An Algebra of Modular Systems: Static and Dynamic Perspectives

Eugenia Ternovska(✉)

Simon Fraser University, Burnaby, Canada
ter@sfu.ca

Abstract. We introduce static and dynamic algebras for specifying combinations of modules communicating among them via shared second-order variables. In the *static* algebra, atomic modules are classes of structures. They are composed using operations of extended Codd's relational algebra, or, equivalently, first-order logic with least fixed point. The *dynamic* algebra has essentially the same syntax, but with a specification of inputs and outputs in addition. In the dynamic setting, atomic modules are formalized in any framework that allows for the specification of their input-output behaviour by means of model expansion. Algebraic expressions are interpreted by binary relations on structures. We demonstrate connections of the dynamic algebra with a modal temporal logic and deterministic while programs.

1 Introduction

In this paper, we introduce a formalism for specifying and reasoning about modular systems. The goal is to be able to combine reusable components, potentially written in different languages, for solving complex computational tasks.[1] We start with first-order logic with fixpoints. We use an algebraic syntax, similar to Codd's relational algebra, but the idea is the same. We redefine FO(LFP), i.e., first-order logic with the least fixpoint operator, over a vocabulary of *atomic module symbols* that replaces a relational vocabulary. In this static setting, each atomic symbol is interpreted as a set of structures rather than a relational table (set of tuples). That is, by a boolean query, a decision procedure. Thus, while the syntax is first-order, the semantics is second-order because variables range over relations. This gives us the first logic.

The second stage is a dynamic setting where we add information flows. An information flow is a *propagation of information from inputs to outputs*. Formally, it is given by two functions, I and O that partition the relational variables of atomic modules into inputs and outputs. Semantically, modules are *binary relations on structures*. The relations describe how information propagates. This gives us an *algebra of binary relations*, where we can reason about information

[1] The heterogeneous components could be web services, knowledge bases, declarative specifications such as Integer Liner Programs, Constraint Satisfaction Problems, Answer Set Programs etc.

© Springer Nature Switzerland AG 2019
A. Herzig and A. Popescu (Eds.): FroCoS 2019, LNAI 11715, pp. 94–111, 2019.
https://doi.org/10.1007/978-3-030-29007-8_6

flows and control the expressive power by means of restricting the algebraic operations and the logics for axiomatizing atomic modules.

Algebras of binary relations have been studied before. Such an algebra was first introduced by De Morgan. It has been extensively developed by Peirce and then Schröder. It was abstracted to relation algebra RA by Jónsson and Tarski in [1]. For a historic perspective please see Pratt's informative historic overview paper [2]. More recently, relation algebras were studied by Fletcher, Van den Bussche, Surinx and their collaborators in a series of paper, see, e.g. [3,4]. The algebras of relations consider various subsets of operations on binary relations as primitive, and other as derivable. In another direction, [5,6] and others study partial functions and their algebraic equational axiomatizations.

When our algebra is interpreted over a pointed Kripke structure, it becomes a modal temporal (dynamic) logic. The logic allows one to specify patterns of execution inside (definable) modalities, similar to Dynamic Logic (see, e.g., [7]) and LDL$_f$ [8]. Just like in PDL and LDL$_f$, the main constructs of imperative programming (e.g., *while* loops) are definable. The main difference of our logic from PDL and LDL$_f$ is that we impose the Law of Inertia for atomic components: the interpretation of the variables not affected by a direct change must remain the same. In this way, the logic is similar to Reiter's situation calculus [9] and Golog [10]. However unlike the first-order successor state axioms of the situation calculus, we allow atomic changes to be non-deterministic, to be specified in a logic with any expressive power where the main computational task can be formalized as the task of Model Expansion, of any complexity. We formulate the main computational task, the Model Expansion task for processes, in the modal setting of our logic as the existence of an information flow that results in a successful computation.

This paper continues the line of research that started at FROCOS 2011 [11] and continued in [12] and then in [13] and [14]. Unlike the previous work, we base our static formalism in classical logic, so the set of our algebraic operations is different. We also develop a novel dynamic perspective, through an algebra of binary relations, and then a modal temporal (dynamic) logic. The development of the dynamic view constitutes most of this paper. Since, in our logic, all the variables that are not constrained by the algebraic expressions are implicitly cylinrified, the closest related work is that on cylindric algebras [15]. These algebras were introduced by Tarski and others as a tool in the algebraization of the first-order predicate calculus.[2] However, a fundamental difference is that, in our logic, unconstrained variables are not only cylindrified, but their interpretation, if not modified, is transferred to the next state by inertia. This property gives us, mathematically, a very different formalism, which is suitable for reasoning about the dynamics of information flows.

The rest of the paper is organized as follows. In Sect. 2, we define the main computational task of Model Expansion in the context of related tasks. Then, in Sect. 3, we introduce the syntax and two different semantics, static and dynamic, of our algebras. The algebra under the dynamic semantics is called a Logic of

[2] See [16] for a historic context in applications to Database theory.

Information Flows (LIF). In Sect. 4, we show that, just by adding input and output specifications to classical logic (here, in an algebraic form), we obtain multiple operations that are given as primitive in many algebras of binary relations. In Sect. 5, we show a connection with modal logic. Finally, we conclude, in Sect. 6, with a broader perspective and future research directions.

2 Model Expansion, Related Tasks

Model Expansion [17] is the task of expanding a structure to satisfy a specification (a formula in some logic). It is the central task in declarative programming: in Answer Set Programming, Constraint Satisfaction Problem, Integer Linear Programming, Constraint Programming, etc. In this section, we define Model Expansion and compare it to two other related computational problems.

For a formula ϕ in any logic $\mathcal{L}$ with model-theoretic semantics, we can associate the following three tasks (all three for the same formula), satisfiability (SAT), model checking (MC) and model expansion (MX). We now define them for the case where ϕ has no free object variables.

Definition 1 (Satisfiability (SAT_ϕ)). *Given: Formula ϕ. Find: structure $\mathfrak{B}$ such that $\mathfrak{B} \models \phi$. (The decision version is: Decide: $\exists \mathfrak{B}$ such that $\mathfrak{B} \models \phi$?).*

Definition 2 (Model Checking (MC_ϕ)). *Given: Formula ϕ, structure $\mathfrak{A}$ for vocab(ϕ). Decide: $\mathfrak{A} \models \phi$? There is no search counterpart for this task.*

The following task (introduced in [17]) is at the core of this paper. The decision version of it can be seen as being of the form "guess and check", where the "check" part is the model checking task we just defined.

Definition 3 (Model Expansion (MX^σ_ϕ)). *Given: Formula ϕ with designated input vocabulary $\sigma \subseteq vocab(\phi)$ and σ-structure $\mathfrak{A}$. Find: structure $\mathfrak{B}$ such that $\mathfrak{B} \models \phi$ and expands σ-structure $\mathfrak{A}$ to vocab(ϕ). (The decision version is: Decide: $\exists \mathfrak{B}$ such that $\mathfrak{B} \models \phi$ and expands σ-structure $\mathfrak{A}$ to vocab(ϕ)?).*

Any logic that can be interpreted over first-order (Tarski) structures can be used for writing specifications ϕ. In general, vocabulary σ can be empty, in which case the input structure $\mathfrak{A}$ consists of a domain only. When $\sigma = vocab(\phi)$, model expansion collapses to model checking, $\mathsf{MX}^\sigma_\phi = \mathsf{MC}_\phi$. Note that, in general, the domain of the input structure in MC and MX can be infinite. For complexity analysis, in this paper, we focus on finite input structures.

Let ϕ be a sentence, i.e., has no free object variables. Data complexity [18] is measured in terms of the size of the finite active domain. For the decision versions of the problems, data complexity of MX lies in-between model checking (full structure is given) and satisfiability (no part of structure is given):

$$\mathsf{MC}_\phi \leq \mathsf{MX}^\sigma_\phi \leq \mathsf{SAT}_\phi.$$

For example, for FO logic, MC is non-uniform AC^0, MX captures NP (Fagin's theorem), and SAT is undecidable. In SAT, the domain is not given. In MC and

MX, at least, the (active) domain is always given, which significantly reduces the complexity of these tasks compared to SAT. The relative complexity of the three tasks for several logics, including ID-logic of [19] and guarded logics, has been studied in [20].

In this paper, we will view Model Expansion as a (nondeterministic) transduction, i.e., a binary relation from input to outputs, that are τ-structures. We will develop an algebra of such transductions. The following example illustrates what we will consider as an atomic transduction. In the development of our algebra, we will abstract away from what exactly the atomic transductions are. We will become more specific towards the end of the paper, when we restrict our attention to a specific logic and prove a complexity result.

Example 1. Consider the following first-order formula with free relational variables. $\phi_{3\text{Col}}(V, E, R, G, B) :=$

$$\begin{array}{l}\forall x\ (V(x) \rightarrow [R(x) \vee B(x) \vee G(x)]) \wedge \\ \forall x\ (V(x) \rightarrow \neg[(R(x) \wedge B(x)) \vee (R(x) \wedge G(x)) \vee (B(x) \wedge G(x))]) \\ \quad \wedge \quad \forall x \forall y\ [V(x) \wedge V(y) \wedge E(x,y) \rightarrow \\ \neg((R(x) \wedge R(y)) \vee (B(x) \wedge B(y)) \vee (G(x) \wedge G(y)))].\end{array}$$

This formula axiomatizes a class of structures. A class of structures, which is closed under isomorphism, represents a boolean query. In this case, the query specifies all 3-colourable graphs with all their proper clourings. If we identify $I_1 = \{E, V\}$ as the input vocabulary, and $O_1 = \{R, G, B\}$ as the output (solution) vocabulary, then we obtain the classic 3-Colouring computational problem. It can be viewed as a transduction or a binary relation on structures, defined by the binary semantics below. We can also identify $I_2 = \{V, R, G, B\}$ as an input vocabulary, and $O_2 = \{E\}$ as the output, and it will give us a rather different computational problem, with no specific name.

One of the parameters to control the expressive power of the logic is the formalism for the atomic transductions (atomic modules). In the example above, the axiomatization is first-order, and the free second-order variables implicitly make it $\exists$SO. But later in the paper, we consider axiomatizations that are output-monadic non-recursive Datalog programs, which are much less expressive.

3 Algebras: Static and Dynamic

For essentially 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, which corresponds to solving MX task for those modules. The algebras correspond to classical and modal logics (as we will see later), respectively. We use a version of Codd's relational algebra instead of first-order logic, since we need an algebraic notation, however, the equivalence of the two formalisms is well-known.

Syntax. Assume we have a countable sequence Vars $= (X_1, X_2, \dots)$ of *relational variables* each with an associated finite *arity*. For convenience, we use X, Y, Z, etc. Let ModAt $= \{M_1, M_2, \dots\}$ be a fixed vocabulary of *atomic module symbols*. Each $M_i \in$ 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 $vvoc(M) = (X_{i_1}, \dots, X_{i_k})$. Similarly, ModVars $= \{Z_1, Z_2, \dots\}$ is a countable sequence of *module variables*, where each $Z_j \in$ 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 of the form $M_i(\bar{X})$, δ is any finite set of relational variables in Vars, Θ is any expression of the form $X = Y$, for relational variables of equal arity that occur in Vars, Z_j is a module variable in ModVars which must occur positively in the expression α, i.e., under an even number of the complementation $(^-)$ operator. By equality symbol '=' in Selection condition Θ, we mean the equality of the interpretations. It is a slight abuse of notations, however the definition of the semantics specifies the intended meaning precisely.

Atomic modules can be specified in any formalism with a model-theoretic semantics. For example, we saw an axiomatization of 3Colouring in Example 1. Modules occurring within one algebraic expression can even be axiomatized in different logics, if needed. They can also be viewed as abstract decision procedures. But, as far as the static algebra is concerned,

their only relevant feature is the *classes of structures they induce*.

When the domain is specified, we talk about sets of structures rather than classes.

Static (Unary) Semantics. Fix a finite relational vocabulary τ. Algebraic expressions will be used as "constraints". A *variable assignment* s is a function that assigns, to each relational variable, a symbol in τ of the same arity. We introduce notation $V := s^{-1}(\tau)$. Clearly, $V \subset Vars$. Function s gives us the flexibility to apply the same algebraic expression in multiple contexts, without a priori binding to a specific vocabulary.

Now fix a domain *Dom*.[3] The domain can be finite or infinite. Let $\mathbf{U}$ be the set of all τ-structures over the domain *Dom*. The following definition is mathematically necessary in defining the semantics of atomic modules.

Definition 4. *Given a sub-vocabulary γ of τ, a subset $W \subseteq \mathbf{U}$ is* determined by γ *if it satisfies*

$$\begin{array}{c} \textit{for all } \mathfrak{A}, \mathfrak{B} \in \mathbf{U} \textit{ such that } \mathfrak{A}|_\gamma = \mathfrak{B}|_\gamma \textit{ we have} \\ \mathfrak{A} \in W \textit{ iff } \mathfrak{B} \in W. \end{array}$$

[3] Usually, in applications, domain *Dom* is the (active) domain of an input structure for a task of interest such as MX.

Given a well-formed algebraic expression α defined by (1), we say that structure $\mathfrak{A}$ *satisfies* α (or that is a *model* of α) under variable assignment s, notation $\mathfrak{A} \models_s \alpha$, if $\mathfrak{A} \in [\alpha]^s$, where *unary interpretation* $[\ \cdot\ ]^s$ is defined as follows. Given a variable assignment s, function $[\ \cdot\ ]^s$ assigns a subset $[M_i]^s \subseteq \mathbf{U}$ and a subset $[Z_j]^s \subseteq \mathbf{U}$ to each atomic module symbol $M_i \in \text{ModAt}$ and each module variable $Z_j \in \text{ModVars}$, with the property that $[M_i]^s$ is determined by $s(vvoc(M_i))$ (respectively, $[Z_j]^s$ is determined by $s(vvoc(Z_j))$). The unary interpretation of atomic modules $[\ \cdot\ ]^s$ (parameterized with s) can be viewed as a function that provides "oracles" or decision procedures, or answers to boolean queries. In general, these oracles can be of arbitrary computational complexity.

We extend the definition of $[\ \cdot\ ]^s$ to all algebraic expressions.

$$
\begin{array}{l}
[\mathrm{id}]^s := \mathbf{U}. \\
[\alpha_1 \cup \alpha_2]^s := [\alpha_1]^s \cup [\alpha_2]^s. \\
[\alpha^-]^s := \mathbf{U} \setminus [\alpha]^s. \\
[\pi_\delta(\alpha)]^s := \{\mathfrak{A} \in \mathbf{U} \mid \exists \mathfrak{B}\ (\mathfrak{B} \in [\alpha]^s \text{ and } \mathfrak{A}|_{s(\delta)} = \mathfrak{B}|_{s(\delta)})\}. \\
[\sigma_{X=Y}(\alpha)]^s := \{\mathfrak{A} \mid \mathfrak{A} \in [\alpha]^s \text{ and } \mathfrak{A}|_{s(X)} = \mathfrak{A}|_{s(Y)}\}. \\
[\mu Z_j.\alpha]^s := \bigcap \{R \subseteq \mathbf{U} \mid [\alpha]^s_{[Z:=\alpha]} \subseteq R\}.
\end{array}
$$

Here, $[\alpha]^s_{[Z:=\alpha]}$ means an interpretation that is exactly like given by the function $[\ \cdot\ ]^s$, except Z is interpreted as α. Note that Projection $\pi_\delta(\alpha)$ is equivalent to cylindrification $C_\gamma(\alpha)$, where $\gamma = V \setminus \delta$.

Free and Bound Variables. These notions are exactly the same as in classical logic. The role of an existentional quantifier is played by Cylinderfication. We define them as follows. $free(M) := vvoc(M)$, $free(\mathrm{id}) := \varnothing$, $free(\alpha \cup \beta) := free(\alpha) \cup free(\alpha)$, $free(\alpha^-) := free(\alpha)$, $free(\pi_\delta(\alpha)) := \delta$, $free(\sigma_{X=Y}(\alpha)) := free(\alpha) \cup \{X, Y\}$, $free(\mu_{\bar{X},Z}\alpha[\bar{X} : \bar{t}] := free(\bar{t}) \cup (free(\alpha) \setminus \{\bar{X}, Z\})$. Taking into account that Projection $\pi_\delta(\alpha)$ is equivalent to cylindrification $C_\gamma(\alpha)$, where $\gamma = vvoc(\alpha) \setminus \delta$, we also have: $free(C_\gamma(\alpha)) := free(\alpha) \setminus \gamma$. Bound variables are defined as those that are not free.

Implicit Cylindrification. Algebraic expressions can be viewed as constraints on the free variables. The following proposition shows that everything outside the free variables of α is implicitly cylindrified. Recall that $V := s^{-1}(\tau)$.

Proposition 1. *If α is an atomic module symbol, then* $[\alpha] = [\pi_{free(\alpha)}(\alpha)]$.

Proof. The proposition holds for the atomic case because, by the static semantics of atomic modules, the set of structures $[M_i]^s$ that interprets an atomic module is determined by $s(vvoc(M_i))$, see Definition (4), and $vvoc(M_i) = free(M_i)$.

We now give a binary semantics to the algebra. The algebra under this semantics is called a **Logic of Information Flows (LIF)**.[4]

Dynamic (Binary) Semantics. The Dynamic semantics is produced by adding information flows. Such flows are initiated by Model Expansion task,

[4] Please note that the goals of this paper have no connection with information flows in security.

where we provide inputs by giving a part of a structure (say, a graph), and expand to obtain the solution part (say, a possible 3-colouring). Since information propagates from inputs to outputs, we introduce two functions that specify inputs and outputs of atomic module symbols, respectively. As a consequence of specifying inputs and outputs, we transition to generalized binary expressions. Projection now has two cases, for the left and the right parts of the binary relation it applies to, and Selection has three – left, right and mixed.

Let $\mathrm{ModAt}_{\mathrm{I/O}}$ denote the set of all atomic module symbols M with all possible *partitions* of $vvoc(M)$ into inputs and outputs, i.e., $I(M) \cup O(M) = vvoc(M)$ and $I(M) \cap O(M) = \varnothing$.[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 $\mathrm{ModVars}_{\mathrm{I/O}}$. The well-formed algebraic expression α is defined, again, by (1), except, in the atomic case, we have atomic module symbols (respectively, module variables) from $\mathrm{ModAt}_{\mathrm{I/O}}$ (respectively, from $\mathrm{ModVars}_{\mathrm{I/O}}$).

While inputs and outputs of atomic modules are always given, the situation with inputs $I(\alpha)$ and outputs $O(\alpha)$ of a general algebraic expression α is much more complicated. The problem is that it is not always possible to syntactically identify the variables whose interpretations are needed as *conditions* for applying algebraic expression α, and those that are the *effects* of α, i.e., can potentially be modified by the expression. A detailed analysis, for a general setting, is a subject of an ongoing collaborative work.

Let s be as above. Given a well-formed α, we say that pair of structures $(\mathfrak{A}, \mathfrak{B})$, *satisfies* α under variable assignment s, notation $(\mathfrak{A}, \mathfrak{B}) \models_s \alpha$, if $(\mathfrak{A}, \mathfrak{B}) \in [\![\alpha]\!]^s$, where *binary interpretation* $[\![\cdot]\!]^s$ is defined by I and II below.

I. Binary Semantics: Atomic Modules and Variables

Definition 5. *For atomic modules in* $\mathrm{ModAt}_{\mathrm{I/O}}$*, we have:*

$$[\![M]\!]^s := \big\{(\mathfrak{A}, \mathfrak{B}) \in \mathbf{U} \times \mathbf{U} \mid \text{ there exists } \mathfrak{C} \in [M]^s \text{ such that}$$

$$\mathfrak{C}|_{s(I(M))} = \mathfrak{A}|_{s(I(M))}, \tag{2}$$

$$\mathfrak{C}|_{s(O(M))} = \mathfrak{B}|_{s(O(M))} \tag{3}$$

$$\text{and } \ \mathfrak{A}|_{\tau \setminus s(O(M))} = \mathfrak{B}|_{\tau \setminus s(O(M))}\big\}. \tag{4}$$

That is, in each pair of structures in the interpretation of an atomic module, the structure on the left agrees with the unary semantics on the inputs, and the structure on the right agrees with the unary semantics on the outputs. While in the unary semantics, everything that is not explicitly mentioned is implicitly cylindrified, here the situation is different. Intuitively, on states where it is defined, an atomic module produces a replica of the current structure except the interpretation of the output vocabulary changes as specified by the action. This preservation of unmodified information, while intuitively obvious, is an

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

important technical property. For this reason, we call it, rather ostentatiously, the Law of Inertia. The semantics is defined in a way so that atomic modules impose constraints on the $vvoc(M)$ only. The semantics of module variables $Z \in \mathrm{ModVars}_{\mathrm{I/O}}$ is defined in exactly the same way as the semantics for atomic module constants.

Properties of Binary Atomic Modules. Before giving binary semantics to the operations, we clarify the properties of the semantics of atomic modules by the following proposition.[6]

Proposition 2. *For all atomic modules, we have, for all structures* $\mathfrak{A}$, $\mathfrak{B}$*:*

$$(a)\ (\mathfrak{A},\mathfrak{B}) \in [\![M]\!]^s \ \Rightarrow (\mathfrak{B},\mathfrak{B}) \in [\![M]\!]^s,$$
$$(b)\ (\mathfrak{B},\mathfrak{B}) \in [\![M]\!]^s \ \Leftrightarrow \mathfrak{B} \in [M]^s.$$

Proof. (a) Assume, towards a contradiction that (a1) $(\mathfrak{A},\mathfrak{B}) \in [\![M]\!]^s$, but (a2) $(\mathfrak{B},\mathfrak{B}) \notin [\![M]\!]^s$. Assumption (a1) implies, by the definition of the binary semantics of atomic modules, that there exists $\mathfrak{C} \in [M]^s$ such that conditions (2)–(4) hold. By (4), which is the Law of Inertia, $\mathfrak{A}|_{s(I(M))} = \mathfrak{B}|_{s(I(M))}$. This is because $s(I(M)) \subseteq \tau \setminus s(O(M))$ since $I(M) \cap O(M) = \varnothing$, so the Law of Inertia applies. Thus, $\mathfrak{C}|_{s(I(M))} = \mathfrak{A}|_{s(I(M))} = \mathfrak{B}|_{s(I(M))}$. Assumption (a2) implies that for all $\mathfrak{C} \in [M]^s$, at least one of the conditions (2)–(4), where $\mathfrak{A} = \mathfrak{B}$, must be violated for all structures $\mathfrak{B}$. Violation of (2) and (3) is impossible by our conclusion from the assumption (1a). Violation of (4) is impossible because $\mathfrak{A}$ is the same as $\mathfrak{B}$ in this case.

(b, $\Rightarrow$) Assume $(\mathfrak{B},\mathfrak{B}) \in [\![M]\!]^s$. Then, by Definition 5 of the binary semantics for atomic modules, there exists $\mathfrak{C} \in [M]^s$ such that $\mathfrak{C}|_{s(I(M))} = \mathfrak{A}|_{s(I(M))}$ and $\mathfrak{C}|_{s(O(M))} = \mathfrak{B}|_{s(O(M))}$. By Proposition 1, in the case of atomic modules, $[M] = [\pi_{free(M)}(M)]$. Thus, since $free(M) = vvoc(M) = I(M) \cup O(M)$, it does not matter how $\mathfrak{B}$ interprets symbols outside $s(free(M))$, and $\mathfrak{C}$ can be taken to be $\mathfrak{B}$.

(b, $\Leftarrow$) Assume $\mathfrak{B} \in [M]^s$. Then, by the unary semantics, there exists $\mathfrak{C} \in [M]^s$ such that $\mathfrak{C}|_{s(I(M))} = \mathfrak{B}|_{s(I(M))}$ and $\mathfrak{C}|_{s(O(M))} = \mathfrak{B}|_{s(O(M))}$. Take $\mathfrak{C} = \mathfrak{B}$. Obviously, $\mathfrak{B} = \mathfrak{B}$ outside of the outputs of M. Thus, all three conditions of Definition 5 are satisfied and $(\mathfrak{B},\mathfrak{B}) \in [\![M]\!]^s$.

II. Binary Semantics: the Remaining Cases. We are now ready to extend the binary interpretation $[\![\cdot]\!]^s$ to all algebraic expressions α:

[6] Part (b) of this proposition is stated without proof as Theorem 4.1 for *compound expressions* in Shahab Tasharrofi thesis. The language has Projection, Sequential Composition, Union and Feedback. The operations of Projection and Sequential Composition have a different semantics than ours, and we do not have Feedback.

$$
\begin{array}{l}
[\![\mathrm{id}]\!]^s := \{(\mathfrak{A},\mathfrak{B}) \in \mathbf{U}\times\mathbf{U} \mid \mathfrak{A}=\mathfrak{B}\},\\
[\![\alpha_1 \cup \alpha_2]\!]^s := [\![\alpha_1]\!]^s \cup [\![\alpha_2]\!]^s,\\
[\![\alpha^-]\!]^s := \mathbf{U}\times\mathbf{U} \setminus [\![\alpha]\!]^s,\\
[\![\mu Z_j.\alpha]\!]^s := \bigcap \{R \subseteq \mathbf{U}\times\mathbf{U} \mid [\![\alpha]\!]^s_{[Z:=R]} \subseteq R\},\\
[\![\pi^l_\delta(\alpha)]\!]^s := \{(\mathfrak{A},\mathfrak{B}) \in \mathbf{U}\times\mathbf{U} \mid\\
\qquad \exists\ (\mathfrak{A}',\mathfrak{B}) \in [\![\alpha]\!]^s \text{ such that } \mathfrak{A}'|_{s(\delta)} = \mathfrak{A}|_{s(\delta)}\},\\
[\![\pi^r_\delta(\alpha)]\!]^s := \{(\mathfrak{A},\mathfrak{B}) \in \mathbf{U}\times\mathbf{U} \mid\\
\qquad \exists\ (\mathfrak{A},\mathfrak{B}') \in [\![\alpha]\!]^s \text{ such that } \mathfrak{B}'|_{s(\delta)} = \mathfrak{B}|_{s(\delta)}\},\\
[\![\sigma^l_{X=Y}(\alpha)]\!]^s := \{(\mathfrak{A},\mathfrak{B}) \in [\![\alpha]\!]^s \mid (s(X))^{\mathfrak{A}} = (s(Y))^{\mathfrak{A}}\},\\
[\![\sigma^r_{X=Y}(\alpha)]\!]^s := \{(\mathfrak{A},\mathfrak{B}) \in [\![\alpha]\!]^s \mid (s(X))^{\mathfrak{B}} = (s(Y))^{\mathfrak{B}}\},\\
[\![\sigma^{lr}_{X=Y}(\alpha)]\!]^s := \{(\mathfrak{A},\mathfrak{B}) \in [\![\alpha]\!]^s \mid (s(X))^{\mathfrak{A}} = (s(Y))^{\mathfrak{B}}\}.
\end{array}
$$

Operation id is sometimes called the "*nil*" action, or it can be seen as an empty word which is denoted ε in the formal language theory. It is convenient to extend the selection operation to $\Theta \in \{X = Y, X \neq Y, X = R, X \neq R\}$, where R is a relational constant. This extension is done in an obvious way. According to the semantics, Left Projection keeps the interpretation of a subset of the vocabulary in the first element of the binary relation defined by α while cylindrifying everything else on the left. It keeps the second element of the binary relation intact. The semantics of Right Projection is defined symmetrically.[7]

Standard Models: Induction Principle. The semantics of the algebra of binary relations on $\mathbf{U}$ gives us transition systems (Kripke structures) with states that are elements of $\mathbf{U}$ and transition given by the binary semantics. In this paper, we are interested in reachability from the input structure. We need to ensure categoricity of the theories in the logic, to avoid non-standard models that, in particular, do not originate in the input structure. For that purpose, we *semantically* impose the following restriction:

> *only structures reachable from the input structures by means of applying atomic modules are in the allowable Kripke models.*

This semantic constraint can also be imposed axiomatically, although we do not do it in this paper. For example, in Dynamic Logic, which is a fragment of the Logic of Information Flows, it would be expressed by an axiom schema $p \wedge [a^*](p \rightarrow [a]p) \rightarrow [a^*]p$. This schema is a form of an inductive definition. Such a definition always has a construction principle that specifies how to construct a set, and an induction principle that says "nothing else is" in the set being defined. Together, the two principles produce, depending on the logic, an axiom similar to the the second-order induction axiom of Peano Arithmetic or the Dynamic Logic axiom above [21].[8]

[7] Equivalently, we could have introduced appropriate Cylindrification operations instead of the two Projections.

[8] The idea of connecting dynamic systems with Peano Arithmetic goes back to Reiter [9]. He introduced second-order Induction axiom to the Basic Action Theory of the situation calculus, which is a formalism for reasoning about actions based on classical first and second-order logic.

In addition to the algebraic operations above, we will also use Sequential Composition $(\alpha;\beta)$. This operation is sometimes also called relative, dynamic, or multiplicative conjunction as its properties are similar to the properties of the logical (additive, static) conjunction $(\alpha \cap \beta)$. The semantics of sequential composition is given as follows.

$$[\![\alpha;\beta]\!] := \{(\mathfrak{A},\mathfrak{B}) \mid \exists \mathfrak{C}((\mathfrak{A},\mathfrak{C}) \in [\![\alpha]\!] \text{ and } (\mathfrak{C},\mathfrak{B}) \in [\![\beta]\!])\}.$$

This operation is definable, under some conditions on inputs and outputs, through the other operations. The full study of the primitivity of this operation is an ongoing collaborative work.

As a decision task, we are interested in checking whether a program α has a successful execution, including a witness for its free relational variables, starting from an input structure $\mathfrak{A}$. This is specified by $\mathfrak{A} \models_s |\alpha\rangle\mathbf{T}$, where $|\alpha\rangle$ is a right-facing possibility modality, and $\mathbf{T}$ represents *true*, that is, all states. We formally introduce and explain this modality in Sect. 5 on Modal Logic. To evaluate α in $\mathfrak{A}$, we use s to match the vocabulary of $\mathfrak{A}$ with the relational input variables of α, while matching the arities as well, and then apply the binary semantics as defined above. We will come back to this decision task in Definition 7.

Static-Dynamic Duality for Atomic Modules. Note that, for a given domain, each atomic module is, simultaneously, (a) a set of structures, according to the unary semantics, and (b) a binary relation, i.e., a set of pairs of structures, according to the binary semantics.

4 Definable Constructs

We now introduce several *definable* operations, and we study some of their properties. All of those constructs are present in algebras of binary relations and partial functions. There are studies on which operations are primitive and which are definable [3]. It turns out that the only thing lacking in classical logic to define most of these constructs is information propagation, i.e., a specification of inputs and outputs. By adding it, we obtain a surprisingly rich logic. In the following, we assume that all structures range over universe $\mathbf{U}$, and all pairs of structures over $\mathbf{U} \times \mathbf{U}$.

Set-Theoretic Operations

$$\begin{array}{ll}
\mathrm{di} := \mathrm{id}^-, & \text{(diversity)} \\
\top := \mathrm{id}^- \cup \mathrm{id}, & \text{(all)} \\
\bot := \top^-, & \text{(empty)} \\
\alpha \cap \beta := (\alpha^- \cup \beta^-)^-, & \text{(intersection)} \\
\alpha - \beta := (\alpha^- \cup \beta)^-, & \text{(difference)} \\
\alpha \sim \beta := (\alpha^- \cup \beta) \cap (\beta^- \cup \alpha). & \text{(similar)}
\end{array}$$

In the following, we use $I(\alpha)$ and $O(\alpha)$ as a generalization of inputs and outputs of atomic modules to compound algebraic expressions.[9]

Projection onto the Inputs (Domain). $\mathrm{Dom}(\alpha) \ := \pi^l_{I(\alpha)}(\alpha) \cap \mathrm{id}$. This operation is also called "projection onto the first element of the binary relation". It identifies the states in V where there is an outgoing α-transition. Thus,

$$[\![\mathrm{Dom}(\alpha)]\!] = \{(\mathfrak{B}, \mathfrak{B}) \mid \exists \mathfrak{B}' \ \ (\mathfrak{B}, \mathfrak{B}') \in [\![\alpha]\!]\}.$$

Projection onto the Outputs (Image). $\mathrm{Img}(\alpha) \ := \pi^r_{O(\alpha)}(\alpha) \cap \mathrm{id}$. This operation can also be called "projection onto the second element of the binary relation". It follows that

$$[\![\mathrm{Img}(\alpha)]\!] = \{(\mathfrak{B}, \mathfrak{B}) \mid \exists \mathfrak{B}' \ \ (\mathfrak{B}', \mathfrak{B}) \in [\![\alpha]\!]\}.$$

Forward Unary Negation (Anti-domain). Regular complementation includes all possible transitions except α. We introduce a stronger negation which is essentially unary (binary with equal elements in the pair) and excludes states where α originates.

$$\curvearrowright \alpha \ := \ (\pi_{I(\alpha)}(\alpha))^- \cap \mathrm{id}.$$

It says "there is no outgoing α-transition". By this definition,

$$[\![\curvearrowright \alpha]\!] = \{(\mathfrak{B}, \mathfrak{B}) \mid \ \forall \mathfrak{B}' \ \ (\mathfrak{B}, \mathfrak{B}') \notin [\![\alpha]\!]\}.$$

Backwards Unary Negation (Anti-image). We define a similar operation for the opposite direction.

$$\curvearrowleft \alpha \ := \ (\pi_{O(\alpha)}(\alpha))^- \cap \mathrm{id}.$$

It says "there is no incoming α-transition". We obtain:

$$[\![\curvearrowleft \alpha]\!] = \{(\mathfrak{B}, \mathfrak{B}) \mid \ \forall \mathfrak{B}' \ \ (\mathfrak{B}', \mathfrak{B}) \notin [\![\alpha]\!]\}.$$

Each of the unary negations is a restriction of the regular negation (complementation). Unlike regular negation, these operations preserve determinism of the components. In particular, De Morgan's Law does not hold for $\curvearrowright$ and $\curvearrowleft$. We have demonstrated that these connectives have the properties of the Intuitionistic negation. The proofs do not fit into the conference format and will be given in a journal version of this paper.

[9] We do not give a formal definition of the more general concept of inputs and outputs here since it is long and an informal understanding is sufficient. The formal definition will be given in another paper (with coauthors).

Iteration (Kleene Star). This operator is the *iteration* operator, also called the Kleene star. The expression α^* means "execute α some nondeterministically chosen finite number of times. We define it as follows: $\alpha^* := \mu Z.(\mathrm{id} \cup Z; \alpha)$. By this definition,

$$\begin{aligned}[\![\alpha^*]\!] = \{(\mathfrak{A}, \mathfrak{B}) \mid\ & \mathfrak{A} = \mathfrak{B} \text{ or there exists } n < 0 \\ & \text{and } \mathfrak{C}_0, \ldots, \mathfrak{C}_n \in \mathbf{U} \text{ such that} \\ \mathfrak{A} = \mathfrak{C}_0, \mathfrak{B} = \mathfrak{C}_n, & \text{ and for all } i < n, (\mathfrak{C}_i, \mathfrak{C}_{i+1}) \in [\![\alpha]\!]\}.\end{aligned}$$

That is, α^* is a transitive reflexive closure of α.

Converse. This operation is equivalent to switching $I(\alpha)$ and $O(\alpha)$. It changes the direction of information propagation. The semantics is as follows.

$$[\![\alpha^\smile]\!] := \{(\mathfrak{A}, \mathfrak{B}) \mid (\mathfrak{B}, \mathfrak{A}) \in \alpha\}.$$

Converse is *implicitly* definable: $\beta = \alpha^\smile$ iff $\begin{array}{l}\mathrm{Dom}(\alpha) = \mathrm{Img}(\beta), \\ \mathrm{Dom}(\beta) = \mathrm{Img}(\alpha).\end{array}$

Logical Equivalence (Equality of Algebraic Terms). We say that α and β are *logically equivalent*, notation $\alpha = \beta$ if $\big((\mathfrak{A}, \mathfrak{B}) \models_s \alpha \text{ iff } (\mathfrak{A}, \mathfrak{B}) \models_s \beta\big)$, for all τ-structures $\mathfrak{A}$, $\mathfrak{B}$, for any variable assignment s.

The following proposition clarifies semantical connections between the operations.

Proposition 3.

$$\begin{aligned}\curvearrowright\alpha &= \mathrm{Dom}(\alpha)^- \cap \mathrm{id} = \mathrm{Dom}(\alpha^-) - \mathrm{Dom}(\alpha) = \curvearrowright\mathrm{Dom}(\alpha) = \curvearrowleft\mathrm{Dom}(\alpha), \\ \curvearrowleft\alpha &= \mathrm{Img}(\alpha)^- \cap \mathrm{id} = \mathrm{Img}(\alpha^-) - \mathrm{Img}(\alpha) = \curvearrowright\mathrm{Img}(\alpha) = \curvearrowleft\mathrm{Img}(\alpha),\end{aligned}$$

$$\begin{aligned}\mathrm{Dom}(\alpha) &= \curvearrowright\curvearrowleft\alpha, \quad \mathrm{id} = \curvearrowright\bot = \curvearrowleft\bot, \qquad\qquad \curvearrowright\curvearrowleft\curvearrowright\alpha = \curvearrowright\alpha, \\ \mathrm{Img}(\alpha) &= \curvearrowleft\curvearrowright\alpha, \quad \bot = \curvearrowright\mathrm{id} = \curvearrowleft\mathrm{id} = \curvearrowright\top = \curvearrowleft\top, \ \curvearrowleft\curvearrowright\curvearrowleft\alpha = \curvearrowleft\alpha.\end{aligned}$$

Proof. The logical equivalences follow directly from the semantics of the operations.

Notice that $\curvearrowright$ inherits a property of intuitionistic negation: $\curvearrowright\curvearrowright\alpha \neq \alpha$. This is because Anti-domain ($\curvearrowright$), when applied twice, gives us Domain of α, which is clearly different from α itself. But Domain of Anti-domain is Anti-domain, so $\curvearrowright\curvearrowright\curvearrowright\alpha = \mathrm{Dom}(\curvearrowright\alpha) = \curvearrowright\alpha$. It is also possible to show that $\curvearrowright$ and $\curvearrowleft$ distribute over $\cap$ and $\cup$, so De Morgan Law does not hold for them. Also, $\curvearrowright\top = \bot$, but $\curvearrowright\bot \neq \top$. Indeed, $\curvearrowright\bot = \mathrm{id}$.

5 Modal Logic

We now define a modal logic which we call $L\mu\mu$, since it is similar to the mu-calculus $L\mu$, but has two fixed points, unary and binary. The modal logic is used, in particular, to formalize the main computational task, the Model Expansion task for processes in Definition 7.

5.1 Two-Sorted Syntax, $L\mu\mu$

The algebra with information flows can be equivalently represented in a "two-sorted" syntax, with sorts for processes (α) and state formulae (ϕ). This syntax gives us a modal logic, similar to Dynamic Logic. The syntax is given by the grammar:

$$\begin{array}{c} \alpha ::= \mathrm{id} \mid M_a \mid Z_j \mid \alpha \cup \alpha \mid \alpha^- \mid \pi_\delta(\alpha) \mid \sigma_\Theta(\alpha) \mid \phi? \mid \mu Z_j.\alpha \\ \phi ::= \mathbf{T} \mid M_p \mid X_i \mid \ \phi \vee \phi \mid \neg\phi \mid |\alpha\rangle\ \phi \mid \langle\alpha|\ \phi \mid \mu X_i.\phi. \end{array} \tag{5}$$

The first line defines *process formulae*. It is essentially our original syntax (1). The second line specifies *state formulae*. There, we have two possibility modalities, $|\alpha\rangle$ is a forward "exists execution of α" modality, and $\langle\alpha|$ is its backwards counterpart. We can also introduce their duals, the two necessity modalities: $|\alpha]\ \phi := \neg(\ |\alpha\rangle\ \neg\phi)$ and $[\alpha|\ \phi := \neg(\langle\alpha|\ \neg\phi)$. Symbols M_a stand for modules that are "actions". Symbols M_p stand for modules that are "propositions". Operation $\mathbf{T}$ represents a proposition that is true in every state. It replaces id under unary semantics.[10]

Test $\phi?$ turns every unary operation in the second line into a binary one by repeating the arguments, such as in e.g. going from $p(x)$ to $p(x, x)$, i.e., they are (partial) identities on $\mathbf{U}$. *Atomic tests* are (a) atomic modules-propositions (MC modules) and (b) expressions of the form $\pi_\delta(\mathrm{id})$ and $\sigma_\Theta(\mathrm{id})$.

We will see that the state formulae "compile out", i.e., are expressible using the operations in the first line. Despite state formulae being redundant, they are useful for expressing properties of processes relative to states, as in other modal temporal logics. In particular, they give an easy way to express quantification over executions (sequences of transitions) by means of modalities.

Semantics of $L\mu\mu$. The modal logic is interpreted over a transition system, where the set of states $\mathbf{U}$ is the set of all τ-structures over the same domain *Dom*. [11]

State Formulae (line 2 of (5)): Atomic modules M_p (modules-propositions) and module variables X_i are interpreted exactly like in the unary semantics. That is, M_p are Model Checking (MC) modules, i.e., those where the expansion (output) vocabulary is empty. The rest of the formulae are interpreted exactly as in the μ-calculus, except we have a backwards modality in addition:

$$\begin{array}{l} [\mathbf{T}] := \mathbf{U}, \\ [\phi_1 \vee \phi_2] := [\phi_1] \cup [\phi_2]\,, \\ [\neg\phi] := \mathbf{U} \setminus [\phi]\,, \\ [|\alpha\rangle\phi] := \{\mathfrak{A} \mid \exists\mathfrak{B}\ (\ (\mathfrak{A},\mathfrak{B}) \in [\![\alpha]\!] \text{ and } \mathfrak{B} \in [\phi]\ )\ \}, \\ [\langle\alpha|\phi] := \{\mathfrak{B} \mid \exists\mathfrak{A}\ (\ (\mathfrak{A},\mathfrak{B}) \in [\![\alpha]\!] \text{ and } \mathfrak{A} \in [\phi]\ )\ \}, \\ [\mu Z_j.\phi] := \bigcap \{R \subseteq \mathbf{U} :\ [\phi]^{[Z:=R]} \subseteq R\}. \end{array}$$

[10] Note that $\mathbf{T}$ is unary, as every other state formula in the second line of (5), which makes it different from the binary $\top$ and id.

[11] In the case of solving Model Expansion task, the domain is determined by the domain of the input structure.

Process Formulae (line 1 of (5)): These formulae are interpreted exactly as in the binary semantics. In particular, modules-actions are interpreted as Model Expansion (MX) tasks, since they have inputs and outputs. In addition, tests are interpreted as in Dynamic Logic: $[\![\phi?]\!] := \{(\mathfrak{A}, \mathfrak{A}) \mid \mathfrak{A} \in [\phi]\}$. In particular, $[\![\mathbf{T}?]\!] = [\![\mathrm{id}]\!]$, where id is the relative multiplicative identity (using the terminology introduced for algebras in the style of Tarski and Givant) in the syntax of $L\mu\mu$ (5).

Satisfaction Relation for $L\mu\mu$. We say that state $\mathfrak{A}$, where $\mathfrak{A} \in \mathbf{U}$, *satisfies* ϕ under variable assignment s, notation $\mathfrak{A} \models_s \phi$, if $\mathfrak{A} \in [\phi]$. For process formulae α, the definition of the satisfaction relation is exactly as in the binary semantics.

Structures as Transitions and States. Note that, for each $\alpha \in L\mu\mu$, its model is a Kripke structure where transitions represent MX tasks for all subformulae of α, according to the binary semantics. In that Kripke structure, states are Tarski's structures, and atomic transitions are also Tarski's structures, over the same vocabulary.[12]

5.2 Two-Sorted = One-Sorted Syntax

The two representations of the algebra, one-sorted (1) and two-sorted (5), are equivalent.[13] We show that all operations in the second line of (5) are reducible to the operations in the first line.

Theorem 1. *For every state formula ϕ in two-sorted syntax (5), there is a formula $\hat{\phi}$ in the one-sorted syntax (1) such that $\mathfrak{B} \models_s \phi$ iff $(\mathfrak{B}, \mathfrak{B}) \models_s \mathrm{Dom/Img}(\hat{\phi})$. For every process formula α there is an equivalent formula $\hat{\alpha}$ in the one-sorted syntax.*

The notation Dom/Img above means that either of the two operations can be used.

Proof. We need to translate all the state formulae into process formulae. We do it by induction on the structure of the formula. Atomic constant modules and module variables remain unchanged by the transformation, except, monadic variables are now considered as binary. Similarly, $\mathbf{T}$ is translated into binary as $\hat{\mathbf{T}} := \mathrm{id}$.

- If $\phi = \phi_1 \vee \phi_2$, we set $\hat{\phi} := \hat{\phi_1} \cup \hat{\phi_2}$.
- If $\phi = \neg\phi_1$, we set $\hat{\phi} := \curvearrowright(\hat{\phi_1})$. Equivalently, we can set $\hat{\phi} := \curvearrowleft(\hat{\phi_1})$, since state formulae are unary, and the two negations are essentially unary, i.e., are subsets of the Diagonal relation id.
- If $\phi = |\alpha_1\rangle\ \phi_1$, we set $\hat{\phi} := \mathrm{Dom}(\hat{\alpha_1}; \hat{\phi_1})$.

[12] Structures can be viewed as computing devices. They store information and expand an interpretation of an input sub-vocabulary to an output sub-vocabulary to satisfy a specification.

[13] Similar statements have been shown for other logics, e.g. [22].

- If $\phi = \langle \alpha_1 |\ \phi_1$, we set $\hat{\phi} :=\ \mathrm{Img}(\hat{\phi_1}; \hat{\alpha_1})$.
- If $\phi\ =\ \mu X.(\phi_1)$, we set $\hat{\phi}\ :=\ \mu X.\mathrm{Dom}(\hat{\phi_1})$. Equivalently, we can set $\hat{\phi}\ :=\ \mu X.\mathrm{Img}(\hat{\phi_1})$, since, again, we are dealing with unary formulae here.

Operations $\curvearrowright$, $\curvearrowleft$, Dom and Img are expressible using the basic operations of the algebra, under the binary semantics. This gives us a transformation for the state formulae.

All process formulae α except test $\phi_1?$ remain unchanged under this transformation. For test, we have:

- If $\alpha = \phi_1?$, we set $\hat{\alpha} := \mathrm{Dom}(\hat{\phi_1})$. Equivalently, we can set $\hat{\alpha} := \mathrm{Img}(\hat{\phi_1})$.

It is easy to see that, under this transformation, the semantic correspondence holds.

We now comment on a connection of the propositional version of $L\mu\mu$ (i.e., a fragment without projection and selection) with well-known logics. Propositional Dynamic Logic (PDL) [23,24] and Linear Dynamic Logic (LDL_f) [8]. Both logics have the same syntax:[14]

$$\begin{array}{c} \alpha ::= \mathrm{id} \mid M_a \mid \alpha; \alpha \mid \alpha \cup \alpha \mid \alpha^* \mid\ \phi?, \\ \phi ::= \mathbf{T} \mid M_p \mid\ \phi \vee \phi \mid \neg\phi \mid |\alpha\rangle\ \phi. \end{array} \tag{6}$$

However, the semantics is different is each case. In particular, LDL_f is interpreted over finite paths. Both logics are fragments of the propositional version (no projection, selection) of the modal logic $L\mu\mu$. To see it, recall that the Kleene star is expressible by $\alpha^* := \mu Z.(\mathrm{id} \cup Z; \alpha)$. Note also that unary negation is implicit in the process line of (6). This is because, in our translation from the two-sorted to one-sorted syntax, if $\phi = \neg\phi_1$, we set $\hat{\phi} := \curvearrowright(\hat{\phi_1})$.

It is known that we can use non-deterministic operations of Union and the Kleene star, which are used in PDL, to define basic imperative constructs called Deterministic Regular programs.[15]

Definition 6. DetRegular (While) programs *are defined by restricting the constructs* $\cup$, $*$ *and* $?$ *to appear only in the following expressions:*

$$\begin{array}{l} \textbf{skip}\ :=\ \mathbf{T}?, \\ \textbf{fail}\ :=\ (\neg\mathbf{T})?, \\ \textbf{if}\ \phi\ \textbf{then}\ \alpha\ \textbf{else}\ \beta\ :=\ (\phi?; \alpha) \cup ((\neg\phi)?; \beta), \\ \textbf{while}\ \phi\ \textbf{do}\ \alpha\ :=\ (\phi?; \alpha)^*; (\neg\phi)?. \end{array} \tag{7}$$

An unrestricted use of sequential composition is allowed.

While the programs are deterministic, their definition uses non-deterministic operations. For a complexity-theoretic analysis, it is possible to show that some

[14] Some description logics have a similar syntax and may include Converse operation.

[15] Please note that Deterministic Regular expressions and the corresponding Glushkov automata are unrelated to what we study here. In those terms, expressions $a; a^*$ are Deterministic Regular, while $a^*; a$ are not. Here, the term Deterministic Regular comes from another name for While programs.

deterministic algebraic operations are sufficient to define the same imperative constructs. We leave such an analysis to future work. As an application, Deterministic Regular programs can be used to specify dynamic behaviour of complex modular systems, in the style of Golog programming language [25].

5.3 The Main Decision Task: Definition

We now present a counterpart of Definition 3 of Model Expansion task for processes. It will be our main task in the rest of the paper. Recall that $\mathfrak{A} \models_s |\alpha\rangle\mathbf{T}$ means that program α has a successful execution starting from an input structure $\mathfrak{A}$. We show now that checking $\mathfrak{A} \models_s |\alpha\rangle\mathbf{T}$ corresponds to the decision version of the MX task for process α. Recall that, by the translation in the proof of Theorem 1, $|\alpha\rangle\mathbf{T} = \mathrm{Dom}(\alpha) = \curvearrowright\curvearrowright \alpha$. Recall also that $[\![\mathrm{Dom}(\alpha)]\!]^s = \{(\mathfrak{B}, \mathfrak{B}) \mid \exists \mathfrak{B}' \ (\mathfrak{B}, \mathfrak{B}') \in [\![\alpha]\!]^s\}$. Thus, we have: $\mathfrak{A} \models_s |\alpha\rangle\mathbf{T}$ iff $(\mathfrak{A}, \mathfrak{A}) \in [\![\mathrm{Dom}(\alpha)]\!]^s$ iff $\exists \mathfrak{B} \ (\mathfrak{A}, \mathfrak{B}) \in [\![\alpha]\!]^s$ iff $\exists \mathfrak{B}$ over the same vocabulary as $\mathfrak{A}$ such that if $\mathfrak{A}|_{s(I(\alpha))}$ interprets the inputs of α, then $\mathfrak{B}|_{s(O(\alpha))}$ interprets the outputs of α. This is an MX task. Thus, we formulate our problem as follows:

Definition 7 (MX task for Processes (Decision Version)).

Problem: MX task for Processes (Decision Version) Input: τ-structure $\mathfrak{A}$, formula α with variables $I(\alpha) \cup O(\alpha)$, variable assignment $s : vvoc(\alpha) \to \tau$. Question: $\mathfrak{A} \models_s |\alpha\rangle\mathbf{T}$?

6 Conclusion

Motivated by the need to combine preexisting components for solving complex problems, we developed two algebras, static and dynamic, for combining systems that communicate through common variables. The variables are second-order, i.e., they range over sets. Atomic modules are axiomatized in any formalism where the task of finding solutions can be specified as the task of Model Expansion. The dynamic algebra treats such specifications as "black boxes". We showed that, many operations studied in algebras of binary relations become definable if we add information propagation, i.e., specify inputs and outputs of atomic modules. We also showed that, when interpreted over transition systems, the dynamic algebra is equivalent to a modal temporal (dynamic) logic.

The logic can be viewed as a significant generalization of Reiter's situation calculus and Golog [9,10] in that first-order successor state axioms are replaced with potentially non-deterministic atomic modules that can be of arbitrary expressive power and computational complexity, and can be axiomatized in multiple logics. In place of Golog programs, we have algebraic terms inside the modalities that specify desired patterns of execution. Since our "successor state axioms" – atomic modules – are no longer first-order, a Prolog implementation, as in the case of Golog, is no longer possible. Different solving methods are needed. Some methods for solving modular systems for fragments of the current

language have already been developed [26,27]. One method, [26], took inspiration in the CDCL algorithm for SAT solving, where modules are called as oracles during the execution. In the other method, [27], a parallel algebra of propagators has been defined and used for solving Model Expansion task for modular systems. An important research direction is to extend the previous methods to the full algebra, as well as to develop new solving techniques.

Another research direction is to analyze the computational complexity of the main computational task in the Logic of Information Flows, under various assumptions on the expressiveness of atomic modules and allowable algebraic operations. In particular, it is very important to provide guarantees to the user that: (1) all problems in a particular complexity class are expressible in a particular fragment, to guarantee completeness of the fragment with respect to that complexity class; and (2) no more than the problems in that class are expressible, to ensure implementability of the system by a chosen technique. Providing such guarantees is at the core of the Model Expansion project and its connection to Descriptive Complexity [28], and a lot of work is currently under way in this direction.

Acknowledgements. Many thanks to Brett McLean, Jan Van den Bussche, Bart Bogaerts and other colleagues for useful discussions. The research is supported by NSERC.

References

1. Jónsson, B., Tarski, A.: Representation problems for relation algebras. Bull. Amer. Math. Soc. **74**, 127–162 (1952)
2. Pratt, V.R.: Origins of the calculus of binary relations. In: Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS 1992), Santa Cruz, California, USA, 22–25 June 1992, pp. 248–254 (1992)
3. Surinx, D., den Bussche, J.V., Gucht, D.V.: The primitivity of operators in the algebra of binary relations under conjunctions of containments. In: LICS 2017 (2017)
4. Fletcher, G., et al.: Relative expressive power of navigational querying on graphs. Inf. Sci. **298**, 390–406 (2015)
5. Jackson, M., Stokes, T.: Modal restriction semigroups: towards an algebra of functions. IJAC **21**(7), 1053–1095 (2011)
6. McLean, B.: Complete representation by partial functions for composition, intersection and anti-domain. J. Log. Comput. **27**(4), 1143–1156 (2017)
7. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic (Foundations of Computing) (2000)
8. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013, Beijing, China, 3–9 August 2013, pp. 854–860 (2013)
9. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press, Cambridge (2001)
10. Levesque, H., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.: GOLOG: a logic programming language for dynamic domains. J Log. Program. Spec. Issue Actions **31**(1–3), 59–83 (1997)

11. Tasharrofi, S., Ternovska, E.: 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), October 2011, pp. 259–274 (2011)
12. Tasharrofi, S.: Arithmetic and modularity in declarative languages for knowledge representation. Ph.D. dissertation, School of Computing Science, Simon Fraser University, December 2013
13. Ternovska, E.: An algebra of combined constraint solving. In: Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, 16–19 October 2015, pp. 275–295 (2015)
14. Ternovska, E.: Recent progress on the algebra of modular systems. In: Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, 7–9 June 2017 (2017)
15. Henkin, L., Monk, J., Tarski, A.: Cylindric Algebras Part I. North-Holland, Amsterdam (1971)
16. Bussche, J.: Applications of Alfred Tarski's ideas in database theory. In: Fribourg, L. (ed.) CSL 2001. LNCS, vol. 2142, pp. 20–37. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44802-0_2
17. Mitchell, D.G., Ternovska, E.: A framework for representing and solving NP search problems. In: Proceedings of AAAI 2005, pp. 430–435 (2005)
18. Vardi, M.Y.: The complexity of relational query language. In: 14th ACM Symposium Theory of Computing, Springer Verlag (Heidelberg, FRG and NewYork NY, USA)-Verlag, 1982
19. Denecker, M., Ternovska, E.: A logic of non-monotone inductive definitions. ACM Trans. Comput. Log. (TOCL) **9**(2), 1–52 (2008)
20. Kolokolova, A., Liu, Y., Mitchell, D., Ternovska, E.: On the complexity of model expansion. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR 2010. LNCS, vol. 6397, pp. 447–458. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16242-8_32
21. Ternovskaia, E.: Inductive definability and the situation calculus. In: Freitag, B., Decker, H., Kifer, M., Voronkov, A. (eds.) DYNAMICS 1997. LNCS, vol. 1472, pp. 227–248. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055501
22. Abu Zaid, F., Grädel, E., Jaax, S.: Bisimulation safe fixed point logic. In: Invited and Contributed Papers from the Tenth Conference on Advances in Modal Logic 10. Advances in Modal Logic. Held in Groningen, The Netherlands, 5–8 August 2014, pp. 1–15 (2014)
23. Pratt, V.R.: Semantical considerations on Floyd-Hoare logic. In: 17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25–27 October 1976, pp. 109–121 (1976)
24. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. Comput. Syst. Sci. **18**(2), 194–211 (1979)
25. Levesque, H., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.: GOLOG: a logic programming language for dynamic domains. J. Log. Program. **31**, 59–84 (1997)
26. Mitchell, D., Ternovska, E.: Clause-learning for modular systems. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) LPNMR 2015. LNCS (LNAI), vol. 9345, pp. 446–452. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23264-5_37
27. Bogaerts, B., Ternovska, E., Mitchell, D.: Propagators and solvers for the algebra of modular systems. In: Logic for Programming, Artificial Intelligence and Reasoning (LPAR) (2017)
28. Immerman, N.: Descriptive Complexity. Graduate Texts in Computer Science. Springer, Heidelberg (1999). https://doi.org/10.1007/978-1-4612-0539-5