Extending Dynamic Logics with First-Class Relational Reasoning

***Abstract*—Many important properties are relational properties, which are often difficult to express and verify. Dynamic logics are well-known formalisms for program verification. We present a general extension, called the REL extension, of dynamic logics to support first-class relational reasoning. The extension provides intuitive syntax to express relational properties, which may be difficult or impossible to express in the host dynamic logic. The REL extension can be instantiated for different host logics and may or may not add expressive power over the host logic. Verification of relational properties expressed by the REL extension benefits from existing tools developed for the host logic.**

We validate the applicability of the REL extension by instantiating it for two well-known and distinct dynamic logics: differential dynamic logic ($\mathsf{d}\mathcal{L}$) and linear dynamic logic on finite traces (LDL_f). As a result, both instantiations can express important relational properties that cannot be easily expressed with the host logics. We investigate the expressive power of both instantiations and prove that the one for $\mathsf{d}\mathcal{L}$ is equally expressive as $\mathsf{d}\mathcal{L}$ and the one for LDL_f is more expressive than LDL_f. Moreover, we introduce two kinds of encodings of relational properties expressible by the instantiations, and leverage existing verification tools to verify these properties.

I. Introduction

Many important properties, such as noninterference [1] and robustness [2], are relational properties [3], i.e., properties over sets of pairs of executions. They are fundamental properties but often harder to express and reason about than safety properties, as they require reasoning simultaneously about multiple executions. Moreover, there is less tool support for verification of relational properties, compared to safety properties.

Dynamic logics are multi-modal logics widely used for verifying safety properties of imperative programs [4], [5], [6], [7]. They are specified over a set of programs α and a set of formulas ϕ. Program specifications are stated with modality of necessity $[\alpha]\phi$ that reads "after any execution of α, ϕ is true" and modality of existence $\langle\alpha\rangle\phi$ that reads "after some execution of α, ϕ is true". Relational properties have been explored in the setting of dynamic logic using trace modalities [8], [9], [10], [11]. However, these works consider relational properties in deterministic programs and their proof obligations are often cumbersome. The limitation to deterministic programs is significant because nondeterminism is essential to the utility of dynamic logics, especially in AI-enabled systems [12]. Proof automation is also difficult for such heavyweight approaches.

A lightweight yet efficient approach for expressing relational properties is extending the host logic with a *biprogram* construct [13], [14], which explicitly specifies a pair of programs and thus allows reasoning about executions of the program pair. We argue that extending dynamic logics with a biprogram provides intuitive and effective support for expressing relational properties. Elements from both programs can be directly used in expressing relational properties. Modalities can be designed to quantify over executions of the program pair. As such, a dynamic logic can be cleanly extended with first-class support for expressing relational properties.

We contribute a general extension of dynamic logics, called the REL extension, that builds upon the biprogram construct to support first-class relational reasoning for different dynamic logics. A REL extension has three major components[1]:

- REL programs that specify pairs of programs. A key construct is *biprogram* (α, β), which specifies a *left* program (i.e., α) and a *right* program (i.e., β), of the host logic.
- REL modalities that express executions of program pairs. They play the same role as modalities in dynamic logics. They can naturally express different quantifications over executions of program pairs. For example, REL modality of necessity $[\![(\alpha, \beta)]\!]\phi_{\mathtt{t}}$ expresses that *for all* executions of α and β, $\phi_{\mathtt{t}}$ holds at the last states of the two executions.
- REL formulas that directly capture relational properties. For example, the following formula
$$(\lfloor x \rfloor_{\mathtt{L}} < \lfloor x \rfloor_{\mathtt{R}}) \rightarrow [\![(\alpha, \beta)]\!](\lfloor x \rfloor_{\mathtt{L}} < \lfloor x \rfloor_{\mathtt{R}})$$
specifies that any pair of executions of α and β preserve the natural order on real-valued variable x, i.e., if program β starts its execution with a larger value of x than that of program α, then its execution would end with a larger value of variable x. (The subscripts $\mathtt{L}$ and $\mathtt{R}$ of variable x respectively refer to its value in the left and right executions.)

We design the REL extension in a general and abstract manner, by focusing on the core constructs, i.e., programs and formulas, of all dynamic logics. It can be instantiated for different dynamic logics to support diverse program constructs. To validate its applicability, we conduct two case studies by instantiating the REL extension for two well-known dynamic logics designed for different application domains: (1) differential dynamic logic ($\mathsf{d}\mathcal{L}$) [6], [15], a logic for verifying safety properties in cyber-physical systems, and (2) linear dynamic logic on finite traces [16] (LDL_f), a logic often used for reasoning about temporal constraints by the AI community. The two dynamic logics share the core constructs, but differ significantly in other constructs and semantic interpretation. Together they cover most features of mainstream dynamic logics. The case studies indicate that instantiations can express

[1] *Color scheme:* we use blue color for syntactic constructs of host dynamic logics, and red color for those of the REL extension.

important relational properties that cannot be easily expressed by the host logics.

A REL instantiation may or may not add expressive power over the host logic. For example, we analyze the expressive power of the REL instantiation for $\mathsf{d}\mathcal{L}$ and LDL_f. We prove that the instantiation for $\mathsf{d}\mathcal{L}$ is equally expressive as $\mathsf{d}\mathcal{L}$, but the instantiation for LDL_f is more expressive than LDL_f.

Verification of REL formulas specified by a REL instantiation benefits from existing verification tools. We introduce two kinds of encodings of the REL formulas that permit reusing existing tools to verify these formulas. The first encoding is inspired by the technique of self-composition [17], which reduces relational verification of a program to standard verification of a composition of two copies of the program. We develop a sound and complete encoding for the instantiation for $\mathsf{d}\mathcal{L}$, and a sound but incomplete encoding for the instantiation for LDL_f. Existing tools developed for $\mathsf{d}\mathcal{L}$ and LDL_f can be used to verify REL formulas using this encoding. The second encoding captures REL formulas as first-order constraints, and thus reduces the verification problem into a satisfiability problem. We demonstrate with a sound and complete encoding for the REL extension of LDL_f, and use the Z3 solvers to verify properties specified by the extension.

Contribution. The key contribution of this paper is the design and validation of a general extension for different dynamic logics to support lightweight yet effective first-class relational reasoning. In particular, we make the following contributions:

- The design of a general and lightweight extension for dynamic logics to support first-class relational reasoning. We introduce the design of the REL extension in the setting of propositional dynamic logic (PDL) [7], which contains the fundamental elements of most dynamic logics. (Section III)
- We explore the applicability of the REL extension by instantiating it for two mainstream dynamic logics: (1) $\mathsf{d}\mathcal{L}$, a first-order dynamic logic that adds real-valued arithmetic, assignments, and differential equations over PDL, and (2) LDL_f, a temporal style dynamic logic whose semantics is interpreted on finite traces rather than the common state transitions. We develop two REL instantiations, $\mathsf{d}\mathcal{L}_{\text{REL}}$ and LDL_{REL}, respectively, for $\mathsf{d}\mathcal{L}$ and LDL_f, and use them to express an important relational property that cannot be directly expressed with the two host logics. We investigate the expressive power of both instantiations, and prove that $\mathsf{d}\mathcal{L}_{\text{REL}}$ is equal expressive as $\mathsf{d}\mathcal{L}$, and LDL_{REL} is more expressive than LDL_f. (Section IV and V)
- We develop two encodings of REL formulas that allow reusing existing verification tools to verify these formulas. The first encoding reduces the verification of REL formulas into the verification of formulas of the host dynamic logics. The second encoding captures the semantics of REL formulas as first-order constraints, so we can use existing constraint solvers to verify the formulas. (Section VI)

Section II introduces the syntax and semantics of PDL. Section VII discusses related work and Section VIII concludes.

$$\begin{aligned} \text{Program: } & \alpha, \beta \ ::= P \mid ?\phi \mid \alpha\,;\beta \mid \alpha \cup \beta \mid \alpha^* \\ \text{Formula: } & \phi, \psi \ ::= \top \mid A \mid \neg\phi \mid \phi \wedge \psi \mid [\alpha]\phi \end{aligned}$$

Fig. 1: Syntax of PDL

Formula semantics

$$\begin{aligned} \omega \models \top &\text{ iff } \omega \in \mathcal{W} \\ \omega \models A &\text{ iff } \omega \in \mathcal{V}(A) \\ \omega \models \neg\phi &\text{ iff } \omega \not\models \phi \\ \omega \models \phi \wedge \psi &\text{ iff } \omega \models \phi \text{ and } \omega \models \psi \\ \omega \models [\alpha]\phi &\text{ iff } \nu \models \phi \text{ for all state } \nu \text{ with } (\omega,\nu) \in [\![\alpha]\!] \end{aligned}$$

Program semantics

$$\begin{aligned} [\![P]\!] &= \mathcal{R}(P) \\ [\![?\phi]\!] &= \{(\omega,\omega) \mid \omega \models \phi\} \\ [\![\alpha\,;\beta]\!] &= \{(\omega,\nu) \mid \exists\,\mu, (\omega,\mu) \in [\![\alpha]\!] \text{ and } (\mu,\nu) \in [\![\beta]\!]\} \\ [\![\alpha \cup \beta]\!] &= [\![\alpha]\!] \cup [\![\beta]\!] \\ [\![\alpha^*]\!] &= [\![\alpha]\!]^* \text{ the transitive, reflexive closure of } [\![\alpha]\!] \end{aligned}$$

Fig. 2: Semantics of PDL

II. Propositional Dynamic Logic

Propositional dynamic logic (PDL) is a subsystem of most, if not all, dynamic logics. It extends propositional logic with modalities to reason with properties of programs. The language of regular PDL has expressions of two sorts: (1) propositions or formulas and (2) programs. Its syntax is defined upon Π, a set of atomic programs, and Φ, a set of atomic propositions. Programs and propositions are mutually inductively defined from the atomic ones as shown in Figure 1. Programs include the operations of Kleene algebra with tests [18]: sequential composition, nondeterministic choice, nondeterministic repetition, and test of a formula. Formulas include the standard propositional connectives, program necessity $[\alpha]\phi$, and program existence $\langle\alpha\rangle\phi$. Program existence can be used to encode program necessity and vice versa, e.g., $[\alpha]\phi = \neg\langle\alpha\rangle\neg\phi$. Common abbreviations for logical connectives apply, e.g., $\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi)$.

The semantics of PDL formulas and programs is interpreted over a Kriple structure $(\mathcal{W}, \mathcal{R}, \mathcal{V})$, where $\mathcal{W}$ is a nonempty set of states, $\mathcal{R}$ is a mapping from the set Π of atomic programs into binary relations on $\mathcal{W}$, and $\mathcal{V}$ is a mapping from the set Φ of atomic propositions into subsets of $\mathcal{W}$. That is, $\mathcal{R}(P) \subseteq \mathcal{W} \times \mathcal{W}$ for $P \in \Pi$, and $\mathcal{V}(A) \subseteq \mathcal{W}$ for $A \in \Phi$. $\mathcal{R}$ and $\mathcal{V}$ are extended inductively to give meanings to all programs and formulas of PDL as shown in Figure 2. We write $\omega \models \phi$ if formula ϕ is true at state ω, i.e., $\omega \in \mathcal{V}(\phi)$. We write $[\![\alpha]\!]$ to denote the semantics of α, i.e., if $(\omega,\nu) \in [\![\alpha]\!]$, then there is an execution of α that starts in state ω and ends in state ν.

Relations and states in PDL are *abstract*, i.e., states in PDL are abstract points and atomic programs in PDL are abstract binary relations. Such a level of abstraction lets us focus on the fundamental design of the REL extension, which we introduce next. These abstract notions will be later instantiated with concrete atomic constructs in the case studies.

III. The REL Extension

This section introduces the design of the REL extension in the setting of PDL. We present its syntax and semantics, and then show how to use it to express various relational properties.

$$\alpha_{\mathfrak{r}},\ \beta_{\mathfrak{r}} \ ::= P_{\mathfrak{r}} \mid ?\phi_{\mathfrak{r}} \mid \alpha_{\mathfrak{r}}\,;\ \beta_{\mathfrak{r}} \mid \alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}} \mid \alpha_{\mathfrak{r}}^{*} \mid (\alpha, \beta)$$
$$\phi_{\mathfrak{r}},\ \psi_{\mathfrak{r}} \ ::= A_{\mathfrak{r}} \mid \neg\phi_{\mathfrak{r}} \mid \phi_{\mathfrak{r}} \wedge \psi_{\mathfrak{r}} \mid [\![\, \alpha_{\mathfrak{r}} \,]\!]\phi_{\mathfrak{r}} \mid \lfloor \phi \rfloor_{\mathtt{B}} \quad (\mathtt{B} \in \{\mathtt{L}, \mathtt{R}\})$$

(L and R, respectively, denotes the left and right states)

Fig. 3: Syntax of the REL extension

A. Syntax and Semantics

The REL extension extends PDL with programs, modalities, and formulas. Figure 3 shows its syntax. It builds on $\Pi_{\mathfrak{r}}$, a set of REL atomic programs, and $\Phi_{\mathfrak{r}}$, a set of REL atomic propositions. REL programs are analogous to PDL: sequential composition, nondeterministic choice, nondeterministic repetition, and test of formula, with the addition of a biprogram construct (α, β), where α and β specify two PDL programs to run, respectively, by the *left* and *right* execution. The syntax of REL formulas is also analogous to the syntax of PDL formulas, with the addition of projection formulas $\lfloor \phi \rfloor_{\mathtt{B}}$, which refers to a PDL formula ϕ in one of the two executions specified by $\mathtt{B} \in \{\mathtt{L}, \mathtt{R}\}$. REL modality of existence can be used to encode REL modality of necessity, i.e., $[\![\, \alpha_{\mathfrak{r}} \,]\!]\phi_{\mathfrak{r}} = \neg\langle\!\langle \alpha_{\mathfrak{r}} \rangle\!\rangle \neg\phi_{\mathfrak{r}}$.

The semantics of REL programs and formulas is interpreted over the semantics of REL atomic programs $\mathcal{R}_{\mathfrak{r}}(P_{\mathfrak{r}})$ (for $P_{\mathfrak{r}} \in \Pi_{\mathfrak{r}}$) and REL atomic propositions $\mathcal{V}_{\mathfrak{r}}(A_{\mathfrak{r}})$ (for $A_{\mathfrak{r}} \in \Phi_{\mathfrak{r}}$), as well as the semantics of PDL programs and formulas. $\mathcal{R}_{\mathfrak{r}}(P_{\mathfrak{r}})$ is a transition relation between two pairs of states: if $((\omega_{\mathtt{L}}, \omega_{\mathtt{R}}), (\nu_{\mathtt{L}}, \nu_{\mathtt{R}})) \in \mathcal{R}_{\mathfrak{r}}(P_{\mathfrak{r}})$, then an execution of the left program specified in $P_{\mathfrak{r}}$ runs from $\omega_{\mathtt{L}}$ to $\nu_{\mathtt{L}}$, and an execution of the right program in $P_{\mathfrak{r}}$ runs from $\omega_{\mathtt{R}}$ to $\nu_{\mathtt{R}}$. $\mathcal{V}_{\mathfrak{r}}(A_{\mathfrak{r}})$ is a relation on states, i.e., $\mathcal{V}_{\mathfrak{r}}(A_{\mathfrak{r}}) \subseteq \mathcal{W} \times \mathcal{W}$ for $A_{\mathfrak{r}} \in \Phi_{\mathfrak{r}}$. Semantics of all REL programs and formulas, as shown in Figure 4, is defined inductively from the semantics of REL atomic programs and formulas, and the projection formulas. A projection formula $\lfloor \phi \rfloor_{\mathtt{B}}$ tests if PDL formula ϕ holds in the state specified by B (left or right). REL modality of existence $\langle\!\langle \alpha_{\mathfrak{r}} \rangle\!\rangle \phi_{\mathfrak{r}}$ holds if and only if formula $\phi_{\mathfrak{r}}$ holds in some pairs of states that are reachable from $(\omega_{\mathtt{L}}, \omega_{\mathtt{R}})$ by running $\alpha_{\mathfrak{r}}$. We write $[\![\alpha_{\mathfrak{r}}]\!]_{\mathrm{RL}}$ to denote the semantics of $\alpha_{\mathfrak{r}}$, and $(\omega_{\mathtt{L}}, \omega_{\mathtt{R}}) \models_{\mathrm{RL}} \phi_{\mathfrak{r}}$ if the formula $\phi_{\mathfrak{r}}$ holds in bi-state $\omega_{\mathtt{L}}, \omega_{\mathtt{R}}$.

B. Quantification over Executions

Expressing relational properties often involves quantifications over executions, e.g., for all executions of a program α, there exists an execution of β that gets the same results. REL modalities can express quantifications over two executions as shown below. We write σ_α (or σ_β) to denote an execution of program α (or β) and σ_α[-1] to denote the last state of σ_α.

$$\forall \sigma_\alpha \forall \sigma_\beta.((\sigma_\alpha[\text{-}1], \sigma_\beta[\text{-}1]) \models_{\mathrm{RL}} \phi_{\mathfrak{r}}) \equiv [\![\, (\alpha, \beta) \,]\!]\phi_{\mathfrak{r}}$$
$$\exists \sigma_\alpha \exists \sigma_\beta.((\sigma_\alpha[\text{-}1], \sigma_\beta[\text{-}1]) \models_{\mathrm{RL}} \phi_{\mathfrak{r}}) \equiv \langle\!\langle (\alpha, \beta) \rangle\!\rangle \phi_{\mathfrak{r}}$$
$$\forall \sigma_\alpha \exists \sigma_\beta.((\sigma_\alpha[\text{-}1], \sigma_\beta[\text{-}1]) \models_{\mathrm{RL}} \phi_{\mathfrak{r}}) \equiv [\![\, (\alpha, ?\top) \,]\!]\langle\!\langle (?\top, \beta) \rangle\!\rangle \phi_{\mathfrak{r}}$$
$$\forall \sigma_\beta \exists \sigma_\alpha.((\sigma_\alpha[\text{-}1], \sigma_\beta[\text{-}1]) \models_{\mathrm{RL}} \phi_{\mathfrak{r}}) \equiv [\![\, (?\top, \beta) \,]\!]\langle\!\langle (\alpha, ?\top) \rangle\!\rangle \phi_{\mathfrak{r}}$$
$$\exists \sigma_\alpha \forall \sigma_\beta.((\sigma_\alpha[\text{-}1], \sigma_\beta[\text{-}1]) \models_{\mathrm{RL}} \phi_{\mathfrak{r}}) \equiv \langle\!\langle (\alpha, ?\top) \rangle\!\rangle [\![\, (?\top, \beta) \,]\!] \phi_{\mathfrak{r}}$$
$$\exists \sigma_\beta \forall \sigma_\alpha.((\sigma_\alpha[\text{-}1], \sigma_\beta[\text{-}1]) \models_{\mathrm{RL}} \phi_{\mathfrak{r}}) \equiv \langle\!\langle (?\top, \beta) \rangle\!\rangle [\![\, (\alpha, ?\top) \,]\!] \phi_{\mathfrak{r}}$$

By combining the biprogram construct and a special program $?\top$, we are able to express various forms of quantification. For example, the *for all, exists* scenario, i.e., $\forall \sigma_\alpha\ \exists \sigma_\beta$ is encoded by $[\![\, (\alpha, ?\top) \,]\!]\langle\!\langle (?\top, \beta) \rangle\!\rangle$, where modality $[\![\, (\alpha, ?\top) \,]\!]$ quantifies over executions of program α, and modality $\langle\!\langle (?\top, \beta) \rangle\!\rangle$ quantifies over executions of program β.

C. Expressing Relational Properties

The REL extension can express relational properties in an intuitive manner. We show several examples here. Note that all properties we can express with PDL can be specified the same way for other dynamic logics.

Refinement Relation. The first example is expressing the refinement relation between two programs. We say that program β refines α if behaviors of α subsume those of β. That is, all states reachable from a state ω by following a transition of β could also be reached from ω by following *some* transitions of α. Such a refinement relation can be encoded by the $\forall \sigma_\beta, \exists \sigma_\alpha$ quantification over executions, that is:

$$(\lfloor \vec{A} \rfloor_{\mathtt{L}} \leftrightarrow \lfloor \vec{A} \rfloor_{\mathtt{R}}) \rightarrow [\![\, (?\top, \beta) \,]\!]\langle\!\langle (\alpha, ?\top) \rangle\!\rangle (\lfloor \vec{A} \rfloor_{\mathtt{L}} \leftrightarrow \lfloor \vec{A} \rfloor_{\mathtt{R}})$$

Where programs α and β refer to the same set of atomic propositions and programs. Formula $(\lfloor \vec{A} \rfloor_{\mathtt{L}} \leftrightarrow \lfloor \vec{A} \rfloor_{\mathtt{R}})$ is a shorthand for $\bigwedge_{i \in 1 \ldots n} (\lfloor A_i \rfloor_{\mathtt{L}} \leftrightarrow \lfloor A_i \rfloor_{\mathtt{R}})$, which encodes that the left and right states are identical. A_i are atomic propositions appear in α and β. The following formula shows an example:

$$(\lfloor A \rfloor_{\mathtt{L}} \leftrightarrow \lfloor A \rfloor_{\mathtt{R}}) \rightarrow [\![\, (?\top, ?A) \,]\!]\langle\!\langle ((?A \cup ?\neg A), ?\top) \rangle\!\rangle (\lfloor A \rfloor_{\mathtt{L}} \leftrightarrow \lfloor A \rfloor_{\mathtt{R}})$$

Here A is an atomic proposition. This formula says program $?A$ refines program $?A \cup ?\neg A$, which indeed holds because the latter program has all possible behaviors of the former one.

Noninterference. The second example concerns relational reasoning on executions of a single program: secure information flow. Noninterference [1], [19] is a well-known strong information security property that, intuitively, guarantees that low-security outputs of a system do not reveal any high-security information (i.e., confidentiality), or dually that low-security inputs of a system do not modify high-security contents (i.e., integrity).

Noninteference has various forms, especially in a language that involves nondeterminism. Here, we express a few variants of noninterference for a PDL program α.

First, a common notion of noninterference for confidentiality of a *deterministic* program ensures that attackers who have access to the program's low-security input and output won't be able to infer the program's high-security input. Intuitively, the property states that if a deterministic program α receives the same low-security inputs, then it should produce the same values for low-security outputs. Such a noninterference notion can be expressed with the following REL formula:

$$(\lfloor A_i \rfloor_{\mathtt{L}} \leftrightarrow \lfloor A_i \rfloor_{\mathtt{R}}) \rightarrow [\![\, (\alpha, \alpha) \,]\!](\lfloor A_o \rfloor_{\mathtt{L}} \leftrightarrow \lfloor A_o \rfloor_{\mathtt{R}})$$

Where atomic proposition A_i and A_o represent, respectively, low-security input and output of α. Modality $[\![\, (\alpha, \alpha) \,]\!]$ ensures that α produces the same low-security output if the same low-security input is received.

Various forms of noninterference can be defined when α has nondeterminism. One of them is *possibilistic noninterference* [1] or *nondeducibility* [20], which states that an attacker cannot infer confidential information *for certain* in a program with non-observable nondeterminism. Intuitively, the attacker

Semantics of REL program

$$
\begin{aligned}
[\![P_{\mathfrak{r}}]\!]_{\mathrm{RL}} &= \mathcal{R}_{\mathfrak{r}}(P_{\mathfrak{r}}) \\
[\![?\phi_{\mathfrak{r}}]\!]_{\mathrm{RL}} &= \{((\omega_{\mathrm{L}},\omega_{\mathrm{R}}),(\omega_{\mathrm{L}},\omega_{\mathrm{R}})) \mid (\omega_{\mathrm{L}},\omega_{\mathrm{R}}) \models_{\mathrm{RL}} \phi_{\mathfrak{r}}\} \\
[\![\alpha_{\mathfrak{r}}\,;\,\beta_{\mathfrak{r}}]\!]_{\mathrm{RL}} &= \{((\omega_{\mathrm{L}},\omega_{\mathrm{R}}),(\nu_{\mathrm{L}},\nu_{\mathrm{R}})) \mid \text{exists } \mu_{\mathrm{L}},\ \mu_{\mathrm{R}} \text{ that } ((\omega_{\mathrm{L}},\omega_{\mathrm{R}}),(\mu_{\mathrm{L}},\mu_{\mathrm{R}})) \in [\![\alpha_{\mathfrak{r}}]\!]_{\mathrm{RL}} \text{ and } ((\mu_{\mathrm{L}},\mu_{\mathrm{R}}),(\nu_{\mathrm{L}},\nu_{\mathrm{R}})) \in [\![\beta_{\mathfrak{r}}]\!]_{\mathrm{RL}}\} \\
[\![\alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}}]\!]_{\mathrm{RL}} &= \{((\omega_{\mathrm{L}},\omega_{\mathrm{R}}),(\nu_{\mathrm{L}},\nu_{\mathrm{R}})) \mid ((\omega_{\mathrm{L}},\omega_{\mathrm{R}}),(\nu_{\mathrm{L}},\nu_{\mathrm{R}})) \in [\![\alpha_{\mathfrak{r}}]\!]_{\mathrm{RL}} \text{ or } ((\omega_{\mathrm{L}},\omega_{\mathrm{R}}),(\nu_{\mathrm{L}},\nu_{\mathrm{R}})) \in [\![\beta_{\mathfrak{r}}]\!]_{\mathrm{RL}}\} \\
[\![\,\alpha_{\mathfrak{r}}^{*}\,]\!]_{\mathrm{RL}} &= \textstyle\bigcup_{n\in\mathbb{N}}([\![\,\alpha_{\mathfrak{r}}^{n}\,]\!]_{\mathrm{RL}}), \text{ where } \alpha_{\mathfrak{r}}^{0} \text{ is } (?\top,?\top),\ \alpha_{\mathfrak{r}}^{1} \text{ is defined as } \alpha_{\mathfrak{r}}, \text{ and } \alpha_{\mathfrak{r}}^{n+1} \text{ is defined as } \alpha_{\mathfrak{r}}^{n}\,;\,\alpha_{\mathfrak{r}} \text{ for } n \geq 1 \\
[\![(\alpha,\beta)]\!]_{\mathrm{RL}} &= \{((\omega_{\mathrm{L}},\omega_{\mathrm{R}}),(\nu_{\mathrm{L}},\nu_{\mathrm{R}})) \mid (\omega_{\mathrm{L}},\nu_{\mathrm{L}}) \in [\![\alpha]\!] \text{ and } (\omega_{\mathrm{R}},\nu_{\mathrm{R}}) \in [\![\beta]\!]\ \}
\end{aligned}
$$

Semantics of REL formula

Truth of formula $\phi_{\mathfrak{r}}$ in bi-state $\omega_{\mathrm{L}},\omega_{\mathrm{R}}$, denoted $(\omega_{\mathrm{L}},\omega_{\mathrm{R}}) \models_{\mathrm{RL}} \phi_{\mathfrak{r}}$ is defined inductively as follows:

$$
\begin{aligned}
(\omega_{\mathrm{L}},\omega_{\mathrm{R}}) &\models_{\mathrm{RL}} A_{\mathfrak{r}} \text{ iff } (\omega_{\mathrm{L}},\omega_{\mathrm{R}}) \in \mathcal{V}_{\mathfrak{r}}(A_{\mathfrak{r}}) \\
(\omega_{\mathrm{L}},\omega_{\mathrm{R}}) &\models_{\mathrm{RL}} \neg\phi_{\mathfrak{r}} \text{ iff } (\omega_{\mathrm{L}},\omega_{\mathrm{R}}) \not\models_{\mathrm{RL}} \phi_{\mathfrak{r}} \\
(\omega_{\mathrm{L}},\omega_{\mathrm{R}}) &\models_{\mathrm{RL}} \phi_{\mathfrak{r}} \wedge \psi_{\mathfrak{r}} \text{ iff } (\omega_{\mathrm{L}},\omega_{\mathrm{R}}) \models_{\mathrm{RL}} \phi_{\mathfrak{r}} \text{ and } (\omega_{\mathrm{L}},\omega_{\mathrm{R}}) \models_{\mathrm{RL}} \psi_{\mathfrak{r}} \\
(\omega_{\mathrm{L}},\omega_{\mathrm{R}}) &\models_{\mathrm{RL}} [\![\,\alpha_{\mathfrak{r}}\,]\!]\phi_{\mathfrak{r}} \text{ iff for all } \nu_{\mathrm{L}},\ \nu_{\mathrm{R}} \text{ that } ((\omega_{\mathrm{L}},\omega_{\mathrm{R}}),(\nu_{\mathrm{L}},\nu_{\mathrm{R}})) \in [\![\alpha_{\mathfrak{r}}]\!]_{\mathrm{RL}},\ (\nu_{\mathrm{L}},\nu_{\mathrm{R}}) \models_{\mathrm{RL}} \phi_{\mathfrak{r}} \text{ holds} \\
(\omega_{\mathrm{L}},\omega_{\mathrm{R}}) &\models_{\mathrm{RL}} \lfloor\phi\rfloor_{\mathtt{B}} \text{ iff } \omega_{\mathrm{L}} \models \phi \text{ when } \mathtt{B} = \mathtt{L} \text{ or } \omega_{\mathrm{R}} \models \phi \text{ when } \mathtt{B} = \mathtt{R}
\end{aligned}
$$

Fig. 4: Semantics of REL programs and formulas

cannot observe the nondeterministic choices made by program α, so the attacker is not certain if the information received is confidential. Such a noninteference definition can be expressed with the *forall, exists* quantifications over executions of α, i.e.,

$$(\lfloor A_i\rfloor_{\mathrm{L}} \leftrightarrow \lfloor A_i\rfloor_{\mathrm{R}}) \rightarrow [\![\,(\alpha,?\top)\,]\!]\langle\!\langle(?\top,\alpha)\rangle\!\rangle(\lfloor A_o\rfloor_{\mathrm{L}} \leftrightarrow \lfloor A_o\rfloor_{\mathrm{R}})$$

To properly reason about two executions in the presence of nondeterminism, some variants of noninterference force (some of) the nondeterminism in the two executions to resolve in the same way [2]. For example, a nondeterministic choice in a program may represent a user's decision, which is assumed to be public input, and so the resolution of the nondeterministic choice should be the same in both executions. Under such a circumstance, we can use the REL nondeterministic choice, i.e., $\alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}}$, to express that two executions take the same choice. The following formula shows an example of such a nondeterminism-aware noninterference:

$$(\lfloor A_i\rfloor_{\mathrm{L}} \leftrightarrow \lfloor A_i\rfloor_{\mathrm{R}}) \rightarrow [\![\,((?A_i\,;\,?A_o,?A_i\,;\,?A_o)) \cup ((?\neg A_o,?\neg A_o))\,]\!]\lfloor A_o\rfloor_{\mathrm{L}} \leftrightarrow \lfloor A_o\rfloor_{\mathrm{R}}$$

This formula indeed holds: two executions with the same nondeterministic choice, i.e., either the first or second branch, would lead to the same value of A.

Lock-step Properties. Note that the semantics of $\alpha_{\mathfrak{r}}^{*}$ means the left and right programs specified in $\alpha_{\mathfrak{r}}$ would always loop for the same number of iterations. Combining it with other constructs, we can express interesting relational properties on loops that involve lock-step. For example, the following REL formula expresses a *relational invariant* of a loop with biprogram, in particular, $\lfloor A\rfloor_{\mathrm{L}} \leftrightarrow \lfloor A\rfloor_{\mathrm{R}}$ holds at the end of every loop iteration of $(\alpha,\beta)^{*}$:

$$(\lfloor A\rfloor_{\mathrm{L}} \leftrightarrow \lfloor A\rfloor_{\mathrm{R}}) \rightarrow [\![\,(\alpha,\beta)^{*}\,]\!](\lfloor A\rfloor_{\mathrm{L}} \leftrightarrow \lfloor A\rfloor_{\mathrm{R}})$$

Where A is an atomic proposition referred to by α and β.

IV. Case Study: Differential Dynamic Logic

To validate the applicability of the REL extension, this section and the next present case studies of the extension on two distinct dynamic logics: (a) differential dynamic logic [6], [21], [15] (dL), and (b) linear dynamic logic on finite traces [16] (LDL$_f$). The two host logics differ in non-trivial ways. Together they cover most features of mainstream dynamic logics. And their differences make them great testbeds for exploring the applicability of the REL extension.

A. Differential Dynamic Logic

Differential dynamic logic (dL) is a first-order dynamic logic that enables verifying high-level cyber-physical systems models featuring real arithmetic, nondeterminism, and differential equations. Programs in dL are referred to as *hybrid programs* [21]. They are a formalism for modeling systems that have both continuous and discrete dynamic behaviors. Hybrid programs can express continuous evolution (as differential equations) as well as discrete transitions.

Figure 5 shows the syntax of constructs that dL add over PDL. Variables are real-valued and can be deterministically assigned ($x := \theta$, where θ is a real-valued arithmetic term) or nondeterministically assigned ($x := *$). Hybrid program $x' = \theta \& \phi$ expresses the continuous evolution of variables x: given the current value of variable x, the system follows the differential equation $x' = \theta$ for some (nondeterministically chosen) amount of time so long as the formula ϕ, the *evolution domain constraint*, holds for all of that time. Note that x can be a vector of variables and then θ is a vector of terms of the same dimension. Atomic formulas of dL are first-order atomic formulas. dL can be viewed as an instantiation of PDL. It instantiates abstract states as valuations of a set of variables over a *domain of computation*: the set of real numbers. It instantiates the set of atomic programs with assignments and a construct that can express continuous evolution. It instantiates the set of atomic formulas with first-order atomic formulas.

Similar to PDL, the semantics of dL [6], [15] is a Kripke semantics in which the Kripke model's worlds are the states of the system. Let $\mathbb{R}$ denote the set of real numbers and $\mathbb{V}$ denote the set of variables. A state is a map $\omega : \mathbb{V} \mapsto \mathbb{R}$ assigning a real value $\omega(x)$ to each variable $x \in \mathbb{V}$. The set of all states is denoted by STA. The semantics of hybrid programs and dL are shown in Figure 6.

$$
\begin{array}{rl}
\text{Term:} & \theta, \delta \ ::= x \mid c \mid \theta \oplus \delta \\
\text{Program:} & \alpha, \beta \ ::= x := \theta \mid x := * \mid x' = \theta \& \phi \mid \cdots \\
\text{Formula:} & \phi, \psi \ ::= \theta \sim \delta \mid \forall x.\ \phi \mid \cdots
\end{array}
$$

Fig. 5: Syntax of dL

Term semantics

$$
\begin{array}{rcl}
\omega[\![x]\!] & = & \omega(x) \\
\omega[\![c]\!] & = & c \\
\omega[\![\theta \oplus \delta]\!] & = & \omega[\![\theta]\!] \ \oplus \ \omega[\![\delta]\!].\ \oplus \text{ denotes corresponding arithmetic operation for } \oplus \in \{+, \times\}
\end{array}
$$

Program semantics

$$
\begin{array}{rcl}
 & \cdots & \\
[\![x := \theta]\!] & = & \{(\omega, \nu) \mid \nu(x) = \omega[\![\theta]\!] \text{ and for all other variables } z \neq x,\ \nu(z) = \omega(z)\} \\
[\![x := *]\!] & = & \{(\omega, \nu) \mid \nu(z) = \omega(z) \text{ for all variables } z \neq x\} \\
[\![x' = \theta \& \phi]\!] & = & \{(\omega, \nu) \mid \text{exists a solution } \varphi : [0, r] \to \text{STA} \\
 & & \text{of } x' = \theta \text{ with } \varphi(0) = \omega,\ \varphi(r) = \nu, \text{ and} \\
 & & \varphi(t) \models \phi \text{ for all } t \in [0, r]\}
\end{array}
$$

Formula semantics

$$
\begin{array}{rcl}
 & \cdots & \\
\omega \models \theta \sim \delta & \text{iff} & \omega[\![\theta]\!] \sim \omega[\![\delta]\!].\ \sim \text{ denotes corresponding comparison for } \sim \in \{=, \leq, <, \geq, >\} \\
\omega \models \forall x.\ \phi & \text{iff} & \nu \models \phi \text{ for all states } \nu \text{ that agree with } \omega \text{ except for the value of } x
\end{array}
$$

Fig. 6: Semantics of dL programs and formulas

With dL, we are often interested in partial correctness formulas of the form $\phi_{pre} \to [\alpha]\phi_{post}$: if ϕ_{pre} is true then ϕ_{post} holds after any possible execution of α. The hybrid program α often has the form $(ctrl\,;\, plant)^*$, where *ctrl* models atomic actions of the control system and does not contain continuous parts (i.e., differential equations); and *plant* models the evolution of the physical environment and has the form of $x' = \theta \& \phi$. That is, the system is modeled as unbounded repetitions of a controller action followed by an update to the physical environment.

$$
\begin{array}{rl}
\phi_{pre} & \equiv A \geq 0 \wedge B \geq 0 \wedge 2Bd > v^2 \\
\phi_{post} & \equiv d > 0 \\
\psi & \equiv 2Bd > v^2 + (A + B)(A\epsilon^2 + 2v\epsilon) \\
accel & \equiv ?\psi\,;\, a := A \\
brake & \equiv a := -B \\
ctrl & \equiv (accel \cup brake)\,;\, t := 0 \\
plant & \equiv d' = -v, v' = a, t' = 1 \& (v \geq 0 \wedge t \leq \epsilon) \\
\phi_{safety} & \equiv \phi_{pre} \to [(ctrl\,;\, plant)^*]\phi_{post}
\end{array}
$$

Fig. 7: dL model of an autonomous vehicle

Consider, as an example, an autonomous vehicle that needs to stop before hitting an obstacle.[2] For simplicity, we model the vehicle in just one dimension. Figure 7 shows a dL model of such an autonomous vehicle. Let d be the vehicle's distance from the obstacle. The *safety condition* that we would like to enforce (ϕ_{post}) is that d is positive. Let v be the vehicle's velocity towards the obstacle in meters per second (m/s) and let a be the vehicle's acceleration (m/s^2). Let t be the time elapsed since the controller was last invoked. The hybrid program *plant* describes how the physical environment evolves over time interval ϵ: distance changes according to $-v$ (i.e., $d' = -v$), velocity changes according to the acceleration (i.e., $v' = a$), and time passes at a constant rate (i.e., $t' = 1$). The differential equations evolve within the time interval $t \leq \epsilon$ and if v is non-negative (i.e., $v \geq 0$).

The hybrid program *ctrl* models the vehicle's controller. The vehicle can either accelerate at A m/s^2 or brake at $-B$ m/s^2. For the purposes of the model, the controller chooses nondeterministically between these options. Hybrid programs *accel* and *brake* express the controller accelerating or braking (i.e., setting a to A or $-B$ respectively). The controller can accelerate only if condition ψ is true, which captures that the vehicle can accelerate for the next ϵ seconds only if doing so would still allow it to brake in time to avoid the obstacle.

The formula to be verified, ϕ_{safety}, is shown at the last line of Figure 7. Given an appropriate precondition ϕ_{pre}, the axioms and proof rules of dL can be used to prove that the safety condition ϕ_{post} holds. The tactic-based theorem prover KeYmaera X [22] provides support for constructing proofs.

[2] Platzer introduces this autonomous vehicle example [21].

$$
\begin{array}{rl}
 & \theta_r, \delta_r \ ::= \lfloor \theta \rfloor_b \mid \theta_r \oplus \delta_r \\
 & \alpha_r, \beta_r \ ::= x := \theta \mid x := * \mid x' = \theta \& \phi \mid \cdots \\
 & \phi_r, \psi_r \ ::= \theta_r \sim \delta_r \mid \cdots
\end{array}
$$

Fig. 8: Syntax of dL$_{\text{REL}}$

B. dL$_{\text{REL}}$: A REL *Instantiation for* dL

We build dL$_{\text{REL}}$, a REL instantiation for dL. dL$_{\text{REL}}$ instantiates the atomic programs and formulas of the REL extension with constructs designed for dL, so relational properties can be easily expressed. These constructs are shown in Figure 8. (We omit the other constructs shown in Figure 3) The relational atomic programs, i.e., P_r, in dL$_{\text{REL}}$ are a relational version of deterministic assignment ($x := \theta$), nondeterministic assignment ($x := *$), and continuous evolution ($x' = \theta \,\&\, \phi$). The relational atomic formulas, i.e., A_r, in dL$_{\text{REL}}$ are comparisons of dL$_{\text{REL}}$ terms, i.e., $\theta_r \sim \delta_r$. A dL$_{\text{REL}}$ term can be either a *projection* term: $\lfloor \theta \rfloor_b$, which refers to a dL term θ in one of the two executions specified by $b \in \{\mathtt{L}, \mathtt{R}\}$, or an arithmetic operation of two dL$_{\text{REL}}$ terms, i.e., $\theta_r \oplus \delta_r$.

Figure 9 shows the semantics of atomic programs in dL$_{\text{REL}}$. $[\![x := \theta]\!]_{\text{RL}}$ indicates that the same deterministic assignment is run by the left and right executions. $[\![x := *]\!]_{\text{RL}}$ ensures both executions having the same value for variable x. $[\![x' = \theta \,\&\, \phi]\!]_{\text{RL}}$ enforces a constraint that the *durations* used by the physical evolution in both executions are the same (i.e., same t for $\varphi_1(t)$ and $\varphi_2(t)$). Such a design is useful since we often want to compare executions of two systems only if they execute for the same period of duration. Note that evolution constraints in dL$_{\text{REL}}$ programs are dL formulas instead of dL$_{\text{REL}}$ formulas. One can imagine a different dL$_{\text{REL}}$ design that allows relational formulas as evolution constraints. However, they are not good

Semantics of $\mathsf{d}\mathcal{L}_{\text{REL}}$ term

$$(\omega_{\text{L}}, \omega_{\text{R}})[\![\lfloor\theta\rfloor_{\text{B}}]\!]_{\text{RL}} = \omega_{\text{L}}[\![\theta]\!] \text{ if } \text{B} = \text{L} \text{ or } \omega_{\text{R}}[\![\theta]\!] \text{ if } \text{B} = \text{R}$$

$$(\omega_{\text{L}}, \omega_{\text{R}})[\![\theta_{\mathfrak{r}} \oplus \delta_{\mathfrak{r}}]\!]_{\text{RL}} = (\omega_{\text{L}}, \omega_{\text{R}})[\![\theta_{\mathfrak{r}}]\!]_{\text{RL}} \oplus (\omega_{\text{L}}, \omega_{\text{R}})[\![\delta_{\mathfrak{r}}]\!]_{\text{RL}}$$ where $\oplus$ denotes corresponding arithmetic operation for $\oplus \in \{+, \times\}$

Semantics of $\mathsf{d}\mathcal{L}_{\text{REL}}$ program

$$\cdots$$

$$[\![x := \theta]\!]_{\text{RL}} = \{((\omega_{\text{L}}, \omega_{\text{R}}), (\nu_{\text{L}}, \nu_{\text{R}})) \mid (\omega_{\text{L}}, \nu_{\text{L}}) \in [\![x := \theta]\!] \text{ and } (\omega_{\text{R}}, \nu_{\text{R}}) \in [\![x := \theta]\!]\}$$

$$[\![x := *]\!]_{\text{RL}} = \{((\omega_{\text{L}}, \omega_{\text{R}}), (\nu_{\text{L}}, \nu_{\text{R}})) \mid \nu_{\text{L}}(z) = \omega_{\text{L}}(z),\ \nu_{\text{R}}(z) = \omega_{\text{R}}(z) \text{ for variables } z \neq x \text{ and } \nu_{\text{L}}(x) = \nu_{\text{R}}(x)\}$$

$$[\![x' = \theta\ \&\phi]\!]_{\text{RL}} = \{((\varphi_1(0), \varphi_2(0)), (\varphi_1(r), \varphi_2(r))) \mid \varphi_1, \varphi_2 : [0, r] \to \text{STA} \text{ are solutions of } x' = \theta \text{ and } \varphi_1(t) \models \phi \text{ and } \varphi_2(t) \models \phi \text{ for all } t \in [0, r]\}$$

Semantics of $\mathsf{d}\mathcal{L}_{\text{REL}}$ formula

$$\cdots$$

$$(\omega_{\text{L}}, \omega_{\text{R}}) \models_{\text{RL}} \theta_{\mathfrak{r}} \sim \delta_{\mathfrak{r}} \text{ iff } (\omega_{\text{L}}, \omega_{\text{R}})[\![\theta_{\mathfrak{r}}]\!]_{\text{RL}} \sim (\omega_{\text{L}}, \omega_{\text{R}})[\![\delta_{\mathfrak{r}}]\!]_{\text{RL}},$$ where $\sim$ is the comparison of $\sim$.

Fig. 9: Semantics of $\mathsf{d}\mathcal{L}_{\text{REL}}$ programs and formulas

$$\begin{aligned}
\psi &\equiv 2Bd > v^2 + (A+B)(A\epsilon^2 + 2v\epsilon) \\
accel &\equiv (?\psi, ?\psi)\,;\ a := A \\
brake &\equiv a := -B \\
ctrl_t &\equiv (\quad (?temp > T\,;\ thermo := -1) \\
&\quad \cup (?temp < T\,;\ thermo := 1) \\
&\quad \cup (?temp = T)\quad) \\
ctrl'_t &\equiv (\quad (?temp > T\,;\ thermo := -2) \\
&\quad \cup (?temp < T\,;\ thermo := 2) \\
&\quad \cup (?temp = T)\quad) \\
ctrl_v &\equiv (accel \cup brake)\,;\ t := 0 \\
plant &\equiv d' = -v, v' = a, t' = 1, temp' = thermo \\
&\quad \&\ (v \geq 0 \wedge t \leq \epsilon) \\
\phi_{\mathfrak{r}} &\equiv (\lfloor d\rfloor_{\text{L}} = \lfloor d\rfloor_{\text{R}} \wedge \lfloor v\rfloor_{\text{L}} = \lfloor v\rfloor_{\text{R}}) \\
\phi_{robust} &\equiv \phi_{\mathfrak{r}} \to [\![\,((ctrl_t, ctrl'_t);\ ctrl_v;\ plant)^*\,]\!]\phi_{\mathfrak{r}}
\end{aligned}$$

Fig. 10: A $\mathsf{d}\mathcal{L}_{\text{REL}}$ example on robustness

candidates of evolution constraints since they are often not physically meaningful in characterizing the physical evolution.

Figure 10 shows examples of $\mathsf{d}\mathcal{L}_{\text{REL}}$ programs and formulas.[3] It presents a design of an autonomous vehicle with velocity control and interior temperature control. Its velocity control is the same as the example presented in Figure 7. For temperature control, the vehicle detects the interior temperature (*temp*), and then chooses one of the two control modes, specified respectively in program $ctrl_t$ and $ctrl'_t$. Here, both modes compare the current temperature with a target temperature T, and then set the thermostat accordingly. The modes differ only in the values of *thermo*. In the physical environment, the temperature changes according to *thermo* (i.e., $temp' = thermo$).

A system designer may want to ensure that the vehicle's control over velocity is *robust*, i.e., the choice of modes for temperature control won't interfere with the vehicle's control of velocity. This relational property is expressed as a $\mathsf{d}\mathcal{L}_{\text{REL}}$ formula at the last line (i.e., ϕ_{robust}). Intuitively, the formula says for two runs of the vehicle, which choose different modes of temperature control, if the vehicle starts with the same position and velocity (the premise $\phi_{\mathfrak{r}}$ of the implication), makes the same control decisions for acceleration and brake ($ctrl_v$), and runs for the same duration (*plant*), it would end with the same position and velocity (the conclusion $\phi_{\mathfrak{r}}$ of the implication). Proving this formula suggests that the vehicle has robust velocity control.

Expressing such a relational property with $\mathsf{d}\mathcal{L}_{\text{REL}}$ is straightforward and much more succinct than the original version [2].

C. Expressive Power of $\mathsf{d}\mathcal{L}_{\text{REL}}$

We investigate the expressive power of $\mathsf{d}\mathcal{L}_{\text{REL}}$ and prove that it is equally expressive as $\mathsf{d}\mathcal{L}$. We state the theorem about the expressive power of $\mathsf{d}\mathcal{L}_{\text{REL}}$:

Theorem 1 (Expressiveness of $\mathsf{d}\mathcal{L}_{\text{REL}}$)**.** *$\mathsf{d}\mathcal{L}_{\text{REL}}$ and $\mathsf{d}\mathcal{L}$ are equally expressive.*

This theorem holds because every $\mathsf{d}\mathcal{L}$ formula can be encoded as an equivalent $\mathsf{d}\mathcal{L}_{\text{REL}}$ formula and vice versa. For the first direction, a $\mathsf{d}\mathcal{L}$ formula ϕ can be encoded as a $\mathsf{d}\mathcal{L}_{\text{REL}}$ formula $\lfloor\phi\rfloor_{\text{L}}$ or $\lfloor\phi\rfloor_{\text{R}}$. We prove the second direction by constructing a sound and complete encoding of $\mathsf{d}\mathcal{L}_{\text{REL}}$ in $\mathsf{d}\mathcal{L}$, which we will introduce in Section VI.

V. Case Study: Linear Dynamic Logic

This case study further explores the applicability of the REL extension by developing LDL_{REL}, a REL instantiation for *linear dynamic logic on finite traces* (LDL_f), a logic that is often used by the AI community for reasoning about actions and planning, such as expressing temporal constraints in task planning [23]. LDL_f presents a great testbed, as it differs largely from $\mathsf{d}\mathcal{L}$ and common dynamic logics, in that its semantics is interpreted over finite traces rather than state transitions. Verification of LDL_f formulas often focuses on satisfiability rather than validity.

We first present the syntax and semantics of LDL_f and LDL_{REL}, and then use LDL_{REL} to express two relational properties that cannot be directly expressed with LDL_f. We prove that LDL_{REL} is more expressive than LDL_f.

A. Linear Dynamic Logic on Finite Traces

LDL_f has the same syntax as PDL (shown in Figure 1), with the set of atomic programs, i.e., Π, instantiated as propositional formulas over the atomic propositions. For presentation purposes, we use metavariable $\phi_{\mathcal{AP}}$ (instead of P in Figure 1) to range over these atomic programs.

Different from PDL (and $\mathsf{d}\mathcal{L}$), the semantics of LDL_f is interpreted over finite traces. A trace is a finite sequence of states: $(\sigma_0, \sigma_1, \sigma_2 ... \sigma_n)$, where each state is a subset of atomic propositions, i.e., $\sigma_i \in (2^{\Phi})$ for all $i \in 0, 1, 2...n$. A state σ_i satisfies proposition A if $A \in \sigma_i$. The position ($i \in 0, 1, 2...n$)

[3] Xiang et.al introduce the original version of this example [2].

can be used to index the states, that is, we write $\sigma(i)$ to indicate the ith state of a trace σ.

Given a set of atomic propositions Φ and a finite trace $\sigma \in (2^{\Phi})^*$, the truth of formula ϕ in trace σ at a position $0 \le i \le |\sigma|-1$, denoted $\sigma, i \models \phi$, is inductively defined [16]:

$\sigma, i \models A$ iff $A \in \sigma(i)$

$\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$

$\sigma, i \models \phi \wedge \psi$ iff $\sigma, i \models \phi$ and $\sigma, i \models \psi$

$\sigma, i \models [\alpha]\phi$ iff for all $i \le j \le |\sigma|-1$ that $(i,j) \in \mathcal{R}(\alpha, \sigma)$, $\sigma, j \models \phi$ holds

Where relation $\mathcal{R}(\alpha, \sigma)$ for a program α and a trace σ is inductively defined as a set of pairs of trace indices, as follows:

$$\mathcal{R}(\phi_{AP}, \sigma) = \{(i, i+1) \mid \sigma, i \models \phi_{AP}\}$$
$$\mathcal{R}(?\phi, \sigma) = \{(i,i) \mid \sigma, i \models \phi\}$$
$$\mathcal{R}(\alpha\,;\,\beta, \sigma) = \{(i,j) \mid \text{exists } k \text{ such that } (i,k) \in \mathcal{R}(\alpha,\sigma) \text{ and } (k,j) \in \mathcal{R}(\beta,\sigma)\}$$
$$\mathcal{R}(\alpha \cup \beta, \sigma) = \mathcal{R}(\alpha,\sigma) \cup \mathcal{R}(\beta,\sigma)$$
$$\mathcal{R}(\alpha^*, \sigma) = \{(i,i)\} \cup \{(i,j) \mid \text{exists } k \text{ such that } (i,k) \in \mathcal{R}(\alpha,\sigma) \text{ and } (k,j) \in \mathcal{R}(\alpha^*,\sigma)\}$$

The semantics is very temporal style, i.e., it focuses on whether a state at a specific position of a trace satisfies propositions. A representative case is the semantics of atomic programs $\mathcal{R}(\phi_{AP}, \sigma)$: it says ϕ_{AP}, a propositional formula on atomic propositions, holds at the current state, i.e., $\sigma(i)$, yet has no relation with the next state (i.e., $\sigma(i+1)$).

B. Model a Gridworld Problem with LDL$_f$

We use LDL$_f$ to model a gridworld problem. Consider the problem of finding a path for a robot towards the destination in a 2×3 gridworld with obstacles. The top part of Figure 11 shows the example. The destination is location `3` and nonterminal locations are $S = \{0,1,2\}$. There are four actions possible in each location, i.e., `up`, `down`, `right`, and `left`, which deterministically cause the corresponding location transitions, except for actions that would take the robot off the grid or hit the obstacle.

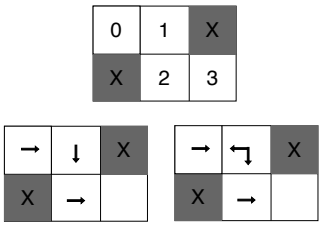

Fig. 11: 2 × 3 gridworld with obstacles (shaded grids with crossmarks) and two planning strategies. Valid locations are {0,1,2,3} and 3 is the destination

A *strategy* of the gridworld problem guides the robot to the destination from any starting location. A strategy is optimal if it takes the least steps to reach the destination. The bottom part of Figure 11 shows two strategies for this problem. The arrows show the actions the robot should take when following a strategy. The strategy on the left is a deterministic one, while the strategy on the right lets the robot nondeterministically choose between `left` and `down` at location `1`.

We use four atomic propositions s_0, s_1, s_2, s_3 to mean that the robot currently sits at the corresponding location. For example, the robot is at location `0` if and only if the following formula holds: $(s_0 \wedge \neg s_1 \wedge \neg s_2 \wedge \neg s_3)$. We write $\widehat{s_i}$ to denote the formula $(s_i \wedge \bigwedge_{j \neq i}(\neg s_j))$, where $i, j \in S = \{0,1,2,3\}$.

We model strategies with LDL$_f$ programs. Intuitively, a strategy can be modeled as repetitions of control actions. At every control step, a strategy checks the current location and decides the next state, until the robot reaches the destination. Specifically, the program modeling a strategy is a repetition of a nondeterministic choice between possible actions. Every action is modeled as a sequence of two programs: a propositional formula that checks the current location followed by a test that specifies the next location to which the action leads.

For example, in both strategies shown in Figure 11, location `1` follows location `0`. It is modeled as $\widehat{s_0}\,;\,?\widehat{s_1}$. And the left strategy can be modeled by a program $\alpha_s = (?\neg\widehat{s_3}\,;\,\alpha_l)^*;\ ?\widehat{s_3}$ where α_l models possible actions at every step, the loop ends when the robot reaches the destination:

$$\alpha_l \equiv (\ \widehat{s_0}\,;\,?\widehat{s_1}\ ) \cup (\ \widehat{s_1}\,;\,?\widehat{s_2}\ ) \cup (\ \widehat{s_2}\,;\,?\widehat{s_3}\ )$$

The right strategy can be similarly modeled by a program $\beta_s = (?\neg\widehat{s_3}\,;\,\beta_l)^*;\ ?\widehat{s_3}$ where β_l is:

$$\beta_l \equiv (\ \widehat{s_0}\,;\,?\widehat{s_1}\ ) \cup (\ \widehat{s_1}\,;\,?\widehat{s_2}\ ) \cup (\ \widehat{s_1}\,;\,?\widehat{s_0}\ ) \cup (\ \widehat{s_2}\,;\,?\widehat{s_3}\ )$$

For example, if a trace satisfies $\widehat{s_0} \wedge \langle\alpha_s\rangle\widehat{s_3}$, then we find a trajectory of the robot that reaches the destination from location `0` by following the left strategy. The length of the trace (also the length of the trajectory) is equal to 1+ number of iterations α_l^* runs.

C. LDL$_{\text{REL}}$: An REL Instantiation for LDL$_f$

Many important properties of planning, such as robustness and privacy, are relational properties on multiple traces [24]. We introduce LDL$_{\text{REL}}$, a REL instantiation for LDL$_f$, and use it to express relational properties of the gridworld problem.

LDL$_{\text{REL}}$ has the same syntax as the REL extension previously shown in Figure 4. Its semantics, shown in Figure 12, is interpreted over a pair of finite traces σ_{L} and σ_{R}, and their indices. The semantics of LDL$_{\text{REL}}$ programs is defined as a transition relation between pairs of indices. For example, the semantics for a biprogram construct, $\mathcal{R}_{\mathfrak{r}}((\alpha,\beta), \sigma_{\text{L}}, \sigma_{\text{R}})$, is defined on two pairs of indices whose first and second elements belong, respectively, to program relation of α, i.e., $(i,x) \in \mathcal{R}(\alpha, \sigma_{\text{L}})$ and β, i.e., $(j,y) \in \mathcal{R}(\beta, \sigma_{\text{R}})$.

We use LDL$_{\text{REL}}$ to express two important relational properties: the first one is a relational property of a single strategy, and the second compares two strategies. The properties cannot be directly expressed with LDL$_f$, especially the second one.

Privacy of Initial Location. Location privacy, i.e., keeping individual locations private while they are partially observable for planning purposes, is an important issue in mobile navigation [25], [26], [27]. We adopt a definition of *opacity* from existing work [24]. A strategy is opaque if it satisfies that there exist at least two paths with the same observation but bearing different secrets, such that the secret of each path cannot be identified exactly only from the observation. Assume that the private information we want to protect is the initial location of the robot, and the publicly observable

LDL_{REL} formula semantics

Truth of formula $\phi_{\mathfrak{r}}$ at position i of a finite trace σ_{L} and position j of a finite trace σ_{R} is inductively defined as follows:

$(\sigma_{\text{L}}, \sigma_{\text{R}}, i, j) \models_{\text{RL}} \neg\phi_{\mathfrak{r}}$ iff $(\sigma_{\text{L}}, \sigma_{\text{R}}, i, j) \not\models_{\text{RL}} \phi_{\mathfrak{r}}$

$(\sigma_{\text{L}}, \sigma_{\text{R}}, i, j) \models_{\text{RL}} \phi_{\mathfrak{r}} \wedge \psi_{\mathfrak{r}}$ iff $(\sigma_{\text{L}}, \sigma_{\text{R}}, i, j) \models_{\text{RL}} \phi_{\mathfrak{r}}$ and $(\sigma_{\text{L}}, \sigma_{\text{R}}, i, j) \models_{\text{RL}} \psi_{\mathfrak{r}}$

$(\sigma_{\text{L}}, \sigma_{\text{R}}, i, j) \models_{\text{RL}} [\![\alpha_{\mathfrak{r}}]\!]\phi_{\mathfrak{r}}$ iff for all position x, y that $((i,j),(x,y)) \in \mathcal{R}_{\mathfrak{r}}(\alpha_{\mathfrak{r}}, \sigma_{\text{L}}, \sigma_{\text{R}})$, $(\sigma_{\text{L}}, \sigma_{\text{R}}, x, y) \models_{\text{RL}} \phi_{\mathfrak{r}}$ holds

$(\sigma_{\text{L}}, \sigma_{\text{R}}, i, j) \models_{\text{RL}} \lfloor\phi\rfloor_{\text{B}}$ iff $\sigma_{\text{L}}, i \models \phi$ when $\text{B} = \text{L}$ or $\sigma_{\text{R}}, j \models \phi$ when $\text{B} = \text{R}$

LDL_{REL} program semantics

The semantics for LDL_{REL} program $\alpha_{\mathfrak{r}}$ and trace σ_{L} σ_{R} is defined inductively as follows:

$$\mathcal{R}_{\mathfrak{r}}(?\phi_{\mathfrak{r}}, \sigma_{\text{L}}, \sigma_{\text{R}}) = \{((i,j),(i,j)) \mid (\sigma_{\text{L}}, \sigma_{\text{R}}, i, j) \models_{\text{RL}} \phi_{\mathfrak{r}}\}$$
$$\mathcal{R}_{\mathfrak{r}}(\alpha_{\mathfrak{r}}\,;\,\beta_{\mathfrak{r}}, \sigma_{\text{L}}, \sigma_{\text{R}}) = \{((i,j),(x,y)) \mid \exists m,n \text{ that } ((i,j),(m,n)) \in \mathcal{R}_{\mathfrak{r}}(\alpha_{\mathfrak{r}}, \sigma_{\text{L}}, \sigma_{\text{R}}) \text{ and } ((m,n),(x,y)) \in \mathcal{R}_{\mathfrak{r}}(\beta_{\mathfrak{r}}, \sigma_{\text{L}}, \sigma_{\text{R}})\}$$
$$\mathcal{R}_{\mathfrak{r}}(\alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}}, \sigma_{\text{L}}, \sigma_{\text{R}}) = \{((i,j),(x,y)) \mid ((i,j),(x,y)) \in \mathcal{R}_{\mathfrak{r}}(\alpha_{\mathfrak{r}}, \sigma_{\text{L}}, \sigma_{\text{R}}) \text{ or } ((i,j),(x,y)) \in \mathcal{R}_{\mathfrak{r}}(\beta_{\mathfrak{r}}, \sigma_{\text{L}}, \sigma_{\text{R}})\}$$
$$\mathcal{R}_{\mathfrak{r}}((\alpha,\beta), \sigma_{\text{L}}, \sigma_{\text{R}}) = \{((i,j),(x,y)) \mid (i,x) \in \mathcal{R}(\alpha, \sigma_{\text{L}}) \text{ and } (j,y) \in \mathcal{R}(\beta, \sigma_{\text{R}})\}$$
$$\mathcal{R}_{\mathfrak{r}}(\alpha^*_{\mathfrak{r}}, \sigma_{\text{L}}, \sigma_{\text{R}}) = \bigcup_{n\in\mathbb{N}}(\mathcal{R}_{\mathfrak{r}}(\alpha^n_{\mathfrak{r}}, \sigma_{\text{L}}, \sigma_{\text{R}})), \text{ where } \alpha^0_{\mathfrak{r}} \text{ is } (?\top, ?\top),\ \alpha^1_{\mathfrak{r}} \text{ is defined as } \alpha_{\mathfrak{r}}, \text{ and } \alpha^{n+1}_{\mathfrak{r}} \text{ is } \alpha^n_{\mathfrak{r}}\,;\,\alpha_{\mathfrak{r}} \text{ for } n \geq 1$$

Fig. 12: Semantics of LDL_{REL} programs and formulas

location is the destination. Thus, the following formula should be *satisfiable* if a strategy, e.g., α_s, ensures privacy of the initial location:

$$(\bigvee_{i,j\in S\wedge j\neq i}(\lfloor\widehat{s_i}\rfloor_{\text{L}} \wedge \lfloor\widehat{s_j}\rfloor_{\text{R}})) \wedge \langle\!\langle(\alpha_s,\alpha_s)\rangle\!\rangle(\lfloor\widehat{s_3}\rfloor_{\text{L}} \wedge \lfloor\widehat{s_3}\rfloor_{\text{R}})$$

Where formula $(\bigvee_{i,j\in S\wedge j\neq i}(\lfloor\widehat{s_i}\rfloor_{\text{L}} \wedge \lfloor\widehat{s_j}\rfloor_{\text{R}}))$ states that the two executions start with different initial location, and the formula of LDL_{REL} modality of existence, i.e., $\lfloor\widehat{s_3}\rfloor_{\text{L}} \wedge \lfloor\widehat{s_3}\rfloor_{\text{R}}$, indicates two executions have the same observable information.

Superiority of Strategies. This property expresses that one strategy, e.g., α_s, is better (no worse) than another strategy, e.g., β_s. Recall that the number of iterations is monotonically related to the length of the robot's trajectory. Therefore, if the left and right executions start from the same initial location, and follow α_s and β_s respectively for the same number of iterations, then the right execution won't reach the destination before the left execution. In particular, the following formula is *valid* (or its negation is unsatisfiable), when strategy α_s is superior to β_s:

$$(\bigvee_{i\in S}(\lfloor\widehat{s_i}\rfloor_{\text{L}} \wedge \lfloor\widehat{s_i}\rfloor_{\text{R}})) \rightarrow [\![(\alpha_l,\beta_l)^*]\!](\lfloor\widehat{s_3}\rfloor_{\text{L}} \vee \neg\lfloor\widehat{s_3}\rfloor_{\text{R}})$$

The formula $(\bigvee_{i\in S}(\lfloor\widehat{s_i}\rfloor_{\text{L}} \wedge \lfloor\widehat{s_i}\rfloor_{\text{R}}))$ indicates the two executions start with the same location, i.e., both start at location `0`, `1`, `2`, or `3`. Repetition of the biprogram construct, i.e., $(\alpha_l,\beta_l)^*$, means both programs run for the same number of iterations. $(\lfloor\widehat{s_3}\rfloor_{\text{L}} \vee \neg\lfloor\widehat{s_3}\rfloor_{\text{R}})$ specifies that the left execution has reached the destination, or the right execution has not. The formula should always hold if strategy α_s is no worse than β_s.

D. Expressive Power of LDL_{REL}

We investigate the expressive power of LDL_{REL} and prove that LDL_{REL} is more expressive than LDL_f. Intuitively, the increased power is caused by two factors: (1) the semantics of LDL_f is interpreted over traces and its expressive power is the same as *regular expression* [16], and (2) program construct $\alpha^*_{\mathfrak{r}}$ in LDL_{REL}, which can be used to encode lock-step, introduces the ability of *counting* to some extent. Regular languages do not have such an ability. The theorem about expressiveness of LDL_{REL} is as follows:

Theorem 2 (Expressiveness of LDL_{REL})**.** LDL_{REL} *is strictly more expressive than* LDL_f.

We prove by finding LDL_{REL} formulas that are not expressible by the host LDL_f. The detailed proof can be found in the Appendix.

VI. Verification Techniques

Verifying REL formulas can take advantage of existing techniques and tools. Reusing existing tools are particularly helpful when the tools designated for relational verification are often lacking.

In this section, we introduce two encodings. The first one is *host logic encodings*, which transform REL formulas into formulas of the host logics, so existing tools and techniques developed for the host logics can be leveraged. The second encoding is *constraints-based encodings*, in which we encode a REL formula as a set of first-order constraints and then use constraint solvers, e.g., Z3, to verify the constraints. Both encodings work for $\mathsf{d}\mathcal{L}_{\text{REL}}$ and LDL_{REL}.

For presentation purpose, we use $\mathsf{d}\mathcal{L}_{\text{REL}}$ and LDL_{REL}, respectively, to introduce the first and second encodings.

A. Host Logic Encoding

The first encoding reduces the verification of REL formulas to the verification of formulas of the host logics. The encoding is inspired by self-composition [28], [17], [29], [30], a proof technique often used for proving noninterference for deterministic programs. We briefly introduce self-composition first, and then introduce the encoding in the setting of $\mathsf{d}\mathcal{L}_{\text{REL}}$.

Self-Composition. To develop an intuition for how the self-composition technique is used to prove noninterference, consider the problem of checking whether low-security outputs of a deterministic program reveals high-security inputs. Construct two copies of the program, renaming the program variables so that the variables in the two copies are disjoint. Set the low-security inputs in both copies to identical values but allow the high-security inputs to take different values. Now, sequentially compose these two programs together. If the composed

program can terminate in a state where the corresponding low-security outputs differ, then the original program does not satisfy noninterference; conversely, if in all executions of the composed program, the low-security outputs are the same, then the original program satisfies noninterference. Intuitively, the composition of the two copies allows a single program to represent two executions of the original program, reducing checking a relational property of the original problem to checking a safety property of the composed program.

Using the same insights, we develop an encoding for dL_{REL} formulas. The encoding build on two steps: (1) renaming a dL_{REL} formula into an equivalent dL formula whose left and right executions use disjoint variables, and (2) making a composition of the left and right programs and then verifying properties on the composition.

Renaming $\mathbf{DL}_{\text{REL}}$ Formulas. Note that the left and right executions often refer to the same set of variables. To compare them in one formula, we need to rename variables used by one of the two executions. To help with renaming, we define *renaming functions* that map all variables accessible by the *right* execution to fresh variables.

Definition 1 (Renaming function for DL_{REL})**.** *For a* DL_{REL} *formula* $\phi_{\mathfrak{r}}$*, a function* $\xi : \text{VAR}_{\mathfrak{R}}(\phi_{\mathfrak{r}}) \to V$ *(where* V *is a set of variables) is a* renaming function for $\phi_{\mathfrak{r}}$ *if:*

1) ξ *is a bijection;*

2) For all $x \in \text{VAR}_{\mathfrak{R}}(\phi_{\mathfrak{r}})$*,* $\xi(x) \notin \text{VAR}_{\mathfrak{L}}(\phi_{\mathfrak{r}})$*;*

Where function $\text{VAR}_{\mathfrak{L}}(\phi_{\mathfrak{r}})$ *(or* $\text{VAR}_{\mathfrak{R}}(\phi_{\mathfrak{r}})$*) returns the set of variables accessed by the left (or right) execution.*

We write $\xi(\phi_{\mathfrak{r}})$ for the formula identical to $\phi_{\mathfrak{r}}$ but whose variables accessible by the right execution have been renamed according to ξ. Renaming functions similarly apply to dL_{REL} programs. We also write $\xi(\omega)$ for the state identical to ω but whose domain variables have been renamed according to ξ. More details about the definitions of variable sets and renaming functions can be found in the Appendix.

As an example, consider a formula $\phi_{\mathfrak{r}}$:

$$(\lfloor x \rfloor_{\mathfrak{L}} = \lfloor x \rfloor_{\mathfrak{R}}) \;\to\; [\![(y := x, y := x)]\!] (\lfloor y \rfloor_{\mathfrak{L}} = \lfloor y \rfloor_{\mathfrak{R}})$$

A renaming function for $\phi_{\mathfrak{r}}$ is $\xi = \{x \mapsto x_1, y \mapsto y_1\}$. Then $\xi(\phi_{\mathfrak{r}})$ is the following formula:

$$(\lfloor x \rfloor_{\mathfrak{L}} = \lfloor x_1 \rfloor_{\mathfrak{R}}) \;\to\; [\![(y := x, y_1 := x_1)]\!] (\lfloor y \rfloor_{\mathfrak{L}} = \lfloor y_1 \rfloor_{\mathfrak{R}})$$

Formula $\phi_{\mathfrak{r}}$ is trivially equivalent to $\xi(\phi_{\mathfrak{r}})$.

Now we introduce the encoding.

From dL_{REL} to dL. We first introduce a function π that will be used to transform renamed dL_{REL} formulas, i.e., $\xi(\phi_{\mathfrak{r}})$, into an equivalent dL formula. A key step of the function is that it converts the biprogram construct (α, β) into a sequential composition of α and β. Also, it directly extracts the contents from all projection constructs, e.g., $\lfloor \phi \rfloor_{\mathfrak{B}}$ and $\lfloor \theta \rfloor_{\mathfrak{B}}$. The π function is defined as follows:

Definition 2 (π for dL_{REL})**.** *For a* dL_{REL} *formula* $\phi_{\mathfrak{r}}$ *whose variables for the left and right executions are disjoint (i.e.,* $\text{VAR}_{\mathfrak{L}}(\phi_{\mathfrak{r}}) \cap \text{VAR}_{\mathfrak{R}}(\phi_{\mathfrak{r}}) = \varnothing$*), a function* π *that can transform a* dL_{REL} *formula* $\phi_{\mathfrak{r}}$ *(or a* dL_{REL} *program* $\alpha_{\mathfrak{r}}$*) to an equivalent* dL *formula (or program), is defined inductively as follows.*

$$\pi(\theta_{\mathfrak{r}} \sim \delta_{\mathfrak{r}}) = \pi(\theta_{\mathfrak{r}}) \sim \pi(\delta_{\mathfrak{r}}) \qquad \pi(\alpha_{\mathfrak{r}} ;\ \beta_{\mathfrak{r}}) = \pi(\alpha_{\mathfrak{r}}) ; \pi(\beta_{\mathfrak{r}})$$
$$\pi(\neg\phi_{\mathfrak{r}}) = \neg\pi(\phi_{\mathfrak{r}}) \qquad \pi(\alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}}) = \pi(\alpha_{\mathfrak{r}}) \cup \pi(\beta_{\mathfrak{r}})$$
$$\pi(\phi_{\mathfrak{r}} \wedge \psi_{\mathfrak{r}}) = \pi(\phi_{\mathfrak{r}}) \wedge \pi(\psi_{\mathfrak{r}}) \qquad \pi(?\phi_{\mathfrak{r}}) = ?\pi(\phi_{\mathfrak{r}})$$
$$\pi([\![\alpha_{\mathfrak{r}}]\!]\phi_{\mathfrak{r}}) = [\pi(\alpha_{\mathfrak{r}})]\pi(\phi_{\mathfrak{r}}) \qquad \pi(\alpha_{\mathfrak{r}}^*) = (\pi(\alpha_{\mathfrak{r}}))^*$$
$$\pi(\lfloor \phi \rfloor_{\mathfrak{B}}) = \phi \qquad \pi((\alpha, \beta)) = \alpha ; \beta$$

And $\pi(\theta_{\mathfrak{r}})$ *on terms is inductively defined:*

$$\pi(\lfloor \theta \rfloor_{\mathfrak{B}}) = \theta \qquad \pi(\theta_{\mathfrak{r}} \oplus \delta_{\mathfrak{r}}) = \pi(\theta_{\mathfrak{r}}) \oplus \pi(\delta_{\mathfrak{r}})$$

Here, $\pi(\alpha_{\mathfrak{r}})$ considers only five program constructs since the other constructs, e.g., $x := *$, can be encoded with the five program constructs.

Encoding Atomic Programs. The atomic programs of dL_{REL} are helpful for programmers to express interesting relational properties of cyber-physical systems. However, they are just syntactic sugar, i.e., they *can* be encoded with the other program constructs of dL_{REL}, as follows:

$$x := \theta \equiv (x := \theta, x := \theta)$$
$$x := * \equiv (x := *, x := *) ;\ ?(\lfloor x \rfloor_{\mathfrak{L}} = \lfloor x \rfloor_{\mathfrak{R}})$$
$$x' = \theta \,\&\, \phi \equiv (t := 0, t := 0) ;$$
$$(x' = \theta, t' = 1 \,\&\, \phi, x' = \theta, t' = 1 \,\&\, \phi) ;\ ?(\lfloor t \rfloor_{\mathfrak{L}} = \lfloor t \rfloor_{\mathfrak{R}})$$

Atomic program $x := \theta$ can be encoded as a biprogram whose left and right programs are the same. The constraint of nondeterministic assignment $x := *$ is encoded as a test of formula $(\lfloor x \rfloor_{\mathfrak{L}} = \lfloor x \rfloor_{\mathfrak{R}})$. Program $x' = \theta \,\&\, \phi$, can be encoded with the help of a continuous variable t, which is a fresh variable that represents time. A special differential equation $t' = 1$ is added to both executions to represent the passage of time. The variable t is set to 0 before the physical evolution; a test of formula $(\lfloor t \rfloor_{\mathfrak{L}} = \lfloor t \rfloor_{\mathfrak{R}})$ is added after the biprograms of physical dynamics to enforce the constraint that both evolutions last for the same duration.

Soundness. With a renaming function ξ, we can encode a dL_{REL} formula $\phi_{\mathfrak{r}}$ as $\pi(\xi(\phi_{\mathfrak{r}}))$. The soundness theorem of this encoding is as follows:

Theorem 3 (Soundness of the encoding of dL_{REL})**.** *For states* $\omega_{\mathfrak{L}}\ \omega_{\mathfrak{R}}$*, a* dL_{REL} *formula* $\phi_{\mathfrak{r}}$*, and a renaming function* ξ *for* $\phi_{\mathfrak{r}}$*,*

$(\omega_{\mathfrak{L}}, \omega_{\mathfrak{R}}) \models_{\text{RL}} \phi_{\mathfrak{r}}$ *iff*

$((\omega_{\mathfrak{L}} \Downarrow \text{VAR}_{\mathfrak{L}}(\phi_{\mathfrak{r}})) \otimes (\xi(\omega_{\mathfrak{R}} \Downarrow \text{VAR}_{\mathfrak{R}}(\phi_{\mathfrak{r}})))) \models \pi(\xi(\phi_{\mathfrak{r}}))$

Where $\omega \Downarrow V$ *denotes the projection of state* ω *on a set of variables* V*: the map* $\{x \mapsto \omega(x)\}$ *for all* $x \in V$*, and* $\otimes$ *means the join of two non-overlapping states.*

The proof can be done by simultaneous induction on formula $\phi_{\mathfrak{r}}$ and program $\alpha_{\mathfrak{r}}$. An important case is the biprogram (α, β). Intuitively, π is sound for (α, β) because the semantics of dL and dL_{REL} is defined on state transitions and sequential composition of α and β is semantically consistent with the semantics of (α, β). More details about the proof can be found in the Appendix.

Verifying dL_{REL} Formulas. With Theorem 3, we can verify a dL_{REL} formula $\phi_{\mathfrak{r}}$ by verifying a dL formula $\pi(\xi(\phi_{\mathfrak{r}}))$.

Verifying dL formulas can done with the theorem prover KeYmaera X. We encode the example shown in Figure 10 (ϕ_{robust} in particular) as a dL formula $\pi(\xi(\phi_{robust}))$, which KeYmaera X is able to prove in a fully-automated manner.

Leverage other verification techniques. An important step of π function is to sequentially compose biprogram, i.e., $\pi((\alpha,\beta)) = \alpha;\beta$. It may still be challenging to verify a formula like $[\alpha;\beta]\phi$ in dL, especially for complicated relational properties. Fortunately, verifying these complicated dL_{REL} formulas can benefit from advances in relational verification. We exemplify with verifying certain complicated *forall, forall* relational properties, which are often expressible as the following dL_{REL} formulas:

$$\phi_{\mathfrak{r}} \rightarrow [\![(\alpha,\beta)]\!]\psi_{\mathfrak{r}}$$

Where $\phi_{\mathfrak{r}}$ and $\psi_{\mathfrak{r}}$ often do not have modalities, and α may be the same as β. One particular interesting kind is when both programs have loops:

$$\phi_{\mathfrak{r}} \rightarrow [\![(\alpha^*,\beta^*)]\!]\psi_{\mathfrak{r}}$$

Our dL encoding for this formula would be:

$$\pi(\phi_{\mathfrak{r}}) \rightarrow [\alpha^*;\beta^*]\pi(\psi_{\mathfrak{r}})$$

This kind of properties may be non-trivial to verify. Prior work on self-composition based approaches often combine the two loops together into a composition with one loop:

$$\pi(\phi_{\mathfrak{r}}) \rightarrow [SC(\alpha,\beta)^*]\pi(\psi_{\mathfrak{r}})$$

The verification is then reduced to discovering a *invariant* of $SC(\alpha,\beta)^*$, like the program safety verification. How to design a sound *SC* may be challenging. An intuitive approach is to sequentially compose α and β: $SC(\alpha,\beta) = \alpha;\beta$. At every loop iteration, the composition executes α and β in lockstep. As noted in prior work [31], such a execution schedule (also known as loop alignment) doesn't work for many verification problems, especially when the loops in α^* and β^* run different numbers of iterations. And more importantly, the numbers they run are determined at runtime.

Consider the following example program α adapted from prior work [31], which computes $\sum_{a\le n<b} n^2$ for inputs a and b:

$$\begin{aligned}\alpha &\equiv init;(body)^*;?a\ge b\\ init &\equiv ?0<a<b;c:=0;\\ body &\equiv (?a<b;c:=c+a*a;a:=a+1)\end{aligned}$$

We are interested in verifying the following dL_{REL} formula: $\phi_{sum} \equiv (\lfloor a\rfloor_{\text{L}} < \lfloor a\rfloor_{\text{R}} \wedge \lfloor b\rfloor_{\text{L}} > \lfloor b\rfloor_{\text{R}}) \rightarrow [\![(\alpha,\alpha)]\!](\lfloor c\rfloor_{\text{L}} > \lfloor c\rfloor_{\text{R}})$ which trivially holds.

With a renaming function that renames the right execution with subscript 1 and applying π, we get a dL formula:

$$(a<a_1 \wedge b>b_1) \rightarrow [\alpha;\alpha_1](c>c_1)$$

If we design *SC* to run the loop body of α and α_1 in lockstep, we get a dL formula like the following:

$$(a<a_1 \wedge b>b_1) \rightarrow [init;init_1;(body;body_1)^*;?a\ge b;?a_1\ge b_1](c>c_1)$$

However, such a lockstep *SC cannot* verify the property: α and α_1 run different numbers of iterations before termination.

We can leverage recent advances in relational verification to handle these complicated dL_{REL} formulas. These work introduce semantic-based approaches whose compositions dynamically schedule the program(s) to execute next [31], [32], [33]. The choice of which program to execute next depends on the states of the compositions, as opposed to the syntactic structures of the composition, such as lockstep. Inspired by these approaches, we design the following $SC(\alpha,\alpha_1)$:

$$\begin{aligned}SC(\alpha,\alpha_1) \equiv\ & (?sch(V,V_1);\alpha)\\ \cup\ & (?sch_1(V,V_1);\alpha_1)\\ \cup\ & (?sch_2(V,V_1);\alpha;\alpha_1)\end{aligned}$$

Where the three test formulas respectively capture the conditions under which the program(s) to execute by the composition. V and V_1 are the set of variables used in α and α_1. We can successfully verify the example property ϕ_{sum} with $sch(V,V_1)$, $sch_1(V,V_1)$, and $sch_2(V,V_1)$ being $a<a_1$, *false*, and $a\ge a_1$ respectively. Intuitively, the composition executes the left program until the two executions have the same value for a and a_1. Then, programs α and α_1 execute in lock-step.

Prior work on the semantic-based approaches often focus on *automatically* discovering the composition like $SC(\alpha,\alpha_1)$, in particular, the test formulas and its relational invariant. They often use constraint solvers, e.g., Z3, to automatically find solutions to uninterpreted functions that encode the unknown test formulas and invariants. Unfortunately, these approaches may be ineffective in verifying certain dL_{REL} formulas, which involves differential equations, real numbers, and non-linear arithmetics. It is difficult for a solver to find solutions of an uninterpreted function when these elements are involved.

The setting of dL_{REL} offers an alternative approach: the users can directly specify the execution conditions, e.g., $sch(V,V_1)$, $sch_1(V,V_1)$, and $sch_2(V,V_1)$, as parts of dL_{REL} formulas. For the example property ϕ_{sum} above, we can specify these conditions as formulas sch_i in the following dL_{REL} formula:

$$\begin{aligned}&(\lfloor a\rfloor_{\text{L}} < \lfloor a\rfloor_{\text{R}} \wedge \lfloor b\rfloor_{\text{L}} > \lfloor b\rfloor_{\text{R}}) \rightarrow\\ &\quad [\![(init,init);(\ ?sch;(body,?\top)\\ &\qquad \cup ?sch_1;(?\top,body)\\ &\qquad \cup ?sch_2;(body,body)\ )^*;(?a\le b,?a\le b)]\!]\\ &\qquad\qquad (\lfloor c\rfloor_{\text{L}} > \lfloor c\rfloor_{\text{R}})\end{aligned}$$

Users can explicitly provide these execution conditions: $sch \equiv \lfloor a\rfloor_{\text{L}} < \lfloor a\rfloor_{\text{R}}$, $sch_1 \equiv$ *false*, and $sch_2 \equiv \lfloor a\rfloor_{\text{L}} \ge \lfloor a\rfloor_{\text{R}}$. The corresponding dL formula can therefore be verified.

In addition, we can develop techniques to synthesize these conditions, for example, develop interactive synthesis techniques [34] to produce these conditions.

In short, verifying dL_{REL} formulas benefit from advances in relational verification, so long as we can develop suitable SC functions. Moreoever, dL_{REL} opens opportunities to develop new techniques to verify other variants of relational properties, e.g., *forall, exists* properties. We leave that to future work.

A sound host encoding for LDL_{REL}. We have also developed a sound but incomplete encoding for LDL_{REL} in LDL_f. This encoding is *incomplete* as LDL_{REL} is more expressive than LDL_f. We can use this encoding to verify a *subset* of LDL_{REL} formulas, especially, loop-free formulas. However, the encoding won't work for formulas that contain lockstep loops. The details about this encoding can be found in the Appendix.

The host logic encoding, arguably, works better for REL instantiations that are equally expressive as their host logics, e.g., $\mathsf{d}\mathcal{L}_{\text{REL}}$. We *cannot* design a sound and complete encoding for REL instantiations that introduce extra expressiveness, e.g., LDL_{REL}. The second encoding works better for LDL_{REL}, which we introduce next.

B. Constraints-based Encoding

The second approach for verifying a REL property is by encoding it as a set of first-order constraints, and then use constraint solvers to find if the constraints are satisfiable. Intuitively, for a REL formula of concern, the constraints are generated by closely following the semantics of all subprograms and subformulas inside the formula. Thus the formula holds if the corresponding constraints are satisfiable. We elaborate on the encoding in the setting of LDL_{REL}.

Encoding of LDL_{REL}. LDL_{REL} is the REL extension of LDL_f, whose semantics is interpreted over finite traces. We encode LDL_{REL} formulas as first-order constraints with functions, integers and arrays. In particular, the encoding builds on several ingrediants:

- Using an integer number to represent an execution state of LDL_f. The number is treated as a bitvector. Every atomic proposition has a corresponding bit. The bit is set to 1 in an integer number if the atomic proposition holds in the state to which the number corresponds. For example, the number 33 (i.e., 100001 in binary) represents a state where the first and last atomic propositions hold.
- Using an array of integers to represent a trace. The semantics of LDL_f are interpreted over finite traces. Arrays of integers represent traces naturally.
- Encoding *every* program construct appears in the targeted LDL_{REL} formula as an independent boolean function. We then can capture the semantics of all involved LDL_{REL} and LDL_f programs as first-order formulas on these functions. For example, the implication $\mathtt{F}_{\alpha\cup\beta}(\sigma,i,j) \rightarrow \mathtt{F}_{\alpha}(\sigma,i,j) \vee \mathtt{F}_{\beta}(\sigma,i,j)$ captures the semantics of $\alpha\cup\beta$, where $\mathtt{F}_{\alpha}(\sigma,i,j)$ is a function that returns true if $\{i,j\}\in\mathcal{R}(\alpha,\sigma)$.

In addition to all the formulas and functions that help specifying program semantics, we define another function λ that returns a first-order formula for every LDL_{REL} formula. λ and $\mathtt{F}$ are inductively defined as follows. The definition closely follows the semantics of LDL_{REL} and LDL_f.

Definition 3 (λ for LDL_{REL})**.** *A function λ that can transform a LDL_{REL} formula $\phi_{\mathfrak{r}}$ to a first-order formula, is defined inductively as follows:*

$$\lambda(\neg\phi_{\mathfrak{r}},\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}}) = \neg\lambda(\phi_{\mathfrak{r}},\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}})$$
$$\lambda(\lfloor\phi\rfloor_{B},\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}}) = \lambda(\phi,\sigma_{\text{L}},i_{\text{L}}) \text{ if } B = \text{L} \text{ else } \lambda(\phi,\sigma_{\text{R}},i_{\text{R}})$$
$$\lambda(\phi_{\mathfrak{r}}\wedge\psi_{\mathfrak{r}},\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}}) = \lambda(\phi_{\mathfrak{r}},\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}})\wedge\lambda(\psi_{\mathfrak{r}},\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}})$$
$$\lambda([\![\,\alpha_{\mathfrak{r}}\,]\!]\phi_{\mathfrak{r}},\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}}) = \forall k_{\text{L}}\,k_{\text{R}},(\ k_{\text{L}}\le|\sigma_{\text{L}}|-1\wedge k_{\text{R}}\le|\sigma_{\text{R}}|-1 \wedge F_{\alpha_{\mathfrak{r}}}(\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}},k_{\text{L}},k_{\text{R}})\ ) \rightarrow \lambda(\phi_{\mathfrak{r}},\sigma_{\text{L}},\sigma_{\text{R}},k_{\text{L}},k_{\text{R}})$$

Where the functions $\mathtt{F}$ are defined for all program constructs as follows, where the programs $\alpha_{\mathfrak{r}}$, $\beta_{\mathfrak{r}}$, or $\phi_{\mathfrak{r}}$ are instantiated with concrete program constructs appear in $\phi_{\mathfrak{r}}$. Note that the arguments in all functions are (implicitly) universally quantified.

$$F_{\alpha_{\mathfrak{r}}\,;\,\beta_{\mathfrak{r}}}(\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}},j_{\text{L}},j_{\text{R}}) \rightarrow \exists\, k_{\text{L}}\,k_{\text{R}},(i_{\text{L}}\le k_{\text{L}}\le j_{\text{L}})\wedge(i_{\text{R}}\le k_{\text{R}}\le j_{\text{R}}) \wedge F_{\alpha_{\mathfrak{r}}}(\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}},k_{\text{L}},k_{\text{R}}) \wedge F_{\beta_{\mathfrak{r}}}(\sigma_{\text{L}},\sigma_{\text{R}},k_{\text{L}},k_{\text{R}},j_{\text{L}},j_{\text{R}})$$
$$F_{\alpha_{\mathfrak{r}}\cup\beta_{\mathfrak{r}}}(\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}},j_{\text{L}},j_{\text{R}}) \rightarrow F_{\alpha_{\mathfrak{r}}}(\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}},j_{\text{L}},j_{\text{R}}) \vee F_{\beta_{\mathfrak{r}}}(\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}},j_{\text{L}},j_{\text{R}})$$
$$F_{?\phi_{\mathfrak{r}}}(\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}},j_{\text{L}},j_{\text{R}}) \rightarrow (i_{\text{L}}==j_{\text{L}}\wedge i_{\text{R}}==j_{\text{R}}) \wedge\lambda(\phi_{\mathfrak{r}},\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}})$$
$$F_{\alpha_{\mathfrak{r}}^*}(\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}},j_{\text{L}},j_{\text{R}}) \rightarrow (i_{\text{L}}==j_{\text{L}}\wedge i_{\text{R}}==j_{\text{R}}) \vee(\exists\, k_{\text{L}}\,k_{\text{R}},F_{\alpha_{\mathfrak{r}}}(\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}},k_{\text{L}},k_{\text{R}}) F_{\alpha_{\mathfrak{r}}^*}(\sigma_{\text{L}},\sigma_{\text{R}},k_{\text{L}},k_{\text{R}},j_{\text{L}},j_{\text{R}})\ )$$
$$F_{(\alpha,\beta)}(\sigma_{\text{L}},\sigma_{\text{R}},i_{\text{L}},i_{\text{R}},j_{\text{L}},j_{\text{R}}) \rightarrow F_{\alpha}(\sigma_{\text{L}},i_{\text{L}},j_{\text{L}})\wedge F_{\beta}(\sigma_{\text{R}},i_{\text{R}},j_{\text{R}})$$

The $\mathtt{F}$ function extends to LDL_f programs:

$$F_{\alpha\,;\,\beta}(\sigma,i,j) \rightarrow \exists\, k,(i\le k\le j)\wedge F_{\alpha}(\sigma,i,k)\wedge F_{\beta}(\sigma,k,j)$$
$$F_{\alpha\cup\beta}(\sigma,i,j) \rightarrow F_{\alpha}(\sigma,i,j)\vee F_{\beta}(\sigma,i,j)$$
$$F_{?\phi}(\sigma,i,j) \rightarrow i==j\wedge\lambda(\phi,\sigma,i)$$
$$F_{\phi_{\mathcal{AP}}}(\sigma,i,j) \rightarrow j==i+1\wedge\lambda(\phi_{\mathcal{AP}},\sigma,i)$$

The λ function also extends to LDL_f formulas:

$$\lambda(A,\sigma,i) = (\mathit{bitEncoding}(A)\ \&\&\ \sigma[i]) == \mathit{bitEncoding}(A)$$
$$\lambda(\neg\phi,\sigma,i) = \neg\lambda(\phi,\sigma,i)$$
$$\lambda(\phi\wedge\psi,\sigma,i) = \lambda(\phi,\sigma,i)\wedge\lambda(\psi,\sigma,i)$$
$$\lambda([\alpha]\phi,\sigma,i) = \forall\, k,((i\le k\le|\sigma|-1)\wedge F_{\alpha}(\sigma,i,k)) \rightarrow \lambda(\phi,\sigma,k)$$

$\lambda(A,\sigma,i)$ checks if the atomic proposition A holds at the state $\sigma[i]$. The $\mathit{bitEncoding}(A)$ function returns the integer (bitvector) that represents A, and Bitwise AND (&&) with the integer at $\sigma[i]$. The result would still be the value of $\mathit{bitEncoding}(A)$ if A holds at the state $\sigma[i]$.

With functions $\mathtt{F}$ and λ, we can produce a set of first-order formulas that is satisfiable if and only if the $\phi_{\mathfrak{r}}$ of interest is satisfiable. In particular, we define the set of constraints $\mathtt{S}(\phi_{\mathfrak{r}})$ that contains the following first-order formulas:

- $\exists\,\sigma_{\text{L}}\,\sigma_{\text{R}},\lambda(\phi_{\mathfrak{r}},\sigma_{\text{L}},\sigma_{\text{R}},0,0)$.
- All formulas (implications) that specify the semantics of all LDL_{REL} and LDL_f program constructs appear in $\phi_{\mathfrak{r}}$.

Theorem 4 (Soundness of the constraints-based encoding for LDL_{REL})**.** *A LDL_{REL} formula $\phi_{\mathfrak{r}}$ holds iff $\mathtt{S}(\phi_{\mathfrak{r}})$ is satisfiable.*

The theorem holds since the definition of $\mathtt{S}(\phi_{\mathfrak{r}})$ follows closely the semantics of LDL_{REL}. It can be proven by mutually induction on programs and formulas of $\phi_{\mathfrak{r}}$.

Verifying LDL_{REL} formulas. We develop set $\mathtt{S}(\phi_{\mathfrak{r}})$ for the two example properties of the gridworld problem introduced in Section V, and successfully verify them using Z3.

Constraints-based Encoding for $\mathsf{d}\mathcal{L}_{\text{REL}}$. The approach of constraints-based encoding relies on the underlying theories support to specify and verify the properties of REL formulas. In the setting of LDL_{REL}, the encoding builds on first-order constraints with functions, integers, and arrays. And the tool Z3 can verify these constraints.

A constraints-based encoding can be similarly developed for $\mathsf{d}\mathcal{L}_{\text{REL}}$, by leveraging existing support on specifying constraints on real numbers and differential equations. For example, the

constraint solver dReal3 [35], [36] allows the users to specify first-order formulas on real numbers as well as non-linear functions such as differential equations. dReal3 focuses on the problem of δ-*sat*: when a formula is δ-sat, either it is indeed satisfiable, or it is unsatisfiable but a δ-perturbation on its numerical terms would make it satisfiable [35]. Comparing the verification power of KeYmaera X and dReal3, especially on $\mathsf{d}\mathcal{L}_{\text{REL}}$ formulas, is an interesting problem, which we leave to future work.

VII. Related work

Relational Reasoning for Dynamic Logic. Beckert et.al have done a series of work on extending dynamic logic with trace modalities [9], [37], [10]. They further apply trace modalities to check secure information flow in the setting of concurrent programs [10]. Their work focus on first-order dynamic logic with deterministic programs. Gutsfeld et.al introduce an expressive extension to propositional dynamic logic to check hyperproperties by introducing path quantifiers [38]. Algorithms for model checking these hyperproperties are introduced. Tool support is left for future work. In comparison, we aim for lightweight yet effective first-class relational reasoning for dynamic logics of different features and reusing standard tools.

Relational Reasoning for Differential Dynamic Logic. Various approaches have been proposed to analyze specific relational properties in $\mathsf{d}\mathcal{L}$. A primitive is introduced to express a refinement relation between two hybrid programs [39]. An expressive modal logic based on $\mathsf{d}\mathcal{L}$ has been introduced to reason with nondeducibility [40]. Both work do not provide tool support. Kolčák et al. introduce a relational extension of $\mathsf{d}\mathcal{L}$ that focuses on reasoning about two dynamics [41]. Xiang et al. introduce a formal framework in the setting of $\mathsf{d}\mathcal{L}$ for modeling and analyzing the robustness of cyber-physical systems under sensor attacks [2]. In our case study for $\mathsf{d}\mathcal{L}$, we express and verify this robustness property with $\mathsf{d}\mathcal{L}_{\text{REL}}$, which is more succinct than the original example.

Biprogram. Our biprogram construct is an adaption of the work of Pottier and Simonet on information flow analysis for ML [13]. The work introduces an extension of ML that encodes a pair of ML terms, i.e., (e, e), as a single bracket construct, i.e., $(e \,|\, e)$. The bracket constructs cannot be nested. This is analogous to our design in that the biprogram contains two programs of the host logics. This bracket-based approach has been often used in proving noninterference [42], [43], [44]. The name "biprogram" is also used in other recent work on relational reasoning [14].

Self-Composition and Product Program. The encodings introduced in this work are inspired by self-composition, which is a common approach for relational reasoning [17], [45], [46], [47]. Approaches based on self-composition are often syntax-directed, that is, they compose two programs that have similar, if not identical, syntactic constructs. Alternatively, property directed self-composition [31] tackles programs that cannot be easily aligned in a composition. It composes programs (or copies of the same program) by finding good alignment between the copies in order to have expressible assertions. The encodings in this work can be viewed as syntax-based self-compositions adapted for dynamic logics, which are less developed in the literature.

Relational Verification in General. Verification of relational properties has been an active research area. A common approach is developing specialized language extensions that target specific relational properties, e.g., secure information flow [13], [48], [49], [50], [51], [52], [53]. Our REL extension is an extension of dynamic logics that targets general relational properties. Another approach for relational verification is to use relational program logic [54], [55], [56]. Benton introduced Relational Hoare Logic for verifying program transformations [54], which provides a general framework for relational correctness proofs. A recent work proposes a notion to evaluate the design of Relational Hoare Logic [56].

SMT-based approaches have emerged as a popular approach for the verification of relational (and beyond) properties. Sousa and Dillig introduce a program logic, named Cartesian Hoare Logic (CHL), for verifying k-safety properties [57]. The work has been further extended for proving the correctness of 3-way merge [58]. They introduce a generalized form of Hoare triples to express relations between different program executions, and use SMT solvers to determine their satisfiability. SMT-based frameworks have also been introduced to reason about program equivalence in both high-level languages [59] and low-level languages [60], [61].

VIII. Conclusion

We introduce the REL extension, a general extension of dynamic logics to support first-class relational reasoning in a lightweight and intuitive manner. Though we don't have a formal proof of the applicability, we expect that the REL extension can be instantiated for almost any dynamic logic, in the same way that one would expect that almost any dynamic logic could be extended (syntactically and semantically) with first-order operators. We have empirically shown the applicability by instantiating the REL extension for two well-known yet distinct dynamic logics. The two case studies demonstrate that both REL instantiations can easily and succinctly express important relational properties that cannot be easily captured by the host logic. In both case studies, existing tools can be used to verify specified relational properties.

Future Work. In this work, verification of REL formulas relies on our encodings of REL formulas in either the host logics or first-order constraints. The encodings allow us to reuse existing tools for relational verification. Meanwhile, our verification capability is also limited by existing tools and techniques. One direction we plan to explore is new proof techniques. We believe the natural expression of relational properties by the REL extension opens new opportunities to explore relational verification. Additional work, e.g., a novel proof system for the extension, might allow us to prove relational properties that cannot be proven by the existing tools and techniques. For example, the ability to verify mixed quantification of

executions is worth pursuing further, in particular, forall-exists properties which are often beyond the reach of many prior works. Another possible future work is a comprehensive evaluation and comparison with other relational reasoning tools.

References

[1] A. Sabelfeld and A. C. Myers, "Language-based information-flow security," *Journal on Selected Areas in Communications*, vol. 21, no. 1, pp. 5–19, 2003.

[2] J. Xiang, N. Fulton, and S. Chong, "Relational analysis of sensor attacks on Cyber-Physical Systems," in *CSF*, 2021.

[3] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *Journal of Computer Security*, vol. 18, no. 6, pp. 1157–1210, 2010.

[4] Y. Satake and H. Unno, "Propositional dynamic logic for higher-order functional programs," in *CAV*, 2018, pp. 105–123.

[5] W. Ahrendt, B. Beckert, R. Bubel, R. Hähnle, P. H. Schmitt, and M. Ulbrich, *Deductive Software Verification–The KeY Book: From Theory to Practice*. Springer, 2016, vol. 10001.

[6] A. Platzer, "Differential dynamic logic for hybrid systems," *Journal of Automated Reasoning*, vol. 41, no. 2, pp. 143–189, 2008.

[7] D. Harel, D. Kozen, and J. Tiuryn, *Dynamic Logic*. MIT Press, 2000.

[8] C. Scheben and P. H. Schmitt, "Verification of information flow properties of Java programs without approximations," in *FoVeOOS*, 2011, pp. 232–249.

[9] B. Beckert and S. Schlager, "A sequent calculus for first-order dynamic logic with trace modalities," in *IJCAR*, 2001, pp. 626–641.

[10] D. Grahl, "Deductive verification of concurrent programs and its application to secure information flow for Java," Ph.D. dissertation, Karlsruhe Institute of Technology, 29 Oct. 2015.

[11] C. Scheben, "Program-level specification and deductive verification of security properties," Ph.D. dissertation, Karlsruhe Institute of Technology, 2014.

[12] N. Fulton and A. Platzer, "Verifiably safe off-model reinforcement learning," in *TACAS*, 2019, pp. 413–430.

[13] F. Pottier and V. Simonet, "Information flow inference for ML," in *POPL*, 2002, pp. 319–330.

[14] A. Banerjee, R. Nagasamudram, D. A. Naumann, and M. Nikouei, "A relational program logic with data abstraction and dynamic framing," *arXiv preprint arXiv:1910.14560*, 2019.

[15] A. Platzer, "A complete uniform substitution calculus for differential dynamic logic," *Journal of Automated Reasoning*, vol. 59, no. 2, pp. 219–265, 2017.

[16] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *IJCAI*, 2013.

[17] G. Barthe, P. R. D'Argenio, and T. Rezk, "Secure information flow by self-composition," in *CSF*, 2004, pp. 100–114.

[18] D. Kozen, "Kleene algebra with tests," *TOPLAS*, vol. 19, no. 3, pp. 427–443, 1997.

[19] J. A. Goguen and J. Meseguer, "Security policies and security models," in *S&P*, 1982, pp. 11–20.

[20] P. Allen, "A comparison of non-interference and non-deducibility using CSP," in *CSF*, 1991, pp. 43–44.

[21] A. Platzer, *Logical foundations of cyber-physical systems*. Springer, 2018, vol. 662.

[22] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, "KeYmaera X: An axiomatic tactical theorem prover for hybrid systems," in *CADE*, 2015, pp. 527–538.

[23] H. Rahmani and J. M. O'Kane, "Optimal temporal logic planning with cascading soft constraints," in *IROS*, 2019, pp. 2524–2531.

[24] Y. Wang, S. Nalluri, and M. Pajic, "Hyperproperties for robotics: Planning via HyperLTL," in *ICRA*, 2020, pp. 8462–8468.

[25] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H. I. Christensen, and F. Dellaert, "Distributed trajectory estimation with privacy and communication constraints: a two-stage distributed Gauss-Seidel approach," in *ICRA*, 2016, pp. 5261–5268.

[26] A. Saboori and C. N. Hadjicostis, "Verification of initial-state opacity in security applications of DES," in *WODES*, 2008, pp. 328–333.

[27] L. Li, A. Bayuelo, L. Bobadilla, T. Alam, and D. A. Shell, "Coordinated multi-robot planning while preserving individual privacy," in *ICRA*, 2019, pp. 2188–2194.

[28] N. Francez, "Product properties and their direct verification," *Acta informatica*, vol. 20, no. 4, pp. 329–344, 1983.

[29] T. Terauchi and A. Aiken, "Secure information flow as a safety problem," in *SAS*, 2005, pp. 352–367.

[30] D. A. Naumann, "Thirty-seven years of relational hoare logic: remarks on its principles and history," in *ISoLA*, 2020, pp. 93–116.

[31] R. Shemer, A. Gurfinkel, S. Shoham, and Y. Vizel, "Property directed self composition," in *CAV*, 2019, pp. 161–179.

[32] H. Unno, T. Terauchi, and E. Koskinen, "Constraint-based relational verification," in *CAV*. Springer, 2021, pp. 742–766.

[33] S. Itzhaky, S. Shoham, and Y. Vizel, "Hyperproperty verification as chc satisfiability," *arXiv preprint arXiv:2304.12588*, 2023.

[34] J. Hu, E. Lu, D. A. Holland, M. Kawaguchi, S. Chong, and M. Seltzer, "Towards porting operating systems with program synthesis," *ACM Transactions on Programming Languages and Systems*, vol. 45, no. 1, pp. 1–70, 2023.

[35] S. Gao, J. Avigad, and E. M. Clarke, "δ-complete decision procedures for satisfiability over the reals," in *IJCAR*, 2012, pp. 286–300.

[36] S. Gao, S. Kong, and E. M. Clarke, "dReal: An SMT solver for nonlinear theories over the reals," in *CADE*, 2013, pp. 208–214.

[37] B. Beckert and D. Bruns, "Dynamic logic with trace semantics," in *CADE*, 2013, pp. 315–329.

[38] J. O. Gutsfeld, M. Müller-Olm, and C. Ohrem, "Propositional dynamic logic for hyperproperties," in *CONCUR*, 2020.

[39] S. M. Loos and A. Platzer, "Differential refinement logic," in *LICS*, 2016, pp. 505–514.

[40] B. Bohrer and A. Platzer, "A hybrid, dynamic logic for hybrid-dynamic information flow," in *LICS*, 2018, pp. 115–124.

[41] J. Kolčák, J. Dubut, I. Hasuo, S.-y. Katsumata, D. Sprunger, and A. Yamada, "Relational differential dynamic logic," in *TACAS*, 2020, pp. 191–208.

[42] L. Zheng and A. C. Myers, "Dynamic security labels and static information flow control," *International Journal of Information Security*, vol. 6, no. 2-3, pp. 67–84, 2007.

[43] P. Li and D. Zhang, "Towards a flow-and path-sensitive information flow analysis," in *CSF*, 2017, pp. 53–67.

[44] O. Arden and A. C. Myers, "A calculus for flow-limited authorization," in *CSF*, 2016, pp. 135–149.

[45] Á. Darvas, R. Hähnle, and D. Sands, "A theorem proving approach to analysis of secure information flow," in *PerCom*, 2005, pp. 193–209.

[46] G. Barthe, J. M. Crespo, and C. Kunz, "Relational verification using product programs," in *FM*, 2011, pp. 200–214.

[47] B. Churchill, O. Padon, R. Sharma, and A. Aiken, "Semantic program alignment for equivalence checking," in *PLDI*, 2019, pp. 1027–1040.

[48] A. C. Myers, "JFlow: practical mostly-static information flow control," in *POPL*, 1999, pp. 228–241.

[49] D. Stefan, A. Russo, J. C. Mitchell, and D. Mazières, "Flexible dynamic information flow control in Haskell," in *Haskell*, 2011, pp. 95–106.

[50] T. H. Austin and C. Flanagan, "Efficient purely-dynamic information flow analysis," in *PLAS*, 2009, pp. 113–124.

[51] P. Buiras, D. Vytiniotis, and A. Russo, "HLIO: mixing static and dynamic typing for information-flow control in Haskell," in *ICFP*, 2015, pp. 289–301.

[52] J. Xiang and S. Chong, "Co-Inflow: coarse-grained information flow control for Java-like languages," in *S&P*, 2021.

[53] D. Stefan, E. Z. Yang, P. Marchenko, A. Russo, D. Herman, B. Karp, and D. Mazières, "Protecting users by confining JavaScript with COWL," in *OSDI*, 2014, pp. 131–146.

[54] N. Benton, "Simple relational correctness proofs for static analyses and program transformations," in *POPL*, 2004, pp. 14–25.

[55] H. Yang, "Relational separation logic," *Theoretical Computer Science*, vol. 375, no. 1-3, pp. 308–334, 2007.

[56] R. Nagasamudram and D. A. Naumann, "Alignment completeness for relational hoare logics," in *LICS*, 2021, pp. 1–13.

[57] M. Sousa and I. Dillig, "Cartesian hoare logic for verifying k-safety properties," in *PLDI*, 2016, pp. 57–69.

[58] M. Sousa, I. Dillig, and S. K. Lahiri, "Verified three-way program merge," in *OOPSLA*, 2016, pp. 145–164.

[59] D. Felsing, S. Grebing, V. Klebanov, P. Rümmer, and M. Ulbrich, "Automating regression verification," in *ASE*, 2014, pp. 349–360.

[60] R. Sharma, E. Schkufza, B. Churchill, and A. Aiken, "Data-driven equivalence checking," in *OOPSLA*, 2013, pp. 391–406.

[61] M. Balliu, M. Dam, and R. Guanciale, "Automating information flow analysis of low level code," in *CCS*, 2014, pp. 1080–1091.

APPENDIX

A. Definitions for dℒ

Definition 4 (Variable set of dℒ). *The variable set of dℒ formulas, programs, and terms are defined inductively:*

$$\begin{aligned}
&\mathrm{VAR}(\theta \sim \delta) &&= \mathrm{VAR}(\theta) \cup \mathrm{VAR}(\delta)\\
&\mathrm{VAR}(\neg\phi) &&= \mathrm{VAR}(\phi)\\
&\mathrm{VAR}(\phi \wedge \psi) &&= \mathrm{VAR}(\phi) \cup \mathrm{VAR}(\psi)\\
&\mathrm{VAR}(\forall x.\ \phi) &&= \{x\} \cup \mathrm{VAR}(\phi)\\
&\mathrm{VAR}([\alpha]\phi) &&= \mathrm{VAR}(\alpha) \cup \mathrm{VAR}(\phi)\\
&\mathrm{VAR}(\langle\alpha\rangle\phi) &&= \mathrm{VAR}(\alpha) \cup \mathrm{VAR}(\phi)\\
&\mathrm{VAR}(x := \theta) &&= \{x\} \cup \mathrm{VAR}(\theta)\\
&\mathrm{VAR}(x := *) &&= \{x\}\\
&\mathrm{VAR}(?\phi) &&= \mathrm{VAR}(\phi)\\
&\mathrm{VAR}(\alpha\,;\,\beta) &&= \mathrm{VAR}(\alpha) \cup \mathrm{VAR}(\beta)\\
&\mathrm{VAR}(\alpha \cup \beta) &&= \mathrm{VAR}(\alpha) \cup \mathrm{VAR}(\beta)\\
&\mathrm{VAR}(\alpha^*) &&= \mathrm{VAR}(\alpha)\\
&\mathrm{VAR}(x' = \theta \,\&\, \phi) &&= \{x\} \cup \mathrm{VAR}(\theta) \cup \mathrm{VAR}(\phi)\\
&\mathrm{VAR}(x) &&= \{x\}\\
&\mathrm{VAR}(c) &&= \varnothing\\
&\mathrm{VAR}(\theta \oplus \delta) &&= \mathrm{VAR}(\theta) \cup \mathrm{VAR}(\delta)
\end{aligned}$$

Definition 5 (Left variable set). *The left variable sets of $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ formulas, programs, and terms are defined inductively:*

$$\begin{aligned}
&\mathrm{VAR}_{\mathrm{L}}(\theta_{\mathrm{T}} \sim \delta_{\mathrm{T}}) &&= \mathrm{VAR}_{\mathrm{L}}(\theta_{\mathrm{T}}) \cup \mathrm{VAR}_{\mathrm{L}}(\delta_{\mathrm{T}})\\
&\mathrm{VAR}_{\mathrm{L}}(\neg\phi_{\mathrm{T}}) &&= \mathrm{VAR}_{\mathrm{L}}(\phi_{\mathrm{T}})\\
&\mathrm{VAR}_{\mathrm{L}}(\phi_{\mathrm{T}} \wedge \psi_{\mathrm{T}}) &&= \mathrm{VAR}_{\mathrm{L}}(\phi_{\mathrm{T}}) \cup \mathrm{VAR}_{\mathrm{L}}(\psi_{\mathrm{T}})\\
&\mathrm{VAR}_{\mathrm{L}}(\langle\!\langle \alpha_{\mathrm{T}} \rangle\!\rangle \phi_{\mathrm{T}}) &&= \mathrm{VAR}_{\mathrm{L}}(\alpha_{\mathrm{T}}) \cup \mathrm{VAR}_{\mathrm{L}}(\phi_{\mathrm{T}})\\
&\mathrm{VAR}_{\mathrm{L}}(\lfloor\phi\rfloor_{B}) &&= \mathrm{VAR}(\phi) \text{ if } B = \mathsf{L} \text{ else } \varnothing\\
&\mathrm{VAR}_{\mathrm{L}}((\alpha,\beta)) &&= \mathrm{VAR}(\alpha)\\
&\mathrm{VAR}_{\mathrm{L}}(\alpha_{\mathrm{T}}\,;\,\beta_{\mathrm{T}}) &&= \mathrm{VAR}_{\mathrm{L}}(\alpha_{\mathrm{T}}) \cup \mathrm{VAR}_{\mathrm{L}}(\beta_{\mathrm{T}})\\
&\mathrm{VAR}_{\mathrm{L}}(\alpha_{\mathrm{T}} \cup \beta_{\mathrm{T}}) &&= \mathrm{VAR}_{\mathrm{L}}(\alpha_{\mathrm{T}}) \cup \mathrm{VAR}_{\mathrm{L}}(\beta_{\mathrm{T}})\\
&\mathrm{VAR}_{\mathrm{L}}(\alpha_{\mathrm{T}}^*) &&= \mathrm{VAR}_{\mathrm{L}}(\alpha_{\mathrm{T}})\\
&\mathrm{VAR}_{\mathrm{L}}(?\phi_{\mathrm{T}}) &&= \mathrm{VAR}_{\mathrm{L}}(\phi_{\mathrm{T}})\\
&\mathrm{VAR}_{\mathrm{L}}(\lfloor\theta\rfloor_{B}) &&= \mathrm{VAR}(\theta) \text{ if } B = \mathsf{L} \text{ else } \varnothing\\
&\mathrm{VAR}_{\mathrm{L}}(\theta_{\mathrm{T}_1} \oplus \theta_{\mathrm{T}_2}) &&= \mathrm{VAR}_{\mathrm{L}}(\theta_{\mathrm{T}}) \cup \mathrm{VAR}_{\mathrm{L}}(\delta_{\mathrm{T}})
\end{aligned}$$

The definition of $\mathrm{VAR}_{\mathrm{R}}(\phi_{\mathrm{T}})$, the right variable set, is identical to $\mathrm{VAR}_{\mathrm{L}}(\phi_{\mathrm{T}})$, except for the projection term and formula:

$$\mathrm{VAR}_{\mathrm{R}}(\lfloor\phi\rfloor_{B}) = \mathrm{VAR}(\phi) \text{ if } B = \mathsf{R} \text{ else } \varnothing$$
$$\mathrm{VAR}_{\mathrm{R}}(\lfloor\theta\rfloor_{B}) = \mathrm{VAR}(\theta) \text{ if } B = \mathsf{R} \text{ else } \varnothing$$

Definition 6 (Renaming functions for $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$). *Renaming $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ formulas, programs, and terms with a renaming function ξ are defined inductively by the following:*

$$\begin{aligned}
&\xi(\theta_{\mathrm{T}} \sim \delta_{\mathrm{T}}) &&\equiv \xi(\theta_{\mathrm{T}}) \sim \xi(\delta_{\mathrm{T}})\\
&\xi(\neg\phi_{\mathrm{T}}) &&\equiv \neg\xi(\phi_{\mathrm{T}})\\
&\xi(\phi_{\mathrm{T}} \wedge \psi_{\mathrm{T}}) &&\equiv \xi(\phi_{\mathrm{T}}) \wedge \xi(\psi_{\mathrm{T}})\\
&\xi(\langle\!\langle \alpha_{\mathrm{T}} \rangle\!\rangle \phi_{\mathrm{T}}) &&\equiv \langle\!\langle \xi(\alpha_{\mathrm{T}}) \rangle\!\rangle \xi(\phi_{\mathrm{T}})\\
&\xi(\lfloor\phi\rfloor_{B}) &&\equiv \lfloor\xi(\phi)\rfloor_{B} \text{ if } B = \mathsf{R} \text{ else } \lfloor\phi\rfloor_{B}\\
&\xi((\alpha,\beta)) &&\equiv (\alpha, \xi(\beta))\\
&\xi(\alpha_{\mathrm{T}}\,;\,\beta_{\mathrm{T}}) &&\equiv \xi(\alpha_{\mathrm{T}})\,;\,\xi(\beta_{\mathrm{T}})\\
&\xi(\alpha_{\mathrm{T}} \cup \beta_{\mathrm{T}}) &&\equiv \xi(\alpha_{\mathrm{T}}) \cup \xi(\beta_{\mathrm{T}})\\
&\xi(\alpha_{\mathrm{T}}^*) &&\equiv (\xi(\alpha_{\mathrm{T}}))^*\\
&\xi(\lfloor\theta\rfloor_{B}) &&\equiv \lfloor\xi(\theta)\rfloor_{B} \text{ if } B = \mathsf{R} \text{ else } \lfloor\theta\rfloor_{B}\\
&\xi(\theta_{\mathrm{T}} \oplus \delta_{\mathrm{T}}) &&\equiv \xi(\theta_{\mathrm{T}}) \oplus \xi(\delta_{\mathrm{T}})
\end{aligned}$$

Where $\xi(\beta)$, $\xi(\phi)$, and $\xi(\theta)$ do substitution for LDL_f program β formula ϕ, and term θ according to ξ.

Renaming functions also apply to states as follows:

$$\xi(\omega) = \begin{cases} \xi(x) \mapsto \omega(x) & \text{if } x \in dom(\xi)\\ x \mapsto \omega(x) & x \in dom(\omega) \setminus dom(\xi) \end{cases}$$

B. Definitions for LDL_f

Definition 7 (LDL_f Variable set). *The variable set of LDL_f formulas and programs are defined inductively as follows:*

$$\begin{aligned}
&\mathrm{VAR}(A) &&= \{A\}\\
&\mathrm{VAR}(\neg\phi) &&= \mathrm{VAR}(\phi)\\
&\mathrm{VAR}(\phi \wedge \psi) &&= \mathrm{VAR}(\phi) \cup \mathrm{VAR}(\psi)\\
&\mathrm{VAR}(\langle\alpha\rangle\phi) &&= \mathrm{VAR}(\alpha) \cup \mathrm{VAR}(\phi)\\
&\mathrm{VAR}(\phi_{\mathcal{AP}}) &&= \textit{all atomic propositions used in } \phi_{\mathcal{AP}}\\
&\mathrm{VAR}(?\phi) &&= \mathrm{VAR}(\phi)\\
&\mathrm{VAR}(\alpha\,;\,\beta) &&= \mathrm{VAR}(\alpha \cup \beta) = \mathrm{VAR}(\alpha) \cup \mathrm{VAR}(\beta)\\
&\mathrm{VAR}(\alpha^*) &&= \mathrm{VAR}(\alpha)
\end{aligned}$$

The definitions on left (and right) variable sets and renaming for LDL_f are the same as (or subset of) the ones for dℒ.

C. Proof of Theorem 2

$\mathrm{LDL}_{\mathrm{REL}}$ has at least the expressive power of LDL_f, since all LDL_f formulas can be encoded in $\mathrm{LDL}_{\mathrm{REL}}$ as projection formulas. We focus on proving that some $\mathrm{LDL}_{\mathrm{REL}}$ formulas cannot be expressed with LDL_f. We proceed by *disproving* the following statement: for all $\mathrm{LDL}_{\mathrm{REL}}$ formula ϕ_{T}, and trace σ_{L}, σ_{R}, there exists a LDL_f formula ϕ, a trace σ_ϕ constructed by interleaving trace σ_{L} and σ_{R} such that $(\sigma_{\mathrm{L}}, \sigma_{\mathrm{R}}) \models_{\mathrm{RL}} \phi_{\mathrm{T}}$ if and only if $\sigma_\phi \models \phi$.

As a counterexample, we show that ϕ *cannot* be encoded as a regular expression. Consider the following formula ϕ_{T}:

$$\langle\!\langle (a\,;a,b)^* \rangle\!\rangle \langle\!\langle (?\top, \neg b)^* \rangle\!\rangle (\lfloor\neg a\rfloor_{\mathrm{L}} \wedge \lfloor b\rfloor_{\mathrm{R}})$$
$$\wedge \quad \langle\!\langle (a,b)^* \rangle\!\rangle \langle\!\langle (a,\neg b)^* \rangle\!\rangle (\lfloor\neg a\rfloor_{\mathrm{L}} \wedge \lfloor b\rfloor_{\mathrm{R}})$$

Where, a (or b) is an atomic proposition of the left trace σ_{L} (or right trace σ_{R}). This $\mathrm{LDL}_{\mathrm{REL}}$ formula is satisfiable by a pair of traces, σ_{L} and σ_{R}, if (1) the number of states in σ_{R} where b holds is the same as the number of states in σ_{R} where $\neg b$ holds, (2) the number of states in σ_{L} where a holds is twice the number of states in σ_{R} where b (or $\neg b$) holds, and (3) σ_{L} and σ_{R} end, respectively, with a state where $\neg a$ and b hold. For example, σ_{L} and σ_{R} can be, respectively, $a^{2k}(\neg a)$ and $b^k(\neg b)^k b$, for some $k > 0$. Here, the notation $a^{2k}\neg a$ means σ_{L} is a trace with 2k+1 states, where the first 2k states satisfy a and the last state satisfies $\neg a$.

Suppose that there exist a LDL_f formula ϕ and a trace σ_ϕ constructed by interleaving trace σ_{L} and σ_{R}, such that $(\sigma_{\mathrm{L}}, \sigma_{\mathrm{R}}) \models_{\mathrm{RL}} \phi_{\mathrm{T}}$ if and only if $\sigma_\phi \models \phi$. Assume ϕ is a regular language. Traces that satisfy ϕ (e.g., σ_ϕ) correspond to strings in the language. Let L denote the language specified by ϕ, and so by the pumping lemma, there exists an integer $n \geq 1$, such that for all string (traces) $w \in L$ with $|w| \geq n$, there exist x, y, z such that $w = xyz$, and (1) $|xy| \leq n$, (2) $|y| \geq 1$, and (3) for all $i \geq 0 : xy^i z \in L$. Assume that some integer n exists as required by the lemma. Let w be a string that has $2n$ instances of a, n instances of b and $\neg b$ before the last state on atomic proposition b, and its last state for atomic proposition a (or b)

satisfies $\neg a$ (or b). Then, for xy^iz to satisfy the second conjunct of $\phi_\mathfrak{r}$, y can only be a substring that contains the same number of a and b, since $|xy| \leq n$ and w should have n instances of b followed by n instances of $\neg b$. Assume y contains j instances of a and b, then xz (i.e., i=0) would have $2n-j$ instances of a and $n-j$ instances of b. Such a string contradicts the first conjunct, in particular, the part of $(a\,;a,b)^*$. That means, xy^iz cannot satisfy the first conjunct for all $i \geq 0$. Therefore, ϕ cannot be regular, and some $\mathrm{LDL}_{\mathrm{REL}}$ formulas cannot be encoded with LDL_f.

D. Proof of Theorem 3

We use $\omega_\mathtt{L} \otimes_{\phi_\mathfrak{r}} \omega_\mathtt{R}$ as a shorthand of $((\omega_\mathtt{L} \Downarrow \mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r})) \otimes (\omega_\mathtt{R} \Downarrow \mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r})))$. Same notations apply for $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ terms and programs. We write $\omega \Downarrow \alpha$ to mean $\omega \Downarrow \mathrm{VAR}(\alpha)$, which also applies to terms and formulas, and $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ terms, programs, and formulas. We write IH for induction hypothesis. In induction on $\phi_\mathfrak{r}$, we only consider the following cases: $\theta_{\mathfrak{r}_1} \sim \theta_{\mathfrak{r}_2}$, $\lfloor\phi\rfloor_\mathtt{B}$, $\neg\phi_\mathfrak{r}$, $\phi_\mathfrak{r} \wedge \psi_\mathfrak{r}$, and $\langle\!\langle \alpha_\mathfrak{r} \rangle\!\rangle \phi_\mathfrak{r}$. Other formulas can be encoded.

Lemma 1 (Renaming preserves term value)**.** *For states $\omega_\mathtt{L}, \omega_\mathtt{R}$, a $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ term $\theta_\mathfrak{r}$ and a renaming function ξ on $\theta_\mathfrak{r}$,*

$$(\omega_\mathtt{L}, \omega_\mathtt{R})[\![\theta_\mathfrak{r}]\!]_{\mathrm{RL}} = (\omega_\mathtt{L}, \xi(\omega_\mathtt{R}))[\![\xi(\theta_\mathfrak{r})]\!]_{\mathrm{RL}}$$

Proof. By induction on $\theta_\mathfrak{r}$. □

Lemma 2 (Renaming preserves formula value)**.** *For states $\omega_\mathtt{L}, \omega_\mathtt{R}$, a $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ formula $\phi_\mathfrak{r}$ and a renaming function ξ of $\phi_\mathfrak{r}$,*

$$(\omega_\mathtt{L}, \omega_\mathtt{R}) \models_{\mathrm{RL}} \phi_\mathfrak{r} \text{ iff } (\omega_\mathtt{L}, \xi(\omega_\mathtt{R})) \models \xi(\phi_\mathfrak{r})$$

Proof. By induction on $\phi_\mathfrak{r}$ (and simultaneous induction on $\alpha_\mathfrak{r}$ for programs), and Lemma 1. □

Lemma 3. *For a $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ formula $\phi_\mathfrak{r}$ such that $\mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r}) \cap \mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r}) = \varnothing$, then $\mathrm{VAR}_\mathtt{L}(\psi_\mathfrak{r}) \cap \mathrm{VAR}_\mathtt{R}(\psi_\mathfrak{r}) = \varnothing$ for all subformulas $\psi_\mathfrak{r}$ in $\phi_\mathfrak{r}$.*

Proof. By simultaneous induction on $\phi_\mathfrak{r}$ and $\alpha_\mathfrak{r}$. □

Similar lemmas can be proven for subprograms and subterms in $\phi_\mathfrak{r}$.

Lemma 4. *For a state ω, a formula ϕ such that $\mathrm{VAR}(\phi) \subseteq \mathrm{VAR}(\omega)$, then $\omega \models \phi$ iff $(\omega \Downarrow \phi) \models \phi$.*

Proof. By the semantics of $\models$ and induction on ϕ (and α for programs). □

Lemma 5. *For states ω_1, ω_2, a formula ϕ such that $\mathrm{VAR}(\omega_1) \cap \mathrm{VAR}(\omega_2) = \varnothing$, if $\omega_1 \models \phi$ then $\omega_1 \otimes \omega_2 \models \phi$.*

Proof. By induction on the number of variables in ω_1. □

Lemma 6. *For states $\omega_\mathtt{L}, \omega_\mathtt{R}, \nu_\mathtt{L}, \nu_\mathtt{R}$, programs α, β such that $\mathrm{VAR}(\alpha) \cap \mathrm{VAR}(\beta) = \varnothing$, then $(\omega_\mathtt{L}, \nu_\mathtt{L}) \in [\![\alpha]\!]$ and $(\omega_\mathtt{R}, \nu_\mathtt{R}) \in [\![\beta]\!]$ iff $(\omega_\mathtt{L} \Downarrow \alpha \otimes \omega_\mathtt{R} \Downarrow \beta, \nu_\mathtt{L} \Downarrow \alpha \otimes \nu_\mathtt{R} \Downarrow \beta) \in [\![\alpha\,;\beta]\!]$.*

Proof. By Lemma 4 and the semantics. □

Lemma 7. *For states $\omega_\mathtt{L}, \omega_\mathtt{R}, \omega_1, \omega_2$, a formula $\phi_\mathfrak{r}$ such that $\mathrm{VAR}(\omega_1) \cap \mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r}) = \varnothing$, $\mathrm{VAR}(\omega_2) \cap \mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r}) = \varnothing$, $\mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r}) \subseteq \mathrm{VAR}(\omega_\mathtt{L})$, $\mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r}) \subseteq \mathrm{VAR}(\omega_\mathtt{R})$, $\mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r}) \cap \mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r}) = \varnothing$, $(\omega_\mathtt{L} \otimes \omega_1, \omega_\mathtt{R} \otimes \omega_2) \models_{\mathrm{RL}} \phi_\mathfrak{r}$, then $(\omega_\mathtt{L}, \omega_\mathtt{R}) \models_{\mathrm{RL}} \phi_\mathfrak{r}$.*

Proof. By simultaneous induction on $\phi_\mathfrak{r}$ and $\alpha_\mathfrak{r}$. □

Lemma 8. *For states $\omega_\mathtt{L}, \omega_\mathtt{R}, \omega_1, \omega_2$, a formula $\phi_\mathfrak{r}$ such that $\mathrm{VAR}(\omega_1) \cap \mathrm{VAR}(\omega_\mathtt{L}) = \varnothing$, $\mathrm{VAR}(\omega_2) \cap \mathrm{VAR}(\omega_\mathtt{R}) = \varnothing$, $\mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r}) \cap \mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r}) = \varnothing$, $(\omega_\mathtt{L}, \omega_\mathtt{R}) \models_{\mathrm{RL}} \phi_\mathfrak{r}$, then $(\omega_\mathtt{L} \otimes \omega_1, \omega_\mathtt{R} \otimes \omega_2) \models_{\mathrm{RL}} \phi_\mathfrak{r}$.*

Proof. By simultaneous induction on $\phi_\mathfrak{r}$ and $\alpha_\mathfrak{r}$. □

Lemma 9. *For states $\omega_\mathtt{L}, \omega_\mathtt{R}$, a $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ term $\theta_\mathfrak{r}$ that $\mathrm{VAR}_\mathtt{L}(\theta_\mathfrak{r}) \cap \mathrm{VAR}_\mathtt{R}(\theta_\mathfrak{r}) = \varnothing$, then $(\omega_\mathtt{L}, \omega_\mathtt{R})[\![\theta_\mathfrak{r}]\!]_{\mathrm{RL}} = (\omega_\mathtt{L} \otimes_{\theta_\mathfrak{r}} \omega_\mathtt{R})[\![\pi(\theta_\mathfrak{r})]\!]$*

Proof. By induction on $\theta_\mathfrak{r}$. □

Lemma 10. *For a state ω, a $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ formula $\phi_\mathfrak{r}$ that $\mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r}) \cap \mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r}) = \varnothing$ and $\omega \models \pi(\phi_\mathfrak{r})$, then $(\omega \Downarrow \mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r}), \omega \Downarrow \mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r})) \models_{\mathrm{RL}} \phi_\mathfrak{r}$.*

Proof. By induction on $\phi_\mathfrak{r}$ (and simultaneous induction on $\alpha_\mathfrak{r}$ for programs):

- Case $\phi_\mathfrak{r}$ is of the form $\theta_\mathfrak{r} \sim \delta_\mathfrak{r}$: by Lemma 9.
- Case $\phi_\mathfrak{r}$ is of the form $\lfloor\phi\rfloor_\mathtt{B}$: by the definition of π and case analysis of $\mathtt{B}$.
- Case $\phi_\mathfrak{r}$ is of the form $\neg\psi_\mathfrak{r}$: by IH.
- Case $\phi_\mathfrak{r}$ is of the form $\phi_{\mathfrak{r}_1} \wedge \phi_{\mathfrak{r}_2}$: by IH and Lemma 8.
- Case $\phi_\mathfrak{r}$ is of the form $\langle\!\langle \alpha_\mathfrak{r} \rangle\!\rangle \psi_\mathfrak{r}$: Cases $\alpha_{\mathfrak{r}_1}\,;\,\beta_{\mathfrak{r}_2}$, $\alpha_{\mathfrak{r}_1} \cup \beta_{\mathfrak{r}_2}$, $?\phi_\mathfrak{r}$, and $\alpha^*_{\mathfrak{r}_1}$ can be proved using IH and Lemma 8. For the case that $\alpha_\mathfrak{r}$ is of the form (α, β): by the semantics of $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ programs, we know that there exists ν_1, ν_2 such that $(\omega, \nu_1) \in [\![\alpha]\!]$, $(\nu_1, \nu_2) \in [\![\beta]\!]$, and $\nu_2 \models \pi(\psi_\mathfrak{r})$. Also, $\omega \Downarrow \mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r}) = \nu_1 \Downarrow \mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r})$ and $\nu_1 \Downarrow \mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r}) = \nu_2 \Downarrow \mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r})$. Thus, we know $((\omega \Downarrow \mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r}), \omega \Downarrow \mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r})), (\nu_2 \Downarrow \mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r}), \nu_2 \Downarrow \mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r}))) \in [\![(\alpha, \beta)]\!]_{\mathrm{RL}}$ Then by the semantics of $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ programs, IH, and Lemma 7, this case is proven.

That concludes the proof. □

Lemma 11. *For states $\omega_\mathtt{L}$ $\omega_\mathtt{R}$, a $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ formula $\phi_\mathfrak{r}$ that $\mathrm{VAR}_\mathtt{L}(\phi_\mathfrak{r}) \cap \mathrm{VAR}_\mathtt{R}(\phi_\mathfrak{r}) = \varnothing$,*

$$(\omega_\mathtt{L}, \omega_\mathtt{R}) \models_{\mathrm{RL}} \phi_\mathfrak{r} \text{ iff } (\omega_\mathtt{L} \otimes_{\phi_\mathfrak{r}} \omega_\mathtt{R}) \models \pi(\phi_\mathfrak{r})$$

Proof. Both directions can be proved by induction on $\phi_\mathfrak{r}$ (and simultaneous induction on $\alpha_\mathfrak{r}$ for programs). Many cases use Lemma 3 and 5, which we refer to as the auxiliary lemmas. We first prove if $(\omega_\mathtt{L}, \omega_\mathtt{R}) \models_{\mathrm{RL}} \phi_\mathfrak{r}$ then $(\omega_\mathtt{L} \otimes_{\phi_\mathfrak{r}} \omega_\mathtt{R}) \models \pi(\phi_\mathfrak{r})$:

- Case $\phi_\mathfrak{r}$ is of the form $\theta_\mathfrak{r} \sim \delta_\mathfrak{r}$: by examining the definition of π and Lemma 9.
- Case $\phi_\mathfrak{r}$ is of the form $\lfloor\phi\rfloor_\mathtt{B}$: by examining the definition of π and case analysis of $\mathtt{B}$.
- Case $\phi_\mathfrak{r}$ is of the form $\neg\psi_\mathfrak{r}$: by IH.
- Case $\phi_\mathfrak{r}$ is of the form $\phi_{\mathfrak{r}_1} \wedge \phi_{\mathfrak{r}_2}$: by IH and the auxiliary lemmas.
- Case $\phi_\mathfrak{r}$ is of the form $\langle\!\langle \alpha_\mathfrak{r} \rangle\!\rangle \psi_\mathfrak{r}$: cases $\alpha_{\mathfrak{r}_1}\,;\,\beta_{\mathfrak{r}_2}$, $\alpha_{\mathfrak{r}_1} \cup \beta_{\mathfrak{r}_2}$, $?\phi_\mathfrak{r}$, and $\alpha^*_{\mathfrak{r}_1}$ can be proved using IH and auxiliary lemmas. For the case of the form (α, β): by the semantics of $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ programs, we know that there exists $\nu_\mathtt{L}, \nu_\mathtt{R}$ such that $(\omega_\mathtt{L}, \nu_\mathtt{L}) \in [\![\alpha]\!]$ and $(\omega_\mathtt{R}, \nu_\mathtt{R}) \in [\![\beta]\!]$, then by IH and Lemma 6.

The other direction can be proven by Lemma 10 and 8. □

Proof of Theorem 3. By Lemma 11 and 2. □

E. Encoding LDL_{REL} Formulas with LDL_f

By Theorem 2, we know it is *impossible* to design a sound and complete encoding of LDL_{REL} formulas in LDL_f. However, we can still take advantage of existing tools for LDL_f to verify LDL_{REL} formulas, by carefully designing a sound encoding of a *subset* of LDL_{REL} formulas in LDL_f.

Before diving into the details of the encoding. We demonstrate why the encoding of $\mathsf{d}\mathcal{L}_{\text{REL}}$ will not fit LDL_{REL}. A key step of the encoding of $\mathsf{d}\mathcal{L}_{\text{REL}}$ is to sequentially compose two host logic programs in a biprogram. This is problematic for LDL_{REL}. In particular, it may create a *misalignment* of states in the case of conjunctions. For example, consider a LDL_{REL} formula $\langle\!\langle (a,b)\rangle\!\rangle \lfloor \top \rfloor_{\text{R}} \wedge \langle\!\langle (?\top, \neg b)\rangle\!\rangle \lfloor \top \rfloor_{\text{R}}$, where a and b are atomic propositions, the formula is trivially *unsatisfiable* due to conflicting values of b. However, applying the encoding of $\mathsf{d}\mathcal{L}_{\text{REL}}$ on the formula produces a *satisfiable* formula $\langle a;b\rangle\top \wedge \langle ?\top;\neg b\rangle\top$. The encoding won't work for LDL_{REL}.

Now, we introduce the encoding for LDL_{REL}, which differs greatly from the one for $\mathsf{d}\mathcal{L}_{\text{REL}}$. Here, we only consider renamed LDL_{REL} formulas, i.e., $\phi_{\mathfrak{r}}$ that $\text{VAR}_{\text{L}}(\phi_{\mathfrak{r}}) \cap \text{VAR}_{\text{R}}(\phi_{\mathfrak{r}}) = \varnothing$. The definition of renaming functions for LDL_{REL} are the same as the ones for $\mathsf{d}\mathcal{L}_{\text{REL}}$. Definitions of $\text{VAR}_{\text{L}}(\phi_{\mathfrak{r}})$ and $\text{VAR}_{\text{R}}(\phi_{\mathfrak{r}})$ for LDL_{REL} can be found in Appendix A.

a) From LDL_{REL} formulas to LDL_f formulas: We develop a function π that will be used to transform LDL_{REL} formulas into LDL_f formulas. It is designed based on the following insight: a LDL_{REL} formula (after renaming) is satisfiable if it is a conjunction of a characterization of the left trace and a characterization of the right trace. For example, consider again the formula $\langle\!\langle (a,b)\rangle\!\rangle \lfloor \top \rfloor_{\text{R}} \wedge \langle\!\langle (?\top, \neg b)\rangle\!\rangle \lfloor \top \rfloor_{\text{R}}$, it is satisfiable if and only if two LDL_f formulas are satisfiable: $\langle a\rangle\top$ (left trace) and $\langle b\rangle\top \wedge \langle \neg b\rangle\top$ (right trace). Since the two formulas refer to different variables, a single trace that satisfies their conjunction corresponds to a pair of traces that satisfy the LDL_{REL} formula.

For a LDL_{REL} formula $\phi_{\mathfrak{r}}$, there could be many such LDL_f conjunctions each of which is satisfiable only if $\phi_{\mathfrak{r}}$ is satisfiable. These conjunctions correspond to different execution paths introduced by disjunction and nondeterministic choice in $\phi_{\mathfrak{r}}$. For example, a LDL_{REL} formula $\langle\!\langle ((a;a)\cup a, b)\rangle\!\rangle \lfloor \top \rfloor_{\text{R}}$ is satisfiable if either of the following LDL_f formulas is satisfiable: $\langle a\rangle\top \wedge \langle b\rangle\top$ or $\langle a;a\rangle\top \wedge \langle b\rangle\top$.

Based on these insights, we develop a π function to find a set of LDL_f conjunctions each of which is satisfiable only if the input LDL_{REL} formula is satisfiable. π is a *partial* function that works on a subset of LDL_{REL} formulas. It is sound but incomplete for LDL_{REL}, in particular, for LDL_{REL} formulas that have loop constructs. For formulas of the form $\langle\!\langle \alpha_{\mathfrak{r}}^* \rangle\!\rangle \phi_{\mathfrak{r}}$, the π function unfolds the loop for a *finite* number of times. It considers some possible execution paths of $\alpha_{\mathfrak{r}}$, but not all of them. For formulas of the form $[\![\alpha_{\mathfrak{r}}^*]\!] \phi_{\mathfrak{r}}$, the π function can only process the formula if $\alpha_{\mathfrak{r}}$ is test-only, i.e., its left and right programs are both test-only. We will discuss techniques to help relax such a limitation later.

Now we introduce the definition of π. We put LDL_{REL} formulas in *negation normal form*, by exploiting abbreviations and pushing negation inside (e.g., $\neg [\![\alpha_{\mathfrak{r}}]\!] \phi_{\mathfrak{r}} = \langle\!\langle \alpha_{\mathfrak{r}} \rangle\!\rangle \neg \phi_{\mathfrak{r}}$) as much as possible, leaving negations only in front of atomic formulas. Essentially, we restrict our attention to LDL_{REL} formulas formed by the following syntax:

$$\phi_{\mathfrak{r}}, \psi_{\mathfrak{r}} ::= \lfloor \phi \rfloor_{\mathfrak{s}} \mid \neg \lfloor \phi \rfloor_{\mathfrak{s}} \mid \phi_{\mathfrak{r}} \vee \psi_{\mathfrak{r}} \mid \phi_{\mathfrak{r}} \wedge \psi_{\mathfrak{r}} \mid \langle\!\langle \alpha_{\mathfrak{r}} \rangle\!\rangle \phi_{\mathfrak{r}} \mid [\![\alpha_{\mathfrak{r}}]\!] \phi_{\mathfrak{r}}$$
$$\alpha_{\mathfrak{r}}, \beta_{\mathfrak{r}} ::= (\alpha, \beta) \mid \alpha_{\mathfrak{r}}; \beta_{\mathfrak{r}} \mid \alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}} \mid \alpha_{\mathfrak{r}}^* \mid ?\phi_{\mathfrak{r}}$$

The π function is defined as follows:

Definition 8 (π for LDL_{REL})**.** *For a LDL_{REL} formula $\phi_{\mathfrak{r}}$ such that $\text{VAR}_{\text{L}}(\phi_{\mathfrak{r}}) \cap \text{VAR}_{\text{R}}(\phi_{\mathfrak{r}}) = \varnothing$, a partial function $\pi(\phi_{\mathfrak{r}})$ that produces a set of pairs of LDL_f formulas is inductively defined as follows:*

$$\pi(\lfloor \phi \rfloor_{\mathfrak{s}}) \equiv \{(\phi, ?\top)\} \text{ if } \mathfrak{s}=\text{L} \text{ or } \{(?\top, \phi)\} \text{ if } \mathfrak{s}=\text{R}$$
$$\pi(\neg \lfloor \phi \rfloor_{\mathfrak{s}}) \equiv \{(\neg\phi, ?\top)\} \text{ if } \mathfrak{s}=\text{L} \text{ or } \{(?\top, \neg\phi)\} \text{ if } \mathfrak{s}=\text{R}$$
$$\pi(\phi_{\mathfrak{r}} \wedge \psi_{\mathfrak{r}}) \equiv \pi(\phi_{\mathfrak{r}}) \otimes \pi(\psi_{\mathfrak{r}}) \text{ where } S_A \otimes S_B \text{ is defined as } \{(\phi \wedge \phi', \psi \wedge \psi') \mid (\phi,\psi) \in S_A, (\phi',\psi') \in S_B\}$$
$$\pi(\phi_{\mathfrak{r}} \vee \psi_{\mathfrak{r}}) \equiv \pi(\phi_{\mathfrak{r}}) \cup \pi(\psi_{\mathfrak{r}}) \text{ where } \cup \text{ is set union}$$
$$\pi(\langle\!\langle \alpha_{\mathfrak{r}}; \beta_{\mathfrak{r}} \rangle\!\rangle \phi_{\mathfrak{r}}) \equiv \pi(\langle\!\langle \alpha_{\mathfrak{r}} \rangle\!\rangle \langle\!\langle \beta_{\mathfrak{r}} \rangle\!\rangle \phi_{\mathfrak{r}})$$
$$\pi(\langle\!\langle \alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}} \rangle\!\rangle \phi_{\mathfrak{r}}) \equiv \pi(\langle\!\langle \alpha_{\mathfrak{r}} \rangle\!\rangle \phi_{\mathfrak{r}} \vee \langle\!\langle \beta_{\mathfrak{r}} \rangle\!\rangle \phi_{\mathfrak{r}})$$
$$\pi(\langle\!\langle \alpha_{\mathfrak{r}}^* \rangle\!\rangle \phi_{\mathfrak{r}}) \equiv \begin{cases} \pi(\langle\!\langle \alpha_{\mathfrak{r}} \rangle\!\rangle \phi_{\mathfrak{r}}) & \text{if } \alpha_{\mathfrak{r}} \text{ is test-only} \\ \pi(\langle\!\langle (?\top, ?\top) \cup \alpha_{\mathfrak{r}} \cup \cdots \alpha_{\mathfrak{r}}^n \rangle\!\rangle \phi_{\mathfrak{r}}) & \text{else} \end{cases}$$
$$\pi(\langle\!\langle ?\psi_{\mathfrak{r}} \rangle\!\rangle \phi_{\mathfrak{r}}) \equiv \pi(\psi_{\mathfrak{r}} \wedge \phi_{\mathfrak{r}})$$
$$\pi(\langle\!\langle (\alpha,\beta) \rangle\!\rangle \phi_{\mathfrak{r}}) \equiv \{(\langle\alpha\rangle\phi, \langle\beta\rangle\psi) \mid (\phi,\psi) \in \pi(\phi_{\mathfrak{r}})\}$$
$$\pi([\![\alpha_{\mathfrak{r}}; \beta_{\mathfrak{r}}]\!] \phi_{\mathfrak{r}}) \equiv \pi([\![\alpha_{\mathfrak{r}}]\!] [\![\beta_{\mathfrak{r}}]\!] \phi_{\mathfrak{r}})$$
$$\pi([\![\alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}}]\!] \phi_{\mathfrak{r}}) \equiv \pi([\![\alpha_{\mathfrak{r}}]\!] \phi_{\mathfrak{r}} \wedge [\![\beta_{\mathfrak{r}}]\!] \phi_{\mathfrak{r}})$$
$$\pi([\![\alpha_{\mathfrak{r}}^*]\!] \phi_{\mathfrak{r}}) \equiv \pi([\![\alpha_{\mathfrak{r}}]\!] \phi_{\mathfrak{r}}) \text{ if } \alpha_{\mathfrak{r}} \text{ is test-only}$$
$$\pi([\![?\psi_{\mathfrak{r}}]\!] \phi_{\mathfrak{r}}) \equiv \pi(\phi_{\mathfrak{r}} \vee \neg\psi_{\mathfrak{r}})$$
$$\pi([\![(\alpha,\beta)]\!] \phi_{\mathfrak{r}}) \equiv \begin{cases} \pi(\phi_{\mathfrak{r}}) & \text{if } \alpha,\beta = ?\top \\ \{([\alpha]\phi, [\beta]\psi)\} & \text{else if } \pi(\phi_{\mathfrak{r}}) = \{(\phi,\psi)\} \\ \pi([\![(\alpha, ?\top)]\!] [\![(?\top, \beta)]\!] \phi_{\mathfrak{r}}) & \text{else if } \alpha,\beta \neq ?\top \\ \text{cases below} & \text{else} \end{cases}$$
$$\pi([\![(\alpha \cup \beta, ?\top)]\!] \phi_{\mathfrak{r}}) \equiv \pi([\![(\alpha, ?\top)]\!] \phi_{\mathfrak{r}} \wedge [\![(\beta, ?\top)]\!] \phi_{\mathfrak{r}})$$
$$\pi([\![(?\top, \alpha \cup \beta)]\!] \phi_{\mathfrak{r}}) \equiv \pi([\![(?\top, \alpha)]\!] \phi_{\mathfrak{r}} \wedge [\![(?\top, \beta)]\!] \phi_{\mathfrak{r}})$$
$$\pi([\![(\alpha; \beta, ?\top)]\!] \phi_{\mathfrak{r}}) \equiv \pi([\![(\alpha, ?\top)]\!] [\![(\beta, ?\top)]\!] \phi_{\mathfrak{r}})$$
$$\pi([\![(?\top, \alpha; \beta)]\!] \phi_{\mathfrak{r}}) \equiv \pi([\![(?\top, \alpha)]\!] [\![(?\top, \beta)]\!] \phi_{\mathfrak{r}})$$
$$\pi([\![(\alpha^*, ?\top)]\!] \phi_{\mathfrak{r}}) \equiv \pi([\![(\alpha, ?\top)^*]\!] \phi_{\mathfrak{r}})$$
$$\pi([\![(?\top, \alpha^*)]\!] \phi_{\mathfrak{r}}) \equiv \pi([\![(?\top, \alpha)^*]\!] \phi_{\mathfrak{r}})$$
$$\pi([\![(\phi_{\mathcal{AP}}, ?\top)]\!] \phi_{\mathfrak{r}}) \equiv \pi(\phi_{\mathfrak{r}} \vee \neg \lfloor \phi_{\mathcal{AP}} \rfloor_{\text{L}})$$
$$\pi([\![(?\top, \phi_{\mathcal{AP}})]\!] \phi_{\mathfrak{r}}) \equiv \pi(\phi_{\mathfrak{r}} \vee \neg \lfloor \phi_{\mathcal{AP}} \rfloor_{\text{R}})$$
$$\pi([\![(?\phi, ?\top)]\!] \phi_{\mathfrak{r}}) \equiv \pi(\phi_{\mathfrak{r}} \vee \neg \lfloor ?\phi \rfloor_{\text{L}})$$
$$\pi([\![(?\top, ?\phi)]\!] \phi_{\mathfrak{r}}) \equiv \pi(\phi_{\mathfrak{r}} \vee \neg \lfloor ?\phi \rfloor_{\text{R}})$$

Most cases in Definition 8 follow the semantics of LDL_{REL} formulas. We explain several interesting cases here.

- $\pi(\langle\!\langle \alpha_{\mathfrak{r}}^* \rangle\!\rangle \phi_{\mathfrak{r}})$ yields $\pi(\langle\!\langle \alpha_{\mathfrak{r}} \rangle\!\rangle \phi_{\mathfrak{r}})$ if $\alpha_{\mathfrak{r}}$ is test-only. If $\alpha_{\mathfrak{r}}$ is not test-only, $\pi(\langle\!\langle \alpha_{\mathfrak{r}}^* \rangle\!\rangle \phi_{\mathfrak{r}})$ unfolds the loop a finite number of times. The original formula is satisfiable if the formula after unrolling is satisfiable.
- $\pi([\![\alpha_{\mathfrak{r}}^*]\!] \phi_{\mathfrak{r}})$ proceeds only if $\alpha_{\mathfrak{r}}$ is test-only.
- $\pi(\phi_{\mathfrak{r}} \wedge \psi_{\mathfrak{r}})$ yields a special product of $\pi(\phi_{\mathfrak{r}})$ and $\pi(\psi_{\mathfrak{r}})$ by taking conjunctions, respectively, of the first and second

elements. We are looking for a trace that satisfies both.

- $\pi(\phi_\mathfrak{r} \vee \psi_\mathfrak{r})$ combines the elements in both sets. The formula is satisfiable if any element in either set is satisfiable.
- $\pi([\![(\alpha,\beta)]\!]\phi_\mathfrak{r})$ expands cases of α and β to look for all possible paths of $[\![(\alpha,\beta)]\!]$. By contrast, $\pi(\langle\!\langle(\alpha,\beta)\rangle\!\rangle\phi_\mathfrak{r})$ directly combines α and β, respectively, with ϕ and ψ into the modality of existence, i.e. $(\langle\alpha\rangle\phi, \langle\beta\rangle\psi)$. The two modalities are treated differently because of their semantic difference. In particular, $\langle\!\langle(\alpha,\beta)\rangle\!\rangle(\phi_\mathfrak{r} \vee \psi_\mathfrak{r})$ is equivalent to $\langle\!\langle(\alpha,\beta)\rangle\!\rangle\phi_\mathfrak{r} \vee \langle\!\langle(\alpha,\beta)\rangle\!\rangle\psi_\mathfrak{r}$ for any $\phi_\mathfrak{r}$ and $\psi_\mathfrak{r}$. But $[\![(\alpha,\beta)]\!](\phi_\mathfrak{r} \vee \psi_\mathfrak{r})$ is *not* equivalent to $[\![(\alpha,\beta)]\!]\phi_\mathfrak{r} \vee [\![(\alpha,\beta)]\!]\psi_\mathfrak{r}$ in general.
- $\pi([\![(\alpha,\beta)]\!]\phi_\mathfrak{r}) \equiv \{([\alpha]\phi, [\beta]\psi)\}$ if $\pi(\phi_\mathfrak{r}) = \{(\phi,\psi)\}$, i.e., $\pi(\phi_\mathfrak{r})$ has only one element. Here, ϕ and ψ are the formulas that the left and right executions expect to satisfy.

The soundness theorem of the encoding is stated as follows:

Theorem 5 (Soundness of the encoding of LDL_{REL})**.** *For a* LDL_{REL} *formula* $\phi_\mathfrak{r}$ *and a renaming function* ξ *for* $\phi_\mathfrak{r}$*,*

$$\textit{if}\ \ \exists\sigma, (\sigma,0) \models \bigvee_{(\phi,\psi)\in\pi(\xi(\phi_\mathfrak{r}))}(\phi\wedge\psi)$$

$$\textit{then}\ \ \exists\sigma_{\text{L}}\sigma_{\text{R}}, (\sigma_{\text{L}},\sigma_{\text{R}},0,0) \models_{\text{RL}} \phi_\mathfrak{r}$$

The proof can be done by induction on $\phi_\mathfrak{r}$ (and simultaneous induction on $\alpha_\mathfrak{r}$). Most cases are proven by the induction hypothesis, as π follows closely the semantics of $\phi_\mathfrak{r}$ and $\alpha_\mathfrak{r}$.

F. Proof of Theorem 5

We consider $\phi_\mathfrak{r}$ formulas in negation normal form.

Lemma 12. *For a* LDL_{REL} *formula* $\phi_\mathfrak{r}$*,* LDL_f *formulas* ϕ,ψ*, if* $\text{VAR}_{\text{L}}(\phi_\mathfrak{r}) \cap \text{VAR}_{\text{R}}(\phi_\mathfrak{r}) = \varnothing$ *and* $(\phi,\psi) \in \pi(\phi_\mathfrak{r})$*, then* $\text{VAR}(\phi) \subseteq \text{VAR}_{\text{L}}(\phi_\mathfrak{r})$ *and* $\text{VAR}(\psi) \subseteq \text{VAR}_{\text{R}}(\phi_\mathfrak{r})$*.*

Proof. By simultaneous induction on $\phi_\mathfrak{r}$ and $\alpha_\mathfrak{r}$. □

Lemma 13. *For a trace* σ*,* LDL_f *formula, if* $\sigma, m \models \phi$*, then* $\exists\ \sigma'$ *that* $\sigma', 0 \models \phi$*.*

Proof. By the definition of $\models\phi$. □

Lemma 14. *For trace* $\sigma_{\text{L}},\sigma_{\text{R}}$*,* LDL_f *formulas* ϕ,ψ*,* $m,n \in \mathbb{N}$*, if* $\text{VAR}(\phi)\cap\text{VAR}(\psi) = \varnothing$*,* $\sigma_{\text{L}}, m \models \phi$*, and* $\sigma_{\text{R}}, n \models \psi$*, then* $\exists\sigma$ *that* $\sigma, 0 \models \phi\wedge\psi$*.*

Proof. By the definition of $\models$. □

Lemma 15. *For a loop-free* LDL_{REL} *formula* $\phi_\mathfrak{r}$ *that* $\text{VAR}_{\text{L}}(\phi_\mathfrak{r}) \cap \text{VAR}_{\text{R}}(\phi_\mathfrak{r}) = \varnothing$*,* $m,n \in \mathbb{N}$*, then* $\exists\sigma_{\text{L}}\ \sigma_{\text{R}}$*,* $(\sigma_{\text{L}},\sigma_{\text{R}},m,n) \models_{\text{RL}} \phi_\mathfrak{r}$ *iff* $\exists\sigma$*,* $(\sigma,0) \models \bigvee_{(\phi,\psi)\in\pi(\xi(\phi_\mathfrak{r}))}(\phi\wedge\psi)$*.*

Proof. Both directions can proven with induction on $\phi_\mathfrak{r}$ (and simultaneous induction on $\alpha_\mathfrak{r}$) and Lemma 13, 14. □

Lemma 16. *For a* LDL_{REL} *formula* $\phi_\mathfrak{r}$*, a trace* σ*, a* LDL_f *formula* ϕ,ψ*,* $m,n \in \mathbb{N}$*, if* $\text{VAR}_{\text{L}}(\phi_\mathfrak{r})\cap\text{VAR}_{\text{R}}(\phi_\mathfrak{r}) = \varnothing$*,* $(\phi,\psi) \in \pi(\phi_\mathfrak{r})$*,* $\sigma_{\text{L}}, m \models \phi$*, and* $\sigma_{\text{R}}, n \models \psi$ *then* $(\sigma_{\text{L}},\sigma_{\text{R}},m,n) \models \phi_\mathfrak{r}$*.*

Proof. By induction of formula $\phi_\mathfrak{r}$ (and simultaneous induction on $\alpha_\mathfrak{r}$ for programs):

- Case $\phi_\mathfrak{r}$ is of the form $\lfloor\phi\rfloor_{\text{B}}$ (or $\neg\lfloor\phi\rfloor_{\text{B}}$): by examining the definition of π and the semantics of LDL_{REL} formulas.
- Case $\phi_\mathfrak{r}$ is of the form $\phi_{\mathfrak{r}_1} \wedge \phi_{\mathfrak{r}_2}$: by IH
- Case $\phi_\mathfrak{r}$ is of the form $\phi_{\mathfrak{r}_1} \vee \phi_{\mathfrak{r}_2}$: by IH.
- Case $\phi_\mathfrak{r}$ is of the form $\langle\!\langle\alpha_\mathfrak{r}\rangle\!\rangle\psi_\mathfrak{r}$: The cases of $\alpha_{\mathfrak{r}_1}\,;\,\beta_{\mathfrak{r}_2}$, $\alpha_{\mathfrak{r}_1}\cup\beta_{\mathfrak{r}_2}$, $?\phi_\mathfrak{r}$ and $\alpha^*_{\mathfrak{r}_1}$. They can be proved using IH. For the case that of (α,β): we know that $\pi(\langle\!\langle(\alpha,\beta)\rangle\!\rangle\psi_\mathfrak{r})$ is $\{(\langle\alpha\rangle\phi_1, \langle\beta\rangle\phi'_1) \mid (\phi_1,\phi'_1) \in \pi(\phi_\mathfrak{r})\}$. That means $\phi = \langle\alpha\rangle\phi_1$, $\phi' = \langle\beta\rangle\phi'_1$, for some $(\phi_1,\phi'_1) \in \pi(\phi_\mathfrak{r})$. Meanwhile, since $\sigma_{\text{L}}, m \models \langle\alpha\rangle\phi_1$, and $\sigma_{\text{R}}, n \models \langle\beta\rangle\phi'_1$, then exists trace $\sigma_{\text{L}}', \sigma_{\text{R}}'$, m', n' such that $\sigma_{\text{L}}', m' \models \phi_1$, and $\sigma_{\text{R}}', n' \models \phi'_1$. Thus by IH, $(\sigma_{\text{L}}', \sigma_{\text{R}}', m', n') \models \psi_\mathfrak{r}$. Moreover, by the definition of LDL_{REL} semantics for $\langle\!\langle(\alpha,\beta)\rangle\!\rangle\psi_\mathfrak{r}$, we have $(\sigma_{\text{L}},\sigma_{\text{R}},m,n) \models \langle\!\langle(\alpha,\beta)\rangle\!\rangle\psi_\mathfrak{r}$.
- Case $\phi_\mathfrak{r}$ is of the form $[\![\alpha_\mathfrak{r}]\!]\psi_\mathfrak{r}$: The definition of π for all cases follows closely the semantics of $\alpha_\mathfrak{r}$ and α, and can be proved with IH.

That concludes the proof. □