ which, intuitively, models the case when we have two alternatives to do a task (that we can choose from).
4. **Feedback** $(M[R = S])$ which connects output $S$ of $M$ to its inputs $R$. As its name suggests, the feedback operator models systems with feedbacks or loops. Intuitively, feedbacks represent fixpoints (not necessarily minimal) of modules viewed as operators, since they state that some outputs must be equal to some inputs.
5. **Complementation** $(\overline{M})$ which does “the opposite” of what $M$ does.

These operations are similar to the operations of Codd’s relational algebra, but they work on sets of structures instead of relational tables. Thus, our algebra can be viewed as a higher-order counterpart of Codd’s algebra, with loops. One can introduce other operations, e.g. as combinations of the ones above. The algebra of modular systems is formally defined recursively starting from primitive modules.

**Definition 1 (Primitive Module)** A primitive module $M$ is a model expansion task (or, equivalently, a class of structures) with distinct instance (input) vocabulary $\sigma$ and expansion (output) vocabulary $\varepsilon$.

A primitive module $M$ can be given, for example, by a decision procedure $D_M$ that decides membership in $M$. It can also be given by a first- or second-order formula $\phi$. In this case, $M$ is all the models of $\phi$, $M = Mod(\phi)$. It could also be given by an ASP program. In this case, $M$ would be the stable models of the program, $M = StableMod(\phi)$.

**Remark 1** A module $M$ can be given through axiomatizing it by a formula $\phi$ in some logic $\mathcal{L}$ such that $\text{vocab}(\phi) = \sigma \cup \varepsilon \cup \varepsilon$. That is, $\phi$ may contain auxiliary expansion symbols that are different from the output symbols $\varepsilon$ of $M$. (It may not even be possible to axiomatize $M$ in that particular logic $\mathcal{L}$ without using any auxiliary symbols). In this case, we take $M = Mod(\phi)_{(\sigma \cup \varepsilon)}$, the models of $\phi$ restricted to $\sigma \cup \varepsilon$.

**Example 3** For example, formula $\phi$ of Example 7 describes the model expansion task for the problem of Graph 3-colouring. Thus, $\phi$ can be the representation of a module $M_{col}$ with instance vocabulary $\{E\}$ and expansion vocabulary $\{R, G, B\}$.

Before recursively defining our algebraic language, we have to define composable and independent modules [Harvisalo et al. 2009]:

**Definition 2 (Composable, Independent) 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 module dependencies). Independence is needed for the definition of union, both properties, comparability and independence are needed for sequential composition, non-empty $\sigma$ is needed for feedback.

**Definition 3 (Well-Formed Modular Systems (MS($\sigma, \varepsilon$)))** The set of all well-formed modular systems $\text{MS}($($\sigma, \varepsilon$)$)$ for a given input, $\sigma$, and output, $\varepsilon$, vocabularies is defined as follows.

**Base Case, Primitive Modules:** If $M$ is a primitive module with instance (input) vocabulary $\sigma$ and expansion (output) vocabulary $\varepsilon$, then $M \in \text{MS}($($\sigma, \varepsilon$)$)$.

**Projection** If $M \in \text{MS}($($\sigma, \varepsilon$)$)$ and $\tau \subseteq \sigma \cup \varepsilon$, then $\pi_{\nu}(M) \in \text{MS}($($\sigma \cap \tau, \varepsilon \cap \tau$)$)$.

**Sequential Composition:** If $M \in \text{MS}($($\sigma, \varepsilon$)$)$, $M' \in \text{MS}($($\sigma', \varepsilon'$)$)$, $M$ is composable (no output interference) with $M'$, and $M$ is independent from $M'$ (no cyclic dependencies) then $(M \circ M') \in \text{MS}($($\sigma \cup \varepsilon', \varepsilon \cup \varepsilon'$)$)$.

**Union:** If $M \in \text{MS}($($\sigma, \varepsilon$)$)$, $M' \in \text{MS}($($\sigma', \varepsilon'$)$)$, $M$ is independent from $M'$, and $M'$ is also independent from $M$ then $(M \cup M') \in \text{MS}($($\sigma \cup \sigma', \varepsilon \cup \varepsilon'$)$)$.

**Feedback:** If $M \in \text{MS}($($\sigma, \varepsilon$)$)$, $R \in \sigma$, $S \in \varepsilon$, and $R$ and $S$ are symbols of the same type and arity, then $M[R = S] \in \text{MS}($($\sigma \setminus \{R\}, \varepsilon \cup \{R\}$)$)$.

**Complementation:** If $M \in \text{MS}($($\sigma, \varepsilon$)$)$, then $\overline{M} \in \text{MS}($($\sigma, \varepsilon$)$)$.

Nothing else is in the set $\text{MS}($($\sigma, \varepsilon$)$)$.

Note that the feedback (loop) operator is not defined for the case $\sigma = \emptyset$. However, composition with a module that selects structures where interpretations of two expansion predicates are equal is always possible. The feedback operator was introduced because loops are important in information propagation, e.g. in all software systems and in solvers (e.g. ILP, ASP-CP, DPLL(T)-based) [Tasharrof, Wu, and Ternovska 2011] [Tasharrof, Wu, and Ternovska 2012]. Feedback operation converts an instance predicate to an expansion predicate, and equates it to another expansion predicate.
Feedbacks are, in a sense, fixpoints, not necessarily minimal. They add expressive power to the algebra of modular systems through introducing additional non-determinism, which is not achieved by equating two expansion predicates. We discuss this issue again after the multi-language logic of modular systems is introduced.

The input-output vocabulary of module $M$ is denoted $\text{vocab}(M)$. Modules have “hidden” vocabulary symbols, see Remark 1.

The description of a modular system (as in Definition 3) gives an algebraic formula representing a system. Subsystems of a modular system $M$ are sub-formulas of the formula that represents $M$. Clearly, each subsystem of a modular system is a modular system itself.

**Example 4 (Simple Modular System)** Consider the following axiomatizations of module $M$ in the corresponding logic $\mathcal{L}_i$.

$$
P_{M_1} := \{L_{WF} : a \leftarrow b\},$$
$$P_{M_2} := \{L_{WF} : a \leftarrow c\},$$
$$P_{M_3} := \{L_{SM} : d \leftarrow \text{not } a\},$$
$$P_{M_4} := \{L_P : b' \vee c' \equiv \neg d\}.$$

$L_{WF}$ is the logic of logic programs under the well-founded semantics, $L_{SM}$ is the logic of logic programs under the stable model semantics, $L_P$ is propositional logic.

The modular system in Figure 2 is represented by the following algebraic specification.

$M := \pi_{\{a,b,c,d\}}(((\{M_1 \cup M_2\} \triangleright M_3) \triangleright M_4)[c = c'][\lnot b = b']).$

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 $M'' := M'[c = c'][\lnot b = b']$, which turns instance symbols $b$ and $c$ into expansion symbols, so we have $\sigma_{M''} = \emptyset$ and $\varepsilon_{M''} = \{a,b',c',d\}$, and in addition, 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 denoted by the box with dotted borders. 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, see Remark 7.

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.

The paper [Tasharrofi and Ternovska 2011] contains a more applied example, of a business process planner, where each module represents a business partner.

**Multi-Language Logic of Modular Systems** It is possible to introduce a multi-language logic of modular systems, where formulas of different languages are combined using conjunctions (standing for $\triangleright$), disjunctions ($\cup$), existential second-order quantification ($\pi_x$), etc. For example, model expansion for the following formula

$$\phi_M := \exists b' \exists c' (((\{L_{WF} : a \leftarrow b\} \cup \{L_{WF} : a \leftarrow c\})$$
$$\land \{L_{SM} : d \leftarrow \text{not } a\} \land \{L_P : b' \land \text{not } a\})$$
$$\lor [\lnot b = b' \land c = c']$$

with $\sigma_M = \emptyset$ and $\varepsilon = \{a,b,c,d\}$ and “hidden” (auxiliary, see Remark 1) vocabulary $\varepsilon_a = \{b',c'\}$ corresponds to the modular system in Figure 2 from Example 4.

Feedback is a meta-logic operation that does not have a counterpart among logic connectives. Feedback does not exist for model generation ($\sigma = \emptyset$) and increases the number of symbols in the expansion vocabulary. In our example, former instance symbols ($b$ and $c$ in this case) become expansion symbols, and become equal to the outputs $b'$ and $c'$ thus forming loops.

Note also that projections (thus quantifiers) over variables ranging over domain objects can be achieved if such variables are considered to be a part of the vocabularies of modules. In this logic, the full version of ID-logic, for example, would correspond to the case without feedbacks and all modules limited to either those axiomatized in first-order logic or definitions under well-founded semantics. A formal study of such a multi-language logic in connection with existing KR formalisms (such as, e.g. ID-logic, combinations such as ASP and Description logic, etc.) is left as a future research direction.

Note that if all modules are axiomatized in second-order logic, our task is just model expansion for classic second-order logic that is naturally expressible by adding existential second-order quantifiers at the front. If there are multiple languages, we can talk about the complexity of model expansion for the combined formula (or modular system) as a function of the expressiveness of the individual languages, which is a study of practical importance.

---

2Modular systems under supported semantics [Tasharrofi 2013] allow one to focus on minimal models.

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

4It will be clear from the semantics that the operation $\triangleright$ is commutative.
Model-Theoretic Semantics

So far, we introduced the syntax of the algebraic language using the notion of a well-formed modular system. Those are primitive modules (that are sets of structures) or are constructed inductively by the algebraic operations of composition, union, projection, loop. Model-theoretic semantics associates, with each modular system, a set of structures. Each such structure is called a model of that modular system. Let us assume that the domains of all modules are included in a (potentially infinite) universal domain \( U \).

**Definition 4 (Models of a Modular System)** Let \( M \in MS(\sigma, \varepsilon) \) be a modular system and \( B \) be a \((\sigma \cup \varepsilon)\)-structure. We construct the set \( \text{Mod}(M) \) of models of module \( M \) under model-theoretic semantics recursively, by structural induction on the structure of a module.

**Base Case, Primitive Module**: \( B \) is a model of \( M \) if \( B \in M \).

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

**Composition**: \( B \) is a model of \( M := M_1 \triangleright M_2 \) (with \( M_1 \in MS(\sigma_1, \varepsilon_1) \) and \( M_2 \in MS(\sigma_2, \varepsilon_2) \)) if \( B_1(\sigma_1 \cup \varepsilon_1) \) is a model of \( M_1 \) and \( B_2(\sigma_2 \cup \varepsilon_2) \) is a model of \( M_2 \).

**Union**: \( B \) is a model of \( M := M_1 \cup M_2 \) (with \( M_1 \in MS(\sigma_1, \varepsilon_1) \) and \( M_2 \in MS(\sigma_2, \varepsilon_2) \)) if either \( B_1(\sigma_1 \cup \varepsilon_1) \) is a model of \( M_1 \), or \( B_2(\sigma_2 \cup \varepsilon_2) \) is a model of \( M_2 \).

**Feedback**: \( B \) is a model of \( M := M[R = S] \) (with \( M' \in MS(\sigma', \varepsilon') \)) if \( R^B = S^B \) and \( B \) is model of \( M' \). Nothing else is derivable.

Note that, by this semantics, sequential composition is a commutative operation (one could have used \( \bowtie \) notation), however the direction of information propagation is uniquely given by the separations of the input and output vocabularies. Notice that it’s not possible to compose two modules in different ways. If it was possible, then in the compound module we would had that the intersection of the input and the output vocabularies would not be empty, and this is not allowed. So, we prefer to use \( \triangleright \) instead of \( \bowtie \) for both historic and mnemonic reasons, and encourage the reader to write algebraic formulas in a way that corresponds to their visualizations of the corresponding modular systems.

An example illustrating the semantics of the feedback operator, as well as non-determinism introduced by this operator is given in the appendix.

The task of model expansion for modular system \( M \) takes a \( \sigma \)-structure \( A \) and finds (or reports that none exists) a \((\sigma \cup \varepsilon)\)-structure \( B \) that expands \( A \) and is a model of \( M \). Such a structure \( B \) is a solution of \( M \) for input \( A \).

**Remark 2** The semantics does not put any finiteness restriction on the domains of structures. Thus, the framework works for modules with infinite structures.

Structural Operational Semantics

In this section, we introduce a novel Structural Operational Semantics of modular systems.

We now focus on potentially infinite all-inclusive vocabulary \( \tau \) that subsumes the vocabularies of all modules considered. Thus, we always have \( \text{vocab}(M) \subseteq \tau \).

**Definition 5 (State of a Modular Systems)** A \( \tau \)-state of a modular system \( M \in MS(\sigma, \varepsilon) \) is a \( \tau \)-structure such that \((\sigma \cup \varepsilon) \subseteq \tau \).

The semantics we give is structural because, for example, the meaning of the sequential composition, \( M_1 \triangleright M_2 \), is defined through the meaning of \( M_1 \) and the meaning of \( M_2 \).

**Definition 6 (Modules as Operators)** We say that a well-formed modular system \( M \) (non-deterministically) maps \( \tau \)-state \( B_1 \) to \( \tau \)-state \( B_2 \), notation \( (M, B_1) \rightarrow B_2 \), if we can apply the rules of the structural operational semantics (below) starting from this expression and arriving to \( B_2 \). In that case, we say that transition \( (M, B_1) \rightarrow B_2 \) is derivable.

**Primitive modules** \( M \):

\[
(M, B_1) \rightarrow B_2 \quad \text{true}
\]

if \( B_2(\sigma \cup \varepsilon) \in M \) and \( B_2(\tau \setminus \varepsilon) = B_1(\tau \setminus \varepsilon) \).

We proceed by induction on the structure of modular system \( M \). 

**Projection** \( \pi_\nu(M) \):

\[
(M, B_1) \rightarrow B_2 \quad \text{if } B'_1 |_\nu = B_1 |_\nu \text{ and } B'_2 |_\nu = B_2 |_\nu.
\]

**Composition** \( M_1 \triangleright M_2 \):

\[
(M_1 \triangleright M_2, B_1) \rightarrow B_2
\]

**Union** \( M_1 \cup M_2 \):

\[
(M_1 \cup M_2, B_1) \rightarrow B_2 \quad \text{if } \nu \in (\sigma \cup \varepsilon) \text{ and } \nu \in (\tau \setminus \varepsilon).
\]

**Feedback** \( M[R = S] \):

\[
(M[R = S], B_1) \rightarrow B_2 \quad \text{true}
\]

if \( R^B_1 = S^B_2 \).

**Complementation** \( \overline{M} \):

\[
(\overline{M}, B_1) \rightarrow B_2 \quad \text{true}
\]

if \( (M, B_1) \rightarrow B_2 \) is not derivable.

Nothing else is derivable.

Let us clarify the projection operation \( \pi_\nu(M) \). 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 \( \tau \)-structure \( B_1 \). It (a) expands \( \sigma \)-part of \( B_1 \) to \( \sigma' \) 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 \( \varepsilon \), ignoring everything else, (d) the interpretations of \( \tau \setminus \varepsilon \) are moved from \( B_1 \) by inertia.

**Definition 7 (Operational Semantics)** Let \( M \) be a well-formed modular system in \( MS(\sigma, \varepsilon) \). The semantics of \( M \) is given by the following set.

\[
M^\text{op} := \{ B | (B_1, M) \rightarrow B_2 \text{ and } B |_\sigma = B_1 |_\sigma, B |_\varepsilon = B_2 |_\varepsilon \}.
\]
From now on, by vice versa). Thus, we may use either of these semantics. Still hold when modules are viewed as sets of structures (and its model-theoretic and operational semantics coincide, by inertia, the interpretation of ε is already changed by M, nothing is to be changed, and (M, B2) → B2.

**Theorem 1 (Operational = Model-theoretic Semantics)**

Let M be a well-formed modular system in MS(σ, ε). Then, its model-theoretic and operational semantics coincide,

\[ M^{\text{mt}} = M^{\text{op}}. \]

The most important consequence of this theorem is that all the results obtained when modules are viewed as operators, still hold when modules are viewed as sets of structures (and vice versa). Thus, we may use either of these semantics. From now on, by M we mean either one of these sets M^{\text{mt}} or M^{\text{op}}.

**Proof:** We prove the statement inductively.

**Base case, primitive module** By definition, model-theoretically, B is a model of M if B ∈ M. On the other hand, operationally,

\[ (M, B_1) \rightarrow B_2 \]

is true if \( B_2|_{(\sigma \cup \varepsilon)} \in M \) and \( B_2|_{(\tau \setminus \varepsilon)} = B_1|_{(\tau \setminus \varepsilon)}. \)

Thus, B ∈ M, and the two semantics coincide for primitive modules.

Our inductive hypothesis is that the statement of the theorem holds for M1, M2 and M'. We proceed inductively.

**Projection** M := \( \pi_\nu(M') \). By the hypothesis, \( (M')^{\text{mt}} = (M')^{\text{op}} \), where \( (M')^{\text{op}} \) is constructed "from pieces", M^{\text{op}} := \{ B | (B_1, M') → B_2 and \( B|_\sigma = B_1|_\sigma, B|_\varepsilon = B_2|_\varepsilon \} \).

We apply the rule

\[ (\pi_\nu(M'), B_1) \rightarrow B_2 \]

if \( B_1|_\nu = B_1|_\nu \) and \( B_2|_\nu = B_2|_\nu \)

and obtain that \( (\pi_\nu(M'), B_1) \rightarrow B_2 \) where B1 and B2 are just like B1' and B2' on the vocabulary \( \nu \). Now, M := \( \pi_\nu(M') \) is constructed "from \( \sigma \) and \( \varepsilon \) pieces" of B1 and B2, respectively (where \( \nu = \sigma \cup \varepsilon \)):

\[ M^{\text{op}} := \{ B | (B_1, M) → B_2 and B|_\sigma = B_1|_\sigma, B|_\varepsilon = B_2|_\varepsilon \}. \]

On the other hand, model-theoretically, B is a model of M := \( \pi_{(\sigma \cup \varepsilon)}(M') \) (with M' ∈ MS(\( \sigma', \varepsilon' \))) if a \( (\sigma' \cup \varepsilon') \)-structure B' exists such that B' is a model of M' and B' expands B, which makes the two semantics equal for projection, \( (M)^{\text{mt}} = (M)^{\text{op}} \).

We omit the proofs for the other inductive cases.

**Applications of Operational View** We now discuss how the operational semantics can be used. For example, we can consider modular systems at various levels of granularity. We might be interested in the following question: if M gives a transition from a structure B to structures B', then what are the transitions given by the subsystems of M? While answering this question in its full generality is algorithmically impossible, we may study the question of whether a particular transition by a subsystem exists. To answer it, one has to start from the system and build down to the subsystem using the rules of the structural operation semantics. Reasoning about subsystems of a modular system can be useful in business process modelling. Suppose a particular transition should hold for the entire process. This might be the global task of an organization. In order to make that transition, the subsystems have to perform their own transitions. Those transitions are derivable using the rules of structural operational semantics.

**Complexity** In the following proposition, we assume a standard encoding of structures as binary strings as is common in Descriptive complexity (Immerman 1982). Note that if M is deterministic, it is polytime in the size of the encoding of the input structure. This is because the domain remains the same, the arities of the relations in ε are fixed, so we need \( (n^k) \) steps to construct new interpretations of ε, and move the remaining relations.

**Proposition 1** Let M be a module that performs a (deterministic) polytime computation. Projection \( \pi_\nu(M) \) increases the complexity of M from P to NP. More generally, for an operator M on the k-th level of the Polynomial Time hierarchy (PH), projection can increase the complexity of M from \( \Delta^p_k \) to \( \Sigma^p_{k+1} \).

**Proof:** We will show the property for the jump from P to NP, for illustration. The proof generalizes to all levels of PH. Let M takes an instance of an NP-complete problem, such as a graph in 3-Colourability, encoded in \( \sigma_G \), and what it means to be 3-Colourable, as a formula encoded in the interpretation of \( \sigma_\phi \), and returns an instance of SAT encoded in ε, a CNF formula that is satisfiable if and only if the graph is 3-Colourable, and a yes/no answer bit represented by ε-answer.
Thus, $M$ performs a deterministic (thus, polytime) reduction. Consider $\pi_\nu(M)$, where $\nu = \sigma_G \cup \varepsilon_{\text{answer}}$. This module takes a graph and returns a yes or no answer depending on whether the graph is 3-colourable. Thus, $\pi_\nu(M)$ solves an NP-complete problem. Union and feedback change the complexity as well.

Inference Semantics of Modular Systems

In modular systems, each agent or a knowledge base can have its own way of reasoning, that can be formulated through inferences or propagations. To define inferential semantics for modular systems, we closely follow (Lierler and Truszczynski 2014). Since input/output is not considered by the authors, their case corresponds to the instance vocabulary being empty, $\sigma = \emptyset$, i.e., model generation, and can be viewed as an analysis of the after-grounding phase. Since we want to separate problem descriptions and their instances (and reuse problem descriptions), as well as to define additional algebraic operations (the authors consider conjunctions only), we need to allow $\sigma \neq \emptyset$, and present inferences on partial structures. This is not hard however.

We start by assuming that there is a constant for every element of the domains. We view structures as sets of ground atoms. We now closely follow and generalize the definitions of (Lierler and Truszczynski 2014) from sets of propositional atoms to first-order structures, to establish a connection to the Modular Systems framework presented above. The propositional case then corresponds to structures over the domain $\{\langle \rangle\}$ containing the empty tuple that interprets propositional symbols that are true.

Let a fixed countably infinite set of ground atoms $\tau$ be given. We use $\text{Lit}(\tau)$ to denote the set of all literals over $\tau$. For $S \subseteq \text{Lit}(\tau)$:

- $S^+: = \tau \cap S$
- $S^- := \{a \in \tau \mid \neg a \in S\}$
- $l \in \text{Lit}(\tau)$ is unassigned in $S$ if $l \notin M$ and $\bar{l} \notin S$
- $S$ is consistent if $S^+ \cap S^- \neq \emptyset$
- Let $C(\tau)$ be all consistent subsets of $\text{Lit}(\tau)$.

Definition 8 (Abstract Inference Representation of $M$)

An abstract inference representation $M^\tau$ of module $M$ over a vocabulary $\tau$ is a finite set of pairs of the form $(S, l)$, where $S \in C(\tau)$, $l \in \text{Lit}(\tau)$, and $l \notin \text{Lit}(\tau)$. Such pairs are called inferences of the module $M$.

In the exposition below, we view structures as sets of propositional atoms, $B \subseteq \tau$.

- $S$ is consistent with $B \subseteq \tau$ if $S^+ \subseteq B$ and $S^- \cap B = \emptyset$.
- Literal $l$ is consistent with $B \subseteq \tau$ if $\{l\}$ is consistent with $B$.

Definition 9 (Primitive Module, Inferential Semantics)

A primitive module $M \in MS(\sigma, \varepsilon)$ is a set of $(\sigma \cup \varepsilon)$-structures $B$ such that for every inference $(S, l) \in M^\tau$ such as $S$ is consistent with $B$, $l$ is consistent with $B$, too.

Thus, primitive modules, even when they are represented through abstract inferences, are sets of structures as before, and the definitions of the algebraic operations do not need to be changed.

The inference framework can be viewed as yet another (very useful) way of representing modules. Since the inference framework is abstract, we cannot prove a correspondence between a given individual module presented as a set of structures or as an operator on one hand and an inferential representation on the other in general, without specifying what inference mechanism is used. However, we can do it for particular cases such as $\text{Ent}(T)$ (Lierler and Truszczynski 2014), which is left for a future paper.

With the inference semantics as described, we can now model problems (sets of instances) rather than single instances as a combination of other problems. This semantics allows one to study the details of propagation of information in the process of constructing solutions to modular systems, through incremental construction of partial structures as in (Tasharrofi, Wu, and Ternovska 2011; Tasharrofi, Wu, and Ternovska 2012), but in more detail. This direction is left for future research.

Conclusion and Future Directions

We described a modular system framework, where primitive and compound modules are sets (classes) of structures, and combinations of modules are achieved by applying algebraic operations that are a higher-order counterpart of Codd’s relational algebra operations. An additional operation is the feedback operator that connects output symbols with the input ones and is used to model information propagation such as loops of software systems and solvers.

We defined two novel semantics of modular systems, operational and inferential, that are equivalent to the original model-theoretic semantics (Tasharrofi and Ternovska 2011). We presented a multi-language logic, a syntactic counterpart of the algebra of modular systems. Minimal models of modular systems are introduced in a separate paper on supported modular systems, see also (Tasharrofi 2013).

The framework of modular systems gives us, through its semantic-based approach, a unifying perspective on multi-language formalisms and solvers. More importantly, it gives rise to a whole new family of multi-language KR formalisms, where new formalisms can be obtained by instantiating specific logics defining individual modules.

The framework can be used for analysis of existing KR languages. In particular, expressiveness and complexity results for combined formalisms can be obtained in a way similar to the previous work (Mitchell and Ternovska 2008; Tasharrofi and Ternovska 2010a), where single-module embedded model expansion was used.

References

[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.

[Denecker and Ternovska 2004] Denecker, M., and Ternovska, E. 2004. Inductive situation calculus. In Proc., KR-04.
presentation languages. In *NonMon at 30 (Thirty Years of Nonmonotonic Reasoning)*.

[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 2014] Tasharrofi, S., and Ternovska, E. 2014. Generalized multi-context systems. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR2014)*.

[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.

[Ternovska and Mitchell 2009] Ternovska, E., and Mitchell, D. G. 2009. Declarative programming of search problems with built-in arithmetic. In *Proc. of IJCAI*, 942–947.

[Turner 1996] Turner, H. 1996. Splitting a default theory. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI’96)* - Volume 1, 645–651. AAAI Press.

[Vennekens, Gilis, and Denecker 2006] Vennekens, J.; Gilis, D.; and Denecker, M. 2006. Splitting an operator: Algebraic modularity results for logics with fixpoint semantics. *ACM Transactions on Computational Logic* 7(4):765–797.

**Appendix**

**Example 5** We illustrate models of a simple modular system with feedback operator. Consider the following axiomatization $P_{M_0}$ of a primitive module $M_0$, where $\sigma_{M_0} = \{i\}$ and $\varepsilon_{M_0} = \{a, b\}$.

$$P_{M_0} := \{ L_{SM} : \begin{array}{ll} a \leftarrow i, & \text{not } b, \\ b \leftarrow i, & \text{not } a. \end{array} \}$$

We will demonstrate how the set of models of this program changes when we use the feedback operator. When the input $i$ is true (given by the corresponding instance structure), then

$$\text{StableMod}(P_{M_0}, i = \text{true}) = \{\{a\}, \{b\}\}.$$  

When $i$ is false, there is one model, where everything is false,

$$\text{StableMod}(P_{M_0}, i = \text{false}) = \{\emptyset\}.$$  

Module $M_0$ is the set of structures for the entire $\sigma_{M_0} \cup \varepsilon_{M_0}$ vocabulary. Since we are dealing with a propositional case,
each structure is represented by a set of atoms that are true in that structure.

\[ M_0 = \{\{i, a\}, \{i, b\}, \emptyset\}. \]

Now consider a different module, \( M_1 \), with \( \sigma_{M_1} = \{i, a, b\} \) and \( \varepsilon_{M_1} = \{a', b'\} \), axiomatized by

\[ P_{M_1} := \{ L_{SM} : \ a' \leftarrow i, \text{not } b, \ b' \leftarrow i, \text{not } a. \} \]

This modular system is deterministic, – for each input (each of the eight possible interpretations of \( i, a \) and \( b \)), there is at most one model.

\[
\begin{array}{c|c|c|c}
\text{Models of } M_1 \\
\hline
i & a & b \\
\hline
\bot & \bot & \bot & \{\emptyset\} \\
\bot & \bot & \bot & \{\emptyset\} \\
\bot & \bot & \bot & \{\emptyset\} \\
\bot & \bot & \bot & \{\emptyset\} \\
\top & \bot & \bot & \{\{i, a', b'\}\} \\
\top & \bot & \bot & \{\{i, a, a'\}\} \\
\top & \bot & \bot & \{\{i, b, b'\}\} \\
\top & \bot & \bot & \{\{i, a, b\}\} \\
\end{array}
\]

Thus, we have:

\[ M_1 = \{\emptyset, \{i, a', b'\}, \{i, a, a'\}, \{i, b, b'\}, \{i, a, b\}\}. \]

If we add feedback, we obtain the following system \( M_2 = M_1[a = a'][b = b'] \). Its input is \( i \), all other symbols are in the expansion vocabulary. The models are:

\[
\begin{array}{c|c|c}
\text{Models of } M_2 \\
\hline
i \\
\hline
\bot & \{\emptyset\} \\
\bot & \{\emptyset\} \\
\top & \{\{i, a, a'\}\} \\
\top & \{\{i, b, b'\}\} \\
\end{array}
\]

\[ M_2 = M_1[a = a'][b = b'] = \{\emptyset, \{i, a, a'\}, \{i, b, b'\}\}. \]

As we see here, after adding feedback, for the same input \( i \), we obtain two different models. Thus, by means of feedback, a deterministic system \( M_1 \) was turned into a non-deterministic system \( M_2 \).

This modular system is deterministic, – for each input (each of the eight possible interpretations of \( i, a \) and \( b \)), there is at most one model. Notice also that

\[ \pi_{\{i, a, b\}}(M_1[a = a'][b = b']) = M_0. \]