A Logic of Information Flows
(Preliminary Report)

Eugenia Ternovska
Simon Fraser University, Canada
ter@sfu.ca

Abstract

We propose a KR formalism for combining heterogeneous components – web services, knowledge bases, declarative specifications such as Integer Liner Programs, Constraint Satisfaction Problems, Answer Set Programs etc.. The formalism is a family of logics, where atomic modules – formally, classes of structures – are combined using operations of extended Relational algebra, or, equivalently, first-order logic with a least fixed point construct. Inputs and outputs of atomic modules indicate directionality of the information flows. As a result of this small addition, an interesting modal logic, similar to Dynamic Logic, is obtained. Many binary operations, including those studied in the calculi of binary relations and the standard constructs of imperative programming become definable. We study properties of this logic and identify an efficient fragment where the main computational task is solvable in deterministic polynomial time.

Introduction

Our goal is to introduce a Logic of Information Flows. We do it in two stages. The first idea is that we can use first-order logic as a versatile language for applying and combining *modules* – which are classes of structures – web services, declarative specifications with associated solvers, Integer Liner Programs, Constraint Satisfaction Problems etc.[1] While the syntax of our formalism is first-order, the semantics is second-order because variables range over relations. We use a version of Codd relational algebra instead of first-order logic, but the idea is the same. We also add least fixed points. Essentially, we redefine FO(LFP) over a vocabulary of modules that replaces a relational vocabulary. This gives us the first logic.

The second stage is adding information flows, to develop a logic where modules are input-output relations. Toward this goal, we describe the Model Expansion task (Mitchell and Ternovska 2005a; Kolokolova et al. 2010), a fundamental computational task solved in many declarative approaches to constraint solving.

Model Expansion initiates information flows. To take information propagation into account, we partition the relational variables of atomic modules into inputs and outputs, thus viewing the modules as solving model expansion tasks. Input-output partitioning turns formulae of classical logic into *binary* in terms of generalized variables ranging over structures. We obtain an *algebra of binary relations*.

Algebras of binary relations have been studied before. A calculus of binary relations was first introduced by De Morgan. It has been extensively developed by Peirce and then Schröder. It was abstracted to relation algebra RA in (Jónsson and Tarski 1952). More recently, relation algebras were studied by Fletcher, Van den Bussche, Surinx and their collaborators in a series of paper, see, e.g. (Surinx, den Bussche, and Gucht 2017; Fletcher et al. 2015). The algebras of relations consider various subsets of operations on binary relations as primitive, and other as derivable. In another direction, (Jackson and Stokes 2011; McLean 2017) and others study partial functions and their algebraic equational axiomatizations.

The main contributions of this paper are as follows. We identify a single component – information flows, which is responsible for a rich variety of definable operations – those studied in the algebras of binary relations and partial functions. It is quite surprising that, by a simple step of adding a specification of inputs and outputs to classical logic, we obtain a multitude of such operations. Our work shows that information propagation is a basic and a fundamental concept. We study many properties of the operations we obtain.

Information flows also turn classical logic into modal, similar to the mu-calculus and Dynamic Logic. The modal logic is stronger than classical because e.g. it allows a new kind of quantification – over information flows. Due to the origin in classical logic and Model Expansion, the integration of processes and data is taken to the highest degree possible – both processes and states in the transition system are structures over the same vocabulary.

Despite the high expressiveness of the formalism (communication between modules happens through second-order variables, and fixed points put us into the third order), we identify a PTIME fragment of the language. The fragment is natural because its second-order variables represent computer registers, and the operations define the standard imperative programming constructs. The connection to binary relations allows us to take advantage of the good properties of bounded-variable fragments in the complexity analysis (Vardi 1995).

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

Model Expansion, Related Tasks

Model Expansion (Mitchell and Ternovska 2005b) is the task of expanding a structure to satisfy a specification (a formula in some logic). It is the central task in several declarative programming paradigms: Answer Set Programming, Constraint Satisfaction Problem, Integer Linear Programming, Constraint Programming, etc. We discuss model expansion in the context of two related 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: $\exists \mathfrak{B}$ s.t. $\mathfrak{B} \models \phi$?)*

Definition 2 (Model Checking (MC)). *Given: Formula ϕ, structure $\mathfrak{A}$ for $vocab(\phi)$. Decide: $\mathfrak{A} \models \phi$?*

There is no search counterpart for this task.

The following task (introduced in (Mitchell and Ternovska 2005b)) 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 $vocab(\phi)$. (The decision version is: Decide: $\exists \mathfrak{B}$ s. t. $\mathfrak{B} \models \phi$ and expands σ-structure $\mathfrak{A}$ to $vocab(\phi)$?)*

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.

Let ϕ be a sentence, i.e., has no free object variables. Data complexity (Vardi 1982) 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 (Denecker and Ternovska 2008) and guarded logics) has been studied in (Kolokolova et al. 2010).

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, respectively.

Syntax Assume we have a countable sequence $\mathrm{Vars} = (X_1, X_2, \dots)$ of *relational variables* each with an associated finite *arity*. For convenience, we use X, Y, Z, etc. Let $\mathrm{ModAt} = \{M_1, M_2, \dots\}$ be a fixed vocabulary of *atomic module symbols*. Each $M_i \in \mathrm{ModAt}$ has an associated *variable vocabulary* $vvoc(M_i)$ whose length can depend on M_i. We may write $M_i(X_{i_1}, \dots, X_{i_k})$, (or $M_i(\bar{X})$), to indicate that $vvoc(M) = (X_{i_1}, \dots, X_{i_k})$. Similarly, $\mathrm{ModVars} = \{Z_1, Z_2, \dots\}$ is a countable sequence of *module variables*, where each $Z_j \in \mathrm{ModVars}$ has its own $vvoc(Z_j)$. Algebraic expressions are built by the grammar:

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

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 $X \equiv Y$, for relational variables of equal arity that occur in α, 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.

Atomic modules can be specified in any formalism with a model-theoretic semantics. Modules occurring within one algebraic expression can be axiomatized in different logics. They can also be viewed as abstract decision procedures. But, as far as the algebra is concerned, their only relevant feature is the classes of structures they induce.

Static (Unary) Semantics Fix a finite relational vocabulary τ. A *variable assignment* s is a function that assigns, to each relational variable, a symbol in τ of the same arity. Now fix a domain Dom.[2] The domain can be finite or infinite. Let $\mathbf{U}$ be the set of all τ-structures over the domain Dom. Given a sub-vocabulary γ of τ, a subset $V \subseteq \mathbf{U}$ is *determined by* γ if it satisfies

$$\text{for all } \mathfrak{A}, \mathfrak{B} \in \mathbf{U} \text{ such that } \mathfrak{A}|_\gamma = \mathfrak{B}|_\gamma \text{ we have } \mathfrak{A} \in V \text{ iff } \mathfrak{B} \in V.$$

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 $\mathfrak{A} \in [\alpha]$, where *unary interpretation* $[\,\cdot\,]$ is defined as follows. Given a variable assignment s, function $[\,\cdot\,]$ assigns a subset $[M_i] \subseteq \mathbf{U}$ and a subset $[Z_j] \subseteq \mathbf{U}$ to each $M_i \in \mathrm{ModAt}$ and each $Z_j \in \mathrm{ModVars}$, with the property that $[M_i]$ is determined by $s(vvoc(M_i))$ (respectively, $[Z_j]$ is determined by $s(vvoc(Z_j))$). The unary interpretation of atomic modules $[\,\cdot\,]$ (parameterized with s) can be viewed as a function that provides "oracles" or decision procedures. We extend the definition of $[\,\cdot\,]$ to all algebraic expressions.

[2] Usually, in applications, domain Dom is the (active) domain of an input structure for a task of interest such as MX. The semantics of the algebra can also be given in terms of axiomatizing classes of structures, but this is not necessary for this paper.

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

Here, $[\alpha]^{[Z:=\alpha]}$ means an interpretation that is exactly like $[\,\cdot\,]$, except Z is interpreted as α.

Example 1. Let $M_{\mathrm{HC}}(N, X, Y)$ and $M_{2\mathrm{Col}}(N, X, Z, T)$ be atomic modules "computing" a Hamiltonian Circuit and a 2-Colouring, respectively. For example, M_{HC} can be represented as an Answer Set Programming program, and $M_{2\mathrm{Col}}$ be an imperative program or a human child with two pencils. The first module decides if Y forms a Hamiltonian Circuit (represented as a set of edges) in the graph given by vertex set N and edge set X. The second module decides if unary relations Z, T specify a proper 2-colouring of the graph. The following expression determines whether or not there is a 2-colourable Hamiltonian Circuit.

$$\alpha_{2\mathrm{Col}-\mathrm{HC}}(N, X, Z, T) := \pi_{N,X,Z,T}((M_{\mathrm{HC}}(N, X, Y) \cap M_{2\mathrm{Col}}(N, Y, Z, T)).$$

To check whether a τ-structure $\mathfrak{A}$ satisfies $\alpha_{2\mathrm{Col}-\mathrm{HC}}$, i.e., $\mathfrak{A} \models_s \alpha$, we need to use function s to match the variables $\{N, X, Z, T\}$ with predicate symbols $\{V, E, R, B\} \subseteq \tau$, and then apply the semantics. In particular, we would need to check whether graph $(V^{\mathfrak{A}}, E^{\mathfrak{A}})$ with verticies $V^{\mathfrak{A}}$ and edges $E^{\mathfrak{A}}$ has *some* Hamiltonian Circuit according to M_{HC}, and that the graph with nodes $V^{\mathfrak{A}}$ and edges formed by the circuit is 2-colourable according to $M_{2\mathrm{Col}}$.

Note that communications between modules happen through second-order variables. First-order variables can be imitated by ensuring that second-order variables range over singleton sets.

Dynamic (Binary) Semantics Let $\mathrm{ModAt}_{\mathrm{I/O}}$ denote the set of all atomic module symbols M with all possible *partitions* of $vvoc(M)$ into inputs and outputs, i.e., $I(M) \cup O(M) = vvoc(M)$ and $I(M) \cap O(M) = \varnothing$.[3] 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 modules (resp. variables) from $\mathrm{ModAt}_{\mathrm{I/O}}$ (resp. from $\mathrm{ModVars}_{\mathrm{I/O}}$). Inputs $I(\alpha)$ and outputs $O(\alpha)$ of well-formed algebraic expression α are defined as follows. Inputs and outputs of the *atomic* modules in α must agree with $I(\alpha)$ and $O(\alpha)$ on the free variables, and can be arbitrary on the other variables.[4]

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 $(\mathfrak{A}, \mathfrak{B}) \in [\![\alpha]\!]$, where *binary interpretation* $[\![\,\cdot\,]\!]$ is defined as follows. For atomic modules in $\mathrm{ModAt}_{\mathrm{I/O}}$, we have:

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

Similarly, for $Z \in \mathrm{ModVars}_{\mathrm{I/O}}$. Intuitively, an atomic module produces a replica of the current structure except the interpretation of the output vocabulary changes as specified by the action.[5] An illustration of the binary semantics for atomic modules is given in Example 2 below. We extend the binary interpretation $[\![\cdot]\!]$ to all expressions α:

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

$$\left\{ (\mathfrak{A}, \mathfrak{B}) \in [\![\alpha]\!] \;\middle|\; \begin{array}{l} (s(X))^{\mathfrak{A}} = (s(Y))^{\mathfrak{A}} \text{ if } \{X, Y\} \subseteq I(\alpha), \\ (s(X))^{\mathfrak{B}} = (s(Y))^{\mathfrak{B}} \text{ if } \{X, Y\} \subseteq O(\alpha), \\ (s(X))^{\mathfrak{A}} = (s(Y))^{\mathfrak{B}}) \text{ if } X \in I(\alpha) \text{ and } \\ \qquad Y \in O(\alpha). \end{array} \right\}$$

Operation id is sometimes called the "nil" action, or it can be seen as an empty word denoted ε in the formal language theory.

It is convenient to extend the selection operation to $\Theta \in \{X \equiv Y, X \not\equiv Y, X \equiv \varnothing, X \not\equiv \varnothing\}$, although these cases are already covered by the semantics above.

We now illustrate the fact that each atomic module is, simultaneously, (a) a set of structures, according to the unary semantics, and (b) a set of pairs of structures, according to the binary semantics. We also illustrate the Law of Inertia used in the semantics of atomic modules (2).

Example 2. Consider a 3-Colouring module $M_{3\mathrm{Col}}(\underline{X}, \underline{Y}, Z, T, W)$. The inputs are underlined. Let $s(I(M_{3\mathrm{Col}})) = (V, E)$ and $s(O(M_{3\mathrm{Col}})) = (R, G, B)$. Let a domain Dom and an interpretation of edges (E) and vertices (V) be given on the input of this module, and colours (R, G, B) are obtained on the output. The MX task can be represented as a set of all $\{V, E, R, G, B\}$-structures which expand $\{V, E\}$-structures over this domain to satisfy a specification Ψ of 3-Colouring in some logic.

First, consider the *binary* semantics (2) for this module. On the input, we have a τ-structure $\mathfrak{A}$ such that $\mathfrak{A}|_{\{V,E\}}$, the interpretation of $\{V, E\} \subseteq \tau$, is the graph of interest. Each output structure $\mathfrak{B}$ consists of: (a) $\mathfrak{B}|_{\{R,G,B\}}$, that is a proper 3-Colouring, (b) $\mathfrak{A}|_{\{V,E\}}$, which is the input graph, transferred from $\mathfrak{A}$ to $\mathfrak{B}$ by inertia, (c) the interpretation of all other symbols of τ, also moved from $\mathfrak{A}$ by inertia. In Figure 1, L is the input graph, R is a particular 3-Colouring.

According to the *unary* semantics, the atomic module is the set of τ-structures with the domain Dom, where $\{V, E, R, G, B\} \subseteq \tau$ are interpreted according to the specification Ψ, and the interpretations of the symbols in $\tau \setminus$

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

[4] Free variables are defined as in first-order logic, where $\exists$-quantified variables are those that are not projected onto.

[5] This is similar to the inertia law for primitive actions in the Situation Calculus (Reiter 2001).

$\{V, E, R, G, B\}$ are interpreted arbitrarily. In Figure 1, the structure that describes the transition (shown horizontally) corresponds to a particular 3-Colouring of a particular graph.

Preservation of unmodified parts of a structure is a fundamental law that we call the Law of Inertia. Figure 1 illustrates actions-as-structures and the Law of Inertia for atomic modules.

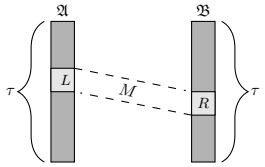

Figure 1: An atomic module M as a set of τ-structures.

Let $I(M) = \sigma$, $O(M) = \varepsilon$, and $s(\sigma) \cup s(\varepsilon) \subseteq \tau$. The Figure shows a transition $\mathfrak{A} \xrightarrow{M} \mathfrak{B}$ of atomic module $M(\underline{\sigma}, \varepsilon)$ according to one of its structures. The structure has interpretation $L = (s(\sigma))^{\mathfrak{A}}$, under the assignment s, of its input relational variables σ on the left, and interpretation $R = (s(\varepsilon))^{\mathfrak{B}}$ of its output variables ε on the right. By the Law of Inertia, the interpretation of everything that is not modified by this action (i.e., of what is in $\tau \setminus s(\varepsilon)$) is transferred from structure $\mathfrak{A}$ to structure $\mathfrak{B}$. Since, in general, M can be non-deterministic, there is such a transition for each of the structures in M. Thus, a module is both a set of structures, and a set of pairs of structures.

In the next example, we will use intersection ($\cap$), which is a definable operation. We will see this and many more definable operations in the next section.

Example 3. Consider again $\alpha_{2\mathrm{Col}-\mathrm{HC}}(N, X, Z, T)$. In each atomic module, we underline designated input symbols:

$$\pi_{N,X,Z,T}(M_{\mathrm{HC}}(\underline{N}, \underline{X}, Y) \cap M_{2\mathrm{Col}}(\underline{N}, \underline{Y}, Z, T)).$$

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

Note that, as in Codd's relational algebra, relational variables can be omitted. That is, we can simply write:

$$\pi_{N,X,Z,T}(M_{\mathrm{HC}} \cap M_{2\mathrm{Col}}).$$

The need for recursion can be seen from e.g. specifications of dynamic programming algorithms on tree decompositions, such as one for 3-Colouring, where a 3-Colouring module is applied recursively. We do not have space for such an example, but we illustrate the recursive constructs by two shorter examples at the end of the paper.

In applications, we check 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. We explain this modality in the section on Modal Logic. To evaluate α in $\mathfrak{A}$, we use s to match the vocabulary of $\mathfrak{A}$ with the relational input variables $I(\alpha) \subseteq vvoc(\alpha)$ of α, while matching the arities as well, and then apply the semantics. One can think of an input vocabulary $I(\alpha)$ as of just a vocabulary $vocab(\mathfrak{A})$ of an input structure $\mathfrak{A}$, in the standard understanding as in model theory.

Definable Constructs

We now introduce several *definable* operations, and we study some of their properties. All of those constructs are studied as primitive in calculi of binary relations and partial functions. It turns out that the only thing lacking in classical logic to define all these constructs is information propagation. 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

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

By these definitions,

$$\begin{array}{l}
[\![\mathrm{di}]\!] = \{(\mathfrak{A}, \mathfrak{B}) \mid \mathfrak{A} \neq \mathfrak{B}\}, \\
[\![\top]\!] = \mathbf{U} \times \mathbf{U}, \\
[\![\bot]\!] = \varnothing, \\
[\![\alpha \cap \beta]\!] = \{(\mathfrak{A}, \mathfrak{B}) \mid (\mathfrak{A}, \mathfrak{B}) \in [\![\alpha]\!] \text{ and } (\mathfrak{A}, \mathfrak{B}) \in [\![\beta]\!]\ \}, \\
[\![\alpha - \beta]\!] = \{(\mathfrak{A}, \mathfrak{B}) \mid (\mathfrak{A}, \mathfrak{B}) \in (\mathbf{U} \times \mathbf{U}) \setminus [\![\alpha]\!] \text{ or } (\mathfrak{A}, \mathfrak{B}) \in [\![\beta]\!]\ \}, \\
[\![\alpha \sim \beta]\!] = \{(\mathfrak{A}, \mathfrak{B}) \mid\ (\mathfrak{A}, \mathfrak{B}) \in [\![\alpha]\!] \text{ and } (\mathfrak{A}, \mathfrak{B}) \in [\![\beta]\!] \text{ or} \\
\qquad (\mathfrak{A}, \mathfrak{B}) \in (\mathbf{U} \times \mathbf{U}) \setminus [\![\alpha]\!] \text{ and } (\mathfrak{A}, \mathfrak{B}) \in (\mathbf{U} \times \mathbf{U}) \setminus [\![\beta]\!]\ \}.
\end{array}$$

In particular, expression $\alpha \sim \beta$ specifies the set of pairs of structures such that each pair is either in both α and β, or it is in neither α or β. That is, it is the set of pairs of structures where α and β behave in exactly the same way – both defined and produce the same outputs on the same inputs, or are both undefined.

Projection onto the inputs (Domain)

$$\mathrm{Dom}(\alpha) := \pi_{I(\alpha)}(\alpha).$$

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)

$$\mathrm{Img}(\alpha) := \pi_{O(\alpha)}(\alpha).$$

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 Law does not hold for $\curvearrowright$and $\curvearrowleft$.

Logical equivalence (equality of algebraic terms) We say that α and β are *logically equivalent*, notation $\alpha = \beta$ if

$$(\mathfrak{A},\mathfrak{B}) \models_s \alpha \text{ iff } (\mathfrak{A},\mathfrak{B}) \models_s \beta,$$

for all τ-structures $\mathfrak{A}$, $\mathfrak{B}$, for any variable assignment s.[6]

Proposition 1.

If α is a (partial) identity on $\mathbf{U}$*, then* $\quad \mathrm{Dom}(\alpha) = \mathrm{Img}(\alpha),\ \curvearrowright\alpha = \curvearrowleft\alpha.$

Proof. The statement is an immediate consequence of the definitions of the operations. □

Proposition 2.

$$\begin{array}{l}
\curvearrowright\alpha = \mathrm{Dom}(\alpha)^{-}\cap\mathrm{id} = \mathrm{Dom}(\alpha^{-}) - \mathrm{Dom}(\alpha)\\
= \curvearrowright\mathrm{Dom}(\alpha) = \curvearrowleft\mathrm{Dom}(\alpha),\\
\curvearrowleft\alpha = \mathrm{Img}(\alpha)^{-}\cap\mathrm{id} = \mathrm{Img}(\alpha^{-}) - \mathrm{Img}(\alpha)\\
= \curvearrowright\mathrm{Img}(\alpha) = \curvearrowleft\mathrm{Img}(\alpha),\\
\mathrm{Dom}(\alpha) = \curvearrowright\curvearrowright\alpha,\\
\mathrm{Img}(\alpha) = \curvearrowleft\curvearrowleft\alpha,\\
\mathrm{id} = \curvearrowright\bot = \curvearrowleft\bot,\\
\bot = \curvearrowright\mathrm{id} = \curvearrowleft\mathrm{id} = \curvearrowright\top = \curvearrowleft\top,\\
\top = \curvearrowright\curvearrowright\mathrm{id} = \curvearrowleft\curvearrowleft\mathrm{id} = \curvearrowright\curvearrowleft\mathrm{id}\\
= \curvearrowleft\curvearrowleft\mathrm{id} = \curvearrowright\bot = \curvearrowleft\bot,\\
\curvearrowright\curvearrowright\curvearrowright\alpha = \curvearrowright\alpha,\\
\curvearrowleft\curvearrowleft\curvearrowleft\alpha = \curvearrowleft\alpha.
\end{array}$$

Proof. The logical equivalences follow directly from the definitions of the operations. □

Sequential composition The operation of sequential composition $(\alpha;\beta)$ 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.

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

If $O(\alpha) = \bar{X}$ and $I(\beta) = \bar{Y}$, then $\alpha;\beta$ is expressible as

$$\sigma_{\bar{X}\equiv\bar{Y}}(\alpha\cap\beta).$$

The following two propositions summarize several properties of sequential composition, intersection and union.

[6]The reason we use the equality symbol ('=') for the meta-logic notion of logical equivalence (instead of, say, '≡') is that '=' is traditionally used to specify equivalence of algebraic terms.

Proposition 3. *Identities and zeros:*

$$\begin{array}{ll}
\mathit{Intersection}: & \top\cap\alpha = \alpha\cap\top = \alpha,\\
& \bot\cap\alpha = \alpha\cap\bot = \bot,\\
\mathit{Composition}: & \alpha;\mathrm{id} = \mathrm{id};\alpha = \alpha,\\
& \alpha;\bot = \bot;\alpha = \bot.\\
\mathit{Union}: & \top\cup\alpha = \alpha\cup\top = \top,\\
& \bot\cup\alpha = \alpha\cup\bot = \alpha.
\end{array}$$

Proposition 4. *Distributivity:*

$$\begin{array}{ll}
\mathit{Intersection}: & \alpha\cap(\beta\cup\gamma) = (\alpha\cap\beta)\cup(\alpha\cap\gamma),\\
& (\alpha\cup\beta)\cap\gamma = (\alpha\cap\gamma)\cup(\beta\cap\gamma),\\
\mathit{Composition}: & \alpha;(\beta\cup\gamma) = (\alpha;\beta)\cup(\alpha;\gamma),\\
& (\alpha\cup\beta);\gamma = (\alpha;\gamma)\cup(\beta;\gamma),\\
\mathit{Unary\ Negations:} & \curvearrowright(\alpha\cap\beta) = \curvearrowright\alpha\cap\curvearrowright\beta,\\
& \curvearrowleft(\alpha\cap\beta) = \curvearrowleft\alpha\cap\curvearrowleft\beta.
\end{array}$$

Relative Disjunction Just as logical disjunction $(\cup)$ is a De Morgan dual of logical conjunction $(\cap)$, relative (or dynamic) disjunction $(\dotplus)$ is a De Morgan dual of relative conjunction $(;)$:

$$\alpha \dotplus \beta \;:=\; (\alpha^{-};\beta^{-})^{-}.$$

Iteration (Kleene star) This operator is the *iteration* operator, also called Kleene star. The expression α^* means "execute α some nondeterministically chosen finite number of times. We define it as follows.

$$\alpha^* := \mu Z.(\mathrm{id}\cup Z;\alpha). \tag{3}$$

Maximum Iterate This operation is a determinization of Kleene star. It outputs only the longest transition out of all possible transitions produced by Kleene star.

$$\alpha^{\uparrow} \;:=\; \mu Z.(\curvearrowright\alpha\cup\alpha;Z).$$

By this definition, $\alpha^{\uparrow} = \bigcup_{1\le n<\omega}\alpha^n$, where

$$\alpha_1 \;:=\; \curvearrowright\alpha, \qquad \alpha_{n+1} := \alpha;\alpha^n. \tag{4}$$

Notice that, if α is a function, then $\alpha^{\uparrow}$ is a function as well, unlike α^* which produces a relation.

Preferential union This operation is defined as follows.

$$\alpha\sqcup\beta \;:=\; \alpha\cup(\curvearrowright\alpha;\beta).$$

By this definition,

$$[\![\alpha\sqcup\beta]\!] = \left\{(\mathfrak{A},\mathfrak{B}) \;\middle|\; \begin{array}{ll} (\mathfrak{A},\mathfrak{B}) \in [\![\alpha]\!] & \text{if } (\mathfrak{A},\mathfrak{A}) \in [\![\mathrm{Dom}(\alpha)]\!],\\ (\mathfrak{A},\mathfrak{B}) \in [\![\beta]\!] & \text{if } (\mathfrak{A},\mathfrak{A}) \notin [\![\mathrm{Dom}(\alpha)]\!]\\ & \text{and } (\mathfrak{A},\mathfrak{A}) \in [\![\mathrm{Dom}(\beta)]\!]. \end{array}\right\}$$

Safe Projection Operation $s\pi_\delta(\alpha)$ is a "safe" version of projection, where the set of symbols δ must include all of the inputs $I(\alpha)$ of α. Since, in this case, none of the inputs are existentially quantified, this operation does not introduce non-determinism.

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:

$$\beta = \alpha^{\smile} \qquad \text{iff} \qquad \begin{array}{l} \mathrm{Dom}(\alpha) = \mathrm{Img}(\beta), \\ \mathrm{Dom}(\beta) = \mathrm{Img}(\alpha). \end{array}$$

It is used in some Description and Dynamic Logics, graph databases. Using Converse, one can also define Residuation, which is an important operation in Lambek calculus (Lambek 1958). Converse, together with complementation, define linear negation, as in Linear Logic (Girard 1987): $\alpha^{\perp} := \alpha^{\smile -}$.

We have been able to define many other operations including those studied in (McLean 2018). But we do not have space to present them here. There are many theorems connecting the relations we defined here and in the previous sections, as is (Tarski 1941). But the operations have never been defined and analyzed from the point of view of information flows in classical logic.

Modal Logic

We call this logic $L\mu\mu$, since it is similar to the mu-calculus $L\mu$, but has two fixed points, unary and binary.

Two-sorted Syntax, $L\mu\mu$

The algebra with information flows can be equivalently represented in a "two-sorted" syntax. This syntax gives us a modal logic, similar to Dynamic Logic. The syntax is given by the grammar:

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

The first line is essentially our original syntax (1). In the second line, 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.[7] 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 $\mathbf{U}$.

The formulae in the first line of (5) are called *process formulae*, and the formulae in the second line are called *state formulae*. 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. [8]

[7] 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.

[8] The domain can be determined by the structure given as an input to a Model Expansion task.

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:

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

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 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.

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 transitions are also Tarski's structures, over the same vocabulary. Please see Figure 1 for clarification of the atomic case.

Two-Sorted = Minimal Syntax

The two representations of the algebra (one-sorted and two-sorted) are equivalent, as we show below.[9] 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 minimal syntax (1) such that $\mathfrak{B} \models_s \phi$ iff $(\mathfrak{B}, \mathfrak{B}) \models_s \mathrm{Dom}/\mathrm{Img}(\hat{\phi})$. For every process formula α there is an equivalent formula $\hat{\alpha}$ in the minimal syntax.*

The notation $\mathrm{Dom}/\mathrm{Img}$ 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 $\hat{\phi} := \hat{\phi_1} \cup \hat{\phi_2}$.

[9] The statement was inspired by a similar theorem for another logic in (Abu Zaid, Grädel, and Jaax 2014).

- 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 we are dealing with self-loops.
- If $\phi = |\alpha_1\rangle\ \phi_1$, we set $\hat{\phi} := \mathrm{Dom}(\hat{\alpha_1}; \hat{\phi_1})$.
- 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 establish a connection with a well-known logic.

Proposition 5. *Propositional Dynamic Logic (PDL) (Pratt 1976; Fischer and Ladner 1979)*

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

is a fragment of of the propositional version of the Logic of Information Flows, and of the equivalent modal logic $L\mu\mu$.

Proof. Immediate from (3). □

Unary negation is implicit in the process line of (6). This is because, in our translation, 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 Kleene star to define basic imperative constructs. But later, we show that some *deterministic* operations are sufficient to define the same imperative constructs.

Definition 4. DetRegular (While) programs *are defined by restricting the constructs $\cup$, $*$ and $?$ to appear only in the following expressions:*

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

An unrestricted use of sequential composition is allowed.

We can also define $\mathbf{repeat}\ \alpha\ \mathbf{until}\ \phi := \mathbf{while}\ \neg\phi\ \mathbf{do}\ \alpha$.

PTIME Fragment

Recall that PTIME is a class of all problems computable by a deterministic polynomial time Turing machine. Our goal is to identify a fragment of the Logic of Information Flows (which is really a family of logics) for which MX task is in PTIME. We demonstrate two closure properties. We show that *if* atomic modules are partial function, then this property is preserved after applying any of the operations of Deterministic Fragment. We also show that, under a condition on second-order variables, *if* MX for atomic modules are deterministic polynomial time computable, then so are the results of applying the operations of Deterministic Fragment.

Deterministic (Functional) Fragment

A grammar describing the Deterministic Fragment is:

$$\begin{array}{rl}\alpha\ ::=\ & \mathrm{id}\ \mid M(\underline{\bar{X}}, \bar{Y}) \mid \alpha;\alpha \mid \curvearrowright\alpha \mid \curvearrowleft\alpha \mid \alpha \sqcup \alpha \mid \alpha^{\uparrow} \mid \alpha^{\smile} \mid\\ & s\pi_{\delta}(\alpha) \mid \sigma_{\Theta}(\alpha),\end{array} \tag{8}$$

where $\Theta \in \{X \equiv Y, X \not\equiv Y, X \equiv \varnothing, X \not\equiv \varnothing\}$ for relational variables X and Y in α. The constructs are restrictions of the binary operations we introduced initially. Notice that the constructs of (8), except Converse, bear a strong resemblance with the constructs of classical logic: conjunction, negation, disjunction, existential quantifier, equality and limited recursion (Deterministic Transitive Closure). However information flows are needed to define them. Thus, the Deterministic Fragment is a *relative* and, as we show next, *function-preserving* counterpart of classical logic.

Theorem 2 (Closure of partial functions). *The set of all partial functions on $\mathbf{U}$ is closed under the operations of the Deterministic Fragment (8).*

Proof. By the condition of the theorem, atomic modules are partial functions on $\mathbf{U}$. Assume that the statement holds for α and β. Operations id and $\curvearrowright$are (partial) identities on $\mathbf{U}$, which clearly produce functions. Composition $\alpha;\beta$ of partial functions is a partial function because when α is defined, and β is defined on the output of α, then $\alpha;\beta$ is a function. If either α or β are undefined, their composition is undefined as well. Preferential union, $\alpha \sqcup \beta$, behaves as α, if α is defined, otherwise it behaves as β. Thus, it is clearly a function. Safe projection $s\pi_{\delta}(\alpha)$ is a function because it is exactly like α, but has fewer outputs. Selection $\sigma_{\Theta}\alpha$ is a function because it simply limits the domain and/or range of α by requiring that some values are equal to each other (or not) or to the empty relation. If α is a function, Maximum Iterate $\alpha^{\uparrow}$ is also a function because it was introduced as a determinization of the non-deterministic operation of Kleene star. □

DetRegular(While) $\subseteq$ Deterministic Fragment We show that imperative programming constructs introduced in Definition 4 are definable in the Deterministic Fragment.

Proposition 6. *Assume ϕ is any partial identity on $\mathbf{U}$. Then*

$$\begin{array}{l}\mathbf{skip}\ =\ \mathrm{id},\\ \mathbf{fail}\ =\ \curvearrowright\mathrm{id},\\ \mathbf{if}\ \phi\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta\ =\ (\phi;\alpha) \sqcup \beta,\\ \mathbf{while}\ \phi\ \mathbf{do}\ \alpha = (\phi;\alpha)^{\uparrow};(\curvearrowright\phi),\\ \mathbf{repeat}\ \alpha\ \mathbf{intil}\ \phi = ((\curvearrowright\phi);\alpha)^{\uparrow};\phi.\end{array}$$

Modalities and the Deterministic Fragment By the following proposition, the possibility and the necessity modalities are also expressible using the operations in the Deterministic Fragment (8).

Proposition 7.

$$\begin{array}{ll}|\alpha\rangle\ \phi\ =\ \curvearrowright\curvearrowright(\alpha;\phi), & |\alpha]\ \phi\ =\ \curvearrowright(\alpha;\curvearrowright\phi),\\ |\alpha\rangle\ \mathbf{T}\ =\ \curvearrowright\curvearrowright\alpha, & |\alpha]\ \mathbf{T}\ =\ \mathrm{id},\\ |\alpha\rangle\ \neg\mathbf{T}\ =\ \bot, & |\alpha]\neg\mathbf{T}\ =\ \curvearrowright\alpha.\end{array}$$

Proof. Recall, from the translation from the two-sorted to the one-sorted syntax that $|\alpha\rangle\ \phi\ =\ \mathrm{Dom}(\alpha;\phi)\ =$

$\curvearrowright\curvearrowright(\alpha;\phi)$. Also, $|\alpha\rangle\ \mathbf{T} = \curvearrowright\curvearrowright(\alpha;\mathbf{T}?) = \curvearrowright\curvearrowright(\alpha;\mathrm{id}) = \curvearrowright\curvearrowright\alpha$, and $|\alpha\rangle\ \neg\mathbf{T} = \curvearrowright\curvearrowright(\alpha;(\neg\mathbf{T})?) = \curvearrowright\curvearrowright(\alpha;\bot) = \curvearrowright\curvearrowright\bot = \bot$. For the necessity modality, we have: $|\alpha]\ \phi = \neg|\alpha\rangle\neg\phi = \neg\mathrm{Dom}(\alpha;\neg\phi) = \curvearrowright\curvearrowright\curvearrowright(\alpha;\curvearrowright\phi) = \curvearrowright(\alpha;\curvearrowright\phi)$. For the more specific cases, $|\alpha]\ \mathbf{T} = \curvearrowright(\alpha;(\neg\mathbf{T})?) = \curvearrowright(\alpha;\curvearrowright\mathrm{id}) = \curvearrowright(\alpha;\bot) = \curvearrowright\bot = \mathrm{id}$, and $|\alpha]\ \neg\mathbf{T} = \curvearrowright\mathrm{Dom}(\alpha;\curvearrowright(\neg\mathbf{T})?) = \curvearrowright\mathrm{Dom}(\alpha;\mathrm{id}) = \curvearrowright\mathrm{Dom}(\alpha) = \curvearrowright\alpha$. □

Thus, if atomic modules are deterministic, then neither modalities, nor deterministic regular (While) programs add nondeterminism. By Propositions 6 and 7, we have:

Corollary 1. *The set of partial functions on* $\mathbf{U}$ *is closed under the operations of DetRegular (While) fragment and the possibility and necessity modalities.*

PTIME Complexity of the Deterministic Fragment

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 $[\![\mathrm{Dom}(\alpha)]\!] = \{(\mathfrak{B},\mathfrak{B}) \mid \exists\mathfrak{B}'\ (\mathfrak{B},\mathfrak{B}') \in [\![\alpha]\!]\}$. Thus, we have: $\mathfrak{A} \models_s |\alpha\rangle\mathbf{T}$ iff $(\mathfrak{A},\mathfrak{A}) \in [\![\mathrm{Dom}(\alpha)]\!]$ iff $\exists\mathfrak{B}\ (\mathfrak{A},\mathfrak{B}) \in [\![\alpha]\!]$ iff $\exists\mathfrak{B}$ over the same vocabulary as $\mathfrak{A}$ s.t. 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:

MX task for Processes (Decision Version)

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

We will study data complexity of this task (Vardi 1982). The theorem below uses computations over a set of registers – monadic singleton-set relations. It turns out that in such computations, the property of modules to be deterministic polynomial time computable (on a Turing machine) is preserved under the operations of Deterministic Fragment.

Theorem 3 (Closure of PTIME MX tasks). *Let α be in the Deterministic Fragment (8), and let all second-order variables be monadic and interpreted by singleton-set relations. Then if MX for all atomic modules is in PTIME, then MX for α is in PTIME.*

Proof. Suppose the conditions of the theorem hold. We argue that the operations of the Deterministic Fragment do not put us outside of PTIME. By Theorem 2, the operations of this fragment do not add non-determinism. We first consider the recursion-free operations. Since the semantics is binary, in terms of generalized variables ranging over structures, we have a bounded-variable fragment, with the generalized formula width at most 3. This implies that each intermediate relation has *generalized* arity of at most 3. Since the definition of the language implies that all structures are essentially l-tuples of domain elements, the size of each intermediate relation is at most n^{3l}, where $l = |vvoc(\alpha)|$, the size of the variable vocabulary of α. The constant l depends on teh formula, which is fixed when we study data complexity. Similarly to the proof in (Vardi 1995) for the combined complexity of FO^k (which is PTIME-complete), we argue that we can evaluate the expression bottom-up, and all intermediate expressions are of bounded generalized arity, thus of size at most n^{3l}. Since combined complexity bounds data complexity from above, and l is fixed, we obtain the desired upper bound for the recursion-free fragment.

It remains to argue for the limited recursion case. Recall that $\alpha^{\uparrow}$ is computed by the process (4). The procedure always terminates because (a) the transition system generated by executing the program is finite (for a finite input domain), and (b) the base case is "no outgoing α-transition". Thus, Maximum Iterate explicitly disregards cycles – if there is an infinite loop, there is no model. Our fragment is in the alternation-fragment of $L\mu$, but nested recursion can be eliminated explicitly by (Gurevich and Shelah 1986) at the price of increasing the arity by the factor of m, the degree of nesting. The number of steps in computing $\alpha^{\uparrow}$ is always bounded by the size of the transition system, which is, since we only use monadic singleton-set relations, at most n^{lm}, the number of tuples of length lm over the input domain of size n. Because of this bound, Maximum Iterate requires at most n^{lm} iterations, where l and m are fixed since the formula is fixed. Each iteration, by the recursion-free case, requires at most n^{3l} steps. Thus, the total number of steps is $n^{3l}n^{lm}$ or $n^{(3+m)l}$, a polynomial in n. □

Corollary 2. *Theorem 3 also holds for DetRegular (While) programs (7).*

Notice that an atomic module does not have to be deterministic to have a deterministic polynomial time computable MX task. Consider, for example, a very simple module that guesses and outputs one domain element. This is a *nondeterministic* module. A Turing machine can perform this computation in *deterministic* polynomial (even linear) time in the size of the domain by simply writing, as an output, say, the last element in the order they appeared on the input tape. The use of such non-deterministic atomic modules, that satisfy the conditions Theorem 3, is demonstrated by the example EVEN below. The example uses atomic modules axiomatized in non-recursive Datalog with limited guessing expressed by means of Hilbert's Epsilon quantifier. The quantifier arbitrarily selects *precisely one* out of all possible ways of instantiating an ε-quantified variable.

Atomic Tests Recall that tests are (partial) identities on $\mathbf{U}$. *Atomic tests* are (a) atomic modules-propositions (MC modules) and (b) expressions of the form $\pi_X(\mathrm{id})$ and $\sigma_\Theta(\mathrm{id})$.

Back Exists ($\mathbf{BE}_\alpha$) and Back Globally ($\mathbf{BG}_\alpha$) are modal operators similar to F (Exists) G (Globally) of Linear Temporal Logic (LTL), respectively. They face backwards in time, *in the same information flow* as produced by the execution of α. We define $\mathbf{BE}_\alpha$ for *atomic* tests, denoted $AtTest$. The program executes the actions of α "backwards" until $AtTest$ is found to be true.

$$\mathbf{BE}_\alpha(AtTest) := \ \mathbf{repeat}\ \sigma_{Act=\langle m1\rangle}(\mathrm{id}); M_1^c \sqcup \cdots \sqcup \sigma_{Act=\langle ml\rangle}(\mathrm{id}); M_l^c\ \mathbf{until}\ AtTest.$$

The body of the loop says: if the last action was m_1, execute M_1^c, otherwise check if it was m_2, and if yes, execute M_2^c, and so on. Constants m_i represent the "names" of atomic modules, and relational variable Act is silently present as an output of every module. It outputs m_i if the just-executed module was M_i. The dual modality is:

$$\mathbf{BG}_\alpha(AtTest) \;:=\; \curvearrowleft \mathbf{BE}_\alpha \;\curvearrowleft AtTest.$$

Example: EVEN

Given: A set represented by a structure $\mathfrak{A}$ with an empty vocabulary. Question: Is $\lvert dom(\mathfrak{A})\rvert$ even?

We construct a 2-coloured path in the transition system using E and O as labels. To avoid infinite loops, we make sure that the elements never repeat. Each information flow gives us an implicit linear order on domain elements. Define:

$$\begin{array}{rcl} GuessP & := & \{\ \varepsilon x\ P(x)\ \}, \\ CopyPO & := & \left\{\ \forall x\ (O(x) \leftarrow \underline{P(x)})\ \right\}, \\ CopyPE & := & \left\{\ \forall x\ (E(x) \leftarrow \underline{P(x)})\ \right\}. \end{array}$$

$$\begin{array}{l} GuessNewO := \\ \big(GuessP; \mathbf{BG}_{\alpha_E}(\sigma_{P\neq E}(\sigma_{P\neq O}(\mathrm{id})))\big)\ ;\ CopyPO, \\ GuessNewE := \\ \big(GuessP; \mathbf{BG}_{\alpha_E}(\sigma_{P\neq E}(\sigma_{P\neq O}(\mathrm{id})))\big)\ ;\ CopyPE. \end{array}$$

The problem EVEN is now axiomatized as:

$$\alpha_E := (GuessNewO; GuessNewE)^\uparrow.$$

The program is successfully executed if each guessed element is different from any elements guessed so far in the current information flow, and if E and O are guessed in alternation. The expression does not depend on an input vocabulary because there is no projection in front of it. Given a structure $\mathfrak{A}$ over an empty vocabulary, $\mathfrak{A} \models \lvert\alpha_E\rangle\mathbf{T}$, holds whenever there is a successful execution of α_E, that is the size of the input domain is even.

Example: Same Generation $\alpha_{\mathrm{SG}}(\underline{E}, \underline{Root}, \underline{A}, \underline{B})$

Input: Tree represented by the binary edge relation E; the root is contained in the unary singleton-set relation $Root$; two nodes a and b represented by singleton-set relations A and B. Question: Do a and b belong to the same generation in the tree?

Again, we use non-recursive Datalog with limited guessing. To illustrate the strength of the fragment, we capture recursion by the constructs of the algebra, not within an atomic module (of course, in general, recursion in atomic modules is allowed). Note that the PTIME fragment does not allow binary *definable* relations (binary inputs are allowed), so we need to capture the notion of being in the same generation through coexistence in the same structure.

$$M_{base\_case} \;:=\; \left\{\begin{array}{l} \forall x\ (Reach_A(x) \leftarrow \underline{Root(x)}) \\ \forall x\ (Reach_B(x) \leftarrow \underline{Root(x)}) \end{array}\right\},$$

We do a simultaneous propagation starting from the root:

$$M_{ind\_case} := \left\{\begin{array}{l} \varepsilon y\ (Reach'_A(y) \leftarrow \underline{Reach_A(x)}, \underline{E(x,y)}), \\ \varepsilon w\ (Reach'_B(w) \leftarrow \underline{Reach_B(v)}, \underline{E(v,w)}) \end{array}\right\},$$

This atomic module specifies that, if x and v coexisted in the previous state, stored in the interpretations of $Reach_A$ and $Reach_B$, respectively, then y and w will coexist in the successor state. Guessing a unique element is essential – if there is a "fork" in the edges, only one branch is selected.

$$Copy := \left\{\begin{array}{l} \forall x\ (Reach_A(x) \leftarrow \underline{Reach'_A(x)}), \\ \forall x\ (Reach_B(x) \leftarrow \underline{Reach'_B(x)}) \end{array}\right\}.$$

The algebraic expression for the problem is:

$$\begin{array}{l} \alpha_{\mathrm{SG}}(\underline{E}, \underline{Root}, \underline{A}, \underline{B}) := s\pi_{\{E,Root,A,B\}}\ (M_{base\_case}; (M_{ind\_case}; \\ \qquad\qquad Copy; \sigma_{Reach_A \equiv A}(\sigma_{Reach_B \equiv B}(\mathrm{id})))^\uparrow). \end{array}$$

An equivalent expression using definable constructs:

$$\begin{array}{rl} \alpha_{\mathrm{SG}}(\underline{E}, \underline{Root}, \underline{A}, \underline{B}) := & s\pi_{\{E,Root,A,B\}}\ (M_{base\_case}; \\ & \mathbf{repeat} \\ & \quad M_{ind\_case}\ ;\ Copy; \\ & \mathbf{until}\ \sigma_{Reach_A \equiv A}(\sigma_{Reach_B \equiv B}(\mathrm{id}))). \end{array}$$

These examples of PTIME programming use very simple atomic modules. It is also possible to show that if modules of any complexity above PTIME are combined using the operations of Deterministic Fragment (8), the overall complexity does not increase.

Conclusion

In this paper, we introduced a Logic of Information Flows, and showed its connections to binary relational calculi. Such calculi have never been defined and analyzed from the point of view of information flows in classical logic before. It turns out that the only thing lacking in classical logic to define all those constructs is information propagation.

The logic provides a connection of Model Expansion, the main task solved in multiple Declarative programming approaches, such as Constraint Satisfaction Problem, Answer Set Programming, Integer Liner Programming etc., to Tarski's et al. calculi of binary relations and a modal logic.

This direction of developing an algebra of modular systems started from our earlier work (Tasharrofi, Wu, and Ternovska 2011), and then Shahab Tasharrofi's PhD thesis. An approach to solving (a simpler version of) modular systems, inspired by CDCL algorithm of SAT solving was proposed in (Mitchell and Ternovska 2015). An approach based on an algebra of propagators was recently developed in (Bogaerts, Ternovska, and Mitchell 2017). We believe that the proposed logic is applicable in multiple areas of KR – in business process modelling, specifications of robot's behaviour, specifying goals of execution, eventual and extended in time.

For future work, it will be interesting to study the conditions for decidability of the satisfiability problem, for finite and infinite domains. It would also be interesting to investigate methods of simplifying systems described in the formalism presented here. In particular, we can use known equational and quasiequational axiomatizations of the binary operations such as (Jackson and Stokes 2011) and theorem proving to simplify algebraic expressions.

Acknowledgements

Many thanks to Brett McLean and Jan Van den Bussche for useful discussions.

References

Abu Zaid, F.; Grädel, E.; and Jaax, S. 2014. Bisimulation safe fixed point logic. In Goré, R.; Kooi, B. P.; and Kurucz, A., eds., *Advances in Modal Logic 10, invited and contributed papers from the tenth conference on "Advances in Modal Logic," held in Groningen, The Netherlands, August 5-8, 2014*, 1–15. College Publications.

Bogaerts, B.; Ternovska, E.; and Mitchell, D. 2017. Propagators and solvers for the algebra of modular systems. In *Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*.

Denecker, M., and Ternovska, E. 2008. A logic of nonmonotone inductive definitions. *ACM transactions on computational logic (TOCL)* 9(2):1–52.

Fischer, M. J., and Ladner, R. E. 1979. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.* 18(2):194–211.

Fletcher, G.; Gyssens, M.; Leinders, D.; Surinx, D.; den Bussche, J. V.; Gucht, D. V.; Vansummeren, S.; and Wu, Y. 2015. Relative expressive power of navigational querying on graphs. *Information Sciences* 298:390–406.

Girard, J. 1987. Linear logic. *Theor. Comput. Sci.* 50:1–102.

Gurevich, Y., and Shelah, S. 1986. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic* 32:265–280.

Jackson, M., and Stokes, T. 2011. Modal restriction semigroups: towards an algebra of functions. *IJAC* 21(7):1053–1095.

Jónsson, B., and Tarski, A. 1952. Representation problems for relation algebras. *Bull. Amer. Math. Soc.* 74:127–162.

Kolokolova, A.; Liu, Y.; Mitchell, D.; and Ternovska, E. 2010. On the complexity of model expansion. In *Proc., 17th Int'l Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-17)*, 447–458. Springer. LNCS 6397.

Lambek, J. 1958. The mathematics of sentence stucture. *Amer. Math. Monthly* 65(3):154–170.

McLean, B. 2017. Complete representation by partial functions for composition, intersection and anti-domain. *J. Log. Comput.* 27(4):1143–1156.

Mitchell, D. G., and Ternovska, E. 2005a. A framework for representing and solving NP search problems. In *Proc. AAAI*, 430–435.

Mitchell, D. G., and Ternovska, E. 2005b. A framework for representing and solving NP search problems. In *Proc. AAAI'05*, 430–435.

Mitchell, D., and Ternovska, E. 2015. Clause-learning algorithms for modular systems. In Francesco Calimeri, Giovambattista Ianni, M. T., ed., *Logic Programming and Nonmonotonic Reasoning, 13th International Conference, LPNMR 2015, Lexington, September 27-30, 2015. Proceedings*, Lecture Notes in Computer Science. Springer.

Pratt, V. R. 1976. Semantical considerations on floyd-hoare logic. In *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976*, 109–121. IEEE Computer Society.

Pratt, V. R. 1992. Origins of the calculus of binary relations. In *Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS '92), Santa Cruz, California, USA, June 22-25, 1992*, 248–254. IEEE Computer Society.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press.

Surinx, D.; den Bussche, J. V.; and Gucht, D. V. 2017. The primitivity of operators in the algebra of binary relations under conjunctions of containments. In *LICS '17*.

Tarski, A. 1941. On the calculus of relations. *J. Symb. Log.* 6(3):73–89.

Tasharrofi, S.; Wu, X. N.; and Ternovska, E. 2011. Solving modular model expansion tasks. In *Proceedings of the 25th International Workshop on Logic Programming (WLP'11)*, volume abs/1109.0583. Computing Research Repository (CoRR).

Vardi, M. Y. 1982. The complexity of relational query language. In *14th ACM Symp.on Theory of Computing, Springer Verlag (Heidelberg, FRG and NewYork NY, USA)-Verlag*.

Vardi, M. Y. 1995. On the complexity of bounded-variable queries. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California*, 266–276. ACM Press.

Appendix

More Operations: Inclusion, Fixset, Tie

Inclusion We define the following operation implicitly, using logical equivalence between algebraic terms.

$$\alpha \leq \beta \quad \text{iff} \quad (\alpha \cap \beta) = \alpha.$$

The definition gives us:

$$[\![\alpha \leq \beta]\!] = \{(\mathfrak{A}, \mathfrak{B}) \mid \text{If } (\mathfrak{A}, \mathfrak{B}) \in [\![\alpha]\!] \text{ then } (\mathfrak{A}, \mathfrak{B}) \in [\![\beta]\!]\}.$$

Expression $\alpha \leq \beta$ specifies a set of pairs of structures such that if executing α produces such a pair, then executing β produces such a pair as well. Thus, whenever α is a label of a transition in TS_α, then β is a label of the same transition.

Fixset This operation takes the diagonal of the fixed points of α. It is defined as follows:

$$\mathrm{Fix}(\alpha) \;:=\; \alpha \cap \mathrm{id}.$$

By this definition, the semantics of this operation is

$$[\![\mathrm{Fix}(\alpha)]\!] = \{(\mathfrak{A}, \mathfrak{A}) \mid (\mathfrak{A}, \mathfrak{A}) \in [\![\alpha]\!]\}.$$

Tie This operation returns all the points where α and β do not disagree, that is, if started from there, the images of both processes are the same, or they are both undefined. We define it as follows.

$$\alpha \bowtie \beta := \mathrm{Dom}(\alpha \sim \beta). \tag{9}$$

Recall that $\alpha \sim \beta$ is the set of pairs of structures where α and β behave in exactly the same way – both defined and

produce the same outputs on the same inputs, or are both undefined. The definition (9) gives us:

$$\begin{aligned}[\![\alpha \bowtie \beta]\!] = \{(\mathfrak{A},\mathfrak{A}) \mid \forall\mathfrak{B}\forall\mathfrak{B}'(&\text{If } (\mathfrak{A},\mathfrak{B}) \in [\![\alpha]\!] \\ &\text{and } (\mathfrak{A},\mathfrak{B}') \in [\![\beta]\!] \text{ then } \mathfrak{B} = \mathfrak{B}') \\ &\text{or neither of } \alpha,\ \beta \text{ is defined in } \mathfrak{A}: \\ \forall\mathfrak{B}(&\text{ If } \mathfrak{B} \neq \mathfrak{A} \text{ then } (\mathfrak{A},\mathfrak{B}) \in (\mathbf{U}\times\mathbf{U}) \setminus [\![\alpha]\!] \\ &\text{and } (\mathfrak{A},\mathfrak{B}) \in (\mathbf{U}\times\mathbf{U}) \setminus [\![\beta]\!])\}.\end{aligned}$$

Residuation With converse as a basic operation, several more operations become definable. In particular, residuation, which is now understood as a form of division, is definable using Converse.

Right Residuation: $\alpha\backslash\beta \ := \ (\alpha^{\smile};\beta^{-})^{-}$,

Left Residuation: $\alpha/\beta \ := \ (\alpha^{-};\beta^{\smile})^{-}$.

Left residuation is a "converse DeMorgan dual" of the right residuation:

$$\alpha/\beta \ = \ (\alpha^{\smile}\backslash\beta^{\smile})^{\smile}.$$

Residuation is an important operation in Lambek calculus (Lambek 1958).

For earlier uses of the operations and a historic perspective please see Pratt's informative overview paper (Pratt 1992). That work also discusses the historic and contemporary notations for the operations, and we used the latter in this paper.