Recent Progress on the Algebra of Modular Systems

Eugenia Ternovska

Simon Fraser University

Abstract. In this short paper, we present some recent and ongoing work on an algebra of modular system aiming at combining large pieces of information.

1 Introduction

Reusing components and services and linking data is important in Computer Science. The essence of our proposal is that we can use first-order logic as a versatile language for applying and combining *modules* – which are classes of structures – web services, declarative specifications with associated solvers, Integer Liner Programs, Constraint Satisfaction Problems etc.[1] While the syntax of our formalism is first-order, the semantics is second-order because variables range over relations. We use a version of Codd relational algebra instead of first-order logic, but the idea is the same. We also add least fixed points. Since most practical systems use inputs and outputs, or directionality in linking data, we introduce an algebra with information flow. Operations such as while loops, if-then-else, reachability, regular expressions over data links, etc. are definable.[2]

The goal of this short paper is to present the two algebras and some of the recent results and ongoing work, namely (1) on the complexity of the evaluation task, (2) on modular system equivalence and containment and (3) on solving modular systems using an algebra of propagators. While item (3) summarizes [1], items (1) and (2) is an ongoing and yet unpublished work. The presentation of the syntax and semantics is also new and improves over previous versions, e.g. [3].

2 Algebras

For exactly the same syntax, we produce two algebras, static and dynamic, by giving different interpretations to the algebraic operations and to the elements of the algebras. In the second algebra, atomic modules have a direction of information propagation. The algebras correspond to classical and modal logics, respectively.

Syntax[3] Assume we have a countable sequence $\mathrm{Vars} = (X_1, X_2, \dots)$ of *relational variables* each with an associated finite *arity*. For convenience, we use X, Y, Z, etc. Let $\mathrm{ModAt} = \{M_1, M_2, \dots\}$ be a fixed vocabulary of *atomic module symbols*. Each $M_i \in \mathrm{ModAt}$ has an associated *variable vocabulary* $vvoc(M_i)$ whose length can depend on M_i. We may write $M_i(X_{i_1}, \dots, X_{i_k})$, (or $M_i(\bar{X})$), to indicate that $vvoc(M) =$

[1] In database terminology, a module is a boolean query.

[2] We call the logic counterpart of the second algebra a Logic of Information Flow.

[3] Thanks to Brett McLean, Bart Bogaerts and anonymous referees of the previous versions whose comments helped to improve the presentation.

$(X_{i_1}, \ldots, X_{i_k})$. Similarly, $\mathrm{ModVars} = \{Z_1, Z_2, \ldots\}$ is a countable sequence of *module variables*, where each $Z_j \in \mathrm{ModVars}$ has its own $vvoc(Z_j)$. Algebraic expressions are built by the grammar:

$$\alpha ::= \mathrm{id} \mid M_i \mid Z_j \mid \alpha \cup \alpha \mid -\alpha \mid \pi_\delta(\alpha) \mid \sigma_\Theta(\alpha) \mid \mu Z_j.\alpha. \tag{1}$$

Here, M_i is any symbol in ModAt, δ is any finite set of relational variables in Vars, Θ is any expression of the form $X \equiv Y$, for relational variables of equal arity that occur in α, Z_j is a module variable in $\mathrm{ModVars}$ which must occur positively in the expression α, i.e., under an even number of the complementation $(-)$ operator.

Static (Unary) Semantics Fix a finite relational vocabulary τ. A *variable assignment* s is a function that assigns, to each relational variable, a symbol in τ of the same arity. Now fix a domain Dom.[4] Let $\mathbf{U}$ be the set of all τ-structures over the domain Dom. Given a sub-vocabulary γ of τ, a subset $V \subseteq \mathbf{U}$ is *determined by* γ if it satisfies: for all $\mathcal{A}, \mathcal{B} \in \mathbf{U}$ such that $\mathcal{A}|_\gamma = \mathcal{B}|_\gamma$ we have $\mathcal{A} \in V$ iff $\mathcal{B} \in V$.

Given a well-formed algebraic expression α defined by (1), we say that structure $\mathcal{A}$ *satisfies* α (or that is a *model* of α) under variable assignment s, notation $\mathcal{A} \models_s \alpha$, if $\mathcal{A} \in [\alpha]$, where *unary interpretation* $[\,\cdot\,]$ is defined as follows. Given a variable assignment s, function $[\,\cdot\,]$ assigns a subset $[M_i] \subseteq \mathbf{U}$ and a subset $[Z_j] \subseteq \mathbf{U}$ to each $M_i \in \mathrm{ModAt}$ and each $Z_j \in \mathrm{ModVars}$, with the property that $[M_i]$ is determined by $s(vvoc(M_i))$ (respectively, $[Z_j]$ is determined by $s(vvoc(Z_j))$). We extend the definition of $[\,\cdot\,]$ to all algebraic expressions.

$$\begin{aligned}
&[\mathrm{id}] := \mathbf{U},\\
&[\alpha_1 \cup \alpha_2] := [\alpha_1] \cup [\alpha_2],\\
&[-\alpha] := \mathbf{U} \backslash [\alpha],\\
&[\pi_\delta(\alpha)] := \{\mathcal{A} \in \mathbf{U} \mid \exists \mathcal{B}\, (\mathcal{B} \in [\alpha] \text{ and } \mathcal{A}|_{\tau \backslash s(\delta)} = \mathcal{B}|_{\tau \backslash s(\delta)})\},\\
&[\sigma_{X \equiv Y}(\alpha)] := \{\mathcal{A} \mid \mathcal{A} \in [\alpha] \text{ and } \mathcal{A}|_{s(X)} = \mathcal{B}|_{s(Y)}\},\\
&[\mu Z_j.\alpha] := \bigcap \{R \subseteq \mathbf{U} \mid [\alpha]^{[Z:=\alpha]} \subseteq R\}.
\end{aligned}$$

Here, $[\alpha]^{[Z:=\alpha]}$ means an interpretation that is exactly like $[\,\cdot\,]$, except Z is interpreted as α. Intuitively, to check whether a τ-structure $\mathcal{A}$ satisfies module M_i, we need to first select predicate symbols $S_{i_1}, \ldots, S_{i_k}$ from τ, whose arities match those of $X_1, \ldots, X_k$, which is done by function s, and then "apply" the module to $S^{\mathcal{A}}_{i_1}, \ldots, S^{\mathcal{A}}_{i_k}$, as we would apply a decision procedure or an "oracle".

Example 1. Let $M_{\mathrm{HC}}(V, X, Y)$ and $M_{2\mathrm{Col}}(V, X, Z, T)$ be atomic modules "computing" a Hamiltonian Circuit and a 2-Colouring. For example, M_{HC} can be represented as an Answer Set Programming program, and $M_{2\mathrm{Col}}$ be an imperative program or a human child with two pencils. The first module decides if Y forms a Hamiltonian Circuit (represented as a set of edges) in the graph given by vertex set V and edge set X. The second module decides if unary relations Z, T specify a proper 2-colouring of the graph. The following expression determines whether or not there is a 2-colourable Hamiltonian Circuit. $M_{2\mathrm{Col}-\mathrm{HC}}(V, X, Z, T) := \pi_{V,X,Z,T}((M_{\mathrm{HC}}(V, X, Y) \cap M_{2\mathrm{Col}}(V, Y, Z, T))$. The need for recursion is illustrated by a much longer specification of a dynamic programming algorithm on tree decompositions [3].

[4] Usually, in applications, domain Dom is the (active) domain of an input structure for a task of interest.

Dynamic (Binary) Semantics Let $\mathrm{ModAt}_{\mathrm{I/O}}$ denote the set of all atomic module symbols M with all possible *partitions* of $vvoc(M)$ into inputs, $I(M)$, and outputs, $O(M)$.[5] This set is larger than the set ModAt (unless both are empty) because the same M can have several different input-output assignments. Similarly, we define $\mathrm{ModVars}_{\mathrm{I/O}}$. Given a well-formed α, we say that pair of structures $(\mathcal{A}, \mathcal{B})$, *satisfies* α under variable assignment s, notation $(\mathcal{A}, \mathcal{B}) \models_s \alpha$, if $(\mathcal{A}, \mathcal{B}) \in [\![\alpha]\!]$, where *binary interpretation* $[\![\cdot]\!]$ is defined as follows. $[\![\mathrm{id}]\!] := \mathbf{U} \times \mathbf{U}$. For atomic modules in $\mathrm{ModAt}_{\mathrm{I/O}}$, we have:

$$[\![M]\!] := \{(\mathcal{A}, \mathcal{B}) \in \mathbf{U} \times \mathbf{U} \mid \mathcal{A}|_{\tau \setminus s(O(M))} = \mathcal{B}|_{\tau \setminus s(O(M))} \text{ and } \mathcal{B} \in [M]\}. \quad (2)$$

Similarly, for $Z \in \mathrm{ModVars}_{\mathrm{I/O}}$. Intuitively, atomic modules produce a replica of the current database except the interpretation of the output vocabulary changes as specified by the action.[6] An illustration of the binary semantics for atomic modules is given in an example below. We extend the binary interpretation $[\![\cdot]\!]$ to all expressions α:

$$\begin{array}{l}
[\![\alpha_1 \cup \alpha_2]\!] := [\![\alpha_1]\!] \cup [\![\alpha_2]\!], \\
[\![-\alpha_2]\!] := \mathbf{U} \times \mathbf{U} \setminus [\![\alpha]\!], \\
[\![\mu Z_j.\alpha]\!] := \bigcap \{R \subseteq \mathbf{U} \times \mathbf{U} \,:\, [\![\alpha]\!]^{[Z:=R]} \subseteq R\}, \\
[\![\pi_\delta(\alpha)]\!] := \{(\mathcal{A}, \mathcal{B}) \in \mathbf{U} \times \mathbf{U} \mid \exists\, (\mathcal{A}', \mathcal{B}') \in [\![\alpha]\!] : \mathcal{A}'|_{\tau \setminus s(\delta)} = \mathcal{A}|_{\tau \setminus s(\delta)} \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{and } \mathcal{B}'|_{\tau \setminus s(\delta)} = \mathcal{B}|_{\tau \setminus s(\delta)}\}, \\
[\![\sigma_{X \equiv Y}(\alpha)]\!] := \left\{ (\mathcal{A}, \mathcal{B}) \in [\![\alpha]\!] \;\middle|\; \begin{array}{l} (s(X))^{\mathcal{A}} = (s(Y))^{\mathcal{A}} \text{ if } \{X, Y\} \subseteq I(\alpha), \\ (s(X))^{\mathcal{B}} = (s(Y))^{\mathcal{B}} \text{ if } \{X, Y\} \subseteq O(\alpha), \\ (s(X))^{\mathcal{A}} = (s(Y))^{\mathcal{B}}) \text{ if } X \in I(\alpha) \text{ and } Y \in O(\alpha). \end{array} \right\}.
\end{array}$$

Example 2. In HC-2Col, in each atomic module, we underline designated input symbols: $\pi_{V,X,Z,T}(M_{\mathrm{HC}}(\underline{V}, \underline{X}, Y) \cap M_{2\mathrm{Col}}(\underline{V}, \underline{Y}, Z, T))$. First, $M_{\mathrm{HC}}(\underline{V}, \underline{X}, Y)$ makes a transition by producing possibly several Hamiltonian Circuits. The interpretation of the output Y changes, everything else is transferred by inertia. Each resulting structure is taken as an input to $M_{2\mathrm{Col}}(\underline{V}, \underline{Y}, Z, T)$, where edges in the cycle, Y, are "fed" to $M_{2\mathrm{Col}}$, although this is hidden from the outside observer. The second module produces non-deterministic transitions, one for each generated colouring, if they exist.

The algebra can be equivalently represented in a "two-sorted" syntax:

$$\begin{array}{c} \alpha ::= \mathrm{id} \mid M_a \mid Z_j \mid \alpha \cup \alpha \mid -\alpha \mid \pi_\delta(\alpha) \mid \sigma_\Theta(\alpha) \mid \phi? \mid \mu Z_j.\alpha \\ \phi ::= \top \mid M_p \mid X_i \mid \phi \vee \phi \mid \neg\phi \mid |\alpha\rangle\, \phi \mid \langle\alpha|\, \phi \mid \mu X_i.\phi. \end{array} \quad (3)$$

The modal logic, which we call $L\mu\mu$ (since we have two fixed points), is interpreted over a transition system where states are τ-structures, M_a represent modules-actions that change states, M_p – modules-propositions that are true in states. Process formulae in the first line are interpreted exactly like in the binary semantic, and tests (i.e., expressions ending with symbol "?") are interpreted as in Dynamic Logic. State formulae in the second line are interpreted exactly like in the μ-calculus.

Example 3. Definable constructs that can appear inside the forward $|\alpha\rangle$ and backwards $\langle\alpha|$ possibility modalities, are, e.g., $\mathbf{if}\ \phi\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta \;:=\; (\phi?;\alpha) \cup (\neg\phi?;\beta)$, $\mathbf{while}\ \phi\ \mathbf{do}\ \alpha \;:=\; (\phi?;\alpha)^*;\neg\phi?$, regular expressions. Here, ';' is sequential composition (a particular case of $\cap$), and $\alpha^* := \mu Z.(\mathrm{id} \cup Z;\alpha)$.

[5] Either one of these sets, $I(\alpha)$, $O(\alpha)$, can be empty.

[6] This is similar to the inertia law for actions in the Situation Calculus.

3 Some Recent and Ongoing Work

(1) Complexity and Expressiveness We consider the task where we give an interpretation to $I(\alpha)$, and ask whether there *exists* an interpretation $O(\alpha)$, such that α is satisfied. We identify "safe" fragments in which the data complexity of this task is essentially the same as the complexity of the modules we start from. Safe (power-*preserving*) operations are, e.g. $\cap$, ;, $\sigma_{X\equiv Y}$, some cases of π_δ and $\cup$ (those that do not add nondeterminism), two definable unary negations, and a restricted version of $*$. If we start with polynomial time atomic modules, some power-*increasing* operations allow us to capture NP, and with the use of a unary negation, all levels of the Polynomial Time Hierarchy. We also characterize other complexity classes. This is yet unpublished work. It extends an earlier initial work on a positive recursion-free fragment of the algebra [2].
(2) Equivalence and Inclusion For compound modules α_1, α_2 with atoms $M_1, \ldots, M_k$, we say that α_1 is *contained* in α_2, denoted $\alpha_1 \sqsubseteq \alpha_2$, if $\mathcal{A} \models_s \alpha_1$ implies $\mathcal{A} \models_s \alpha_2$, for any s and any choice of atoms $M_1, \ldots, M_s$. This is essentially the problem of logical implication. We have $\alpha_1 = \alpha_2$ iff $\alpha_1 \sqsubseteq \alpha_2$ and $\alpha_2 \sqsubseteq \alpha_1$. The task here is, given α_1 and α_2, decide if $\alpha_1 = \alpha_2$ (respectively, $\alpha_1 \sqsubseteq \alpha_2$). We show that equivalence (and thus, also containment) is undecidable in general. We prove that, for a large class of modular systems, namely those specified by conjunctive expressions, containment is in NP. We discuss the conditions under which the containment problem becomes polynomial time solvable. Joint work with Andrei Bulatov. As of now, this work is unpublished.
(3) Solving using an algebra of propagators, CDCL-like algorithm We consider the task of, given an interpretation of $I(\alpha)$, *construct* all interpretations of $O(\alpha)$. We use precision order $\mathbf{u} <_p \mathbf{t} <_p \mathbf{i}$, $\mathbf{u} <_p \mathbf{f} <_p \mathbf{i}$, which is pointwise extended to four-valued (4V) structures: $\mathfrak{A} <_p \mathfrak{A}'$. The set of all 4V structures forms a complete lattice. A propagator is a mapping of 4V structures to 4V structures such that it is $\geqslant_p$-monotone, i.e., $\mathfrak{A} \geqslant_p \mathfrak{A}'$ implies $P(\mathfrak{A}) \geqslant_p P(\mathfrak{A}')$, and is information-preserving, i.e., $P(\mathfrak{A}) \geqslant_p \mathfrak{A}$. A propagator refines a partial (four-valued) structure by deriving consequences of a given module. An α-solver is a procedure that takes as input a 4V structure $\mathfrak{A}$, and whose output is the set of all 2V structures $\mathcal{A}$ with $\mathcal{A} \models \alpha$ and $\mathcal{A} \geqslant_p \mathfrak{A}$. We give examples of how our notion of a propagator generalizes notions from various domains. We define an algebra with the same operators as above, but on propagators, and then on *explaining* propagators. An explaining propagator not only returns the partial structure that is the result of its propagations ($P(\mathfrak{A})$), but also an explanation ($C(\mathfrak{A})$). This explanation takes the form of a propagator itself. Our main algorithm turns such a propagator into a solver that learns from these explanations, similar to the Conflict Driven Clause Learning (CDCL) algorithm, the main algorithm in SAT solving. Our work extends SMT technology to arbitrary modules and to module combinations beyond conjunctive.

References

1. B. Bogaerts, E. Ternovska, and D. Mitchell. Propagators and solvers for the algebra of modular systems. In *Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, 2017.
2. S. Tasharrofi and E. Ternovska. Modular systems. In *Proceedings of Workshop on Hybrid Reasoning at IJCAI'15*, 2015.
3. E. Ternovska. Static and dynamic views on the algebra of modular systems. In *Proc. of the 16th Int. Workshop on Non-Monotonic Reasoning (NMR 2016)*, pages 153–162, 2016.