

Deterministic Linear Dynamic Logic and Hilbert’s Choice Operator

(Version of August 2020)

Eugenia Ternovska

We introduce a new Linear Dynamic Logic where atomic actions are non-deterministic, while programs are deterministic in the sense that they preserve the property of being a function. The logic has a two-level syntax where (1) atomic transitions are axiomatized in any logic interpreted over first-order structures, and (2) algebraic expressions specify programs over those transitions. We define two semantics, the first one as an algebra of binary relations with a dynamic version of Hilbert’s Choice operator ε , and the other as a path semantics. We demonstrate a correspondence between the two. We show that the determinism-preserving algebraic constructs are sufficient to express typical constructs of Imperative programming. Thus, the full power of regular expressions (as in a related logic LDL_f) is not needed for reasoning about such programs.

1 Introduction

A starting question of our research was: *What can we take from classical logic for formalizing efficient computations, what do we need to drop, and what needs to be added?* In the process, we obtained a logic that, we think, is very suitable for reasoning about strategic behaviours in an environment. We are going to present this logic here. But first, let us explain how this logic was obtained. We started from first-order logic with least fixed points (FO(LFP)) and we added an ability to reason about information propagation by specifying inputs and outputs of atomic expressions. The resulting logic was named the Logic of Information Flows (LIF) [25, 24]. The idea was that information would propagate from inputs to outputs similarly to tokens in Petri nets. That way, the logic would “compute”. Adding input and output specification to FO(LFP) gave us an algebra of binary relations, which had a modal logic counterpart, very similar to the modal μ -calculus, with both unary and binary fixed points.

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 [14]. More recently, relation algebras were studied by Fletcher, Van den Bussche, Surinx and their collaborators in a series of paper, see, e.g. [23, 7]. The algebras of relations consider various subsets of operations on binary relations as primitive, and other as derivable. In another direction, [13, 18] and others study partial functions and their algebraic equational axiomatizations.

Then we asked: in this logic, what is really essential for efficient computations, and what is redundant? We restricted the logical connectives (algebraic operators) to exclude any non-determinism they introduce. A criterion was that the standard constructs of Imperative programming must still be definable. We obtained a fragment of LIF that we call here the Determinism-Preserving Fragment. In this fragment, none of the algebraic operations introduce non-determinism, but atomic actions can be non-deterministic.

However, something was still lacking. In many abstract descriptions of polynomial time algorithms, we see phrases like “choose the next element and continue”. A typical example is the Gaussian Elimination algorithm. Examples of choosing a witness are well-known in logic-based reasoning. To construct a derivation in a proof system, we may need to instantiate x in some formula of the form $\exists x \phi(x)$. Similarly,

we may need to construct a witness in a proof of satisfiability of a formula in a fragment with the finite model property. Also, in SAT solving, a solver usually outputs one satisfying truth assignment, out of all possible ones. Moreover, the choice may depend on the choices made earlier in the execution.

To capture this type of reasoning, we added Choice and Converse operators. Our Choice construct can be thought of as a dynamic version of Hilbert’s Epsilon ε operator. The original construct was introduced by Hilbert and Bernays in 1939 [12] for proof-theoretic purposes. General studies of some versions of Choice operator include the work by Blass and Gurevich [3] and by Otto [21]. Our version of Hilbert’s Epsilon operator, however, is different. It acts as a *strategy* function — given a history, it returns the next state, within the constraints imposed by the program. Technically, ε is a free function variable that is instantiated with concrete Choice functions. As a result of adding ε , the computations specified by the logic became linear and logically reversible, i.e., one-to-one functions. The significance of reversibility is that, as in all reversible computations, no information is ever erasable. Thus, for example, if we want to select a new element of the domain, we use Converse to check that it has not been used in the past.

In the context of Strategic Reasoning, Choice can be used to model the choice of an agent, say to formalize a moral choice situation for self-driving cars. Also, Choice can model the choice of the environment that decides on the outcome of agent’s actions, outside of the agent’s control. Our agent’s programs are deterministic, but the outcomes of atomic actions are non-deterministic. This way, deterministic algebraic compositions are detangled from non-deterministic Choice among atomic outcomes.

Below, we present our logic, that can be viewed as a logic of deterministic proofs or programs. It is a “minimal” logic that still expresses the constructs of imperative programming. We give two views on the logic. One is as an algebra of binary relations with the Choice operator. We call this version Linear LIF or LLIF. Another (closely related) view is as a modal temporal logic interpreted over paths. The modal logic is called Deterministic Linear Dynamic Logic over finite traces, or DetLDL_f for short. The main result of the paper is in showing a correspondence between these two semantics. We give examples of cardinality and reachability properties for a particular variant of this logic with output-monadic conjunctive queries as atomic modules. While all free (implicitly, existentially quantified) relational variables are monadic, this logic is not a fragment of MSO because it expresses the query EVEN, which is not in MSO, unless the domain is ordered.

2 An Algebra of Binary Relations

2.1 Preliminaries

In our expositions, we loosely follow the presentation and terminology introduced in [2].

A (*module*) *vocabulary* $\mathcal{M}$ is a triple $(Names, ar, iar)$ where:

- $Names$ is a nonempty set, the elements of which are called *module names*;
- ar assigns an arity to each module name in $Names$;
- iar assigns an input arity to each module name M in $Names$, where $iar(M) \leq ar(M)$.

In this paper, $Names$ will be a set of macros that stand for axiomatizations of atomic actions in some logic, to be specified. We assume familiarity with the basic notions of first-order (FO) and second-order (SO) logic [5] and use ‘:=’ to mean “is by definition”.

We fix a countably infinite universe $\mathbb{A}$ of atomic data elements or, simply, *atoms*. Furthermore, we fix a countably infinite supply of variables $\mathbb{V}$. A *valuation* is a function from $\mathbb{V}$ to $\mathbb{A}$. The set of all valuations is denoted $\mathbf{U}$. The semantics of the logic will be defined in terms of binary relations on $\mathbf{U}$ (abbreviated **BRV**: Binary Relations on Valuations). **In this paper, valuations are relational structures** (defined next). Let τ be a relational vocabulary. An *instantiation function* $s : \mathbb{V} \rightarrow \tau$ maps variables in $\mathbb{V}$ to τ . In

practice, τ is a set of relational symbols that are substituted for variables in an expression, so we may assume it is finite (but is of an unlimited size). Let $\tau := \{S_1, \dots, S_n\}$, each S_i has an associated arity r_i , and $\mathcal{D}$ be a non-empty set. A τ -structure $\mathfrak{A}$ over domain $\mathcal{D}$ is $\mathfrak{A} := (\mathcal{D}; S_1^{\mathfrak{A}}, \dots, S_n^{\mathfrak{A}})$, where $S_i^{\mathfrak{A}}$ is an r_i -ary relation called the *interpretation* of S_i . In this paper, atomic data elements in $\mathbb{A}$ are *relations* that are interpretations of symbols in τ over $\mathcal{D}$, and each structure $\mathfrak{A}$ maps each relational symbol $s(X) \in \tau$ to a relation $(s(X))^{\mathfrak{A}} \in \mathbb{A}$. Thus, in our case, $\mathbf{U}$, the set of all valuations, is the set of all τ -structures over the domain $\mathcal{D}$. For simplicity, we assume that $\mathcal{D}$ is at most countable. We say that $\mathfrak{A}$ and $\mathfrak{B}$ *agree on* $\sigma \subseteq \tau$ if $P^{\mathfrak{A}} = P^{\mathfrak{B}}$ for all $P \in \sigma$, and that they *agree outside of* σ if they agree on $\tau - \sigma$. If $\mathfrak{A}$ is a structure, we use $\mathfrak{A}|_{\sigma}$ to denote its restriction to σ .

Let Σ be an alphabet. A *finite word* is a finite sequence $w = a_0 \cdot a_1 \dots a_n$ of letters in Σ , where the indexes are called *positions*. We use notation w_i for the prefix of w ending in position $i \in \mathbb{N}$. The *length* of w is $|w| := n + 1$. Let $\text{last}(w) := |w| - 1$. We write w_{last} to denote $w_{\text{last}(w)}$. Clearly, $w_{\text{last}} = w$. The empty word, i.e., such that $|w| = 0$, is denoted $\mathbf{e}$. We use $w(i)$ for the i -th letter in word w . A word such that $w(0) = a$, where $a \in \Sigma$, is denoted $w(a)$. The *positions* on the path are denoted as $i, j \in \mathbb{N}$, and the letters (elements of Σ) in those positions as $w(i)$, for $0 \leq i \leq \text{last}(w)$. We use $w(i, j)$ to denote the *subword* obtained from w starting from position i and terminating in position j , where $0 \leq i \leq j \leq \text{last}(w)$. Later in the paper, Σ will be $\mathbf{U}$, the set of all valuations and, in a path semantics, words will be called *paths*.

A *tree* over Σ is a (finite or infinite) nonempty set $Tr \subseteq \Sigma^*$ such that for all $x \cdot c \in Tr$, with $x \in \Sigma^*$ and $c \in \Sigma$, we have $x \in Tr$. The elements of Tr are called *nodes*, and the empty word $\mathbf{e}$ is the *root* of Tr . For every $x \in Tr$, the nodes $x \cdot c \in Tr$ where $c \in \Sigma$ are the *children* of x . A node with no children is a *leaf*. We refer to the length $|x|$ of x as its *level* in the tree. A *branch* b of a tree Tr is a set $b \subseteq Tr$ such that $\mathbf{e} \in b$ and for every $x \in b$, either x is a leaf, or there exists a unique $c \in \Sigma$ such that $x \cdot c \in b$. An $|\Sigma|$ -ary tree is *full* if each node is either a leaf or has exactly $|\Sigma|$ child nodes. It is *Full* if it has no leaves. In a Full tree, every branch is infinite.

Recall that $\mathbf{U}$ is the set of all valuations, here — all τ -structures over the same domain. Let Tr be a Full tree over $\mathbf{U}$. We define an *unconstrained* Dynamic Choice function $f : Tr \rightarrow \mathbf{U}$. Function f , given a node x , returns c such that $x \cdot c \in Tr$, where $c \in \mathbf{U}$. Observe that for every unconstrained Choice function f , there is a unique branch b obtained as follows: $b(f) := \{\mathbf{e}, \mathbf{e} \cdot f(\mathbf{e}), \mathbf{e} \cdot f(\mathbf{e}) \cdot f(f(\mathbf{e})), \dots\}$. This is because, at each level of the Full tree, f makes a unique choice. However, Choice functions are defined for every node in Tr , not just those reachable through a repeated application of f from the root $\mathbf{e}$ of Tr . For each branch, there are infinitely many unconstrained Choice functions that agree on that branch, but act differently from each other on nodes not in that branch. We consider such functions equivalent iff the corresponding branches are the same: $f \cong g \Leftrightarrow b(f) = b(g)$. Each branch $b(f)$ determines an equivalence class $[f]$ of Choice functions.

Let a *path* be a finite word $\mathbf{p} \subseteq \mathbf{U}^*$. Each path has the following form, where $\mathfrak{A}$ is the input structure.

$$\mathfrak{A} = \mathfrak{A}_0 \cdot \mathfrak{A}_2 \cdot \mathfrak{A}_3, \dots \mathfrak{A}_m = \text{last}(\mathbf{p}). \quad (1)$$

We call a path $\mathbf{p}$ an *f-path* if $\mathbf{p} \in b(f)$.

2.2 Syntax of Logic of Linear Information Flows (LLIF)

The one-sorted syntax of the main logic we consider in this paper (LLIF) is given by the grammar below.

$$\alpha ::= \text{id} \mid \varepsilon M(\bar{Z}) \mid \curvearrowright \alpha \mid \curvearrowleft \alpha \mid \alpha ; \alpha \mid \alpha \sqcup \alpha \mid \sigma_{X=Y}^r(\alpha) \mid \alpha^\uparrow \mid \alpha^\sim. \quad (2)$$

Here, M is any module name in $\mathcal{M}$, $\bar{Z}$ is a tuple of variables; and X, Y are variables. For *atomic module expressions*, i.e., expressions of the form $M(\bar{Z})$, the length of $\bar{Z}$ must equal $\text{ar}(M)$. In practice, we will

often write $M(\bar{X}; \bar{Y})$ for atomic module expressions, where $\bar{X}$ is a tuple of variables of length $\text{iar}(M)$ and $\bar{Y}$ is a tuple of variables of length $\text{ar}(M) - \text{iar}(M)$. In this paper, all variables are relational.

In (2), we require that all occurrences of ε in α denote the same free function variable. In addition, we have: Identity (Diagonal) (id), atomic module symbols preceded by Choice operator ($\varepsilon M(\bar{Z})$), Sequential Composition ($;$), Forward-facing ($\curvearrowright$) and Backwards-facing ($\curvearrowleft$) Unary Negations, also called Anti-Domain and Anti-Image, respectively, Preferential Union ($\sqcup$), which is a restriction of Union, Maximum Iterate ($\uparrow$), which is a restriction of a transitive closure construct. While all operation are determinism-preserving, we do not require atomic modules to be deterministic. Note that operations $\sqcup$ and $\uparrow$ are restrictions of operations of regular languages, Union and the Kleene star, respectively.

In general, this logic has a **two-level syntax**, where, in addition to the algebra (2), each atomic module is axiomatized in some logic, such as, for example, (non-recursive) Datalog, conjunctive queries, or, in fact, any logic of an arbitrary expressiveness. Module names M in $\mathcal{M}$ serve as macros for such axiomatizations. The axiomatizations provide a context for the semantics (since they impose restrictions on the atomic transitions). There could be different semantic contexts, e.g., through access-restricted relations in databases [1], through interpretations of atomic modules as edges in graph databases, or through axiomatizations of agents' actions in Strategic Reasoning. Below, we describe one particular context, through the Model Expansion task [19]. It amounts to a **structure-rewriting** operation.

2.3 Semantic Context: Atomic Transitions

A semantic context gives interpretation to module names M (without ε) in $\mathcal{M}$. First, we give an informal intuition. Under a unary interpretation, each module can be viewed as a boolean query or a decision procedure. Thus, a module corresponds to a class of structures (a set when the domain is fixed). For example, it can be the class of all graphs with all associated proper 3-colourings. However, we view each such module dynamically, that is, as a **transduction** represented by a binary relation. E.g., for a graph on the input, an action of 3-colouring produces the colours of the graph on the output.¹ The interpretations of all other variables, except those on the outputs, remain unchanged. We formalize it as follows.

The following definition is mathematically necessary in defining the semantics of atomic module symbols. Given a sub-vocabulary γ of τ , a subset $W \subseteq \mathbf{U}$ is *determined by γ* if it satisfies

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

A *unary interpretation* T of module vocabulary $\mathcal{M}$ determines the following:

- instantiation function $s : \mathbb{V} \rightarrow \tau$, where τ is a relational vocabulary;
- finite or infinite domain $\mathcal{D}$ of elements;
- For each atomic module symbol M in $\mathcal{M}$ and tuples of variables $\bar{Z}$ with $|\bar{Z}| = \text{ar}(M)$, function $T(M)$ assigns a subset $T(M) \subseteq \mathbf{U}$, where $\mathbf{U}$ is a set of all $s(\mathbb{V})$ -structures over domain $\mathcal{D}$, with the property that $T(M)$ is determined by $s(\bar{Z})$.²

Note that $T(M)$ can be viewed as a relation over atoms $\mathbb{A}$ (see Section 2.1), i.e., atomic relations.

Unary interpretation T , for each atomic module expression M , gives rise to a **binary** relation on $\mathbf{U}$:

$$\begin{aligned} \llbracket M(\bar{X}; \bar{Y}) \rrbracket_T &:= \{ (\mathfrak{A}, \mathfrak{B}) \in \mathbf{U} \times \mathbf{U} \mid \text{there exists } \mathfrak{C} \in T(M) \text{ such that} \\ &\quad \mathfrak{C}|_{s(\bar{X})} = \mathfrak{A}|_{s(\bar{X})}, \\ &\quad \mathfrak{C}|_{s(\bar{Y})} = \mathfrak{B}|_{s(\bar{Y})} \\ &\text{and } \mathfrak{A}|_{\tau-s(\bar{Y})} = \mathfrak{B}|_{\tau-s(\bar{Y})} \}. \end{aligned} \tag{3}$$

¹Alternatively, $\mathcal{M}$ may specify that a colouring is given on the input, and edges of a graph are produced on the output.

²We abuse the notations slightly by writing $s(\mathbb{V})$ to denote the set $\{s(Z_1), s(Z_2), \dots\}$, where $Z_1, Z_2, \dots$ are variables in $\mathbb{V}$.

Intuitively, the semantics of an expression $M(\bar{X}; \bar{Y})$ represents a transition from $\mathfrak{A}$ to $\mathfrak{B}$: the inputs of the module are “read” in $\mathfrak{A}$ and the outputs are updated in $\mathfrak{B}$. The value of every variable that is not an output is preserved; this semantic principle is a realization of the commonsense law of inertia [17, 16].

Atomic modules can be axiomatized in any logic with model-theoretic semantics. These modules solve Model Expansion (MX) task [19], which is the task of expanding a structure (in this case, $\mathfrak{A}|_{s(\bar{X})}$), to the full vocabulary (in this case, $s(\bar{X}) \cup s(\bar{Y})$) in order to satisfy a formula in some logic.

Problem: Model Expansion (MX) Given: σ -structure $\mathfrak{A}$, formula ϕ in logic $\mathcal{L}$, $\sigma \subseteq \text{vocab}(\phi)$. Find: $\mathfrak{B}$ that expands $\mathfrak{A}$ to $\text{vocab}(\phi)$ and $\mathfrak{B} \models_{\mathcal{L}} \phi$ (the decision version is Decide: $\mathfrak{B} \models_{\mathcal{L}} \phi?$)

For the same logic, in terms of complexity, the MX task is in-between Model Checking (MC) and satisfiability: In MC, a structure to the full $\text{vocab}(\phi)$ is given, in SAT, no structure is given. The three tasks for different logics have been analyzed in [15].

2.4 Semantics of the Algebra

We define the **binary semantics of (2) in the context of a given unary interpretation T** , which gives rise to atomic transitions (hence, it is abbreviated as T). In addition, the semantics is conditional on a **specific instantiation f of the free Choice variable ε** . Recall that BRV stands for Binary Relation on Valuations. The semantics of a LLIF expression α is given as a BRV which will be denoted by $\llbracket \alpha \rrbracket_T^f$. Thus, adapting Gurevich’s terminology [9, 10], every algebraic expression α denotes a **global BRV** $\llbracket \alpha \rrbracket$: a function that maps interpretations T of $\mathcal{M}$ and instantiations f of ε to the BRV $\alpha(T, f) := \llbracket \alpha \rrbracket_T^f$.

First, we define the semantics of the **Determinism-Preserving Fragment**, which is (2) without ε and Converse. The name comes from the fact that all operations in this fragment preserve functions (i.e., they do not “add” non-determinism), while atomic modules are allowed to be non-deterministic. The meaning of the subscript f will become clear when we define the semantics of the free Choice variable ε . Function f will be a specific instantiation of this variable. Choice plays no role in the Determinism-Preserving Fragment (since it does not have ε).

Atomic module expressions M in $\mathcal{M}$ are interpreted as in (3). In this fragment, they are just binary relations on structures $\llbracket M(\bar{X}; \bar{Y}) \rrbracket_T^f \subseteq \mathbf{U} \times \mathbf{U}$.

We extend the binary interpretation $\llbracket \cdot \rrbracket_T^f$ to all algebraic expressions.

Identity (Diagonal) $\llbracket \text{id} \rrbracket_T^f := \{(\mathfrak{A}, \mathfrak{B}) \in \mathbf{U} \times \mathbf{U} \mid \mathfrak{A} = \mathfrak{B}\}$. Operation id is sometimes called the “*nil*” action, or it can be seen as an empty word in the formal language theory.

Right Selection (Equality) $\llbracket \sigma_{X=Y}^r(\alpha) \rrbracket_T^f := \{(\mathfrak{A}, \mathfrak{B}) \in \mathbf{U} \times \mathbf{U} \mid (\mathfrak{A}, \mathfrak{B}) \in \llbracket \alpha \rrbracket_T^f \text{ and } (s(X))^{\mathfrak{B}} = (s(Y))^{\mathfrak{B}}\}$.

Sequential Composition $\llbracket \alpha; \beta \rrbracket_T^f := \{(\mathfrak{A}, \mathfrak{B}) \in \mathbf{U} \times \mathbf{U} \mid \exists \mathfrak{C} ((\mathfrak{A}, \mathfrak{C}) \in \llbracket \alpha \rrbracket_T^f \text{ and } (\mathfrak{C}, \mathfrak{B}) \in \llbracket \beta \rrbracket_T^f)\}$.

Forward Unary Negation (Anti-Domain) Note that regular complementation includes all possible transitions except α . We introduce a stronger negation which is essentially unary (expressed as binary with two identical elements in the pair) and excludes states where α originates.³ It says “there is no outgoing α -transition”. The semantics is: $\llbracket \neg \alpha \rrbracket_T^f = \{(\mathfrak{B}, \mathfrak{B}) \in \mathbf{U} \times \mathbf{U} \mid \forall \mathfrak{B}' (\mathfrak{B}, \mathfrak{B}') \notin \llbracket \alpha \rrbracket_T^f\}$.

Backwards Unary Negation (Anti-Image) A similar operation for the opposite direction says “there is no incoming α -transition”. Thus, $\llbracket \neg \alpha \rrbracket_T^f = \{(\mathfrak{B}, \mathfrak{B}) \in \mathbf{U} \times \mathbf{U} \mid \forall \mathfrak{B}' (\mathfrak{B}', \mathfrak{B}) \notin \llbracket \alpha \rrbracket_T^f\}$. Each of the unary negations is a restriction of the regular negation (complementation). Unlike regular negation, these operations preserve determinism of the components. The connectives $\neg$ and $\neg$ have the properties of the Intuitionistic negation: $\neg \neg \neg \alpha = \neg \alpha$, $\neg \neg \alpha \neq \alpha$, and the same for the other direction.

³Ideally, these operations should be denoted with a notation closer to the typical negation sign, i.e., more “square”, however we could not find a suitable pair of LaTeX symbols. Any suggestion for a better solution is welcome.

Maximum Iterate Maximum Iterate is a determinization of the Kleene star α^* (reflexive transitive closure). It outputs only the longest transition out of all possible transitions produced by the Kleene star. We define it as: $\alpha^\dagger = \bigcup_{0 \leq n < \omega} (\alpha_n; \curvearrowright \alpha)$, where $\alpha_0 := \text{id}$, and $\alpha_{n+1} := \alpha; \alpha_n$. For readers familiar with the least fixed point operator $\mu Z.$, we mention an alternative definition: $\alpha^\dagger := \mu Z.(\curvearrowright \alpha \cup \alpha; Z)$.

Preferential union $\alpha \sqcup \beta := \alpha \cup (\curvearrowright \alpha; \beta)$. By this definition,

$$\llbracket \alpha \sqcup \beta \rrbracket_T^f = \left\{ (\mathfrak{A}, \mathfrak{B}) \in \mathbf{U} \times \mathbf{U} \mid \begin{array}{ll} (\mathfrak{A}, \mathfrak{B}) \in \llbracket \alpha \rrbracket_T^f & \text{if } (\mathfrak{A}, \mathfrak{A}) \in \llbracket \text{Dom}(\alpha) \rrbracket_T^f, \\ (\mathfrak{A}, \mathfrak{B}) \in \llbracket \beta \rrbracket_T^f & \text{if } (\mathfrak{A}, \mathfrak{A}) \notin \llbracket \text{Dom}(\alpha) \rrbracket_T^f \\ & \text{and } (\mathfrak{A}, \mathfrak{A}) \in \llbracket \text{Dom}(\beta) \rrbracket_T^f \end{array} \right\},$$

where $\text{Dom}(\alpha) := \curvearrowright \curvearrowright \alpha$ is a projection onto the left element of the binary relation. Thus, we perform α if it is defined, otherwise we perform β . This operation is a determinization of Union.

Having defined the semantics of the Deterministic Fragment, we now define the semantics of the full algebra (2). For that, we introduce the semantics of ε and Converse. Recall that ε is a free Choice variable. The semantics is parameterized with a specific instantiation f of this variable.

A Dynamic Version of Hilbert's Choice Operator Epsilon

Let Tr be a Full tree over $\mathbf{U}$, and f be an unconstrained Choice function on Tr .

The semantics of Epsilon A specific Choice function f is a parameter of the semantics that affects the result of the evaluation of an expression. It serves as an instantiation of the free function variable ε .

$$\llbracket \varepsilon M(\bar{X}; \bar{Y}) \rrbracket_T^f := \{ (w(\text{last}), f(w_{\text{last}})) \in \llbracket M(\bar{X}; \bar{Y}) \rrbracket_T \text{ for } w \text{ in } b(f) \}. \quad (4)$$

By this definition, the Epsilon operator, for each τ -structure $\mathfrak{A} = w(\text{last})$ where M is defined, arbitrarily selects *precisely one* structure $\mathfrak{B} = f(w_{\text{last}})$ out of all possible structures that are M -successors of $\mathfrak{A}$, as prescribed by f on the branch $b(f)$. Notice that f -choices outside $b(f)$ are not included.

The semantics of Converse For the atomic case of Converse, we have:

$$\llbracket \varepsilon M(\bar{X}; \bar{Y})^\smile \rrbracket_T^f := \{ (\mathfrak{B}, \mathfrak{A}) \in \mathbf{U} \times \mathbf{U} \mid (\mathfrak{A}, \mathfrak{B}) \in \llbracket \varepsilon M(\bar{X}; \bar{Y}) \rrbracket_T^f \}.$$

We extend the semantics of $\smile$ to all operators of (2) inductively:

$$\begin{aligned} (\alpha; \beta)^\smile &:= \beta^\smile; \alpha^\smile, & (\curvearrowright \alpha)^\smile &:= \curvearrowright \alpha, \\ (\curvearrowleft \alpha)^\smile &:= \curvearrowleft \alpha, & (\alpha^\dagger)^\smile &:= \alpha^\smile{}^\dagger, \\ (\alpha^\smile)^\smile &:= \alpha, & (\sigma_\Theta^r(\alpha))^\smile &:= \sigma_\Theta^r(\alpha^\smile). \end{aligned} \quad (5)$$

Proposition 1. *For each f , the interpretation $\llbracket \alpha \rrbracket_T^f$ of α in (2) induces a partial function on $\mathbf{U}^+$. The domain and the image of that function are in $b(f)$.*

Proof. (induction on the structure of the algebraic expressions in (2)). Pick an arbitrary f .

Base cases, id , $\varepsilon M(\bar{X}; \bar{Y})$ and $\varepsilon M(\bar{X}; \bar{Y})^\smile$: If $\alpha := \text{id}$, its interpretation $\llbracket \text{id} \rrbracket_T^f$ induces the identity function $\mathbf{f}_\alpha$ on $\mathbf{U}^+$. If $\alpha := \varepsilon M(\bar{X}; \bar{Y})$, for each f , we have a partial function

$$\mathbf{f}_\alpha := \{ w_{(\text{last}-1)} \mapsto w \mid (w(\text{last}), f(w_{\text{last}})) \in \llbracket M(\bar{X}; \bar{Y}) \rrbracket_T^f, w \in b(f) \} \text{ on } \mathbf{U}^+.$$

Atomic Converse induces a partial function because it reverses one-to-one $\mathbf{f}_\alpha$, for $\alpha = \varepsilon M(\bar{X}; \bar{Y})$.

Inductive cases Unary negations induce functions on $\mathbf{U}^+$ because id does, and they are its subsets. Composition is a composition of partial functions, and thus is such a function itself. In $\alpha \sqcup \beta$, only one of the

terms gets “executed”, which, by inductive hypothesis, is a partial function on $\mathbf{U}^+$. Selection restricts a partial function, so is also a partial function. Maximum Iterate is a sequential composition of the iterated component, thus, by the inductive hypothesis, is a partial function on $\mathbf{U}^+$. Observe that all the cases so far are one-to-one partial functions on $\mathbf{U}^+$, with both domains and images in $b(f)$, so the general (non-atomic) Converse has these properties too. $\square$

Binary Satisfaction Relation Given a well-formed α , we say that pair of structures $(\mathfrak{A}, \mathfrak{B})$, *satisfies* α under interpretation T and choice function f iff $(\mathfrak{A}, \mathfrak{B}) \in \llbracket \alpha \rrbracket_T^f$. In symbols,

$$(\mathfrak{A}, \mathfrak{B}) \models_T \alpha[\varepsilon/f] \Leftrightarrow (\mathfrak{A}, \mathfrak{B}) \in \llbracket \alpha \rrbracket_T^f. \quad (6)$$

3 2-Way Deterministic Linear Dynamic Logic (DetLDL_f)

We now provide an alternative (and equivalent) two-sorted version of the syntax of (2) in the form of a modal temporal (dynamic) logic. The syntax is given by the following grammar.⁴

$$\begin{aligned} \alpha &::= \text{id} \mid \varepsilon M_a(\bar{Z}) \mid \neg \alpha \mid \neg \alpha \mid \alpha; \alpha \mid \alpha \sqcup \alpha \mid \sigma_{X=Y}^r(\alpha) \mid \alpha^\uparrow \mid \alpha^\sim \mid \phi? \\ \phi &::= \mathbf{T} \mid M_p(\bar{Z}) \mid \neg \phi \mid |\alpha\rangle \phi \mid \langle \alpha| \phi. \end{aligned} \quad (7)$$

The subscripts a and p stand for “actions” and “propositions”, respectively. Intuitively, proposition modules M_p make only self-loop transitions. We write $M_p(\bar{X}; \bar{Y})$, as a particular case of $M_p(\bar{X}; \bar{Y})$, to indicate that only input variables are checked. The expressions in the first line of (7) are typically called *process* formulae, and those in the second line *state* formulae. The idea is that state formulae ϕ in the second line of (7) are “unary” in the same sense as $P(x)$ is a unary notation for $P(x, x)$. Semantically, unary formulae are subsets of the identity relation on $\mathbf{U}$. State formulae $|\alpha\rangle \phi$ and $\langle \alpha| \phi$ are right and left-facing existential modalities. The state formulae in the second line of (7) are shorthands that use the operations in the first line: $\mathbf{T} := \text{id}$, $\neg \phi := \neg \phi$ (or $\neg \phi$), $|\alpha\rangle \phi := \neg \neg \alpha; \phi$, $\langle \alpha| \phi := \neg \neg \phi; \alpha$, $\phi? := \neg \neg \phi$.

We also define $|\alpha| \phi := \neg(\alpha; \neg \phi)$ and $\text{Dom}(\alpha) := \neg \neg \alpha$.

Temporal Dynamic Logics such as (7) are typically interpreted over pointed (rooted) transition systems $\mathcal{T}_{\mathfrak{A}}$. In our case, $\mathfrak{A}$ is an initial structure given on the input. The **states of $\mathcal{T}_{\mathfrak{A}}$ are $\mathbf{U}$** , which is the set of all τ -structures over the domain of the input structure $\mathfrak{A}$. **Transitions are given by binary relations on τ -structures**, as already defined, and state formulae ϕ are interpreted by subsets of the binary identity relation. Thus, they are a particular case of binary relations.

Unary Satisfaction Relation For state formulae ϕ , we define

$$\mathfrak{A} \models_T \phi[\varepsilon/f] \Leftrightarrow (\mathfrak{A}, \mathfrak{A}) \models_T \phi[\varepsilon/f]. \quad (8)$$

Thus, state formulae correspond to sets of states where they are satisfied. When a specific choice function in (8) is not specified, each formula $\phi(\varepsilon)$ with free function variable ε can be seen as defining all possible Choice functions f that witness $\mathfrak{A} \models_T \phi[\varepsilon/f]$.

Expressing Programming Constructs It is well-known that in Propositional Dynamic Logic (PDL) [6], imperative programming constructs are definable using a fragment of regular languages, see the Dynamic Logic book by Harel, Kozen and Tiuryn [11]. A language that corresponds to the expressions composed

⁴Under the path semantics defined in Section 4, the Epsilon notation in front of atomic modules is, in fact, not needed. We will demonstrate that the role of a specific choice function f , in a precise sense to be defined, is fulfilled by a path $\mathbf{p}$.

of the imperative constructs is called *Deterministic Regular (While) Programs* in [11].⁵ It is easy to see that imperative constructs are also definable in (7), the modal version of the logic.

skip := id,
fail := $\neg \text{id}$,
if ϕ **then** α **else** β := $(\phi?; \alpha) \sqcup \beta$,
while ϕ **do** α := $(\phi?; \alpha)^\dagger; (\neg \phi?)$,
repeat α **until** ϕ := $\alpha; ((\neg \phi?); \alpha)^\dagger; \phi?$.

Thus, importantly, the full power of regular languages is not needed to define these constructs.

Composition as Conjunction The reason for excluding intersection in (2) is that it “wastes” computations already performed. It plays the role of confluence, where two process are executed independently, and the result is accepted only if they both produce the same result, and is rejected otherwise. Since we are aiming for a computationally *optimal* fragment, we excluded Intersection. Indeed, none of the operations of (2) add computational power through adding non-determinism, and none reduce it by ignoring computations that are performed already. One may think that intersection is needed to express conjunction of state formulae. This is not the case. In fact, it has been known for a long time that relational composition, also called relative or dynamic product, is an analog of logical or static conjunction. According to Pratt [22], “This view of composition/concatenation as a form of conjunction predates even Peirce and would appear to be due to De Morgan in 1860 [20]”. In some Resource logics, Composition is called Multiplicative Conjunction. It is not hard to check that, in the unary (state formulae) case, ‘;’ and ‘ $\sqcup$ ’ have the properties of ‘ $\wedge$ ’ and ‘ $\vee$ ’, respectively.

4 Path Semantics

In this section, we define an alternative (and equivalent) *path* semantics of (7) inspired by [4]. Then, we prove that all “accepted” paths (i.e., those constrained by α) are f -paths for some choice function f . This theorem shows the equivalence of the binary and the path semantics. Path semantics of the two-sorted syntax (7) is defined by a simultaneous induction on formulae ϕ and path expressions α .

State Formulae (line 2 of (7)) We define when state $\mathbf{p}(i)$ of path $\mathbf{p}$ *satisfies* formula ϕ under T , denoted $\mathbf{p}, i \models_T \phi$, where $0 \leq i \leq |\mathbf{p}|$, as follows:

- $\mathbf{p}, i \models_T M_p(\bar{X}; _)$, for $M_p \in \mathcal{M}$, iff $\mathbf{p}(i) \models_T M_p(\bar{X}; _)$, for proposition module M_p .
- $\mathbf{p}, i \models_T \neg \phi$ iff $\mathbf{p}, i \not\models_T \phi$,
- $\mathbf{p}, i \models_T \langle \alpha \rangle \phi$ iff there is $j, i \leq j \leq \text{last}(\mathbf{p})$, with $\mathbf{p}(i, j) \in \mathcal{L}(\alpha)$ such that $\mathbf{p}(j) \models \phi$,
- $\mathbf{p}, i \models_T [\alpha] \phi$ iff for all $j, i \leq j \leq \text{last}(\mathbf{p})$, such that $\mathbf{p}(i, j) \in \mathcal{L}(\alpha)$, we have $\mathbf{p}(j) \models \phi$.

The cases for $\langle \alpha |$ and $[\alpha |$ are similar.

Process (Path) Formulae (line 1 of (7)) These formulae are interpreted as in the binary semantics, but always with respect to a path. The relation $\mathbf{p}(i, j) \in \mathcal{L}(\alpha)$ is defined as follows.

⁵ Please note that Deterministic Regular expressions and the corresponding Glushkov automata are unrelated to what we study here. In those terms, expression $a; a^*$ is Deterministic Regular, while $a^*; a$ is not. Both expressions are not in our language.

- Atomic modules: $\mathbf{p}(i, i+1) \in \mathcal{L}(\varepsilon M_a(\bar{X}; \bar{Y}))$ if $(\mathbf{p}(i), \mathbf{p}(i+1)) \in \llbracket M_a(\bar{X}; \bar{Y}) \rrbracket_T$ and $M_a \in \mathcal{M}$, for action module M_a .
- Diagonal: $\mathbf{p}(i, j) \in \mathcal{L}(\text{id})$ if $j = i$,
- Test: $\mathbf{p}(i, j) \in \mathcal{L}(\phi?)$ if $j = i$ and $\mathbf{p}, i \models_T \phi$,
- Right Unary Negation: $\mathbf{p}(i, j) \in \mathcal{L}(\neg \alpha)$ if $j = i$ and there is no k such that $\mathbf{p}(i, k) \in \mathcal{L}(\alpha)$,
- Left Unary Negation: $\mathbf{p}(i, j) \in \mathcal{L}(\neg \alpha)$ if $j = i$ and there is no l such that $\mathbf{p}(l, j) \in \mathcal{L}(\alpha)$,
- Sequential Composition: $\mathbf{p}(i, j) \in \mathcal{L}(\alpha; \beta)$ if there exists l , with $i \leq l \leq j$, such that $\mathbf{p}(i, l) \in \mathcal{L}(\alpha)$ and $\mathbf{p}(l, j) \in \mathcal{L}(\beta)$,
- Preferential Union: $\mathbf{p}(i, j) \in \mathcal{L}(\alpha \sqcup \beta)$ if (recall that $\text{Dom}(\alpha) := \neg \neg \alpha$)

$$\left\{ \begin{array}{l} \mathbf{p}(i, j) \in \mathcal{L}(\alpha) \text{ and } \mathbf{p}(i) \models_T \text{Dom}(\alpha), \text{ or} \\ \mathbf{p}(i, j) \in \mathcal{L}(\beta), \mathbf{p}(i) \not\models_T \text{Dom}(\alpha) \text{ and } \mathbf{p}(i) \models_T \text{Dom}(\beta) \end{array} \right\}.$$

- Maximum Iterate: $\mathbf{p}(i, j) \in \mathcal{L}(\alpha^\dagger)$ if $i = j$ and $\mathbf{p}(i, j) \in \mathcal{L}(\neg \alpha)$ or there exists l , with $i \leq l \leq j$, such that $\mathbf{p}(i, l) \in \mathcal{L}(\alpha)$ and $\mathbf{p}(l, j) \in \mathcal{L}(\alpha^\dagger)$.
- Right Selection: $\mathbf{p}(i, j) \in \mathcal{L}(\sigma_{X=Y}^r(\alpha))$ if $(s(X))^{\mathbf{p}(j)} = (s(Y))^{\mathbf{p}(j)}$,
- Converse: $\mathbf{p}(i, j) \in \mathcal{L}(\alpha^\sim)$ if $\mathbf{p}(j, i) \in \mathcal{L}(\alpha)$.

Reasoning Tasks The expression for the last state of a path is: $\text{Last} := |id| \neg \mathbf{T}$. Then we can ask

$$\mathfrak{A} \models_T |\alpha| \text{Last} ? \quad (9)$$

where α may have a free Choice variable ε . The question asks whether the path determined by a specific sequence of choices f , (i.e., a strategy, an instantiation of ε), represents a (terminated) successful execution of α . Note that for input-output disjoint expressions [2], the question (9) is a Model Expansion task defined earlier. Intuitively, the words (1) are computations that witness the “yes” answer to the question (9). Whichever f -path $\mathbf{p}(\mathfrak{A})$ we substitute — the formula “decides” whether to accept or reject it. Equivalently, the formula constrains all such sequences of choices. In general, the sequences (1) witnessing (9) are of arbitrary finite length. But, under certain conditions on the logic for specifying atomic modules, their lengths are polynomial in the size of a (finite) domain, for each fixed formula ϕ . The details and the proof of this property is outside of the scope of this paper.

To express temporal properties, we define $\text{any}_M := M_1 \sqcup \dots \sqcup M_l$, where $\{M_1, \dots, M_l\} = \mathcal{M}$. Exists (**E**) and Globally (**G**) are modal operators analogous to F (Exists) G (Globally) of Linear Temporal Logic (LTL), respectively. We also introduce Back Exists (**BE**) and Back Globally (**BG**), which are similar to P and H , respectively, of the version of LTL with backwards-facing temporal modalities. We define: $\mathbf{E}\phi := |\text{repeat any}_M \text{ until } \phi| \mathbf{T}$. Similarly, for reasoning “backwards in time”, we have: $\mathbf{BE}\phi := \langle (\text{repeat any}_M \text{ until } \phi)^\sim | \mathbf{T}$. The program executes the actions of α “backwards” until ϕ is found to be true. The dual modalities are: $\mathbf{G}\phi := \neg \mathbf{E} \neg \phi$, $\mathbf{BG}\phi := \neg \mathbf{BE} \neg \phi$.

For a *fixed* semantic context $T(M)$ that gives an interpretation to the atomic transitions M in $\mathcal{M}$, we can specify a *agent’s program* by α and a *property* by β , and check $\llbracket \alpha \rrbracket^f \subseteq \llbracket \beta \rrbracket^f$ for any instantiation f of the free Choice variable ε . For example, α may be an imperative program, and β be a temporal property, e.g., safety or liveness. The strategy f can be taken to be a strategy of the agent, or a strategy

of the environment, depending on the application. Of course, checking containment for a fixed f , fixed f and T , or for all f and T is also possible.

Equivalence $\llbracket \alpha \rrbracket_T^f = \llbracket \beta \rrbracket_T^f$ means containment in both directions. If it holds for all f and T , then α and β denote the same global BRV, i.e., a function $\llbracket \alpha \rrbracket$ that maps interpretations T of $\mathcal{M}$ and instantiations f of ε to the BRV $\alpha(T, f) := \llbracket \alpha \rrbracket_T^f$.

Correspondence between Binary and Path semantics

The following lemma shows the connection between path and binary semantics for *process* formulae.

Lemma 1. $\mathbf{p}(i, j) \in \mathcal{L}(\alpha) \Leftrightarrow (\mathbf{p}(i), \mathbf{p}(j)) \in \llbracket \alpha \rrbracket_T^f$, where $\mathbf{p}$ is an f -path, i.e., $\mathbf{p} \in b(f)$ for some concrete Choice function f .

Proof. We show the equivalence by induction on the structure of the process formulae in (7). We demonstrate the existence of f by constructing a subset b of $b(f)$. Any extension of this subset to the full $b(f)$ is an equivalence class of Choice functions. Any function in that class gives us the f needed in the lemma.

Base cases:

- Atomic modules: $\mathbf{p}(i, i+1) \in \mathcal{L}(\varepsilon M_a(\bar{X}; \bar{Y}))$
 $\Leftrightarrow$ (by the path semantics) $(\mathbf{p}(i), \mathbf{p}(i+1)) \in \llbracket M_a(\bar{X}; \bar{Y}) \rrbracket_T$
 $\Leftrightarrow$ (by the binary semantics) $(\mathbf{p}(i), \mathbf{p}(i+1)) \in \llbracket \varepsilon M_a(\bar{X}; \bar{Y}) \rrbracket_T^f$, where $\mathbf{p}_i, \mathbf{p}_{(i+1)} \in b \subset b(f)$ (recall that $\mathbf{p}_i$ is the prefix of $\mathbf{p}$ ending in position i). That is, f must “agree” with M_a , $f(\mathbf{p}_i) = \mathbf{p}(i+1)$.
- Diagonal: $\mathbf{p}(i, j) \in \mathcal{L}(\text{id}) \Leftrightarrow$ (by the path semantics) $j = i$
 $\Leftrightarrow$ (by the binary semantics) $(\mathbf{p}(i), \mathbf{p}(i)) \in \llbracket \text{id} \rrbracket_T^f$. That is, only $\mathbf{p}_i$ is added to $b \subset b(f)$.
- Test: $\mathbf{p}(i, j) \in \mathcal{L}(\phi?) \Leftrightarrow$ (by the path semantics) $j = i$ and $\mathbf{p}, i \models_T \phi$
 $\Leftrightarrow$ (by the binary semantics) $(\mathbf{p}(i), \mathbf{p}(i)) \in \llbracket \phi? \rrbracket_T^f$. Again, only $\mathbf{p}_i$ is added to $b \subset b(f)$.

Inductive cases: Assume that, for the paths $\mathbf{p}(i, l)$ and $\mathbf{p}(k, j)$, the equivalence holds, and $\mathbf{p}_i, \mathbf{p}_l, \mathbf{p}_k$, and $\mathbf{p}_j$ (by construction, all prefixes of the same path $\mathbf{p}$ are in $b \subset b(f)$, for some f). We show that the lemma holds for all applications of the algebraic operations.

- Sequential Composition: $\mathbf{p}(i, j) \in \mathcal{L}(\alpha; \beta) \Leftrightarrow$ (by the path semantics) there exists l , with $i \leq l \leq j$, such that $\mathbf{p}(i, l) \in \mathcal{L}(\alpha)$ and $\mathbf{p}(l, j) \in \mathcal{L}(\beta)$. By inductive hypothesis, $(\mathbf{p}(i), \mathbf{p}(l)) \in \llbracket \alpha \rrbracket_T^f$, $(\mathbf{p}(k), \mathbf{p}(j)) \in \llbracket \beta \rrbracket_T^f$, where $l = k$, and $\mathbf{p}_i, \mathbf{p}_l$ and $\mathbf{p}_j$ are in $b \subset b(f)$, for some f . By the binary semantics of Composition, $(\mathbf{p}(i), \mathbf{p}(j)) \in \llbracket \alpha; \beta \rrbracket_T^f$. Since $\mathbf{p}_i$ and $\mathbf{p}_j$ are already in $b \subset b(f)$, the lemma holds for this case.

For the other inductive cases, the same type of reasoning applies. We conclude that the lemma holds. $\square$

Since $(\mathbf{p}(0), \mathbf{p}(\text{last})) \in \llbracket \alpha \rrbracket_T^f$ witnesses a successful execution of α , the longest word in the subset b of $b(f)$ is a witness of the form (1) to the task (9).⁶

The lemma implies that the containment property $\llbracket \alpha \rrbracket^f \subseteq \llbracket \beta \rrbracket^f$, for any instantiation f of the free Choice variable ε , can now be expressed as formal language containment $\mathcal{L}(\alpha) \subseteq \mathcal{L}(\beta)$, which, in turn, can be reduced to the emptiness problem of the corresponding product automaton.

It also follows from the lemma that the *binary* satisfaction relation (6) of the binary semantics and the path semantics of *process* formulae of (7) coincide. We now show the same correspondence for the *unary* satisfaction relation (8) and the path semantics of *state* formulae of (7).

⁶For readers familiar with categorical grammars, we mention that the set of all substrings $\mathbf{p}(i, j)$, $0 \leq i \leq j \leq \text{last}$, of $(\mathbf{p}(0), \mathbf{p}(\text{last})) \in \llbracket \alpha \rrbracket_T^f$ forms a residuated semigroup.

Theorem 1. For any state formula ϕ of (7), we have: $\mathbf{p}, 0 \models_T \phi \Leftrightarrow \mathfrak{A} \models_T \phi[\varepsilon/f]$, where $\mathbf{p}$ is an f -path and $\mathbf{p}(0) = \mathfrak{A}$.

Proof. The property follows by induction on the structure of the algebraic expressions of (7).

Base case, atomic modules-propositions:

$$\begin{aligned} & \mathbf{p}, 0 \models_T M_p(\bar{X}; \bar{X}) \\ \Leftrightarrow & \text{(by the path semantics)} \mathbf{p}(0) \models_T M_p(\bar{X}; \bar{X}) \\ \Leftrightarrow & \text{(by Lemma 1, assuming } \mathbf{p} \text{ is an } f\text{-path)} (\mathfrak{A}, \mathfrak{A}) \in \llbracket M_p(\bar{X}; \bar{X}) \rrbracket_T^f \\ \Leftrightarrow & \text{(by (6) and (8)) } \mathfrak{A} \models_T M_p(\bar{X}; \bar{X})[\varepsilon/f]. \end{aligned}$$

Choice function does not play any role in the case of atomic modules-propositions.

Case $\neg\phi$:

$$\begin{aligned} & \mathbf{p}, 0 \models_T \neg\phi \Leftrightarrow \text{(by the path semantics)} \mathbf{p}, 0 \not\models_T \phi \\ \Leftrightarrow & \text{(by Lemma 1, assuming } \mathbf{p} \text{ is a } f\text{-path)} (\mathfrak{A}, \mathfrak{A}) \notin \llbracket \phi \rrbracket_T^f \\ \Leftrightarrow & \text{(by (6), (8) and the binary semantics of Unary Negation by recalling } \neg\phi := \neg\phi) \mathfrak{A} \models_T \neg\phi[\varepsilon/f]. \end{aligned}$$

Case $|\alpha\rangle\phi$:

$$\begin{aligned} & \mathbf{p}, 0 \models_T |\alpha\rangle\phi \\ \Leftrightarrow & \text{(by definition of the path semantics of } |\alpha\rangle\phi) \\ & \exists j \ (0 \leq j \leq \text{last}(\mathbf{p}) \text{ such that } \mathbf{p}(0, j) \in \mathcal{L}(\alpha) \text{ and } \mathbf{p}(j) \models_T \phi) \\ \Leftrightarrow & \text{(by Lemma 1, assuming } \mathbf{p} \text{ is an } f\text{-path)} \\ & \exists j \ ((\mathfrak{A}, \mathfrak{B}) \in \llbracket \alpha \rrbracket_T^f \text{ and } (\mathfrak{B}, \mathfrak{B}) \in \llbracket \phi \rrbracket_T^f, \text{ where } \mathbf{p}(j) = \mathfrak{B}) \\ \Leftrightarrow & \text{(by binary semantics of Composition)} \exists \mathfrak{B} \ ((\mathfrak{A}, \mathfrak{B}) \in \llbracket \alpha; \phi \rrbracket_T^f) \\ \Leftrightarrow & \text{(by binary semantics of Domain operation)} (\mathfrak{A}, \mathfrak{A}) \in \llbracket \text{Dom}(\alpha; \phi) \rrbracket_T^f \\ \Leftrightarrow & \text{(by binary semantics of } |\alpha\rangle\phi) \mathfrak{A} \models_T |\alpha\rangle\phi[\varepsilon/f] \end{aligned}$$

The cases for $|\alpha|$, $\langle\alpha|$ and $[\alpha|$ are similar. □

5 Examples

We now give two abstract examples that illustrate the logic. The semantic context, i.e., the interpretation of the atomic modules, is given as Output-Monadic Conjunctive Queries, represented here as non-recursive Datalog programs.

Problem: **EVEN** α_E

Given: A structure $\mathfrak{A}$ with an empty vocabulary.

Question: Is $|\text{dom}(\mathfrak{A})|$ even?

EVEN is PTIME computable, but is not expressible in MSO, Datalog, or any fixed point logic, *unless* a linear order on the domain elements is given. We construct a 2-coloured path in the transition system using E and O as labels. Due to **BG**, the elements in the path never repeat. Each trace gives us an implicit linear order on domain elements. For deterministic atomic modules, we omit the Epsilon operator.

$$\begin{aligned} \text{GuessP} &:= \varepsilon \{ P(x) \leftarrow \quad \}, \\ \text{CopyPO} &:= \{ O(x) \leftarrow P(x) \}, \\ \text{CopyPE} &:= \{ E(x) \leftarrow \underline{P(x)} \}. \end{aligned}$$

$$\begin{aligned} \text{GuessNewO} &:= (\text{GuessP}; \mathbf{BG}(\sigma_{P \neq E}^r(\sigma_{P \neq O}^r(\text{id})))) ; \text{CopyPO}, \\ \text{GuessNewE} &:= (\text{GuessP}; \mathbf{BG}(\sigma_{P \neq E}^r(\sigma_{P \neq O}^r(\text{id})))) ; \text{CopyPE}. \end{aligned}$$

The problem EVEN is now axiomatized as: $\alpha_E := (GuessNewO; GuessNewE)^\dagger; \curvearrowright GuessNewO$. The program is successfully executed if each chosen element is different from any elements selected 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. Given a structure $\mathfrak{A}$ over an empty vocabulary, $\mathfrak{A} \models_s |\alpha_E\rangle Last$, holds whenever there is a successful execution of α_E , that is, the size of the input domain is even.

Problem: s-t Connectivity

Given: Binary relation E , two constants s and t represented as singleton-set relations S and T .

Question: Is t reachable from s by following the edges?

We use the definable constructs of imperative programming.

$$\alpha(\underline{E}, \underline{S}, \underline{T}) := M_{base_case}; \\ \text{repeat } (M_{ind_case}; \mathbf{BG}(\sigma_{Reach' \neq Reach}(\text{id}))) ; \text{Copy until } \sigma_{Reach=T}^r(\text{id}).$$

On the input, we have a binary relation E , two constants s and t represented as singleton-set relations S and T . Inputs E , S and T are underlined. We use a unary relational variable $Reach$. Initially, the corresponding relation contains the same node as S . The execution is terminated when $Reach$ equals T . Variable $Reach'$ is used as a temporary storage. To avoid guessing the same element multiple times, we use the (definable) Back Globally (**BG**) modality. Atomic modules are axiomatized in non-recursive Datalog with monadic output predicates as follows:

$$\begin{aligned} M_{base_case} &:= \{ Reach(x) \leftarrow \underline{S}(x) \}, \\ M_{ind_case} &:= \varepsilon \{ Reach'(y) \leftarrow \underline{Reach}(x), \underline{E}(x, y) \}, \\ Copy &:= \{ Reach(x) \leftarrow \underline{Reach'}(x) \}. \end{aligned}$$

Here, module M_{ind_case} is the only non-deterministic module. For the other two modules, ε in the expression for $\alpha(\underline{E}, \underline{S}, \underline{T})$ is omitted because they are deterministic. Input symbols of each atomic module are underlined. Given structure $\mathfrak{A}$ over a vocabulary that matches the input variable vocabulary $I(\alpha) = \{E, S, T\}$, including matching the arities, of α , by checking $\mathfrak{A} \models_s |\alpha\rangle Last$, we verify that there is a successful execution of α . That is, t is reachable from s by following the edges of the input graph.

6 Conclusions

We have defined a Deterministic Dynamic Logic that can refer back in the executions. The logic is, simultaneously, an algebra of binary relations (here called LLIF) and a modal temporal logic (here, DetLDL_f) under a path semantics. The algebra of binary relations uses a *dynamic* version of Hilbert's Choice operator Epsilon that acts as a strategy function. It maps a history of the execution to the next state allowed by the algebraic expression. We have proved that the meaning of Epsilon in the algebra of binary relations is naturally captured by the path semantics of the Dynamic logic.

While the logic is rather minimal, it defines the main constructs of Imperative programming. Thus, reasoning about such programs avoids the non-determinism of regular expressions. Unlike LDL_f [4], the logic is not propositional. This is due to the use of equality represented here by means of Selection. Similarly to the agent control language ElGolog [8], our process descriptions are non-Markovian. On the other hand, our operations on processes do not introduce non-determinism, and we use arbitrary modules in place of deterministic Successor State axioms.

As a future direction, we plan to develop an automata-theoretic counterpart of DetLDL_f . To do that, with each formula α in (7), we will associate an alternating automaton on words (AFW) $\mathcal{A}_\alpha$ accepting finite words of the form (1). A standard approach is then to transform the AFW to a non-deterministic finite automaton (NFA). However, since we restricted all programs (algebraic expressions) to be deterministic, the NFA is, in fact, a deterministic finite automaton (DFA). An automaton for the full logic (7), needs to “look inside the alphabet” of τ -structures $\mathbf{U}$ to account for Selection, and to “look backwards in time” to account for the backwards-facing Unary Negation (and thus, the backwards-facing modalities) and Converse. Note that the “forward” fragment of (7), i.e., without $\sim$ and $\curvearrowright$, and where Selection is interpreted as an atomic equality test action, is *less expressive than regular languages* (recall that every operation of regular expressions was replaced by its deterministic restriction).

Another future direction is to study the decidability of the semantic input and output problem for compound expressions of LLIF, along the lines of [2]. Since, unlike the full LIF, LLIF no longer embeds first-order logic, we believe that the problem may be decidable.

It is well-known that the temporal logics LDL_f and LTL_f have classical logic counterparts. LDL_f corresponds to MSO *over strings* and regular languages, and LTL_f corresponds to FO, also over strings, and star-free regular languages [4]. We restricted classical connectives $\vee$ and $\neg$ to their deterministic counterparts, Preferential Union and Unary Negations, respectively. Thus, for DetLDL_f , no classical logic counterpart can exist. In fact, Unary Negation has properties of Intuitionistic Negation. It is not idempotent, $\curvearrowright\curvearrowright\alpha \neq \alpha$, but is weakly idempotent, $\curvearrowright\curvearrowright\curvearrowright\alpha = \curvearrowright\alpha$. Moreover, $[\alpha]\phi$ seems to represent intuitionistic implication $\alpha \rightarrow \phi$. It would be interesting to understand further connections to intuitionistic and substructural logics, and to develop a proof system for LLIF.

References

- [1] H. Aamer, B. Bogaerts, D. Surinx, E. Ternovska & J. Van den Bussche (2020): *Executable first-order queries in the logic of information flows*. In: *Proceedings 23rd International Conference on Database Theory, Leibniz International Proceedings in Informatics 155*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 4:1–4:14.
- [2] Heba Aamer, Bart Bogaerts, Dimitri Surinx, Eugenia Ternovska & Jan Van den Bussche (2020): *Inputs, outputs, and composition in the logic of information flows*. In: *KR’2020*.
- [3] Andreas Blass & Yuri Gurevich (2000): *The Logic of Choice*. *J. Symb. Log.* 65(3), pp. 1264–1310.
- [4] Giuseppe De Giacomo & Moshe Y. Vardi (2013): *Linear Temporal Logic and Linear Dynamic Logic on Finite Traces*. In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pp. 854–860.
- [5] Herbert B. Enderton (1972): *A mathematical introduction to logic*. Academic Press.
- [6] Michael J. Fischer & Richard E. Ladner (1979): *Propositional Dynamic Logic of Regular Programs*. *J. Comput. Syst. Sci.* 18(2), pp. 194–211.
- [7] G. Fletcher, M. Gyssens, D. Leinders, D. Surinx, J. Van den Bussche, D. Van Gucht, S. Vansummen & Y. Wu (2015): *Relative expressive power of navigational querying on graphs*. *Information Sciences* 298, pp. 390–406.
- [8] Giuseppe De Giacomo, Yves Lespérance & Eugenia Ternovska (2020): *ElGolog: A High-Level Programming Language with Memory of the Execution History*. In: *AAAI, AAAI Press*, pp. 2806–2813.
- [9] Y. Gurevich (1983): *Algebras of feasible functions*. In: *Proceedings 24th Symposium on Foundations of Computer Science*, IEEE Computer Society, pp. 210–214.
- [10] Y. Gurevich (1988): *Logic and the challenge of computer science*. In E. Börger, editor: *Current Trends in Theoretical Computer Science*, Computer Science Press, pp. 1–57.

- [11] David Harel, Dexter Kozen & Jerzy Tiuryn (2000): *Dynamic Logic (Foundations of Computing)*.
- [12] David Hilbert & Paul Bernays (1939): *Grundlagen der Mathematik*. 2, Springer Verlag.
- [13] Marcel Jackson & Timothy Stokes (2011): *Modal restriction Semigroups: towards an Algebra of Functions*. *IJAC* 21(7), pp. 1053–1095.
- [14] B. Jónsson & A. Tarski (1952): *Representation problems for relation algebras*. *Bull. Amer. Math. Soc.* 74, pp. 127–162.
- [15] Antonina Kolokolova, Yongmei Liu, David Mitchell & Eugenia Ternovska (2010): *On the Complexity of Model Expansion*. In: *Proc., 17th Int’l Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR-17)*, Springer, pp. 447–458. LNCS 6397.
- [16] Vladimir Lifschitz (1987): *Formal Theories of Action (Preliminary Report)*. In: *Proceedings of the 10th International Joint Conference on Artificial Intelligence*. Milan, Italy, August 23–28, 1987, pp. 966–972.
- [17] John McCarthy & Patrick J. Hayes (1969): *Some Philosophical Problems from the Standpoint of Artificial Intelligence*. In B. Meltzer & D. Michie, editors: *Machine Intelligence 4*, Edinburgh University Press, pp. 463–502.
- [18] Brett McLean (2017): *Complete representation by partial functions for composition, intersection and anti-domain*. *J. Log. Comput.* 27(4), pp. 1143–1156.
- [19] D. G. Mitchell & E. Ternovska (2005): *A framework for representing and solving NP search problems*. In: *Proc. AAAI*, pp. 430–435.
- [20] De Morgan (1860): *On the syllogism, no. IV, and on the logic of relations*. *Trans. Cambridge Phil. Soc.* 10, pp. 331–358.
- [21] Martin Otto (2000): *Epsilon-Logic Is More Expressive Than First-Order Logic Over Finite Structures*. *J. Symb. Log.* 65(4), pp. 1749–1757.
- [22] Vaughan R. Pratt (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, pp. 248–254.
- [23] Dimitri Surinx, Jan Van den Bussche & Dirk Van Gucht (2017): *The primitivity of operators in the algebra of binary relations under conjunctions of containments*. In: *LICS ’17*.
- [24] E. Ternovska (2019): *An Algebra of Modular Systems: Static and Dynamic Perspectives*. In: *Proceedings of the 12th International Symposium on Frontiers of Combining Systems (FroCoS)*.
- [25] Eugenia Ternovska (2017): *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, June 7–9, 2017.