Algebra of Modular Systems: Containment and Equivalence

Andrei Bulatov, Eugenia Ternovska
Simon Fraser University
abulatov@sfu.ca, ter@sfu.ca

Abstract

The Algebra of Modular System is a KR formalism 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, ILP, CP program, etc. Formally, a module is a class of structures over the same vocabulary. Modules are combined declaratively, using, essentially, operations of Codd's relational algebra.

In this paper, we address the problem of checking modular system containment, which we relate to a homomorphism problem. We prove that, for a large class of modular systems, the containment problem (and thus equivalence) is in the complexity class NP, and therefore is solvable by, e.g., SAT solvers. We discuss conditions under which the problem is polynomial time solvable.

Introduction

Programming from reusable components is one of the main attributes of good software engineering practice. Such a practice reduces the effort, minimizes bugs, saves the cost and time for more high-level development tasks. The theory of combining conventional imperative programs and digital circuits is relatively well developed. However, in knowledge-intensive computing, characterized by using so-called declarative programming, research on *combining heterogeneous components* is not as advanced. It would be very desirable to be able to write new programs by taking, e.g., a program written in Answer Set Programming (ASP), combining it with a specification of a Constraint Satisfaction Problem (CSP), and then also with an Integer Linear Program (ILP), in a compound specification for solving a more complex task. Moreover, the components should be *substitutable*, so that parts can be replaced with an alternative design without breaking the intended functionality.

A central algorithmic problem in the modular setting is that of *modular system equivalence*. Being able to solve the equivalence problem is usually the first step toward developing *optimization techniques*. Moreover, this task is crucial, for example, in system development and rapid prototyping. A related notion is that of modular systems *containment*. It asks whether one modular system is "contained" in the other. For example, the first expression may represent the system and the second one the property to be satisfied. Containment then implies that the design is good, that is, it satisfies the specification. Moreover, containment property is in the core of the notions of abstraction-refinement and tightening-relaxation often used in reasoning about computational processes and operations research. Algorithms for containment also solve equivalence, i.e., containment in both directions.

While several declarative formalisms provide their own solutions for combining *homogeneous* modules, our goal is distinctly different. We need combinations of components specified in *different languages* (even legacy languages), and those that rely on *different solving technologies*. For instance, consider a company that provides logistics services. It decides how to pack goods and deliver them. The system has to solve two NP-complete tasks interactively – Multiple Knapsack and Travelling Salesman Problem. It takes items from customers to deliver, and considers their profits, weights, and the capacity of trucks available. It has to decide how to pack the items in the trucks, and for each truck, to solve a TSP problem. To save time and other resources, the company utilizes reusable components, that we call *concrete modules*. The Knapsack part can be solved, by, e.g., a concrete module in Integer Linear Programming (ILP), and the TSP part by a concrete module in Answer Set Programming (ASP). The system is specified by an algebraic expression that combines these and possibly other components. The expression can be optimized using modular system equivalence and verified against a specification of a desired behaviour. Clearly, our goal of utilizing modules "as is" implies the need for a semantic, language-independent approach. Moreover, a natural compositionality principle should hold: *a combination of modules is, again, a module that can be further reused*.

Early work on combining heterogeneous modules, with a range of operators, is (Järvisalo et al. 2009) and (Tasharrofi and Ternovska 2011). The latter paper proposed a semantic approach to combining modules, where *modules correspond to classes of structures*. However, when a module is a class of structures over a specific vocabulary, what is the notion of substitutability of a module by a new one, that uses a different but "compatible" vocabulary? The authors have not formalized this notion. Thus, that early work, while making an important transition to considering classes of structures,

does not fully support programming from reusable components. Moreover, it was not clear to us how to even approach the problem of modular system containment in that formalism. The algebra of (Ternovska 2019) is similar to ours, but it also does not differentiate between abstract and concrete modules. The main focus of that work is on a transition from a static algebra to a dynamic one, and to a related modal dynamic logic. Other heterogeneous formalisms have been proposed, but they either use embeddings of one specific logic into another, e.g., (Sebastiani 2007; de Bruijn et al. 2007; Eiter et al. 2008), or limit to conjunctions of modules (Lierler and Truszczynski 2014).

The inspiration of our Algebra of Modular Systems comes from Database theory. In 1970 Edgar (Ted) F. Codd introduced a relational data model and two query languages: relational algebra and relational calculus (Codd 1970). Relational algebra contains five basic operations on relational tables (sets of tuples). Relational calculus, a first-order logic counterpart, was proven to be essentially equivalent to the algebra (Codd 1972). The formalism is in the foundation of thousands of relational database management systems (RDBMS), a software industry generating tens of billions of dollars annually.

While database queries, expressed using Codd's relational algebra, can be viewed as *relations definable with respect to a structure* (a database), declarative problem specifications can be understood as *axiomatizations of classes of structures* that constrain allowable solutions by some rules. The two notions (in italic) are defined in two consecutive chapters in the classic Enderton's textbook on mathematical logic (Enderton 1972). The main idea of the Algebra of Modular Systems is to lift Codd's algebra from operations on relational tables to operations on *classes of structures*. This approach inherits compositionality of classical logic. However, by itself, the notion of a module as a class of structures, as in the previous work, is not sufficient to formalize substitutability of various modules into a modular system, as required by a good software engineering practice.

Contributions Unlike the previous work, we introduce a version of the Algebra of Modular Systems *with variables* that are *distinct from constant symbols of a relational vocabulary*. Applying compound modules then means *binding the variables with relational symbols* of the (reusable) components. Such binding (or, equivalently, substitutions of constant relational symbols for relational variables with matching arities) is performed in accordance with what combinations of modules are needed. Moreover, unlike Codd's algebra, unconstrained variables are implicitly cylindrified, i.e., interpreted arbitrarily.[1]

We introduce a novel distinction between abstract and concrete modules. Formally, a (concrete) module is a *class of structures*, however, such understanding is too restrictive in the context of reusable components. Therefore we introduce *abstract* modules that can be instantiated by a concrete one by fixing a first-order vocabulary. A *concrete* module can be given by a knowledge base in some logic, be a specification of a robotic agent, be, e.g., an ASP, CSP, ILP, CP program, etc. It can even be a human making decisions. In practice, any decision procedure, of arbitrary complexity, could be used. In contract, abstract modules are used in compound specifications. Such an algebraic specification can be "applied" to any matching concrete modules (i.e., equivalently, boolean queries, classes of structures, decision procedures).

We introduce a general notion of representability of a module in a formalism, and apply it to some examples of ASP and CSP concrete modules.

Our **main technical contribution** is a study of the formal problem of *Modular System Containment*. In this problem, we are given two expressions, and ask whether the structures that satisfy the first one, also satisfy the second one. A crucial related notion is that of *homomorphism*. Our study of the Containment problem is inspired by the results of (Chandra and Merlin 1977) on a similar containment problem for database queries. In particular, we show that Modular System Equivalence problem (and thus, the Containment problem) is undecidable in general. Then, similar to (Chandra and Merlin 1977), we consider a restricted class of modular systems, namely Conjunctive Compound Modules (CCM). We prove that CCMs exhibit a similar connection to homomorphisms between relational structures in finite model theory. This implies that for CCMs the modular system containment is in the complexity class NP. This kind of modular systems are analogous to conjunctive queries in database theory that constitute by far *the largest class* of practical queries in databases. The hope is that CCMs will represent an equally common pattern among modular systems. Unlike database queries, CCMs exhibit a significantly more general level of abstraction. Being *abstract* compound modules, they are even more general than classes of structures. Finally, we discuss conditions under which the CCM Containment problem becomes polynomial time solvable.

Algebra: Syntax and Semantics

We need a few preliminary notions. A *first-order vocabulary* (denoted, e.g. $\tau, \varepsilon, \omega$) is a finite sequence of non-logical (predicate and function) symbols, each with an associated arity. A *τ-structure*, e.g. $\mathscr{A} = (A; S_1^{\mathscr{A}}, ..., S_n^{\mathscr{A}}, f_1^{\mathscr{A}}, ..., f_m^{\mathscr{A}}, c_1^{\mathscr{A}}, ..., c_l^{\mathscr{A}})$ is a domain A together with interpretations of predicate symbols, function and constants (0-ary functions) in τ. To simplify presentation, we view functions as particular kinds of relations and consider relational structures only. We use $\mathscr{B}|_\sigma$ to mean structure $\mathscr{B}$ restricted to vocabulary σ. Symbol ":=" means "denotes" or "is by definition".

Syntax of the Algebra. An *atomic module symbol* (or simply an *atom*) is an expression $M(X_1, \ldots, X_k)$, where the X_i's are called *relational variables*. Each X_i has an associated arity a_i. The set $\{X_1, \ldots, X_k\}$ is called the *vocabulary* of M and is denoted $vocab(M)$. The vocabulary of M, $vocab(M)$, which consists of relational variables, is not to be confused with a first-order vocabulary defined above for structures, e.g., τ, which consists of predicate symbols. In the former,

[1] The term comes from Tarski's Cylindric Algebras. These algebras were introduced by Tarski and others as a tool in the algebraization of the first-order predicate calculus. See (Van den Bussche 2001) for a historic context in applications to Database theory.

we have variables, in the latter, we have constants. For a fixed (first order) vocabulary τ an expression $M(S_{i_1},\ldots,S_{i_k})$, where $\{S_{i_1},\ldots,S_{i_k}\}\subseteq\tau$ is called a *concrete module* or instantiation of M in τ, provided the arity of each S_{i_j} equals a_j. We write $\overline{X}$ and $\overline{S}$ to denote tuples $\langle X_1,\ldots,X_k\rangle$ and $\langle S_{i_1},\ldots,S_{i_k}\rangle$, respectively.

A *basic compound module* is built by the grammar (we use the usual rules for such expressions)

$$\alpha ::= \top \mid M_i \mid (\alpha\cap\alpha) \mid (\alpha\cup\alpha) \mid (\alpha-\alpha) \mid \pi_\nu(\alpha) \mid \sigma_\Theta(\alpha). \tag{1}$$

Symbol $\top$ represents a tautological module, and M_i are atomic module symbols. If α is a compound module constructed this way, then $vocab(\alpha)$ is the union of $vocab(M_i)$ for all atomic modules involved in α. The operations (except $\top$) are like in Codd's relational algebra, but are of a higher order, and are defined on *classes of structures* rather than on relational tables. In particular, projection is onto relational variables of $vocab(\alpha)$ rather than onto object variables. The three set-theoretic operations are union ($\cup$), intersection ($\cap$), set difference ($-$). Projection ($\pi_\nu(\alpha)$) is a family of unary operations, one for each ν, where $\nu\subseteq vocab(\alpha)$. The condition Θ in selection $\sigma_\Theta(E)$ is an expression that is built up using $\wedge$, $\vee$, $\neg$, from equivalence operators $\equiv$, $\not\equiv$, applied to variables in α. Selection can be used, in particular, to connect modules by equating relational symbols of equal arity. Notice that the framework works for infinite structures.

Semantics of Atomic Modules. The semantics of an atom $M(X_1,\ldots,X_k)$ is given in two steps. First, for a FO (first-order) vocabulary $\tau=\{S_1,S_2,\ldots\}$ we *instantiate* vocabulary symbols $X_1,\ldots,X_k$ as predicate symbols $S_{i_1},\ldots,S_{i_k}$ from τ so that the arities of X_j and S_{i_j} match. The semantics of the resulting concrete module $M(S_{i_1},\ldots,S_{i_k})$ indicates whether $M(\overline{S})$ is true (notation $\mathscr{B}\models M(\overline{S})$) or false on τ-structure $\mathscr{B}$ (notation $\mathscr{B}\not\models M(\overline{S})$). For any two τ-structures $\mathscr{B}_1$, $\mathscr{B}_2$ which coincide on $\{S_{i_1},\ldots,S_{i_k}\}$, we have $\mathscr{B}_1\models M(\overline{S})$ iff $\mathscr{B}_2\models M(\overline{S})$. Thus, for the vocabulary τ and an instantiation $S_{i_1},\ldots,S_{i_k}$ the concrete module $M(S_{i_1},\ldots,S_{i_k})$ defines a **class of structures** with vocabulary τ.

As a simple example consider the module $M_{3\text{Col}}$ that (intuitively) decides whether or not a given graph, that is, a set of nodes and a set of edges is 3-colourable. Such a module has a very natural semantics on the class of graphs, that is, τ-structures with $\tau=\{E\}$ where E is a binary predicate symbol: For a graph $G=(V;E^G)$ it holds $G\models M_{3\text{Col}}(E)$ if and only if G is 3-colourable. However, the freedom of interpretations of vocabulary variables allows for more flexibility. Let $\tau=\{B,R\}$, where both B and R are binary symbols. In other words τ is the vocabulary of bigraphs, structures with edges of two types, say, blue and red. Then for a bigraph $G=(V;B^G,R^G)$ the expression $G\models M_{3\text{Col}}(B)$ that the blue-edged part of G is 3-colourable, while $G\models M_{3\text{Col}}(R)$ means that the red-edged part of G is 3-colourable.

Modules can also be thought of in a more straightforward way. In this way the variables of a module $M(X_1,\ldots,X_k)$ are instantiated as predicates over some domain and the module "evaluates" them. To define what it means that a τ-structure $\mathscr{A}$ satisfies module M, we need to first select predicate symbols $S_{i_1},\ldots,S_{i_k}$ from τ, whose arities match those of $X_1,\ldots,X_k$, and then "apply" the module to $S_{i_1}^{\mathscr{A}},\ldots,S_{i_k}^{\mathscr{A}}$, as we would apply a decision procedure.

Semantics of Five Basic Operations. Satisfaction relation $\models$ for compound algebraic expressions is built inductively.

1. Product $(\alpha_1(\overline{X})\cap\alpha_2(\overline{Y}))$.[2] For a vocabulary τ, instantiations $\overline{R}$ and $\overline{S}$ of $\overline{X}$ and $\overline{Y}$, respectively, and a structure $\mathscr{B}$ with vocabulary τ, we have $\mathscr{B}\models\alpha_1(\overline{R})\cap\alpha_2(\overline{S})$ iff both $\mathscr{B}\models\alpha_1(\overline{R})$ and $\mathscr{B}\models\alpha_2(\overline{S})$.

2. Union $(\alpha_1(\overline{X})\cup\alpha_2(\overline{Y}))$. For a vocabulary τ, instantiations $\overline{R}$ and $\overline{S}$ of $\overline{X}$ and $\overline{Y}$, respectively, and a structure $\mathscr{B}$ with vocabulary τ, $\mathscr{B}\models\alpha_1(\overline{R})\cup\alpha_2(\overline{S})$ iff at least one of $\mathscr{B}\models\alpha_1(\overline{R})$ and $\mathscr{B}\models\alpha_2(\overline{S})$ holds.[3]

3. Set Difference $(\alpha_1(\overline{X})-\alpha_2(\overline{Y}))$. For a vocabulary τ, instantiations $\overline{R}$ and $\overline{S}$ of $\overline{X}$ and $\overline{Y}$, respectively, and a structure $\mathscr{B}$ with vocabulary τ, we have $\mathscr{B}\models\alpha_1(\overline{R})-\alpha_2(\overline{S})$ iff $\mathscr{B}\models\alpha_1(\overline{R})$ and $\mathscr{B}\not\models\alpha_2(\overline{S})$. Thus, $\alpha_1-\alpha_2$ represents a class of structures over the joint vocabulary that are in $\alpha_1(\overline{R})$, but not in $\alpha_2(\overline{S})$.

4. Projection $(\pi_\nu(\alpha(X_1,\ldots,X_k)))$, $\nu\subseteq\{X_1,\ldots,X_k\}$. For a vocabulary τ, instantiation S_i of $X_i\in\nu$, and a structure $\mathscr{B}$ with vocabulary τ, the semantics is given by: $\mathscr{B}\models\pi_\nu(\alpha)(S_i:X_i\in\nu)$ if there are instantiations S_i in τ of $X_i\in\{X_1,\ldots,X_k\}-\nu$, and a τ-structure $\mathscr{B}'$ such that $\mathscr{B}'\models\alpha(\overline{S})$ and $\mathscr{B}$ and $\mathscr{B}'$ agree on S_i, $X_i\in\nu$. The operation restricts the structures of α to $\nu\subseteq vocab(\alpha)$ leaving the instantiations of other symbols open. Thus, it is more relaxed than the original expression and increases the number of models.

5. Selection is a family of unary operations of the form $\sigma_{\Theta(\overline{X})}(\alpha(\overline{X}))$, where Θ is a condition that can be applied as a test to *each structure* from instantiations of M.

The semantics is given by $\mathscr{B}\models\sigma_{\Theta(\overline{S})}(\alpha(\overline{S}))$ iff $\mathscr{B}\models\alpha(\overline{S})$ and $\mathscr{B}\models_{\text{FO}}\Theta(\overline{S})$, where $\models_{\text{FO}}$ is the satisfaction relation in the sense of classical first-order logic. Notice that as selection imposes extra restrictions on a potential model, it decreases the number of models.

Example 1 Let $M_{\text{HC}}(X,Y)$ and $M_{2\text{Col}}(X,Z,T)$ be modules "computing" a Hamiltonian Circuit, and a 2-colouring. In other words, X,Y are variables of arity 2, and the first module decides if Y forms a Hamiltonian Circuit (represented as a set of edges) in the graph given by edge set X. Variable X of the second module has arity 2, and variables Z,T are unary; the module decides if unary relations Z,T specify a proper 2-colouring of the graph with edge set X. The following algebraic expression determines a combination of 2-Colouring and Hamiltonian Circuit, that is whether or not there is a 2-colourable Hamiltonian Circuit.

$$M_{\text{2Col-HC}}(X,Z,T) \;:=\; \pi_{X,Z,T}(M_{\text{HC}}(X,Y)\cap M_{2\text{Col}}(Y,Z,T)). \tag{2}$$

[2]Note that variables $\overline{X}$ and $\overline{Y}$ do not have to be all different. In this case equal variables must have equal instantiations.

[3]Note that in relational algebra, both arguments to the union and the difference must be relations of the same arity. Here, tuples $\overline{S}$, $\overline{R}$ can be of different length because instantiations are over τ-structures, where τ includes the vocabularies of all modules.

Projection hides the instantiation of Y in M_{HC}, since it is the same as Y's in $M_{2\text{Col}}$.

Example 2 The following algebraic expression specifies a new module that verifies if a given structure with a certain binary relation is a graph that is a cycle $M_{\text{Cycle}}(X) := \pi_X(\sigma_{(X\equiv Y)}(M_{\text{HC}}(X,Y)))$. An alternative way to express the same module is $M_{\text{Cycle}}(X) := M_{\text{HC}}(X,X)$.

Representing Modules in CSP and ASP

We illustrate how abstract modules can be concretely represented in two broadly used formalisms, Answer Set Programming (ASP) and Constraint Satisfaction Problem (CSP). We use well-known combinatorial problems as examples.[4] In computer science, combinatorial decision problems are encoded as sets of binary strings. In finite model theory (Libkin 2004), they correspond to classes of structures. E.g., 3-Colouring corresponds to all graphs that are 3-colourable. We say *module M represents a problem $\mathscr{P}$* if each structure in M consists of an instance and a certificate of $\mathscr{P}$, e.g., a graph and its colouring.

Representing Modules in CSP

CSP in the traditional AI form:

Instance: (V,D,C) where V is a finite set of variables, D is a set of *values* (also called domain), C is a finite set of *constraints* $\{C_1,\ldots,C_n\}$. Each constraint is a pair $(\bar{x}_i,T_i)$, where $\bar{x}_i$ is the *scope*, a list of variables of length m_i, and T_i is a m_i*-ary relation* over D.
Question: Is there $f: V \to D$ such that $f(\bar{x}_i) \in T_i$ for all i? One can also search for such f.

Let $\mathscr{A}$ and $\mathscr{B}$ be τ-structures, where τ is a relational vocabulary $\tau := \{R_1,\ldots,R_l\}$. A *homomorphism* is a function $h: \mathscr{A} \to \mathscr{B}$ such that $\forall i \in \{1,\ldots,l\} \quad \big[\,(a_1,\ldots,a_{n_i}) \in R_i^{\mathscr{A}} \Rightarrow (h(a_1),...,h(a_{n_i})) \in R_i^{\mathscr{B}}\,\big]$.

CSP in homomorphism form:

Instance: Two τ-structures, $\mathscr{C}_1$ and $\mathscr{C}_2$. Question: Is there a homomorphism $h: \mathscr{C}_1 \to \mathscr{C}_2$?

To relate to the AI form of CSP, think of domain elements in $\mathscr{C}_1$ as of variables. Tuples in relations in $\mathscr{C}_1$ correspond to constraint scopes, and elements in $\mathscr{C}_2$ to values. Relations in $\mathscr{C}_2$ are constraint relations.

Structure $\mathscr{C}$ is a *homomorphic pair* if it has the form

$$\mathscr{C} = (D; \underbrace{R_{dom(\mathscr{C}_1)}, R_1^{\mathscr{C}_1}, \ldots R_l^{\mathscr{C}_1}}_{\mathscr{C}_1}, \underbrace{R_{dom(\mathscr{C}_2)}, R_1^{\mathscr{C}_2}, \ldots R_l^{\mathscr{C}_2}}_{\mathscr{C}_2}, H^{\mathscr{C}}),$$

where $dom(\mathscr{C}_1), dom(\mathscr{C}_2) \subseteq D$, $R_{dom(\mathscr{C}_1)}$ and $R_{dom(\mathscr{C}_2)}$ are unary relations that represent the domains of τ-structures $\mathscr{C}_1$ and $\mathscr{C}_2$, respectively, and $H^{\mathscr{C}}$ is a relation that represents the homomorphism function h.

Module $M(X_1,\ldots,X_k,Y_1,\ldots,Y_k,Z)$ is *representable in CSP (in homomorphism form)* if: (a) the arity of X_1,Y_1 is 1, the arity of X_i equals that of Y_i, and the arity of Z is 2, (b) for any instantiation $S_1,\ldots,S_k,R_1,\ldots,R_k,T$ of $X_1,\ldots,X_k,Y_1,\ldots,Y_k,Z$ in a vocabulary τ, for a τ-structure $\mathscr{A}$, it holds that $\mathscr{A} \models M(S_1,\ldots,S_k,R_1,\ldots,R_k,T)$ iff $T^{\mathscr{A}}$ encodes a homomorphism from $(S_1^{\mathscr{A}};S_2^{\mathscr{A}},\ldots,S_k^{\mathscr{A}})$ to $(R_1^{\mathscr{A}};R_2^{\mathscr{A}},\ldots,R_k^{\mathscr{A}})$. Such a representation is clearly not possible when the complexity of checking the membership in the module exceeds NP.

The CSP representation of modules can be sometimes simplified using the *non-uniform* CSP, when the first or the second structure in a pair is fixed. In this case a CSP representation of a module is closer to the standard representation of, e.g., 3-Colouring, and intuitively corresponds to the class of structures homomorphic to a fixed structure (or admitting a homomorphism from a fixed structure). More precisely, let $\mathscr{B} = (B;S_1,\ldots,S_k)$, $B = \{b_1,\ldots,b_{|B|}\}$ be a structure. Then a module $M(X_1,\ldots,X_k,Z_1,\ldots,Z_{|B|})$ is *representable in the right non-uniform* CSP($\mathscr{B}$) if following conditions hold: (a) the arity of $Z_1,\ldots,Z_{|B|}$ is 1, and the arity of X_i equals that of S_i; (b) for any vocabulary τ, and any instantiation $R,\ldots,R_k$ and $T_1,\ldots,T_{|B|}$ of $X_1,\ldots,X_k$ and $Z_1,\ldots,Z_{|B|}$, respectively, for a τ-structure $\mathscr{A}$ it holds $\mathscr{A} \models M(R_1,\ldots,R_k,T_1,\ldots,T_{|B|})$ iff $T_1,\ldots,T_{|B|}$ is a partition of the domain A of $\mathscr{A}$, and the mapping $\varphi: A \to B$ given by $\varphi(a) = b_i$ where $a \in T_i$ for $a \in A$ is a homomorphism from $(A;R_1^{\mathscr{A}},\ldots,R^{\mathscr{A}})$ to $\mathscr{B}$. Representations by the left non-uniform CSP are similar.

Example 3 (Hamiltonian Circuit in CSP (hom. form)) An **abstract** atomic module representing Hamiltonian Circuit problem (denoted by $M_{\text{HC}}(X,Y)$, where X and Y are binary), is such that for any vocabulary τ and any instantiation S,R of X,Y, for a τ-structure $\mathscr{B}$ with domain V it holds that $\mathscr{B} \models M(S,R)$ iff $G = (V;S^{\mathscr{B}})$ is a graph, $R^{\mathscr{B}} \subseteq S^{\mathscr{B}}$ is a set of edges of G that form a Hamiltonian Circuit. It is representable by a **concrete** CSP instance $(\mathscr{C},\mathscr{D})$ in homomorphism form where $\mathscr{C} = (V;E^{\mathscr{C}},(\neq_V)^{\mathscr{C}})$, $\mathscr{D} = (V;E^{\mathscr{D}},(\neq_V)^{\mathscr{D}})$ and there is a homomorphism H from $\mathscr{C}$ to $\mathscr{D}$. This CSP instance represents a class of structures of the form $\mathscr{E} = (V;E^{\mathscr{C}},(\neq_V)^{\mathscr{C}},E^{\mathscr{D}},(\neq_V)^{\mathscr{D}},H^{\mathscr{E}})$. Notice that $\neq_V$ and H are auxiliary, not a part of $M_{\text{HC}}(E,C_V)$, and are specific to the CSP representation.

Example 4 (Betweenness in CSP (AI form)) This problem was originally posed in (Opatrny 1979). An **abstract** atomic module representing Betweenness problem, $M_{\text{B}}(X,Y)$, for any instantiation N,F of X,Y, is a class of structures of the form $\mathscr{B} = (V \cup \mathbb{Q}; N^{\mathscr{B}}, F^{\mathscr{B}})$ where V is a finite set, $N^{\mathscr{B}} \subseteq V^3$, mapping $F^{\mathscr{B}} : V \to \mathbb{Q}$ is such that it generates a linear ordering of V such that, for each triple $(r,b,g) \in M^{\mathscr{B}}$, we have $r < b < g$ or $r > b > g$, and $R_b = \{(a,b,c) \in \mathbb{Q}^3 \mid a < b < c \text{ or } a > b > c\}$.

It is representable by **concrete** CSP instance $(V,\mathbb{Q},C)$ in the traditional AI form, where V is a set variables, $\mathbb{Q}$ is the domain for those variables, and C is a set of constraints, $C = \{C_m \mid m \in N\}$, $C_m = ((u,v,w),R_b)$, and $R_b = \{(a,b,c) \in \mathbb{Q}^3 \mid a < b < c \text{ or } a > b > c\}$. Observe that Betweenness admits representation by the right non-uniform CSP, as every concrete module of this form is the class of all struc-

[4] We give examples in ASP and CSP, but any other formalism, of arbitrary expressiveness, e.g. Essence (Frisch et al. 2008), could also be used.

tures that admit homomorphism to $(\mathbb{Q},N)$, where N is the betweenness relation on $\mathbb{Q}$.

Example 5 (k-Colouring in CSP (homomorphism form)) An **abstract** atomic module representing k-Colouring problem, $M_{\text{kCol}}(X,Y)$, where both X,Y are binary, is such that for any vocabulary τ and any instantiation S,R of X,Y, for a τ-structure $\mathscr{B}$ with domain V it holds that $\mathscr{B} \models M(S,R)$ iff $G = (V;S^{\mathscr{B}})$ is a graph, $R^{\mathscr{B}} \subseteq S^{\mathscr{B}}$ is an equivalence relation on V with at most k equivalence classes such that each equivalence class is an independent set of G.

Let Col refers to the relation that represents the homomorphism function $h : \mathscr{G}_1 \to \mathscr{G}_2$. Consider structures of the form:

$$\mathscr{C} = (V;\underbrace{V_1^{\mathscr{G}_1},E_1^{\mathscr{G}_1}}_{\mathscr{G}_1},\underbrace{V_2^{\mathscr{G}_2},E_2^{\mathscr{G}_2}}_{\mathscr{G}_2},Col^{\mathscr{C}}),$$

concrete 3-Colouring CSP module, $M_{3\text{Col}}(E,Col)$, consists of all the structures $\mathscr{C}$ as above, restricted to the vocabulary of this module.

In general, any class of structures representable by CSP is a (concrete) module. The converse is not true. This is due to the well known property: if there is a homomorphism from $\mathscr{A}$ to $\mathscr{B}$ and from $\mathscr{B}$ to $\mathscr{C}$, there is also a homomorphism from $\mathscr{A}$ to $\mathscr{C}$. Any concrete module that does not satisfy this condition cannot be represented by CSP. For example: pairs of structures $\mathscr{A}$, $\mathscr{B}$ such that $|A|+|B|$ is odd. There are $\mathscr{A}$, $\mathscr{B}$, $\mathscr{C}$ such that $\mathscr{A} \to \mathscr{B}$ and $\mathscr{B} \to \mathscr{C}$, and $|A|+|B|$, $|B|+|C|$ are odd, so $(\mathscr{A},\mathscr{B})$, $(\mathscr{B},\mathscr{C})$ belong to the module. Then if this module is representable by CSP, then, since $\mathscr{A} \to \mathscr{C}$, we must have $(\mathscr{A},\mathscr{C})$ in the module. But we do not because $|A|+|C|$ is even.

Representing Modules in ASP

We assume familiarity of the reader with Answer Set Programming (Niemelä 1999; Marek and Truszczynski 1999). We separate IDB and EDB predicates, as is common in Datalog. Intuitively, EDB predicates are given by a database, and IDB predicates are definable in terms of those. For a program Π, we use Π' to denote an ASP program obtained by augmenting Π with ground atoms representing the database, the interpretations of the EDB predicates.

We say a module M is *representable in ASP* if there is an ASP program Π such that the stable models of Π', for each interpretation of the EDB predicates, when limited to the vocabulary of M, are precisely the structures of M.

Example 6 (Hamiltonian Circuit in ASP) The program is from page 89 of (Hölldobler and Schweizer 2014).

$$1\ \{cycle(X,Y) : edge(X,Y)\}\ 1 :- node(X).$$
$$1\ \{cycle(X,Y) : edge(X,Y)\}\ 1 :- node(Y).$$
$$reachable(Y) :- cycle(s,Y).$$
$$reachable(Y) :- cycle(X,Y), reachable(X).$$
$$:- node(X), not\ reachable(X).$$

A node in the cycle has exactly one incoming and one outgoing edge, according to the first two rules. Since for every node we nondeterministically pick one outgoing and incoming edge, we might have generated a cycle or not. To rule out model candidates not representing cycles, we check whether every node can be reached by every other node and exclude models where there is an unreachable node.

This **concrete** ASP module is a particular instantiation $M_{\text{HC}}(edge,cycle)$ of the abstract module from Example 3, where *node*, *reachable* and *s* are auxiliary and are specific to this concrete representation.

We now give a general translation.

From ASP Programs to Atomic Modules An atomic module representing general ASP program, M_{ASP}, where $\tau = \{R_1,\ldots,R_l,R_1,\ldots,R_k\}$, is a class of structures of the form

$$\mathscr{A} = (D;\underbrace{R_1^{\mathscr{A}},\ldots,R_l^{\mathscr{A}}}_{EDB},\underbrace{R_1^{\mathscr{A}},\ldots,R_k^{\mathscr{A}}}_{IDB}),$$

where D is the active domain, i.e., the constants used in the ground atoms representing the database.

Containment and Equivalence

Equivalence We say that two atomic or compound modules α,α' are *equivalent*,[5] denoted $\alpha = \alpha'$, if $\mathscr{B} \models \alpha(\bar{S})$ iff $\mathscr{B} \models \alpha'(\bar{S})$ for any τ-structure $\mathscr{B}$ and any instantiation $\bar{S}$ of variables by relational symbols from τ. E.g., $(M_1 \cup M_2) \cap M_3 = (M_1 \cap M_3) \cup (M_2 \cap M_3)$, for any choice of atoms M_1,M_2,M_3. Modular System Equivalence *problem* is given two compound modules α^1,α^2, decide whether they are equivalent.

Containment We say that a M_1 is *contained* in M_2, denoted $M_1 \sqsubseteq M_2$, if $\mathscr{B} \models M_2(\bar{S})$ is true whenever $\mathscr{B} \models M_1(\bar{S})$ is true, for any structure $\mathscr{B}$, for any instantiation of variables $\bar{S}$ by relational symbols from τ. For compound modules α^1,α^2 with atoms $M_1,\ldots,M_s$, we say that α^1 is *contained* in α^2, denoted $\alpha^1 \sqsubseteq \alpha^2$, if $\mathscr{B} \models \alpha^2(\bar{S})$ is true whenever $\mathscr{B} \models \alpha^1(\bar{S})$ is true, for any instantiations $\bar{S}$ and any choice of atoms $M_1,\ldots,M_s$. Thus, this is essentially the problem of logical implication for our algebraic expressions. For example, $M_1 \cap M_2 \sqsubseteq M_1 \sqsubseteq M_1 \cup M_2$ for any choice of atoms M_1, M_2. We have $M_1 = M_2$ iff $M_1 \sqsubseteq M_2$ and $M_2 \sqsubseteq M_1$.

Theorem 1 *The Modular System Equivalence problem is undecidable in general, even in the finite case.*

Proof: (*outline*) The Finite Satisfiability problem asks whether a given formula in first-order logic is satisfiable by a structure with a finite domain. By Trakhtenbrot's (Trakhtenbrot 1950) theorem, this problem is undecidable. There is a straightforward reduction of the Finite Satisfiability problem to the Modular System Equivalence problem. Observe that finite satisfiability of first-order logic is essentially the same problem as finite satisfiability of relational algebra, so finite satisfiability of modular systems is undecidable. To finish the argument, let Q be an unsatisfiable modular system. Then "F equivalent Q" is the same problem as checking if F is finitely satisfiable, which is undecidable. □

[5] We use equality symbol (=) as a meta-symbol between algebraic expressions, and equivalence symbol (≡) as a logical connective (biconditional) in formulas.

Thus, the problem of deciding containment of compound modules is also undecidable. However, as we show below, for certain broad classes of modules it is decidable and even NP-complete, or sometimes solvable in polynomial time.

It will be convenient in this section to represent tuples or sets of vocabulary variables and their instantiations as indexed sets, e.g., $\{X_i : i \in I\}$ for a certain (finite) set I. We will mostly assume that I is clear from the context. A **conjunctive compound module (CCM)** is a compound module expressible by a relational algebra expression:

$$\pi_{\{X_i:i\in J\}}(\sigma_\Theta(M_1 \cap \cdots \cap M_n)(X_i : i \in I)),$$

where $J \subseteq I$ are some finite sets, Θ is a conjunction of equivalence ($\equiv$) atomic formulas. CCMs are also expressible in logic form by formulas in prenex normal form built from atomic modules $M_j(X_{i_1},\ldots,X_{i_n})$, and $\wedge$ and $\exists$ (applied to vocabulary variables) only:

$$\exists(X_i : i \in J)\ \ \Phi(X_i : i \in I),$$

where $\Phi(X_i : i \in I)$ is a conjunction of atomic modules of the form $M_j(X_{i_1},\ldots,X_{i_n})$.

We study the complexity of the following problem:

CCM Containment Problem

Input: CCMs α^1, α^2 over the same atomic module symbols $M_1,\ldots,M_s$.
Decide: if $\alpha^1 \sqsubseteq \alpha^2$, for any choice of $M_1,\ldots,M_s$.

Note that the CCM Containment problem makes sense only for abstract modular systems (represented by algebraic expressions), i.e., modular systems where atomic module symbols have relational *variables* as arguments (rather than elements of a fixed relational vocabulary τ).

Let us observe first that the selection operation in a CCM can be eliminated. Indeed, since the only Θ allows is a conjunction of expressions of the form $X \equiv Y$, Θ defines an equivalence relation on the set of variables. Replacing every occurrence of every variable from each equivalence class with a fixed representative of that class, we obtain an equivalent CCM containing no selection operation.

A CCM α is said to be *projection-free* if it has the form $M_1 \cap \cdots \cap M_n$ for some atoms $M_1,\ldots,M_n$.

Canonical Structure For a projection-free CCM $\alpha(\overline{X}) = \Phi(M_1,\ldots,M_s)$, the *canonical structure* $\mathscr{A}_\alpha$ is defined as:

- the domain of $\mathscr{A}_\alpha$ is $\overline{X} = \{X_i : i \in I\}$ for some finite set I;
- the vocabulary is $\delta = \{T_i \mid i \in \{1,\ldots,s\}\}$, where the arity of T_j equals that of module M_j;
- the instantiation of every symbol T_j is given by $(X_{i_1},\ldots,X_{i_k}) \in T_j^{\mathscr{A}_\alpha}$ iff $M_j(X_{i_1},\ldots,X_{i_k})$ is an atom in Φ.

Notice that, in a canonical structure, for a specific CCM $\alpha(\bar{X})$, the variables of α become domain elements.

For a CCM $\alpha = \pi_{\{X_i:i\in J\}}\alpha'$ that is not projection-free and α' is projection-free, $\mathscr{A}_\alpha$ is defined to be $\mathscr{A}_{\alpha'}$.

As an example of a canonical structure consider compound module $\alpha(X_1,X_2,X_3,x_4) = M_1(X_1,X_2) \cap M_2(X_2,X_2,X_3) \cap M_1(X_3,X_4)$. Then the domain of $\mathscr{A}_\alpha$ is the set $\{X_1,X_2,X_3,X_4\}$, its vocabulary consists of two relational symbols T_1 and T_2 of arity 2 and 3, respectively. These symbols are then instantiated as follows: $T_1^{\mathscr{A}_\alpha} = \{(X_1,X_2),(X_3,X_4)\}$ and $T_2^{\mathscr{A}_\alpha} = \{(X_2,X_2,X_3)\}$.

Proposition 1 *Let α^1, α^2 be projection-free CCMs. Then $\alpha^1 \sqsubseteq \alpha^2$ iff the identity mapping is a homomorphism from $\mathscr{A}_{\alpha^2}$ to $\mathscr{A}_{\alpha^1}$.*

Proof: Indeed, if the identity is a homomorphism, it simply means that every atom of α^2 is also an atom of α^1. On the other hand, if $\alpha^1 \sqsubseteq \alpha^2$ every atom of α^2 must be present in α^1, as otherwise there are atomic modules, an instantiation, and a structure that contradicts $\alpha^1 \sqsubseteq \alpha^2$. This immediately implies that the identity mapping is a homomorphism between the two structures. □

Next we extend Proposition 1 to general CCMs, toward obtaining Theorem 2. It will be convenient for us to represent a CCM as $\alpha = \pi_{(X_j:j\in J)}\Phi(M_1,\ldots,M_s)$, where $J \subseteq I$ and $\Phi(M_1,\ldots,M_s)$ is a projection free CCM. Let $M' = \Phi(M_1,\ldots,M_s)$. In addition to canonical structure, we will need the notion of a power structure.

Power Structure. Let $\mathscr{A}$ be a structure with domain A and vocabulary τ, and let $M_1,\ldots,M_s$ be modules. The *power* structure $\mathscr{K} = \mathscr{K}_{M_1,\ldots,M_s}(\mathscr{A})$ is defined as follows:

- let $a_1,\ldots,a_r$ be all the different arities of symbols from τ;
- the domain of $\mathscr{K}$ is
$$K = \mathscr{P}(A^{a_1}) \cup \cdots \cup \mathscr{P}(A^{a_r}),$$
where $\mathscr{P}(B)$ denotes the power set of B;
- the vocabulary of $\mathscr{K}_{M_1,\ldots,M_s}(\mathscr{A})$ is $R_1,\ldots,R_s$, and the arity of R_j is that of the module M_j;
- every relational symbol R_j of arity k is interpreted as follows: tuple $(c_1,\ldots,c_k) \in K^k$ belongs to $R_j^{\mathscr{K}_{M_1,\ldots,M_s}(\mathscr{A})}$ iff $c_t \in \mathscr{P}(A^{a_{i_t}})$, where a_{i_t} is the arity of t'th argument of M_j, for $t \in [k]$, and $\mathscr{B} \models M_j(S_{i_1},\ldots,S_{i_k})$ for any τ-structure $\mathscr{B}$ such that $S_{i_t}^{\mathscr{B}} = c_t$ for $t \in [k]$.

Note that as defined, $\mathscr{K}_{M_1,\ldots,M_s}(\mathscr{A})$ depends only on the modules $M_1,\ldots,M_s$, the domain, and the vocabulary of $\mathscr{A}$, but not on $\mathscr{A}$'s relations.

The following is an example of a simple power-structure. Let $M(X_1,X_2)$ be a module whose arguments have arities 2 and 3, respectively. Let also $\mathscr{A}$ be a relational structure with domain $A = \{0,1\}$ and vocabulary $\tau = \{T_1,T_2\}$ of arities 2 and 3. Then $K = \mathscr{P}(\{0,1\}^2) \cup \mathscr{P}(\{0,1\}^3)$ consists of sets of pairs and sets of triples over $\{0,1\}$ that correspond to all possible binary and ternary relations. Suppose that module M contains all the structures $\mathscr{B}$ with a binary and a ternary symbols such that $M(S_1,S_2)$ is true whenever

$$\begin{array}{c} S_1 \subseteq \{(a,b)\colon (a,b,c) \in S_2\} \cap \\ \{(b,c)\colon (a,b,c) \in S_2\} \cap \{(a,c)\colon (a,b,c) \in S_2\}. \end{array} \quad (3)$$

Then $\mathscr{K}_M(\mathscr{A})$ has only one relational symbol R interpreted as the set of all pairs (S_1,S_2), where $S_1 \subseteq \{0,1\}^2$, $S_2 \subseteq \{0,1\}^3$ satisfying (3). These includes pairs $(\emptyset,\emptyset)$, $(\{(0,0),(1,1)\},\{(0,0,0),(1,1,1)\})$, $(\{(a,b)\colon a \le b\},\{(a,b,c)\colon a \le b \le c\})$ among others.

The following lemma is a generalization of the "Magic Lemma" from (Chandra and Merlin 1977), as it is sometimes referred to, to a much higher level of abstraction. In this lemma, we relate the satisfaction of a concrete compound module, that is, a module where variables are instantiated with specific symbols of a relational vocabulary, to the question of existence of a homomorphism that maps variables to the elements of the domain of the Power Structure. The homomorphism takes into account how variables are instantiated by specific symbols of a relational vocabulary. The difference with the original setting (Chandra and Merlin 1977) is substantial. Instead of queries that are formulae that use database relations directly, we have algebraic combinations of modules that are, essentially, decision procedures, as in the previous section, that accept or reject structures. These structures provide interpretations to second-order variables, given their instantiations in τ. Because of this additional level, and because the proof has to work for all instantiations of variables by relational symbols, our construction of the Power Structure is rather involved. Notice that, in the results below, modules are treated abstractly as classes of structures (sets if a domain is given). As a consequence, the lemma and the subsequent theorem apply to all modules.

Lemma 1 (Magic Lemma II) *Let M be a CCM with the set of variables $\{X_i : i \in I\}$ and the set of variables $J \subseteq I$ in the projection operation, $M_1, \ldots, M_s$ be atoms of M. Let $\mathscr{A}$ be a structure with vocabulary τ, and $\{S_i : i \in J\}$ an instantiation of the variables of M in τ. Then*

$$\mathscr{A} \models M(S_i : i \in J) \text{ iff there is a homomorphism } \varphi \text{ from } \mathscr{A}_M \text{ to } \mathscr{K}_{M_1,\ldots,M_s}(\mathscr{A})$$

such that $\varphi(X_i) = S_i^{\mathscr{A}}$ for $i \in J$.

Proof: ($\Rightarrow$) Suppose $\mathscr{A} \models M(S_i : i \in J)$, and $\mathscr{A}$ has a domain A. This means that there is an instantiation $\{S_i : i \in I - J\}$, a structure $\mathscr{A}'$ with the same domain such that $\mathscr{A}' \models M'(S_i : i \in I)$ and $S_i^{\mathscr{A}'} = S_i^{\mathscr{A}}$ for $i \in J$. Define $\varphi : \{X_i : i \in I\} \to K$ by $\varphi(X_i) = S_i^{\mathscr{A}'}$. Obviously, this mapping satisfies the condition $\varphi(X_i) = S_i^{\mathscr{A}}$ for $i \in J$. Since for every atom $M_j(X_{i_1}, \ldots, X_{i_k})$ of Φ, $\mathscr{A}' \models M_j(S_{i_1}, \ldots, S_{i_k})$, the tuple $(S_{i_1}^{\mathscr{A}'}, \ldots, S_{i_k}^{\mathscr{A}'})$ belongs to $R_j^{\mathscr{K}_{M_1,\ldots,M_s}(\mathscr{A})}$. Thus, φ is a homomorphism.

($\Leftarrow$) Suppose there is a homomorphism φ from $\mathscr{A}_M$ to $\mathscr{K}_{M_1,\ldots,M_s}(\mathscr{A})$ such that $\varphi(X_i) = S_i^{\mathscr{A}}$ for $i \in J$. Define $\mathscr{A}'$ on the same domain A as that of $\mathscr{A}$ by setting, for every S_i, $i \in I$, $S_i^{\mathscr{A}'} = \varphi(X_i)$. Note that if $i \in J$ then by assumption $S_i^{\mathscr{A}'} = S_i^{\mathscr{A}}$. Also, since for every atom $M_j(X_{i_1}, \ldots, X_{i_k})$ of Φ, we have $(S_{i_1}^{\mathscr{A}'}, \ldots, S_{i_k}^{\mathscr{A}'}) \in R_j^{\mathscr{K}_{M_1,\ldots,M_s}(\mathscr{A})}$, $\mathscr{A}' \models M_j(S_{i_1}, \ldots, S_{i_k})$. Thus, $\mathscr{A}' \models M'(S_i : i \in I)$, and so $\mathscr{A} \models M(S_i : i \in J)$. □

Theorem 2 (Homomorphism Theorem) *Let $\alpha^1 = \pi_{(X_i:i\in J)}\Phi_1(M_1, \ldots, M_s)$ and $\alpha^2 = \pi_{(X_i:i\in J)}\Phi_2(M_1, \ldots, M_s)$ be two CCMs for $J \subseteq I$. Then $\alpha^1 \sqsubseteq \alpha^2$ iff there is a homomorphism $\varphi : \mathscr{A}_{\alpha^2} \to \mathscr{A}_{\alpha^1}$ such that $\varphi(X_i) = X_i$ for all $i \in J$.*

Proof: ($\Leftarrow$) Suppose there is a homomorphism $\varphi : \mathscr{A}_{\alpha_2} \to \mathscr{A}_{\alpha_1}$ such that $\varphi(X_i) = X_i$ for $i \in J$. Fix atomic modules $M_1, \ldots, M_s$. Now if $\mathscr{A} \models \alpha^1(S_i : i \in J)$ for some instantiation $\{S_i : i \in J\}$, by the Magic Lemma II there is a homomorphism from $\mathscr{A}_{\alpha^1}$ to $\mathscr{K}_{M_1,\ldots,M_s}(\mathscr{A})$. Composing it with φ we obtain a homomorphism from $\mathscr{A}_{\alpha^2}$ to $\mathscr{K}_{M_1,\ldots,M_s}(\mathscr{A})$. Therefore, $\mathscr{A} \models \alpha^2(S_i : i \in J)$.

($\Rightarrow$) Since $\alpha^1 \sqsubseteq \alpha^2$, we can choose any kind of modules and structures to guarantee that a required homomorphism exists, and we will use this freedom to the fullest.

Given a vocabulary τ we define a specialized structure that only depends on τ. Fix some $\tau = \{S_i : i \in I\}$ so that its symbols correspond to the variables of α^1, α^2 and have the same arities. Let $\mathscr{A}_\tau$ be a structure with domain $A = \tau$ and vocabulary τ. For every $S_i \in \tau$, its interpretation is given by $S_i^{\mathscr{A}_\tau} = \{(S_i, \ldots, S_i)\}$, where the length of the tuple is the arity of S_i. It is valid because S_i also belongs to the domain of $\mathscr{A}_\tau$.

Second, we define modules $M_1, \ldots, M_s$ in such a way that the structure $\mathscr{K}_{M_1,\ldots,M_s}(\mathscr{A}_\tau)$ contains a substructure isomorphic to $\mathscr{A}_\alpha$ for some CCM α. Let M_j be such that under any instantiation $\{S_i : i \in I\}$ it satisfies the following condition:

(*) if $\mathscr{B} \models M_j(S_{i_1}, \ldots, S_{i_k})$ where $\mathscr{B}$ is a τ-structure with domain τ, $S_{i_1}, \ldots, S_{i_k} \in \tau$, then $S_{i_t}^{\mathscr{B}} = (S_{i_t}, \ldots, S_{i_t})$, $t \in [k]$.

It does not matter how M_j behaves with other vocabularies and domains. Let α be a CCM with variables X_i, $i \in I$, and atomic modules $M_1, \ldots, M_s$. Later modules α^1 and α^2 will be used as α here. For α we define modules $M_1, \ldots, M_s$ with property (*) and such that for some instantiation $\{S_i : i \in I\}$ it holds $\mathscr{A}_\tau \models M_j(S_{i_1}, \ldots, S_{i_k})$ iff $M_j(X_{i_1}, \ldots, X_{i_k})$ is an atom in M. We denote the structure $\mathscr{K}_{M_1,\ldots,M_s}(\mathscr{A}_\tau)$ by $\mathscr{K}_\alpha(\mathscr{A}_\tau)$.

Recall that $a_1, \ldots, a_r$ are all the different arities of symbols from τ. We consider a substructure of $\mathscr{K}_\alpha(\mathscr{A}_\tau)$ induced by the set $K'(\mathscr{A}_\tau) \subseteq K = \mathscr{P}(A^{a_1}) \cup \cdots \cup \mathscr{P}(A^{a_r})$, the domain of $\mathscr{K}_\alpha(\mathscr{A}_\tau)$ consisting of all singleton sets of the form $\{(S, \ldots, S)\} \in \mathscr{P}(A^{a_j})$, $S \in \tau$ and a_j is the arty of S. Note that the domain of this structure does not depend on specific modules $M_1, \ldots, M_s$). It is not difficult to see that $\mathscr{K}_\alpha(\mathscr{A}_\tau)$ restricted to $K'(\mathscr{A}_\tau)$ is isomorphic to $\mathscr{A}_M$. By Magic Lemma II, $\mathscr{A}_\tau \models M(S_i : i \in J)$ for any instantiation $\{S_i : i \in J\}$.

Now we are ready to define a homomorphism from $\mathscr{A}_{\alpha^2}$ to $\mathscr{A}_{\alpha^1}$. By the premise of Theorem 2, for any choice of modules $M_1, \ldots, M_s$, vocabulary τ, instantiation $\{S_i : i \in J\}$, and τ-structure $\mathscr{A}$, if $\mathscr{A} \models \alpha^1(S_i : i \in J)$ then also $\mathscr{A} \models \alpha^2(S_i : i \in J)$. Choose $M_1, \ldots, M_s$ as above for module α^1. Then $\mathscr{A}_\tau \models \alpha^1(S_i : i \in J)$, and so $\mathscr{A}_\tau \models \alpha^2(S_i : i \in J)$. By Magic Lemma II there is a homomorphism ψ of $\mathscr{A}_{\alpha^2}$ to $\mathscr{K}_{\alpha^1}(\mathscr{A}_\tau)$. As is easily seen, we may assume that the range of ψ is a subset of $K'(\mathscr{A}_\tau)$. Indeed, if $\psi(X_i) \notin K'(\mathscr{A}_\tau)$ for some $i \in I$, then by condition (*) no tuple in no relation of $\mathscr{A}_{\alpha^2}$ contains X_i, and its image can be chosen arbitrarily. Since $\mathscr{K}_{\alpha^1}(\mathscr{A}_\tau)$ restricted to $K'(\mathscr{A}_\tau)$ is isomorphic to $\mathscr{A}_{\alpha^1}$, we get the result. □

Theorem 2 reduces the CCM Containment problem to the Homomorphism problem of two relational structures. As the latter belongs to NP, we obtain the following corollary.

Corollary 1 (Membership in NP) *The CCM Containment problem belongs to NP.*

The complexity of the Homomorphism problem can be reduced if we restrict the class of allowed structures. The list of useful restrictions is long and includes bounded tree and hypertree width, various forms of acyclicity (see e.g. (Gottlob, Leone, and Scarcello 1999b,a; Gyssens, Jeavons, and Cohen 1994)), and fractional hyper tree width (Grohe and Marx 2006; Marx 2013). By Theorem 2 if we consider CCMs satisfying any of these conditions, the respective CCM Containment problem becomes polynomial time solvable.

Conclusion and Future Work

In this paper, we addressed the problem of checking modular system equivalence and containment. This problem is important, for example, in hierarchical and component-based development, in rapid prototyping and system verification. We proved that, for a large class of modular systems, namely for Conjunctive Compound Modules, system containment (and thus equivalence) problem is in the complexity class NP. This is a typical class of modular systems, conveniently described using an expressive subset of the operations. We discussed cases where the containment problem is solvable in polynomial time. We also introduced the notion of representability of modules in specific formalisms and provided examples of such representations.

An important future extension of our work would be to the Unions of Conjunctive Compound Modules, using Sagiv-Yannakakis theorem (Sagiv and Yannakakis 1980). We would also like to prove a Containment property for an efficient restriction of a dynamic version of the algebra (Ternovska 2019), where inputs and outputs of modules are specified, similarly to (Aamer et al. 2020a) and (Aamer et al. 2020b).

Acknowledgements

The authors are grateful to the anonymous referees for their helpful comments. This research is supported by Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Aamer, H.; Bogaerts, B.; Surinx, D.; Ternovska, E.; and Van den Bussche, J. 2020a. Executable first-order queries in the logic of information flows. In *Proc. ICDT'20*, 4:1–4:14.

Aamer, H.; Bogaerts, B.; Surinx, D.; Ternovska, E.; and Van den Bussche, J. 2020b. Inputs, Outputs, and Composition in the Logic of Information Flows. In *Proc. KR'20*, 2–11.

Chandra, A. K.; and Merlin, P. M. 1977. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC*, 77–90. ACM.

Codd, E. F. 1970. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* 13(6): 377–387.

Codd, E. F. 1972. Relational Completeness of Data Base Sublanguages. *Research Report / RJ / IBM / San Jose, California* RJ987.

de Bruijn, J.; Eiter, T.; Polleres, A.; and Tompits, H. 2007. Embedding Non-Ground Logic Programs into Autoepistemic Logic for Knowledge-Base Combination. In *Proc. IJCAI'07)*. Hyderabad, India: AAAI Press.

Eiter, T.; Ianni, G.; Lukasiewicz, T.; Schindlauer, R.; and Tompits, H. 2008. Combining answer set programming with description logics for the Semantic Web. *Artif. Intell.* 172(12-13): 1495–1539.

Enderton, H. B. 1972. *A mathematical introduction to logic*. Academic Press. ISBN 978-0-12-238450-9.

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.

Gottlob, G.; Leone, N.; and Scarcello, F. 1999a. A Comparison of Structural CSP Decomposition Methods. In *IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999*, 394–399.

Gottlob, G.; Leone, N.; and Scarcello, F. 1999b. On Tractable Queries and Constraints. In *Database and Expert Systems Applications, 10th Int. Conference, DEXA '99, Florence, Italy, August 30 - September 3, 1999*, 1–15.

Grohe, M.; and Marx, D. 2006. Constraint solving via fractional edge covers. In *SODA*, 289–298. ACM Press.

Gyssens, M.; Jeavons, P.; and Cohen, D. A. 1994. Decomposing Constraint Satisfaction Problems Using Database Techniques. *Artif. Intell.* 66(1): 57–89.

Hölldobler, S.; and Schweizer, L. 2014. Answer Set Programming and Clasp, A Tutorial. In *Proc. of the Young Scientists' Int. Workshop on Trends in Information Processing (YSIP)*, 77–95. CEUR Workshop Proceedings.

Järvisalo, M.; Oikarinen, E.; Janhunen, T.; and Niemelä, I. 2009. A Module-Based Framework for Multi-language Constraint Modeling. In *LPNMR*, volume 5753 of *Lecture Notes in Computer Science*, 155–168. Springer.

Libkin, L. 2004. *Elements of Finite Model Theory*. Springer Verlag.

Lierler, Y.; and Truszczynski, M. 2014. Abstract Modular Inference Systems and Solvers. In *Proc. PADL' 2014)*.

Marek, V. W.; and Truszczynski, M. 1999. Stable Models and an Alternative Logic Programming Paradigm. In *The Logic Programming Paradigm*, Artificial Intelligence, 375–398. Springer.

Marx, D. 2013. Tractable Hypergraph Properties for Constraint Satisfaction and Conjunctive Queries. *J. ACM* 60(6): 42.

Niemelä, I. 1999. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. *Ann. Math. Artif. Intell.* 25(3-4): 241–273.

Opatrny, J. 1979. Total ordering problem. *SIAM J. Comput* 8(1): 111–114.

Sagiv, Y.; and Yannakakis, M. 1980. Equivalences Among Relational Expressions with the Union and Difference Operators. *J. ACM* 27(4): 633–655.

Sebastiani, R. 2007. Lazy Satisfiability Modulo Theories. *Journal of Satisfiability, Boolean Modeling and Computation (JSAT)* 3: 141–224.

Tasharrofi, S.; and Ternovska, E. 2011. A Semantic Account for Modularity in Multi-language Modelling of Search Problems. In *FroCoS 2011*, 259–274.

Ternovska, E. 2019. An Algebra of Modular Systems: Static and Dynamic Perspectives. In *FroCoS, 2019*.

Trakhtenbrot, B. 1950. The Impossibility of an Algorithm for the Decidability Problem on Finite Classes. *Proceedings of the USSR Academy of Sciences (in Russian)* 70(4): 569–572.

Van den Bussche, J. 2001. Applications of Alfred Tarski's Ideas in Database Theory. In *CSL*, volume 2142 of *Lecture Notes in Computer Science*, 20–37. Springer.