Extending Dynamic Logics with First-Class Relational Reasoning

Jian Xiang*, Stephen Chong†
*College of Computing and Informatics, UNC Charlotte, *jian.xiang@charlotte.edu*
†SEAS, Harvard University, *chong@seas.harvard.edu*

***Abstract*—Many important properties of computer systems 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. Verifying relational properties expressed by the REL extension can benefit from techniques developed for general relational reasoning and domain-specific relational reasoning, and 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 key relational properties that cannot be easily expressed with the host logics. We state and prove the theorems on the expressive power of the instantiations. We develop two encodings for the instantiations to leverage existing tools for verification. We further demonstrate the usefulness of the encodings with an experiment on verifying a set of non-trivial relational properties.

I. Introduction

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

Dynamic logics are multi-modal logics widely used for verifying single-trace properties of imperative programs [3], [4], [5], [6]. 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". Different variants of dynamic logics have been introduced and used in different application domains. For example, differential dynamic logic is developed for the safety of cyber-physical systems [7], [8], and linear dynamic logic is often used for AI-based planning [9]. Various tools and techniques have been developed for the verification of formulas expressed with these dynamic logics. However, relational verification is often not well-explored across many of these application domains. It would be advantageous if we could easily incorporate relational reasoning support into these dynamic logics and, furthermore, leverage existing tools and techniques for relational verification.

A lightweight yet efficient approach for expressing relational properties is extending the host logic with a *biprogram* construct [10], [11], which explicitly specifies a pair of programs and thus allows reasoning about executions of the pair. We argue that extending dynamic logics with a biprogram provides intuitive and effective support for expressing many useful 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 $[\![(\alpha, \beta)]\!]\phi_{\mathtt{r}}$ expresses that *for all* executions of α and β, $\phi_{\mathtt{r}}$ holds at the last states of the two executions.
- REL formulas that directly capture relational properties. For example, the following formula
$$(\lfloor x \rfloor_{\mathsf{L}} < \lfloor x \rfloor_{\mathsf{R}}) \rightarrow [\![(\alpha, \beta)]\!](\lfloor x \rfloor_{\mathsf{L}} < \lfloor x \rfloor_{\mathsf{R}})$$
specifies that any pair of executions of α and β preserves the natural order on the values of x, i.e., if the execution of program β begins with a higher value for x compared to α, then its execution will end with a higher value for x. (the projections $\lfloor\cdot\rfloor_{\mathsf{L}}$ and $\lfloor\cdot\rfloor_{\mathsf{R}}$ 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) differen-

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

tial dynamic logic (d$\mathcal{L}$) [5], [12], a logic for verifying safety properties in cyber-physical systems, and (2) linear dynamic logic on finite traces [9] (LDL$_f$), a logic used by the AI community for reasoning about temporal constraints. The two logics share the core constructs, but differ significantly in other constructs and semantic interpretation. Together they cover most features of mainstream dynamic logics. Both instantiations can express key relational properties that are difficult, if not impossible, to express by the host logics.

A REL instantiation may or may not add expressive power over the host logic. We prove that the REL instantiation for d$\mathcal{L}$ is equally expressive as d$\mathcal{L}$, but the REL instantiation for LDL$_f$ is strictly more expressive than LDL$_f$.

Verifying REL formulas specified by REL instantiations can benefit from existing verification tools. We introduce two encodings of REL formulas that permit reusing existing tools to verify these formulas. The first encoding is inspired by the technique of self-composition [13], 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 d$\mathcal{L}$, and a sound but incomplete encoding for the instantiation for LDL$_f$. Existing tools developed for d$\mathcal{L}$ and LDL$_f$ can be used to verify REL formulas using this encoding. The second encoding reduces the problem of verifying REL formulas to a satisfiability problem, by encoding a REL formula as a set of constraints, and then using SMT solvers to find solutions. We demonstrate with a sound and complete encoding for the REL extension of LDL$_f$, and use Z3 to verify properties specified by the extension.

These encodings promote verification of relational properties, by leveraging techniques for both general relational reasoning and domain-specific relational reasoning. To investigate the usefulness of the encodings, we conduct an experiment on the encoding for d$\mathcal{L}$. We implement several variants of this encoding and demonstrate that these variants can integrate techniques for general relational reasoning as well as domain-specific relational reasoning. Thanks to this integration, the variants are able to verify non-trivial relational properties. We test these variants on a set of benchmarks, most of which are from existing work. We successfully verify all benchmarks using a theorem prover developed for d$\mathcal{L}$, with little to no manual effort. Some of the benchmarks cannot be automatically verified by prior techniques or tools to our knowledge.

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:

- A lightweight extension for dynamic logics to support first-class relational reasoning. Using the modalities, the REL extension can naturally and concisely express relational properties involving mixed quantifications. We present a set of practical example properties (Section III).
- Two case studies that demonstrate the applicability of the REL extension. We instantiate it for two distinct dynamic logics: d$\mathcal{L}$ and LDL$_f$. Both instantiations can express key domain-specific relational properties that are difficult, if not impossible, to express with the host logics. We formally state and prove the theorems on the expressiveness of both instantiations (Section IV and V).
- Encodings of REL formulas that allow reusing existing techniques and tools to promote verifying non-trivial REL formulas. We have developed two encodings. The first encoding reduces the verification of REL formulas to the verification of formulas of the host logics. The second encoding captures the semantics of REL formulas as constraints that can be solved by SMT solvers. We experiment with the encoding for d$\mathcal{L}$ on a set of benchmarks. The results are promising: we can use an existing tool for d$\mathcal{L}$ to verify non-trivial relational properties, with little to no manual effort (Section VI).

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

$$\begin{aligned}
\text{Program:}\ \ \alpha,\ \beta\ &::= P \in \Pi_0 \mid ?\phi \mid \alpha\,;\beta \mid \alpha \cup \beta \mid \alpha^* \\
\text{Formula:}\ \ \phi,\ \psi\ &::= \top \mid A \in \Phi_0 \mid \neg\phi \mid \phi \wedge \psi \mid [\alpha]\phi
\end{aligned}$$

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 about program executions. The language of regular PDL has expressions of two sorts: (1) propositions or formulas and (2) programs. Its syntax is defined upon Π_0, a set of atomic programs, and Φ_0, 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 [14]: sequential composition, nondeterministic choice, nondeterministic repetition, and test of a formula. Formulas include the standard propositional connectives and program necessity $[\alpha]\phi$. Program existence $\langle\alpha\rangle\phi$ can be encoded with 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 Kripke 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_0 of atomic programs into binary relations on $\mathcal{W}$, and $\mathcal{V}$ is a mapping from the set Φ_0 of atomic propositions into subsets of $\mathcal{W}$. That is, $\mathcal{R}(P) \subseteq$

$$\begin{aligned} \alpha_\mathfrak{r}, \beta_\mathfrak{r} &::= P_\mathfrak{r} \mid ?\phi_\mathfrak{r} \mid \alpha_\mathfrak{r}\, ;\, \beta_\mathfrak{r} \mid \alpha_\mathfrak{r} \cup \beta_\mathfrak{r} \mid \alpha_\mathfrak{r}^* \mid (\alpha, \beta) \\ \phi_\mathfrak{r}, \psi_\mathfrak{r} &::= A_\mathfrak{r} \mid \neg\phi_\mathfrak{r} \mid \phi_\mathfrak{r} \wedge \psi_\mathfrak{r} \mid \llbracket \alpha_\mathfrak{r} \rrbracket \phi_\mathfrak{r} \mid \lfloor \phi \rfloor_{\mathsf{B}} \quad (\mathsf{B} \in \{\mathsf{L}, \mathsf{R}\}) \end{aligned}$$

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

Fig. 3: Syntax of the REL extension

$\mathcal{W} \times \mathcal{W}$ for $P \in \Pi_0$, and $\mathcal{V}(A) \subseteq \mathcal{W}$ for $A \in \Phi_0$. $\mathcal{R}$ 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 relational properties.

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, 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 PDL, with the addition of projection formulas $\lfloor \phi \rfloor_{\mathsf{B}}$, which refers to a PDL formula ϕ in one of the two executions specified by $\mathsf{B} \in \{\mathsf{L}, \mathsf{R}\}$. The REL modality of necessity $\llbracket \alpha_\mathfrak{r} \rrbracket \phi_\mathfrak{r}$ can be used to encode the REL modality of existence $\langle\!\langle \alpha_\mathfrak{r} \rangle\!\rangle \phi_\mathfrak{r}$ and vice versa, e.g., $\llbracket \alpha_\mathfrak{r} \rrbracket \phi_\mathfrak{r} = \neg \langle\!\langle \alpha_\mathfrak{r} \rangle\!\rangle \neg \phi_\mathfrak{r}$.

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 $((\omega_{\mathsf{L}}, \omega_{\mathsf{R}}), (\nu_{\mathsf{L}}, \nu_{\mathsf{R}})) \in \mathcal{R}_\mathfrak{r}(P_\mathfrak{r})$, then an execution of the left program specified by $P_\mathfrak{r}$ runs from ω_{L} to ν_{L}, and an execution of the right program of $P_\mathfrak{r}$ runs from ω_{R} to ν_{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 $\lfloor \phi \rfloor_{\mathsf{B}}$ tests if PDL formula ϕ holds in the state specified by B (left or right). REL modality of necessity $\llbracket \alpha_\mathfrak{r} \rrbracket \phi_\mathfrak{r}$ holds in bi-state $(\omega_{\mathsf{L}}, \omega_{\mathsf{R}})$ if and only if formula $\phi_\mathfrak{r}$ holds in any pairs of states that are reachable from $(\omega_{\mathsf{L}}, \omega_{\mathsf{R}})$ by running $\alpha_\mathfrak{r}$. We write $[\![\alpha_\mathfrak{r}]\!]_{\mathsf{RL}}$ to denote the semantics of $\alpha_\mathfrak{r}$, and $(\omega_{\mathsf{L}}, \omega_{\mathsf{R}}) \models_{\mathsf{RL}} \phi_\mathfrak{r}$ if the formula $\phi_\mathfrak{r}$ holds in bi-state $(\omega_{\mathsf{L}}, \omega_{\mathsf{R}})$.

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. We can use REL formulas to easily and *succinctly* express all quantifications over two executions by using (and mixing) the necessity and existence modalities, 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. The first two entries show how to encode quantifications without alternations. The other four show how to encode quantifications with alternations, by combining the biprogram construct and a special program $?\top$ (i.e., *skip* in common programming languages).

$$\begin{aligned}
\forall \sigma_\alpha \forall \sigma_\beta.((\sigma_\alpha[\text{-}1], \sigma_\beta[\text{-}1]) \models_{\mathsf{RL}} \phi_\mathfrak{r}) &\equiv \llbracket (\alpha, \beta) \rrbracket \phi_\mathfrak{r} \\
\exists \sigma_\alpha \exists \sigma_\beta.((\sigma_\alpha[\text{-}1], \sigma_\beta[\text{-}1]) \models_{\mathsf{RL}} \phi_\mathfrak{r}) &\equiv \langle\!\langle (\alpha, \beta) \rangle\!\rangle \phi_\mathfrak{r} \\
\forall \sigma_\alpha \exists \sigma_\beta.((\sigma_\alpha[\text{-}1], \sigma_\beta[\text{-}1]) \models_{\mathsf{RL}} \phi_\mathfrak{r}) &\equiv \llbracket (\alpha, ?\top) \rrbracket \langle\!\langle (?\top, \beta) \rangle\!\rangle \phi_\mathfrak{r} \\
\forall \sigma_\beta \exists \sigma_\alpha.((\sigma_\alpha[\text{-}1], \sigma_\beta[\text{-}1]) \models_{\mathsf{RL}} \phi_\mathfrak{r}) &\equiv \llbracket (?\top, \beta) \rrbracket \langle\!\langle (\alpha, ?\top) \rangle\!\rangle \phi_\mathfrak{r} \\
\exists \sigma_\alpha \forall \sigma_\beta.((\sigma_\alpha[\text{-}1], \sigma_\beta[\text{-}1]) \models_{\mathsf{RL}} \phi_\mathfrak{r}) &\equiv \langle\!\langle (\alpha, ?\top) \rangle\!\rangle \llbracket (?\top, \beta) \rrbracket \phi_\mathfrak{r} \\
\exists \sigma_\beta \forall \sigma_\alpha.((\sigma_\alpha[\text{-}1], \sigma_\beta[\text{-}1]) \models_{\mathsf{RL}} \phi_\mathfrak{r}) &\equiv \langle\!\langle (?\top, \beta) \rangle\!\rangle \llbracket (\alpha, ?\top) \rrbracket \phi_\mathfrak{r}
\end{aligned}$$

For example, the *for all, exists* scenario, i.e., $\forall \sigma_\alpha\ \exists \sigma_\beta$ is encoded as $\llbracket (\alpha, ?\top) \rrbracket \langle\!\langle (?\top, \beta) \rangle\!\rangle$, where the modality $\llbracket (\alpha, ?\top) \rrbracket$ quantifies over executions of program α, and the modality $\langle\!\langle (?\top, \beta) \rangle\!\rangle$ quantifies over executions of program β.

From now on, we write $\llbracket (\alpha, \beta) \rangle\!\rangle \phi_\mathfrak{r}$ and $\langle\!\langle (\alpha, \beta) \rrbracket \phi_\mathfrak{r}$, respectively, for $\llbracket (\alpha, ?\top) \rrbracket \langle\!\langle (?\top, \beta) \rangle\!\rangle \phi_\mathfrak{r}$, and $\langle\!\langle (\alpha, ?\top) \rangle\!\rangle \llbracket (?\top, \beta) \rrbracket \phi_\mathfrak{r}$.

C. Expressing Relational Properties

The REL extensions can express relational properties intuitively. We show several examples here. Some examples are more than PDL, but their meanings can be easily inferred.

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

$$(\lfloor \vec{A} \rfloor_{\mathsf{L}} \leftrightarrow \lfloor \vec{A} \rfloor_{\mathsf{R}}) \rightarrow \llbracket (\alpha, \beta) \rangle\!\rangle (\lfloor \vec{A} \rfloor_{\mathsf{L}} \leftrightarrow \lfloor \vec{A} \rfloor_{\mathsf{R}})$$

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

Noninterference. Noninterference [1], [15] is a well-known strong information security property that guarantees that public outputs of a system do not reveal any confidential information (i.e., confidentiality), or dually that untrusted inputs of a system do not modify trusted contents (i.e., integrity). Noninterference has various variants, especially in a language that involves nondeterminism. Here, we express a few variants.

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 for any executions of a deterministic program α receiving the same low-security inputs, they should

Semantics of REL program

$$\begin{aligned}
\llbracket P_{\mathfrak{r}} \rrbracket_{\text{RL}} &= \mathcal{R}_{\mathfrak{r}}(P_{\mathfrak{r}}) \\
\llbracket ?\phi_{\mathfrak{r}} \rrbracket_{\text{RL}} &= \{((\omega_{\text{L}},\omega_{\text{R}}),(\omega_{\text{L}},\omega_{\text{R}})) \mid (\omega_{\text{L}},\omega_{\text{R}}) \models_{\text{RL}} \phi_{\mathfrak{r}}\} \\
\llbracket \alpha_{\mathfrak{r}}\,;\ \beta_{\mathfrak{r}} \rrbracket_{\text{RL}} &= \{((\omega_{\text{L}},\omega_{\text{R}}),(\nu_{\text{L}},\nu_{\text{R}})) \mid \text{exists } \mu_{\text{L}},\ \mu_{\text{R}} \text{ that } ((\omega_{\text{L}},\omega_{\text{R}}),(\mu_{\text{L}},\mu_{\text{R}})) \in \llbracket \alpha_{\mathfrak{r}} \rrbracket_{\text{RL}} \text{ and } ((\mu_{\text{L}},\mu_{\text{R}}),(\nu_{\text{L}},\nu_{\text{R}})) \in \llbracket \beta_{\mathfrak{r}} \rrbracket_{\text{RL}}\} \\
\llbracket \alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}} \rrbracket_{\text{RL}} &= \{((\omega_{\text{L}},\omega_{\text{R}}),(\nu_{\text{L}},\nu_{\text{R}})) \mid \llbracket \alpha_{\mathfrak{r}} \rrbracket_{\text{RL}} \cup \llbracket \beta_{\mathfrak{r}} \rrbracket_{\text{RL}}\} \\
\llbracket\, \alpha_{\mathfrak{r}}^{*} \,\rrbracket_{\text{RL}} &= \llbracket \alpha_{\mathfrak{r}} \rrbracket_{\text{RL}}^{*} \text{ the transitive, reflexive closure of } \llbracket \alpha_{\mathfrak{r}} \rrbracket_{\text{RL}}, \text{ where } \alpha_{\mathfrak{r}}^{0} \text{ is } (?\top, ?\top) \\
\llbracket (\alpha,\beta) \rrbracket_{\text{RL}} &= \{((\omega_{\text{L}},\omega_{\text{R}}),(\nu_{\text{L}},\nu_{\text{R}})) \mid (\omega_{\text{L}},\nu_{\text{L}}) \in \llbracket \alpha \rrbracket \text{ and } (\omega_{\text{R}},\nu_{\text{R}}) \in \llbracket \beta \rrbracket\ \}
\end{aligned}$$

Semantics of REL formula

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

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

Fig. 4: Semantics of REL programs and formulas

produce the same low-security outputs. Such a noninterference notion is a $\forall\forall$ property:

$$(\lfloor \vec{A}_{in} \rfloor_{\text{L}} \leftrightarrow \lfloor \vec{A}_{in} \rfloor_{\text{R}}) \rightarrow [\![\,(\alpha,\alpha)\,]\!](\lfloor \vec{A}_{out} \rfloor_{\text{L}} \leftrightarrow \lfloor \vec{A}_{out} \rfloor_{\text{R}})$$

Where atomic propositions $\vec{A}_{in}$ and $\vec{A}_{out}$ represent, respectively, low-security inputs and outputs of α. The 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* [16], which states that an attacker cannot infer confidential information *for certain* in a program with non-observable nondeterminism. Intuitively, the attacker cannot observe the nondeterministic choices made by program α, so the attacker is not certain if the information received is confidential. Such a noninterference notion can be expressed with the *forall, exists* quantifications over executions of α, i.e.,

$$(\lfloor \vec{A}_{in} \rfloor_{\text{L}} \leftrightarrow \lfloor \vec{A}_{in} \rfloor_{\text{R}}) \rightarrow [\![\,(\alpha,\beta)\,\rangle\!\rangle(\lfloor \vec{A}_{out} \rfloor_{\text{L}} \leftrightarrow \lfloor \vec{A}_{out} \rfloor_{\text{R}})$$

To properly reason about two executions in the presence of nondeterminism, some variants of noninterference require (some of) the nondeterminism in the two executions to resolve in the same way [17]. 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 circumstances, 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, where $\alpha_{\mathfrak{r}}$ is $((?A_{in}\,;\,?A_{out},\ ?A_{in}\,;\,?A_{out}))$ and $\beta_{\mathfrak{r}}$ is $((?\neg A_{out},\ ?\neg A_{out}))$:

$$(\lfloor A_{in} \rfloor_{\text{L}} \leftrightarrow \lfloor A_{in} \rfloor_{\text{R}}) \rightarrow [\![\, \alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}} \,]\!](\lfloor A_{out} \rfloor_{\text{L}} \leftrightarrow \lfloor A_{out} \rfloor_{\text{R}})$$

This formula indeed holds: two executions with the same nondeterministic choice, i.e., either the first branch (i.e., $\alpha_{\mathfrak{r}}$) or the second (i.e., $\beta_{\mathfrak{r}}$), would lead to the same value of A_{out}.

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 formula expresses a *relational invariant* of a loop with a biprogram. In particular, an invariant ϕ_{inv} holds at the end of every loop iteration of $(\alpha,\beta)^{*}$: $\phi_{inv} \rightarrow [\![\,(\alpha,\beta)\,]\!]\phi_{inv}$.

Loop Alignments. As shown in prior work, verifying relational properties on loops may require careful *alignment* in executing the left and right loops. The REL extension can readily and clearly express these alignments. Consider the example (DoubleSquare) adapted from [18], [19] on integers x, y, and z:

$$\lfloor x \rfloor_{\text{L}} = \lfloor x \rfloor_{\text{R}} \rightarrow [\![\, y := 0\,;\ (z := 2x, z := x);\ (body^{*}, body^{*})\,]\!]$$
$$[\![\,(?z \leq 0, ?z \leq 0);\ (?\top, y := 2y)\,]\!]\lfloor y \rfloor_{\text{L}} = \lfloor y \rfloor_{\text{R}}$$

Where $body \equiv ?z \geq 0; z := z-1; y := y+x$. An effective alignment for verifying this property can be expressed as another formula that replaces $(body^{*}, body^{*})$ with $(body\,; body, body)^{*}$ which aligns two loop iterations of the left execution with one iteration of the right.

For general $\forall\forall$ formulas of the form $[\![\,(body^{*}, body^{*})\,]\!]\phi_{\mathfrak{r}}$, we can express the alignments required for verifying such a formula in a very general manner, as follows:

$$[\![\,(?Gb;\ (body, body) \cup ?Gl;\ (body, \top) \cup ?Gr;\ (\top, body))^{*}\,]\!]\phi_{\mathfrak{r}}$$

Where formulas Gb, Gl, and Gr specify the conditions for executing, respectively, both the left and right loops, the left loop only, and the right loop only. For example, we can get an effective alignment for verifying the example above by letting $Gb \equiv \lfloor x \rfloor_{\text{L}} = \lfloor x \rfloor_{\text{R}}$, $Gl \equiv \lfloor x \rfloor_{\text{L}} \geq \lfloor x \rfloor_{\text{R}}$, $Gr \equiv \lfloor false \rfloor_{\text{L}} \wedge \lfloor false \rfloor_{\text{R}}$.

Motion Planning. The REL extension can intuitively express many relational properties on motion planning or path finding, e.g., for driving or robots. For example, an important property in motion planning is finding the optimal path (of all possible paths). Given a control system modeled as α, the following $\exists\forall$ formula specifies that an optimal path exists (the left execution arrives no-later than all possible executions):

$$(\lfloor \phi \rfloor_{\text{L}} \wedge \lfloor \phi \rfloor_{\text{R}}) \rightarrow \langle\!\langle\,(\alpha,\alpha)\,]\!](\lfloor \psi \rfloor_{\text{R}} \rightarrow \lfloor \psi \rfloor_{\text{L}})$$

Where $\lfloor \phi \rfloor_{\text{L}} \wedge \lfloor \phi \rfloor_{\text{R}}$ specifies that executions start at the same location, and $\lfloor \psi \rfloor_{\text{R}} \rightarrow \lfloor \psi \rfloor_{\text{L}}$ specifies that the left execution arrives no-later than other executions of α. We can further express useful variants of this property. For example, an optimal path may exist only if certain actions are triggered:

$$(\lfloor\phi\rfloor_{\mathsf{L}}\wedge\lfloor\phi\rfloor_{\mathsf{R}})\rightarrow\langle\!\langle((\alpha_1;\beta;\alpha_2),\alpha)\rrbracket(\lfloor\psi\rfloor_{\mathsf{L}}\rightarrow\lfloor\psi\rfloor_{\mathsf{R}})$$

Where β specifies these actions. And α_1 and α_2 model possible actions the system can take before and after β.

Beyond Relational Properties. By renaming variables used in a program, we may also express hyperproperties that are beyond relational. For example, consider the notion of *generalized noninterference* [2] that states for any two executions t_1 and t_2 of α, there exists another execution t_3 of α whose high inputs are the same as t_1 and whose low events are the same as t_2. We can express this hyperproperty as follows:

$$(\lfloor A_h\rfloor_{\mathsf{L}}\leftrightarrow\lfloor\epsilon(A_h)\rfloor_{\mathsf{R}})\rightarrow\llbracket(\alpha,\alpha)\rrbracket\langle\!\langle(?\top,\epsilon(\alpha))\rangle\!\rangle(\lfloor A_l\rfloor_{\mathsf{R}}\leftrightarrow\lfloor\epsilon(A_l)\rfloor_{\mathsf{R}})$$

Where a function ϵ renames the input programs (or formulas) with fresh variables. Propositions A_h and A_l represent, respectively, the high inputs and low events of α.

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 [5], [7], [12] (d$\mathcal{L}$), and (b) linear dynamic logic on finite traces [9] (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 (d$\mathcal{L}$)*

d$\mathcal{L}$ is a first-order dynamic logic that enables verifying high-level cyber-physical systems models featuring real arithmetic, nondeterminism, and differential equations. Programs in d$\mathcal{L}$ are referred to as *hybrid programs* [7]. They are a formalism for modeling systems that have both continuous and discrete behaviors. Hybrid programs can express continuous evolution (as differential equations) as well as discrete transitions.

Figure 5 shows the syntax of constructs that d$\mathcal{L}$ 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 ($x := *$). The 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 d$\mathcal{L}$ are comparisons of terms, i.e., $(\theta \sim \delta)$.

d$\mathcal{L}$ is considered 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 comparisons of terms.

Similar to PDL, the semantics of d$\mathcal{L}$ [5], [12] 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 d$\mathcal{L}$ are shown in Figure 6.

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

Fig. 5: Syntax of d$\mathcal{L}$

Term semantics

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

Program semantics

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

Formula semantics

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

Fig. 6: Semantics of d$\mathcal{L}$ programs and formulas

With d$\mathcal{L}$, we are often interested in formulas of the form $\phi_{pre} \rightarrow [\alpha]\phi_{post}$: 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.

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

Fig. 7: d$\mathcal{L}$ 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 d$\mathcal{L}$ model of such an autonomous vehicle. Let d be the vehicle's distance from the obstacle. The *safety condition* that we would like

[2]Platzer introduces this autonomous vehicle example [7].

$$\begin{aligned}\theta_{\mathfrak{r}},\ \delta_{\mathfrak{r}} &::= \lfloor\theta\rfloor_{\mathsf{B}} \mid \theta_{\mathfrak{r}} \oplus \delta_{\mathfrak{r}}\\ \alpha_{\mathfrak{r}},\ \beta_{\mathfrak{r}} &::= x := \theta \mid x := * \mid x' = \theta\ \&\phi \mid \cdots\\ \phi_{\mathfrak{r}},\ \psi_{\mathfrak{r}} &::= \theta_{\mathfrak{r}} \sim \delta_{\mathfrak{r}} \mid \cdots\end{aligned}$$

Fig. 8: Syntax of d$\mathcal{L}_{\text{REL}}$

to enforce (ϕ_{post}) is that d is positive, i.e., doesn't hit the obstacle. 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 d$\mathcal{L}$ can be used to prove that the safety condition ϕ_{post} holds. The tactic-based theorem prover KeYmaera X [20] provides support for constructing proofs.

B. d$\mathcal{L}_{\text{REL}}$*: A* REL *Instantiation for* d$\mathcal{L}$

We build d$\mathcal{L}_{\text{REL}}$, a REL instantiation for d$\mathcal{L}$. d$\mathcal{L}_{\text{REL}}$ instantiates the atomic programs and formulas of the REL extension with constructs designed for d$\mathcal{L}$. These constructs are shown in Figure 8. (We omit the other constructs shown in Figure 3) The relational atomic programs, i.e., $P_{\mathfrak{r}}$, in d$\mathcal{L}_{\text{REL}}$ are a relational version of deterministic assignment ($x := \theta$), nondeterministic assignment ($x := *$), and continuous evolution ($x' = \theta\ \&\phi$). The relational atomic formulas, i.e., $A_{\mathfrak{r}}$, in d$\mathcal{L}_{\text{REL}}$ are comparisons of d$\mathcal{L}_{\text{REL}}$ terms, i.e., $\theta_{\mathfrak{r}} \sim \delta_{\mathfrak{r}}$. A d$\mathcal{L}_{\text{REL}}$ term can be either a *projection* term: $\lfloor\theta\rfloor_{\mathsf{B}}$, which refers to a d$\mathcal{L}$ term θ in one of the two executions specified by $\mathsf{B} \in \{\mathsf{L}, \mathsf{R}\}$, or an arithmetic operation of two d$\mathcal{L}_{\text{REL}}$ terms, i.e., $\theta_{\mathfrak{r}} \oplus \delta_{\mathfrak{r}}$.

Figure 9 shows the semantics of atomic programs in d$\mathcal{L}_{\text{REL}}$. Program $[\![x := \theta]\!]_{\text{RL}}$ indicates that the same deterministic assignment is run by the left and right executions. The program $[\![x := *]\!]_{\text{RL}}$ 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 d$\mathcal{L}_{\text{REL}}$ programs are d$\mathcal{L}$ formulas instead of d$\mathcal{L}_{\text{REL}}$ formulas. One can imagine a different d$\mathcal{L}_{\text{REL}}$ design that allows relational formulas as evolution constraints. However, they are not good candidates for evolution constraints since they are often not physically meaningful in characterizing the physical evolution.

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

$$\begin{aligned}(\omega_{\text{L}}, \omega_{\text{R}})[\![\lfloor\theta\rfloor_{\mathsf{B}}]\!]_{\text{RL}} &= \omega_{\text{L}}[\![\theta]\!] \text{ if } \mathsf{B} = \mathsf{L} \text{ or } \omega_{\text{R}}[\![\theta]\!] \text{ if } \mathsf{B} = \mathsf{R}\\ (\omega_{\text{L}}, \omega_{\text{R}})[\![\theta_{\mathfrak{r}} \oplus \delta_{\mathfrak{r}}]\!]_{\text{RL}} &= (\omega_{\text{L}}, \omega_{\text{R}})[\![\theta_{\mathfrak{r}}]\!]_{\text{RL}} \oplus (\omega_{\text{L}}, \omega_{\text{R}})[\![\delta_{\mathfrak{r}}]\!]_{\text{RL}} \quad \text{where}\\ &\quad \oplus \text{ denotes corresponding arithmetic operation for } \oplus \in \{+, \times\}\end{aligned}$$

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

$$\cdots$$

$$\begin{aligned}[\![x := \theta]\!]_{\text{RL}} &= \{((\omega_{\text{L}}, \omega_{\text{R}}), (\nu_{\text{L}}, \nu_{\text{R}})) \mid (\omega_{\text{L}}, \nu_{\text{L}}) \in [\![x := \theta]\!] \text{ and } (\omega_{\text{R}}, \nu_{\text{R}}) \in [\![x := \theta]\!]\}\\ [\![x := *]\!]_{\text{RL}} &= \{((\omega_{\text{L}}, \omega_{\text{R}}), (\nu_{\text{L}}, \nu_{\text{R}})) \mid \nu_{\text{L}}(z) = \omega_{\text{L}}(z),\ \nu_{\text{R}}(z) = \omega_{\text{R}}(z) \text{ for variables } z \neq x \text{ and } \nu_{\text{L}}(x) = \nu_{\text{R}}(x)\}\\ [\![x' = \theta\ \&\phi]\!]_{\text{RL}} &= \{((\varphi_1(0), \varphi_2(0)), (\varphi_1(r), \varphi_2(r))) \mid \varphi_1, \varphi_2 : [0, r] \to \text{STA are solutions of } x' = \theta \text{ and } \varphi_1(t) \models \phi \text{ and } \varphi_2(t) \models \phi \text{ for all } t \in [0, r]\}\end{aligned}$$

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

$$\cdots$$

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

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

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

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

Figure 10 shows examples of d$\mathcal{L}_{\text{REL}}$ programs and formulas adapted from existing work [17]. 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 d$\mathcal{L}_{\text{REL}}$

formula at the last line (i.e., ϕ_{robust}). Intuitively, the formula says for two runs of the vehicle, that have different modes of temperature control, if the vehicle starts with the same position and velocity (the premise ϕ_{τ} 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 ϕ_{τ} of the implication). The validity of 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 [17].

Theorem 1. *$\mathsf{d}\mathcal{L}_{\text{REL}}$ 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 $\lfloor\phi\rfloor_{L}$ or $\lfloor\phi\rfloor_{R}$. 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 describe 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 [21]. LDL_f presents a great testbed, as it differs significantly from $\mathsf{d}\mathcal{L}$, 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 strictly 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_0, instantiated as propositional formulas over the atomic propositions. For presentation purposes, we use metavariable ϕ_{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: $(\sigma_0, \sigma_1, \sigma_2, \cdots, \sigma_n)$, where each state is a subset of atomic propositions, i.e., $\sigma_i \in (2^{\Phi_0})$ for all $i \in \{0, 1, 2, \cdots, n\}$. A state σ_i satisfies an atomic proposition A if $A \in \sigma_i$. The position ($i \in \{0, 1, 2, \cdots, n\}$) 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_0 and a finite trace $\sigma \in (2^{\Phi_0})^*$, the truth of formula ϕ in trace σ at a position $0 \leq i < |\sigma|$, denoted $\sigma, i \models \phi$, is inductively defined [9]:

$\sigma, i \models A$ 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 $i \leq j < |\sigma|$ such that $(i,j) \in \mathcal{R}(\alpha, \sigma)$, $\sigma, j \models \phi$ holds

The semantics of a program α on a trace σ is defined as a binary relation $\mathcal{R}(\alpha, \sigma)$ on indices of the trace, i.e., a pair of indices $(i,j) \in \mathcal{R}(\alpha, \sigma)$ if the semantics of α holds at the sub-trace identified by the indices. The semantics of atomic programs ϕ_{AP} (propositional formulas over atomic propositions) is the key for $\mathcal{R}(\phi_{AP}, \sigma)$. In particular, $(i, i+1) \in \mathcal{R}(\phi_{AP}, \sigma)$ if the formula ϕ_{AP} holds at state $\sigma(i)$. The relation $\mathcal{R}(\alpha, \sigma)$ for all program constructs are then inductively defined:

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

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 on obstacle.

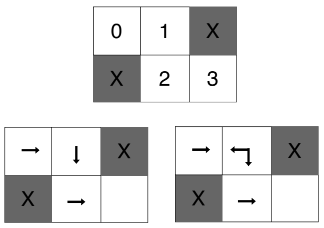

Fig. 11: 2 × 3 gridworld with obstacles (shaded grids with crossmarks) and two planning strategies. Valid locations are {0,1,2,3} and 3 is the destination

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. $\mathrm{LDL}_{\mathrm{REL}}$: An REL Instantiation for LDL_f

Many important properties of planning, such as robustness and privacy, are relational properties on multiple traces [22]. We develop $\mathrm{LDL}_{\mathrm{REL}}$, a REL instantiation for LDL_f, and use it to express relational properties of the gridworld problem.

$\mathrm{LDL}_{\mathrm{REL}}$ 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 $\mathrm{LDL}_{\mathrm{REL}}$ programs is defined as a transition relation between pairs of indices. For example, the semantics for a biprogram construct, $\mathcal{R}_{\mathfrak{r}}((\alpha,\beta),\sigma_{\mathrm{L}},\sigma_{\mathrm{R}})$, is defined on two pairs of indices whose first and second elements belong, respectively, to program relation of α, i.e., $(i,x)\in\mathcal{R}(\alpha,\sigma_{\mathrm{L}})$ and β, i.e., $(j,y)\in\mathcal{R}(\beta,\sigma_{\mathrm{R}})$.

We use $\mathrm{LDL}_{\mathrm{REL}}$ to express two important relational properties. The first one is a relational property of a single strategy, and the second compares two strategies.

Privacy of Initial Location. Location privacy, i.e., keeping individual locations private while they are partially observable for planning, is an important issue in mobile navigation [23], [24], [25]. We adopt a definition of *opacity* from existing work [22]. 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 location is the destination. Thus, the following formula should be *satisfiable* if a strategy, e.g., α_s, ensures privacy of the initial location:

$$\Big(\bigvee\nolimits_{i,j\in S\wedge j\neq i}(\lfloor\widehat{s}_{\mathrm{i}}\rfloor_{\mathrm{L}} \wedge \lfloor\widehat{s}_{\mathrm{j}}\rfloor_{\mathrm{R}})\Big) \wedge \langle\!\langle(\alpha_s,\alpha_s)\rangle\!\rangle(\lfloor\widehat{s}_3\rfloor_{\mathrm{L}} \wedge \lfloor\widehat{s}_3\rfloor_{\mathrm{R}})$$

Where formula $(\bigvee_{i,j\in S\wedge j\neq i}(\lfloor\widehat{s}_{\mathrm{i}}\rfloor_{\mathrm{L}} \wedge \lfloor\widehat{s}_{\mathrm{j}}\rfloor_{\mathrm{R}}))$ states that the two executions start with different initial locations, and the formula of $\mathrm{LDL}_{\mathrm{REL}}$ modality of existence, i.e., $\lfloor\widehat{s}_3\rfloor_{\mathrm{L}} \wedge \lfloor\widehat{s}_3\rfloor_{\mathrm{R}}$, 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 one, 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* (its negation is unsatisfiable), when strategy α_s is superior to β_s:

$$\Big(\bigvee\nolimits_{i\in S}(\lfloor\widehat{s}_{\mathrm{i}}\rfloor_{\mathrm{L}} \wedge \lfloor\widehat{s}_{\mathrm{i}}\rfloor_{\mathrm{R}})\Big) \to [\![(\alpha_l,\beta_l)^*]\!](\lfloor\widehat{s}_3\rfloor_{\mathrm{R}} \to \lfloor\widehat{s}_3\rfloor_{\mathrm{L}})$$

The formula $(\bigvee_{i\in S}(\lfloor\widehat{s}_{\mathrm{i}}\rfloor_{\mathrm{L}} \wedge \lfloor\widehat{s}_{\mathrm{i}}\rfloor_{\mathrm{R}}))$ 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. $(\lfloor\widehat{s}_3\rfloor_{\mathrm{R}} \to \lfloor\widehat{s}_3\rfloor_{\mathrm{L}})$ specifies that the left execution reaches the destination no later than the right one. The formula should always hold if strategy α_s is no worse than β_s.

Note that the host logic LDL_f cannot express the second property. The property is expressible in $\mathrm{LDL}_{\mathrm{REL}}$ because additional expressive power is introduced by $\mathrm{LDL}_{\mathrm{REL}}$.

Theorem 2. *$\mathrm{LDL}_{\mathrm{REL}}$ is strictly 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 expressions* [9], and (2) use of biprogram construct $(\alpha,\beta)^*$ can express that α and β have the same number of iterations, which is not regular, so it is not expressible in LDL_f.

VI. Verification Techniques

Verifying REL formulas can take advantage of existing techniques and tools. In this section, we introduce two encodings to exploit the benefits. The first one is *host logic encoding*, which transforms 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 *constraint-based encoding*, which encodes a REL formula as a set of constraints and then uses SMT solvers, e.g., Z3, to verify the constraints.

To understand how the encodings can promote the verification of REL formulas, we conduct an experiment with the host logic encoding for $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ on a set of benchmarks. We demonstrate that the encoding can integrate (and benefit from) techniques for general relational reasoning and domain-specific reasoning. Thanks to the integration, we can verify the benchmarks with an existing tool developed for $\mathsf{d}\mathcal{L}$, with little to no manual effort.

We first use $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ and $\mathrm{LDL}_{\mathrm{REL}}$ respectively to introduce the first and second encodings, and then discuss the experiment.

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 [26], [13], [27], [28], 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 $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$.

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 reveal high-security inputs. Construct

LDL$_{\text{REL}}$ formula semantics

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

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

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

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

The semantics for LDL$_{\text{REL}}$ program $\alpha_{\mathtt{r}}$ and trace σ_{L} σ_{R} is defined inductively as follows:

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

Fig. 12: Semantics of LDL$_{\text{REL}}$ programs and formulas

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 insights, we develop an encoding for d$\mathcal{L}_{\text{REL}}$. It builds on two steps: (1) renaming a d$\mathcal{L}_{\text{REL}}$ formula into an equivalent one whose left and right programs use disjoint variables, and (2) making a composition of the two programs.

Renaming d$\mathcal{L}_{\text{REL}}$ Formulas. To help with renaming, we define *renaming functions* that map all variables accessible by the *right* execution/program to fresh variables.

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

1) ξ is a bijection;

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

Where function $\text{VAR}_{L}(\phi_{\mathtt{r}})$ and $\text{VAR}_{R}(\phi_{\mathtt{r}})$ return the set of variables accessed by, respectively, the left and right execution.

We write $\xi(\phi_{\mathtt{r}})$ for the formula identical to $\phi_{\mathtt{r}}$ but whose variables accessible by the right execution have been renamed according to ξ. Renaming functions similarly apply to d$\mathcal{L}_{\text{REL}}$ programs. We also write $\xi(\omega)$ for the state identical to ω but whose domain variables have been renamed according to ξ.

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

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

$$\begin{aligned}
\pi(\theta_{\mathtt{r}} \sim \delta_{\mathtt{r}}) &= \pi(\theta_{\mathtt{r}}) \sim \pi(\delta_{\mathtt{r}}) & \pi(\alpha_{\mathtt{r}}\,;\ \beta_{\mathtt{r}}) &= \pi(\alpha_{\mathtt{r}})\,; \pi(\beta_{\mathtt{r}})\\
\pi(\neg\phi_{\mathtt{r}}) &= \neg\pi(\phi_{\mathtt{r}}) & \pi(\alpha_{\mathtt{r}} \cup \beta_{\mathtt{r}}) &= \pi(\alpha_{\mathtt{r}}) \cup \pi(\beta_{\mathtt{r}})\\
\pi(\phi_{\mathtt{r}} \wedge \psi_{\mathtt{r}}) &= \pi(\phi_{\mathtt{r}}) \wedge \pi(\psi_{\mathtt{r}}) & \pi(?\phi_{\mathtt{r}}) &= ?\pi(\phi_{\mathtt{r}})\\
\pi([\![\alpha_{\mathtt{r}}]\!]\phi_{\mathtt{r}}) &= [\pi(\alpha_{\mathtt{r}})]\pi(\phi_{\mathtt{r}}) & \pi(\alpha^*_{\mathtt{r}}) &= (\pi(\alpha_{\mathtt{r}}))^*\\
\pi(\lfloor\phi\rfloor_{\text{B}}) &= \phi & \pi((\alpha, \beta)) &= \alpha\,; \beta
\end{aligned}$$

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

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

Here, $\pi(\alpha_{\mathtt{r}})$ considers only five program constructs since the other constructs can be encoded with these five constructs.

Encoding Atomic Programs. The atomic programs of d$\mathcal{L}_{\text{REL}}$ 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 d$\mathcal{L}_{\text{REL}}$, as follows:

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

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 $(\lfloor x\rfloor_{\text{L}} = \lfloor x\rfloor_{\text{R}})$ to ensure the two executions have the same value for variable x. The program $x' = \theta\,\&\phi$ can be encoded with the help of a fresh continuous variable t to represent 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 $(\lfloor t\rfloor_{\text{L}} = \lfloor t\rfloor_{\text{R}})$ is added after the biprograms of physical dynamics to enforce the constraint that both evolutions last for the same duration.

With a renaming function ξ, we can encode a $\mathsf{d}\mathcal{L}_{\text{REL}}$ formula $\phi_{\mathfrak{r}}$ as $\pi(\xi(\phi_{\mathfrak{r}}))$. This encoding is sound and complete:

Theorem 3 (Soundness and completeness of the host logic encoding for $\mathsf{d}\mathcal{L}_{\text{REL}}$)**.** *For states ω_L ω_R, a $\mathsf{d}\mathcal{L}_{\text{REL}}$ formula $\phi_{\mathfrak{r}}$, and a renaming function ξ for $\phi_{\mathfrak{r}}$,*

$$(\omega_L, \omega_R) \models_{RL} \phi_{\mathfrak{r}} \textit{ iff}$$

$$((\omega_L \Downarrow \text{VAR}_L(\phi_{\mathfrak{r}})) \otimes (\xi(\omega_R \Downarrow \text{VAR}_R(\phi_{\mathfrak{r}})))) \models \pi(\xi(\phi_{\mathfrak{r}}))$$

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 $\phi_{\mathfrak{r}}$ and $\alpha_{\mathfrak{r}}$. An important case is the biprogram (α, β). Intuitively, π is sound for (α, β) because the semantics of $\mathsf{d}\mathcal{L}$ and $\mathsf{d}\mathcal{L}_{\text{REL}}$ are 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 $\mathsf{d}\mathcal{L}_{\text{REL}}$ Formulas. With Theorem 3, we can verify a $\mathsf{d}\mathcal{L}_{\text{REL}}$ formula $\phi_{\mathfrak{r}}$ by verifying a $\mathsf{d}\mathcal{L}$ formula $\pi(\xi(\phi_{\mathfrak{r}}))$. Verifying $\mathsf{d}\mathcal{L}$ formulas can done with the theorem prover KeYmaera X [20]. We encode the example shown in Figure 10 (ϕ_{robust} in particular) as a $\mathsf{d}\mathcal{L}$ formula $\pi(\xi(\phi_{robust}))$, which KeYmaera X is able to prove in a fully-automated manner.

Note that the function π *sequentially* composes the left and right programs, i.e., $\pi((\alpha, \beta)) = \alpha\,;\,\beta$. Though the encoding based on this π is sound and complete, it may be challenging to verify certain $\mathsf{d}\mathcal{L}_{\text{REL}}$ formulas with this encoding. Later in the experiment, we discuss how to improve π and promote verification by integrating techniques for general relational reasoning and domain-specific relational reasoning.

A Host Logic Encoding for LDL$_{\text{REL}}$. We have developed a sound but *incomplete* encoding for LDL$_{\text{REL}}$ in LDL$_f$ (in the appendix). This encoding is incomplete as LDL$_{\text{REL}}$ is strictly more expressive than LDL$_f$: it's impossible to design a sound and complete encoding of LDL$_{\text{REL}}$ in LDL$_f$. We can use this encoding to verify a *subset* of LDL$_{\text{REL}}$ formulas, especially, loop-free formulas. However, the encoding won't work for formulas that contain lockstep loops.

B. Constraint-based Encoding

The second approach for verifying a REL formula is to encode it as a set of constraints, i.e., logical formulas, and then use SMT solvers like Z3 to find if the constraints are satisfiable. We have developed such an encoding for LDL$_{\text{REL}}$ that is both sound and complete: the LDL$_{\text{REL}}$ formula is satisfiable if and only if the corresponding set of constraints is satisfiable. The encoding for $\mathsf{d}\mathcal{L}_{\text{REL}}$ can be similarly developed.

Encoding for LDL$_{\text{REL}}$. We use *arrays of integers* to encode traces, where an integer represents a state. The integers can be used as binary encodings, where each atomic proposition corresponds to a unique bit in an integer. The presence or absence of an atomic proposition in a state can then be represented by the state's binary encoding.

We use a universal quantifier to encode the necessity modality and a *predicate uninterpreted function* to encode the semantics of $\alpha_{\mathfrak{r}}$. Intuitively, the function returns true if its arguments satisfy its semantics.

We develop a function, denoted S, that generates a set of logical formulas for a LDL$_{\text{REL}}$ formula $\phi_{\mathfrak{r}}$. Intuitively, $S(\phi_{\mathfrak{r}})$ starts with a first-order logic formula generated for an input $\phi_{\mathfrak{r}}$, and then adds the semantics of all programs involved in $\phi_{\mathfrak{r}}$ to the output set. The function S builds on two functions: the first function λ takes a LDL$_{\text{REL}}$ (or LDL$_f$) formula and returns a first-order logic formula; the second function μ takes a program $\alpha_{\mathfrak{r}}$ (or α) and returns a first-order formula over uninterpreted functions that encodes the semantics of the program. We introduce the definitions of λ and μ next, and then present the definition of $S(\phi_{\mathfrak{r}})$.

Definition 3 (λ for LDL$_{\text{REL}}$)**.** *A function λ that takes a* LDL$_{\text{REL}}$ *formula $\phi_{\mathfrak{r}}$ as input and outputs a first-order logical formula, is defined as follows:*

$$\lambda(\neg\phi_{\mathfrak{r}}, \sigma_L, \sigma_R, i_L, i_R) = \neg\lambda(\phi_{\mathfrak{r}}, \sigma_L, \sigma_R, i_L, i_R)$$

$$\lambda(\lfloor\phi\rfloor_B, \sigma_L, \sigma_R, i_L, i_R) = \lambda(\phi, \sigma_L, i_L) \textit{ if } \mathsf{B} = \mathsf{L} \textit{ else } \lambda(\phi, \sigma_R, i_R)$$

$$\lambda(\phi_{\mathfrak{r}} \wedge \psi_{\mathfrak{r}}, \sigma_L, \sigma_R, i_L, i_R) = \lambda(\phi_{\mathfrak{r}}, \sigma_L, \sigma_R, i_L, i_R) \wedge \lambda(\psi_{\mathfrak{r}}, \sigma_L, \sigma_R, i_L, i_R)$$

$$\begin{aligned}\lambda([\![\alpha_{\mathfrak{r}}]\!]\phi_{\mathfrak{r}}, \sigma_L, \sigma_R, i_L, i_R) = \forall k_L\, k_R, (\ & k_L < |\sigma_L| \wedge k_R < |\sigma_R| \\ & \wedge F_{\alpha_{\mathfrak{r}}}(\sigma_L, \sigma_R, i_L, i_R, k_L, k_R)\ ) \\ & \rightarrow \lambda(\phi_{\mathfrak{r}}, \sigma_L, \sigma_R, k_L, k_R)\end{aligned}$$

The function λ extends to LDL$_f$ *formulas:*

$$\lambda(A, \sigma, i) = (\textit{binaryEnc}(A)\ \&\&\ \sigma[i]) == \textit{binaryEnc}(A)$$

$$\lambda(\neg\phi, \sigma, i) = \neg\lambda(\phi, \sigma, i)$$

$$\lambda(\phi \wedge \psi, \sigma, i) = \lambda(\phi, \sigma, i) \wedge \lambda(\psi, \sigma, i)$$

$$\lambda([\alpha]\phi, \sigma, i) = \forall k, ((i \leq k < |\sigma|) \wedge F_{\alpha}(\sigma, i, k)) \rightarrow \lambda(\phi, \sigma, k)$$

Most cases closely follow the semantics of the formulas and programs. Note that $\lambda(A, \sigma, i)$ checks if the atomic proposition A holds at the state $\sigma[i]$. The $\textit{binaryEnc}(A)$ function returns the integer that uniquely represents A. The check then proceeds with Bitwise AND (&&) with the integer at $\sigma[i]$. The result would be $\textit{binaryEnc}(A)$ if A holds at $\sigma[i]$.

$\mathtt{F}_{\alpha_{\mathfrak{r}}}(\sigma_L, \sigma_R, i_L, i_R, k_L, k_R)$ is a predicate uninterpreted function to help encode the semantics of the program $\alpha_{\mathfrak{r}}$. It returns true if $((i_L, i_R), (k_L, k_R)) \in \mathcal{R}_{\mathfrak{r}}(\alpha_{\mathfrak{r}}, \sigma_L, \sigma_R)$. The semantics of $\alpha_{\mathfrak{r}}$ is encoded as a logical formula over $\mathtt{F}_{\alpha_{\mathfrak{r}}}$ and uninterpreted functions for programs relevant to defining the semantics of $\alpha_{\mathfrak{r}}$. For example, by the semantics of $\alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}}$, we know the following logical formula holds: $\mathtt{F}_{\alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}}}(\sigma_L, \sigma_R, i_L, i_R, j_L, j_R) \leftrightarrow (\mathtt{F}_{\alpha_{\mathfrak{r}}}(\sigma_L, \sigma_R, i_L, i_R, j_L, j_R) \vee \mathtt{F}_{\beta_{\mathfrak{r}}}(\sigma_L, \sigma_R, i_L, i_R, j_L, j_R))$ where all parameters are universally quantified. This logical formula refers to three uninterpreted functions: $\mathtt{F}_{\alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}}}$, $\mathtt{F}_{\alpha_{\mathfrak{r}}}$, and $\mathtt{F}_{\beta_{\mathfrak{r}}}$.

We develop a function μ to output a logical formula for an input program $\alpha_{\mathfrak{r}}$. Note that the parameters of all uninterpreted functions are (implicitly) universally quantified in the formulas produced by μ.

Definition 4 (μ for LDL$_{\text{REL}}$)**.** *A function μ that takes a* LDL$_{\text{REL}}$ *program $\alpha_{\mathfrak{r}}$ as input and outputs a first-order logical formula, is defined as follows:*

$$\begin{aligned}\mu(\alpha_{\mathfrak{r}}\,;\ \beta_{\mathfrak{r}}) = \ & F_{\alpha_{\mathfrak{r}}\,;\,\beta_{\mathfrak{r}}}(\sigma_L, \sigma_R, i_L, i_R, j_L, j_R) \leftrightarrow \\ & (\exists k_L\, k_R, (i_L \leq k_L \leq j_L) \wedge (i_R \leq k_R \leq j_R) \\ & \wedge F_{\alpha_{\mathfrak{r}}}(\sigma_L, \sigma_R, i_L, i_R, k_L, k_R) \wedge F_{\beta_{\mathfrak{r}}}(\sigma_L, \sigma_R, k_L, k_R, j_L, j_R))\end{aligned}$$

$$\mu(\alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}}) = \ F_{\alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}}}(\sigma_L, \sigma_R, i_L, i_R, j_L, j_R) \leftrightarrow$$

$$
\begin{aligned}
&(\mathtt{F}_{\alpha_\mathfrak{r}}(\sigma_L,\sigma_R,i_L,i_R,j_L,j_R) \lor \mathtt{F}_{\beta_\mathfrak{r}}(\sigma_L,\sigma_R,i_L,i_R,j_L,j_R))\\
\mu(?\phi_\mathfrak{r}) =\ & \mathtt{F}_{?\phi_\mathfrak{r}}(\sigma_L,\sigma_R,i_L,i_R,j_L,j_R) \leftrightarrow\\
&((i_L == j_L \land i_R == j_R) \land \lambda(\phi_\mathfrak{r},\sigma_L,\sigma_R,i_L,i_R))\\
\mu(\alpha_\mathfrak{r}^*) =\ & \mathtt{F}_{\alpha_\mathfrak{r}^*}(\sigma_L,\sigma_R,i_L,i_R,j_L,j_R) \leftrightarrow\\
&((i_L == j_L \land i_R == j_R) \lor \mathtt{F}_{\alpha_\mathfrak{r}\,;\,\alpha_\mathfrak{r}^*}(\sigma_L,\sigma_R,i_L,i_R,j_L,j_R))\\
\mu((\alpha,\beta)) =\ & \mathtt{F}_{(\alpha,\beta)}(\sigma_L,\sigma_R,i_L,i_R,j_L,j_R) \leftrightarrow\\
&(\mathtt{F}_\alpha(\sigma_L,i_L,j_L) \land \mathtt{F}_\beta(\sigma_R,i_R,j_R))
\end{aligned}
$$

The function μ extends to LDL$_f$ *programs:*

$$
\begin{aligned}
\mu(\alpha\,;\,\beta) &= \mathtt{F}_{\alpha\,;\,\beta}(\sigma,i,j) \leftrightarrow (\exists k,(i \le k \le j) \land \mathtt{F}_\alpha(\sigma,i,k) \land \mathtt{F}_\beta(\sigma,k,j))\\
\mu(\alpha \cup \beta) &= \mathtt{F}_{\alpha\cup\beta}(\sigma,i,j) \leftrightarrow (\mathtt{F}_\alpha(\sigma,i,j) \lor \mathtt{F}_\beta(\sigma,i,j))\\
\mu(?\phi) &= \mathtt{F}_{?\phi}(\sigma,i,j) \leftrightarrow (i == j \land \lambda(\phi,\sigma,i))\\
\mu(\phi_{\mathcal{AP}}) &= \mathtt{F}_{\phi_{\mathcal{AP}}}(\sigma,i,j) \leftrightarrow (j == i+1 \land \lambda(\phi_{\mathcal{AP}},\sigma,i))\\
\mu(\alpha^*) &= \mathtt{F}_{\alpha^*}(\sigma,i,j) \leftrightarrow (j == i \lor \mathtt{F}_{\alpha\,;\,\alpha^*}(\sigma,i,j))
\end{aligned}
$$

With λ and μ, we define the following function $S(\phi_\mathfrak{r})$ that outputs a set of first-order logic formulas from an input LDL$_{\text{REL}}$ formula $\phi_\mathfrak{r}$. Intuitively, the set should be satisfiable if and only if the $\phi_\mathfrak{r}$ is satisfiable. Specifically, the set contains the first-order formula that encodes the input $\phi_\mathfrak{r}$, as well as *all* logical formulas that encode relevant (sub)program constructs.

Definition 5 ($S(\phi_\mathfrak{r})$ for LDL$_{\text{REL}}$)**.** *The set $S(\phi_\mathfrak{r})$ is the minimum set that contains the following:*

- $\exists\,\sigma_L\,\sigma_R, \lambda(\phi_\mathfrak{r},\sigma_L,\sigma_R,0,0) \in S(\phi_\mathfrak{r})$.
- *For any logical formula fml $\in S(\phi_\mathfrak{r})$, if fml contains an uninterpreted function $\mathtt{F}_{\alpha_\mathfrak{r}}$ (or $\mathtt{F}_\alpha$), then $\mu(\alpha_\mathfrak{r}) \in S(\phi_\mathfrak{r})$ (or $\mu(\alpha) \in S(\phi_\mathfrak{r})$).*

$\lambda(\phi_\mathfrak{r},\sigma_L,\sigma_R,0,0)$ uses 0 as beginning indices as arrays are 0-indexed. The second part of the definition covers all possible program constructs involved in all formulas or subformulas of $\phi_\mathfrak{r}$. $S(\phi_\mathfrak{r})$ is the minimum set to avoid irrelevant formulas.

A notable case of $S(\phi_\mathfrak{r})$ is when $\phi_\mathfrak{r}$ contains programs of the form $\alpha_\mathfrak{r}^*$ (or $\mu(\alpha^*)$), e.g., $[\![\alpha_\mathfrak{r}^*]\!]\phi_\mathfrak{r}$. The set $S([\![\alpha_\mathfrak{r}^*]\!]\phi_\mathfrak{r})$ refers to the uninterpreted function $\mathtt{F}_{\alpha_\mathfrak{r}^*}$, so $\mu(\alpha_\mathfrak{r}^*) \in S([\![\alpha_\mathfrak{r}^*]\!]\phi_\mathfrak{r})$. The formula $\mu(\alpha_\mathfrak{r}^*)$ refers to $\mathtt{F}_{\alpha_\mathfrak{r}\,;\,\alpha_\mathfrak{r}^*}$, so $\mu(\alpha_\mathfrak{r}\,;\,\alpha_\mathfrak{r}^*) \in S([\![\alpha_\mathfrak{r}^*]\!]\phi_\mathfrak{r})$. The formula $\mu(\alpha_\mathfrak{r}\,;\,\alpha_\mathfrak{r}^*)$ refers back to $\mathtt{F}_{\alpha_\mathfrak{r}^*}$, whereas $\mu(\alpha_\mathfrak{r}^*)$ is already in $S([\![\alpha_\mathfrak{r}^*]\!]\phi_\mathfrak{r})$ (parameters are universally quantified).

Theorem 4 (Soundness and completeness of the constraint-based encoding for LDL$_{\text{REL}}$)**.** *A* LDL$_{\text{REL}}$ *formula $\phi_\mathfrak{r}$ is satisfiable iff $S(\phi_\mathfrak{r})$ is satisfiable.*

The theorem holds since the definition of $S(\phi_\mathfrak{r})$ closely follows the semantics of LDL$_{\text{REL}}$.

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

Constraint-based Encoding for d$\mathcal{L}_{\text{REL}}$. A constraint-based encoding can be similarly developed for d$\mathcal{L}_{\text{REL}}$. The key difference is how to encode the semantics of d$\mathcal{L}_{\text{REL}}$ programs which is based on state transitions. An approach is to use two different sets of variables to represent the states before and after program executions. For example, the function λ for a d$\mathcal{L}_{\text{REL}}$ formula $[\![\alpha_\mathfrak{r}]\!]\phi_\mathfrak{r}$ can be defined as:

$$\lambda([\![\alpha_\mathfrak{r}]\!]\phi_\mathfrak{r},\vec{x}) = \forall\,\vec{y}, \mathtt{F}_{\alpha_\mathfrak{r}}(\vec{x},\vec{y}) \to \lambda(\phi_\mathfrak{r},\vec{y})$$

Where $\vec{x}$ is the set of variables used by the formula $[\![\alpha_\mathfrak{r}]\!]\phi_\mathfrak{r}$, and $\vec{y}$ is a set of fresh variables with the same dimension as $\vec{x}$. The two sets respectively capture the states before and after executing the program $\alpha_\mathfrak{r}$, whose semantics is denoted by the uninterpreted function $\mathtt{F}_{\alpha_\mathfrak{r}}(\vec{x},\vec{y})$. Then μ produces logical formulas over uninterpreted functions like $\mathtt{F}_{\alpha_\mathfrak{r}}(\vec{x},\vec{y})$.

Another challenge is how to specify the semantics of dynamics, i.e., differential equations. We can leverage existing work on computing the weakest preconditions (or strongest postconditions) for hybrid programs [29], [30]. For complex dynamics, their semantics are often specified as *differential invariants*, i.e., invariants that hold during the evolution of the dynamics. Differential invariants are often first-order formulas.

Benchmarks \ Variants	π	π_{align}	$\pi_{\forall\exists}$	π_{ode}	π_{syn}
Fig.10 (Fig.8 in [17])	A	A	A	A	A
Ex.2.1 in [19]		S	S	S	S
Ex.2.4 in [19]	A	A	A	A	A
Ex.7.4 in [19]	A	A	A	A	A
Ex.7.5 in [19]			A	A	A
Ex.7.6 in [19]			A	A	A
Ex.1				A	A
Fig.8 in [31]				S	S
Cs.1 in [32]					UA
Cs.2 in [32]					UA

Fig. 13: Using variants of π to help verify more complicated relational properties. For notation, *Ex.* and *Cs.* respectively denote the *example properties* and *case studies* in the cited work. Contents in the table cells indicate how much automation can KeYmaera X verify the generated d$\mathcal{L}$ formulas. Notations A, S, and UA mean, respectively, fully automated, automated with less than 3 manual proof steps, and automated after user inputs.

C. Verification with Improved Translations

The π and λ functions *translate* REL formulas to formulas that can be verified by existing tooling. These *translations* open great opportunities to utilize recent (and future) advances in both (1) general relational reasoning, and (2) domain-specific relational reasoning. Using d$\mathcal{L}_{\text{REL}}$ as an example, we show how to improve the verification power of a translation by retrofitting it with existing and new reasoning techniques.

In particular, we develop four variants of π, each with increasing verification power, achieved by incorporating additional *heuristics* in π. We either develop these heuristics or adapt them from prior work. We test these variants on a set of benchmarks, by (1) applying these variants on the benchmarks to generate d$\mathcal{L}$ formulas, and then (2) use the theorem prover KeYmaera X to verify the generated formulas. Figure 13 shows the benchmarks and the results.

Benchmarks. The benchmarks include a set of example properties from this paper and some existing work. In particular, we collect many motivating examples from [19], as it focuses on Kleene Algebra with Tests, which is directly relevant to dynamics logics. We are able to verify all examples from [19], except for two examples that require native support of integers (Ex.2.2) and arrays (Ex.2.3), which are not natively supported by KeYmaera X. In addition, we verify a few other representative and non-trivial case studies from existing work.

We elaborate on the four variants of π and their corresponding results. Note that, as shown in Figure 13, the default π can automatically verify three of these benchmarks.

π_{align}**:** π **+ heuristics on alignment**. As shown by existing work [19], [18], verifying relational properties often requires effective *alignment* of programs and computations so that relational invariants can be easier specified. Consider the following property from existing work [19] written in $\mathsf{d}\mathcal{L}_{\text{REL}}$:

$$\lfloor n\rfloor_{\mathtt{L}} = \lfloor n\rfloor_{\mathtt{R}} \to \llbracket i := n;\ r := 1 \rrbracket$$
$$\llbracket (body^*, body^*);\ (?i = 0, ?i = 0) \rrbracket \lfloor r\rfloor_{\mathtt{L}} = \lfloor r\rfloor_{\mathtt{R}}$$

Where the loop body is $body \equiv\ ?i \neq 0;\ r := r \times i;\ i := i - 1$. An effective alignment for verifying this property is to rewrite $(body^*, body^*)$ into $(body, body)^*$ which run the left and right executions in lockstep. We develop π_{align} that adds into π a heuristic that encodes a proof rule from existing work (Rule (3) from [19]) that soundly performs such a rewrite. Apply π_{align} to the property above produces a $\mathsf{d}\mathcal{L}$ formula that can be verified with KeYmaera X with only one manual proof step.

$\pi_{\forall\exists}$**:** π_{align} **+** $\forall\exists$ **heuristics**. Certain $\forall\exists$ properties on loops can be reduced to $\forall\forall$ properties for easier verification. In particular, the following reasoning is sound, as the lockstep of $(\alpha, \beta)^*$ presents a witness for the $\forall\exists$ property:

$$\textit{if}\ \phi_{\mathfrak{r}} \to \llbracket (\alpha, \beta)^* \rrbracket \psi_{\mathfrak{r}}\ \textit{then}\ \phi_{\mathfrak{r}} \to \langle\!| (\alpha^*, \beta^*) \rangle\!\rangle \psi_{\mathfrak{r}}$$

This rule can be generalized to support reasoning with more program constructs, such as $\langle\!| (\alpha_1;\ \alpha^*;\ \alpha_2, \beta_1;\ \beta^*;\ \beta_2) \rangle\!\rangle \psi_{\mathfrak{r}}$.

We develop $\pi_{\forall\exists}$ by adding these $\forall\exists$ rules above into π_{align}, and test it on benchmarks from existing work (Ex.7.5 and Ex.7.6 from [19]). KeYmaera X is able to verify the generated $\mathsf{d}\mathcal{L}$ formulas automatically.

π_{ode}**:** $\pi_{\forall\exists}$ **+ heuristics on dynamics**. The translations π_{align} and $\pi_{\forall\exists}$ can help reason with general relational properties; they are applicable to different REL extensions. Moreover, these translations open opportunities towards better domain-specific relational reasoning that focuses on the domain-specific parts of the host logics. For example, consider the following example on a relationship between two dynamics of exponential decay:

$$\lfloor x\rfloor_{\mathtt{L}} \geq \lfloor x\rfloor_{\mathtt{R}} \land \lfloor x\rfloor_{\mathtt{L}} \geq 0 \land A > 0 \land A \geq B \to$$
$$\langle\!| (x' = A \times x, x' = B \times x) \rangle\!\rangle \lfloor x\rfloor_{\mathtt{L}} \geq \lfloor x\rfloor_{\mathtt{R}} \qquad (1)$$

Using the aforementioned translations, this property cannot be verified with KeYmaera X. However, this $\forall\exists$ property should hold since the x value of the right dynamic would always be lower if *the two dynamics evolve for the same duration of time*. We can derive heuristics based on this observation. Consider a formula of the form $\phi_{\mathfrak{r}} \to \langle\!| (x' = \theta \& \phi_1, y' = \delta \& \phi_2) \rangle\!\rangle \psi_{\mathfrak{r}}$ (this formula has been properly renamed, so the left and right programs refer to disjoint sets of variables). After applying the π function, we get a $\mathsf{d}\mathcal{L}$ formula of the form: $\phi \to [x' = \theta \& \phi_1]\langle y' = \delta \& \phi_2\rangle \psi$, which can be reasoned with the following heuristic:

$$\textit{if}\ \phi \to [x' = \theta, y' = \delta \& (\phi_1 \land \phi_2)]\psi$$
$$\textit{and}\ \phi \to [x' = \theta, y' = \delta](\phi_1 \to \phi_2)$$
$$\textit{then}\ \phi \to [x' = \theta \& \phi_1]\langle y' = \delta \& \phi_2\rangle \psi$$

The first condition of this heuristic merges two dynamics into one and forces the two dynamics to evolve for the same duration of time. A formula of the form $\phi \to [x' = \theta, y' = \delta \& (\phi_1 \land \phi_2)]\psi$ is often easier to prove. The second condition on evolution constraints indicates there exists executions of the right dynamics if the left has valid executions. The second condition is needed to ensure *there exists an execution of* β for a $\langle\!| (\alpha, \beta) \rangle\!\rangle \phi_{\mathfrak{r}}$ modality.

We develop π_{ode} by adding this heuristic into $\pi_{\forall\exists}$, and we manage to automatically verify EQ.1 in KeYmaera X. In addition to Eq.1, we apply π_{ode} to a non-trivial case study on a water tank from existing work [**?**]. Its main property is a $\forall\exists$ one on a system-level model of the water tank. The function π_{ode} first applies the heuristic added in $\pi_{\forall\exists}$, and then applies the heuristic added in π_{ode}. We manage to verify the $\mathsf{d}\mathcal{L}$ formula generated by π_{ode} on this property with only one manual proof step.

π_{syn}**:** π_{ode} **+ User Inputs**. We further investigate the utility of the translation by adding a heuristic derived from recent work on relational verification of $\forall\forall$ properties on dynamics (Thm.24[32]). The heuristic aims to reason with $\mathsf{d}\mathcal{L}_{\text{REL}}$ formulas like the following:

$$\phi_{\mathfrak{r}} \to \llbracket (x' = \theta \& \phi_1, y' = \delta \& \phi_2) \rrbracket \psi_{\mathfrak{r}}$$

However, instead of fixing the duration passed by the left and right dynamics as in π_{ode}, this heuristic asks for user inputs that would allow the two dynamics to *synchronize* at different time points, e.g., compare the values of x and y after the left and right dynamics evolve for different durations.

We develop π_{syn} that asks users to provide inputs when it sees appropriate $\mathsf{d}\mathcal{L}_{\text{REL}}$ formulas, and then produces corresponding $\mathsf{d}\mathcal{L}$ formulas. We test π_{syn} with two case studies from [32]: after obtaining the correct inputs, π_{syn} produces $\mathsf{d}\mathcal{L}$ formulas that can be automatically verified by KeYmaera X.

We have demonstrated that we can integrate existing and new techniques into the translation for $\mathsf{d}\mathcal{L}_{\text{REL}}$, to promote its power in verifying general relational properties, as well as domain-specific ones. Note that we manually develop all these variants of π. How to systematically integrate different techniques is an interesting future work.

VII. Related Work

BiKAT. The closest related work is the BiKAT [19], an extension of KAT to help reason with the alignments of relational properties through equational reasoning. A main focus of BiKAT is to ensure *adequacy*: the aligned program should capture all computations of the originial program. Our biprogram construct is analogous to the design in BiKAT. However, a major difference between our work and BiKAT is applicability: the main focus of BiKAT is alignments of two programs, while we also concern domain-specific reasoning. The reasoning rules for BiKAT can be encoded in the translations (as shown in Section VI-C), but BiKAT cannot be used to reason with many $\mathsf{d}\mathcal{L}_{\text{REL}}$ formulas. In addition, the design of modalities promotes expressing complex relational properties concisely, which is not a motivation behind BiKAT.

Relational Reasoning for Dynamic Logic. Beckert et al. have done a series of work on extending dynamic logic with trace modalities [33], [34], [35]. They further apply trace modalities to check secure information flow in the setting of concurrent programs [35]. Their work focus on first-order dynamic logic with deterministic programs. The limitation to deterministic programs is significant because nondeterminism is essential to the utility of dynamic logics, especially in AI-enabled systems [36]. Gutsfeld et al. introduce an expressive extension to propositional dynamic logic to check hyperproperties by introducing path quantifiers [37]. Algorithms for model checking these hyperproperties are introduced. Tool support is left for future work. In comparison, the REL extension goes beyond propositional dynamic logic, and it can reuse existing tools to verify relational properties.

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 [38]. An expressive modal logic based on $\mathsf{d}\mathcal{L}$ has been introduced to reason with nondeducibility [39]. Neither work provides tool support. Kolčák et al. introduce a relational extension of $\mathsf{d}\mathcal{L}$ that focuses on reasoning about $\forall\forall$ on two dynamics [32]. Our translation for $\mathsf{d}\mathcal{L}_{\text{REL}}$ is able to integrate this work. 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 [17]. 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 is an adaptation of the work of Pottier and Simonet on information flow analysis for ML [10]. The work introduces an extension of ML that encodes a pair of ML terms as a single bracket construct. 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, e.g., [40], [41]. The name "biprogram" is also used in other recent work on relational reasoning [11], [19].

Relational Program Logic. Another approach for relational verification is to use relational program logic [42], [43], [44]. Benton introduced Relational Hoare Logic for verifying program transformations [42], which provides a general framework for relational correctness proofs. Recent work proposes a notion to evaluate the design of Relational Hoare Logic [44]. Sousa and Dillig introduce a program logic, named Cartesian Hoare Logic (CHL), for verifying k-safety properties [45]. The work has been further extended for proving the correctness of 3-way merge [46]. They introduce a generalized form of Hoare triples to express relations between different program executions, and use SMT solvers to determine their satisfiability. D'Osualdo et al. [47] describe a logic for hyper-triple composition (LHC) based on weakest preconditions that can decompose a hypersafety proof along the boundary of hyper tuples, offering ways of combining multiple k-safety proofs. These relational program logics often are based on weakest pre-condition and only support $\forall\forall$ pre/post k-safety properties. In contrast, the REL extensions naturally support properties with mixed modalities. Moreover, the translations for REL extensions open opportunities in combining general relational reasoning and and domain-specific reasoning. We believe these relational program logics can be also integrated as a part of the translations, which we leave to future work.

Self-Composition and Product Program. The encodings introduced in this work are inspired by self-composition, which is a common approach for relational reasoning [13], [48], [49], [50]. 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 [18] 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 special kinds of self-compositions that shall be adapted for different host logics. The encoding can benefit from advances in self-composition, as we have shown in Section VI-C.

Other Related Work. Noninterference is a well-known relational property for secure information flow and has been widely studied, e.g., [10], [51], [52], [53], [54]. REL extensions bring in several major benefits for properties like noninterference. First, noninterference reasons about program executions, and modalities in REL capture executions intuitvely. Different variants of noninterference can be concisely expressed, as shown in the examples of Section III. Second, the notion of noninterference can be easily extended to different domains (host logics), as the REL extension connects the common program constructs with domain-specific constructs. And third, verifying noninterference can leverage more existing tools, as the translations can integrate different techniques.

Relational verification has been studied in the setting of temporal-style logics, such as HyperLTL and HyperCTL, have been introduced to model relational properties [55], [56], [22], [37]. Model checking is often used in verify these properties. In contrast, REL extensions are more expressive on program behaviors than logics like HyperLTL and HyperCTL. Translations for the extensions may leverage the model checkers for HyperLTL or HyperCTL if the correct heuristics are developed, which we leave to future work.

VIII. Conclusion

We introduce a general and lightweight relational extension for dynamic logics. Though we don't have a formal proof of the applicability, we expect that the 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. Verifying REL formulas can leverage existing tools developed for the host logics. More importantly, the encodings for a REL extension open opportunities to combine techniques of general relational reasoning and domain-specific relational reasoning, which may greatly promote proof automation.

Future work One immediate next step is to systematically integrate techniques for general relational reasoning and domain-specific relational reasoning into the encoding. Proof automation with the translations is also an interesting future work.

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

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

[4] 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.

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

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

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

[8] A. Platzer, "The complete proof theory of hybrid systems," in *LICS*, 2012, pp. 541–550.

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

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

[11] 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.

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

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

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

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

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

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

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

[19] T. Antonopoulos, E. Koskinen, T. C. Le, R. Nagasamudram, D. A. Naumann, and M. Ngo, "An algebra of alignment for relational verification," *PLDI*, vol. 7, pp. 573–603, 2023.

[20] 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.

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

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

[23] 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.

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

[25] 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.

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

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

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

[29] S. Foster, J. J. Huerta y Munive, M. Gleirscher, and G. Struth, "Hybrid systems verification with Isabelle/HOL: Simpler syntax, better models, faster proofs," in *FM*, ser. LNCS, vol. 13047. Springer, 2021, pp. 367–386.

[30] J. J. Huerta y Munive and G. Struth, "Predicate transformer semantics for hybrid systems," *Journal of Automated Reasoning*, vol. 66, no. 1, pp. 93–139, 2022.

[31] J. Xiang, R. Lanotte, S. Tini, S. Chong, and M. Merro, "Measuring robustness in cyber-physical systems under sensor attacks," *arXiv preprint arXiv:2403.05829*, 2024.

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

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

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

[35] 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.

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

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

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

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

[40] 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.

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

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

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

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

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

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

[47] E. D'Osualdo, A. Farzan, and D. Dreyer, "Proving hypersafety compositionally," in *OOPSLA2*, 2022, pp. 289–314.

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

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

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

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

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

[53] M. Balliu, M. Dam, and G. Le Guernic, "Epistemic temporal logic for information flow security," in *PLAS*, 2011, pp. 1–12.

[54] A. Chudnov and D. A. Naumann, "Assuming you know: Epistemic semantics of relational annotations for expressive flow policies," in *CSF*, 2018, pp. 189–203.

[55] M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez, "Temporal logics for hyperproperties," in *POST*, 2014, pp. 265–284.

[56] R. Beutner and B. Finkbeiner, "Autohyper: Explicit-state model checking for HyperLTL," in *TACAS*, 2023, pp. 145–163.

Appendix

A. *Proof of Theorem 2*

LDL$_{\mathrm{REL}}$ has at least the expressive power of LDL$_f$, since all LDL$_f$ formulas can be encoded in LDL$_{\mathrm{REL}}$ as projection formulas. We focus on proving that some LDL$_{\mathrm{REL}}$ formulas cannot be expressed with LDL$_f$. We proceed by *disproving* the following: for all LDL$_{\mathrm{REL}}$ formula $\phi_{\mathfrak{r}}$, 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 $(\sigma_{\mathsf{L}}, \sigma_{\mathsf{R}}) \models_{\mathrm{RL}} \phi_{\mathfrak{r}}$ 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 $\phi_{\mathfrak{r}}$:

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

Where, a (or b) is an atomic proposition of the left trace σ_{L} (or right trace σ_{R}). This 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 $(\sigma_{\mathsf{L}}, \sigma_{\mathsf{R}}) \models_{\mathrm{RL}} \phi_{\mathfrak{r}}$ 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 $i \geq 0 : xy^iz \in L$. Assume that an 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 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 $(a\,; a, b)^*$. Thus, xy^iz cannot satisfy the first conjunct for all $i \geq 0$. So ϕ cannot be regular.

B. *Proof of Theorem 3*

We use $\omega_{\mathsf{L}} \otimes_{\phi_{\mathfrak{r}}} \omega_{\mathsf{R}}$ as a shorthand of $((\omega_{\mathsf{L}} \Downarrow \mathrm{VAR}_{\mathsf{L}}(\phi_{\mathfrak{r}})) \otimes (\omega_{\mathsf{R}} \Downarrow \mathrm{VAR}_{\mathsf{R}}(\phi_{\mathfrak{r}})))$. 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: $\theta_{\mathfrak{r}_1} \sim \theta_{\mathfrak{r}_2}$, $\lfloor \phi \rfloor_{\mathsf{B}}$, $\neg\phi_{\mathfrak{r}}$, $\phi_{\mathfrak{r}} \wedge \psi_{\mathfrak{r}}$, and $\langle\!\langle \alpha_{\mathfrak{r}} \rangle\!\rangle \phi_{\mathfrak{r}}$. Other formulas can be encoded.

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

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

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

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

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

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 $\mathrm{VAR}_{\mathsf{L}}(\phi_{\mathfrak{r}}) \cap \mathrm{VAR}_{\mathsf{R}}(\phi_{\mathfrak{r}}) = \varnothing$, then $\mathrm{VAR}_{\mathsf{L}}(\psi_{\mathfrak{r}}) \cap \mathrm{VAR}_{\mathsf{R}}(\psi_{\mathfrak{r}}) = \varnothing$ for all subformulas $\psi_{\mathfrak{r}}$ in $\phi_{\mathfrak{r}}$.

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

Similar lemmas exist 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 $\omega_{\mathsf{L}}, \omega_{\mathsf{R}}, \nu_{\mathsf{L}}, \nu_{\mathsf{R}}$, programs α, β such that $\mathrm{VAR}(\alpha) \cap \mathrm{VAR}(\beta) = \varnothing$, then $(\omega_{\mathsf{L}}, \nu_{\mathsf{L}}) \in [\![\alpha]\!]$ and $(\omega_{\mathsf{R}}, \nu_{\mathsf{R}}) \in [\![\beta]\!]$ iff $(\omega_{\mathsf{L}} \Downarrow \alpha \otimes \omega_{\mathsf{R}} \Downarrow \beta, \nu_{\mathsf{L}} \Downarrow \alpha \otimes \nu_{\mathsf{R}} \Downarrow \beta) \in [\![\alpha\,; \beta]\!]$.

Proof. By Lemma 4 and the semantics. □

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

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

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

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

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

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 $\mathrm{VAR}_{\mathsf{L}}(\phi_{\mathfrak{r}}) \cap \mathrm{VAR}_{\mathsf{R}}(\phi_{\mathfrak{r}}) = \varnothing$ and $\omega \models \pi(\phi_{\mathfrak{r}})$, then $(\omega \Downarrow \mathrm{VAR}_{\mathsf{L}}(\phi_{\mathfrak{r}}), \omega \Downarrow \mathrm{VAR}_{\mathsf{R}}(\phi_{\mathfrak{r}})) \models_{\mathrm{RL}} \phi_{\mathfrak{r}}$.

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 $\lfloor \phi \rfloor_{\mathsf{B}}$: by the definition of π and case analysis of 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, $\omega \Downarrow \mathrm{VAR}_{\mathsf{R}}(\phi_{\mathfrak{r}}) = \nu_1 \Downarrow \mathrm{VAR}_{\mathsf{R}}(\phi_{\mathfrak{r}})$ and $\nu_1 \Downarrow \mathrm{VAR}_{\mathsf{L}}(\phi_{\mathfrak{r}}) = \nu_2 \Downarrow \mathrm{VAR}_{\mathsf{L}}(\phi_{\mathfrak{r}})$. Thus, we know $((\omega \Downarrow \mathrm{VAR}_{\mathsf{L}}(\phi_{\mathfrak{r}}), \omega \Downarrow \mathrm{VAR}_{\mathsf{R}}(\phi_{\mathfrak{r}})), (\nu_2 \Downarrow \mathrm{VAR}_{\mathsf{L}}(\phi_{\mathfrak{r}}), \nu_2 \Downarrow

$\mathrm{VAR}_{R}(\phi_{\mathfrak{r}}))) \in \llbracket(\alpha,\beta)\rrbracket_{RL}$ Then by the semantics of $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ programs, IH, and Lemma 7, this case is proven.

That concludes the proof. $\square$

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

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

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 $(\omega_{L}, \omega_{R}) \models_{RL} \phi_{\mathfrak{r}}$ then $(\omega_{L} \otimes_{\phi_{\mathfrak{r}}} \omega_{R}) \models \pi(\phi_{\mathfrak{r}})$:

- 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 $\lfloor\phi\rfloor_{B}$: by examining the definition of π and case analysis of 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 (α, β): by the semantics of $\mathsf{d}\mathcal{L}_{\mathrm{REL}}$ programs, there exists ν_{L}, ν_{R} such that $(\omega_{L}, \nu_{L}) \in \llbracket\alpha\rrbracket$ and $(\omega_{R}, \nu_{R}) \in \llbracket\beta\rrbracket$, then by IH and Lemma 6.

The other direction can be proven by Lemma 10 and 8. $\square$

Proof of Theorem 3. By Lemma 11 and 2. $\square$

C. *Encoding* LDL$_{\mathrm{REL}}$ *Formulas with* LDL$_f$

We develop a π for LDL$_{\mathrm{REL}}$ based on the following insight: a LDL$_{\mathrm{REL}}$ formula (after renaming) is satisfiable if it is a conjunction of a characterization of the left trace and a characterization of the right one. Consider again the formula $\langle\!\langle(a,b)\rangle\!\rangle\lfloor\top\rfloor_{R} \wedge \langle\!\langle(?\top,\neg b)\rangle\!\rangle\lfloor\top\rfloor_{R}$, 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$_{\mathrm{REL}}$ formula.

For a LDL$_{\mathrm{REL}}$ 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$_{\mathrm{REL}}$ formula $\langle\!\langle((a\,;\,a)\cup a, b)\rangle\!\rangle\lfloor\top\rfloor_{R}$ is satisfiable if either of the following LDL$_f$ formulas is satisfiable: $\langle a\rangle\top \wedge \langle b\rangle\top$ or $\langle a\,;\,a\rangle\top \wedge \langle b\rangle\top$.

Using these insights, we develop π to find a set of LDL$_f$ conjunctions each of which is satisfiable only if the input LDL$_{\mathrm{REL}}$ formula is satisfiable. π is sound but incomplete: it is a *partial* function that works on a subset of LDL$_{\mathrm{REL}}$ formulas. For formulas of the form $\langle\!\langle\alpha^*_{\mathfrak{r}}\rangle\!\rangle\phi_{\mathfrak{r}}$, π 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 $\llbracket\alpha^*_{\mathfrak{r}}\rrbracket\phi_{\mathfrak{r}}$, 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 put LDL$_{\mathrm{REL}}$ formulas in *negation normal form*, by exploiting abbreviations and pushing negation inside (e.g., $\neg\llbracket\alpha_{\mathfrak{r}}\rrbracket\phi_{\mathfrak{r}} = \langle\!\langle\alpha_{\mathfrak{r}}\rangle\!\rangle\neg\phi_{\mathfrak{r}}$) as much as possible, leaving negations only in front of atomic formulas. Essentially, we restrict our attention to LDL$_{\mathrm{REL}}$ formulas formed by the following syntax:

$\phi_{\mathfrak{r}}, \psi_{\mathfrak{r}} ::= \lfloor\phi\rfloor_{B} \mid \neg\lfloor\phi\rfloor_{B} \mid \phi_{\mathfrak{r}} \vee \psi_{\mathfrak{r}} \mid \phi_{\mathfrak{r}} \wedge \psi_{\mathfrak{r}} \mid \langle\!\langle\alpha_{\mathfrak{r}}\rangle\!\rangle\phi_{\mathfrak{r}} \mid \llbracket\alpha_{\mathfrak{r}}\rrbracket\phi_{\mathfrak{r}}$

$\alpha_{\mathfrak{r}}, \beta_{\mathfrak{r}} ::= (\alpha,\beta) \mid \alpha_{\mathfrak{r}}\,;\,\beta_{\mathfrak{r}} \mid \alpha_{\mathfrak{r}} \cup \beta_{\mathfrak{r}} \mid \alpha^*_{\mathfrak{r}} \mid ?\phi_{\mathfrak{r}}$

Definition 6 (π for LDL$_{\mathrm{REL}}$)**.** *For a* LDL$_{\mathrm{REL}}$ *formula* $\phi_{\mathfrak{r}}$ *that* $\mathrm{VAR}_{L}(\phi_{\mathfrak{r}}) \cap \mathrm{VAR}_{R}(\phi_{\mathfrak{r}}) = \varnothing$, *a partial function* π *that produces a set of pairs of* LDL$_f$ *formulas is inductively defined:*

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

Most cases in Definition 6 follow the semantics of LDL$_{\mathrm{REL}}$ 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.
- $\pi(\llbracket\alpha^*_{\mathfrak{r}}\rrbracket\phi_{\mathfrak{r}})$ 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.
- $\pi(\llbracket(\alpha,\beta)\rrbracket\phi_{\mathfrak{r}})$ expands cases of α and β to look for all possible paths of $\llbracket(\alpha,\beta)\rrbracket$. 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$_{\text{REL}}$)**.** *For a* LDL$_{\text{REL}}$ *formula* $\phi_{\mathfrak{r}}$ *and a renaming function* ξ *for* $\phi_{\mathfrak{r}}$*,*

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

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

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 $\alpha_{\mathfrak{r}}$.