Heterogeneous Dynamic Logic: Provability Modulo Program Theories

SAMUEL TEUBER, Karlsruhe Institute of Technology (KIT), Germany
MATTIAS ULBRICH, Karlsruhe Institute of Technology (KIT), Germany
ANDRÉ PLATZER, Karlsruhe Institute of Technology (KIT), Germany
BERNHARD BECKERT, Karlsruhe Institute of Technology (KIT), Germany

Formally specifying, let alone verifying, properties of systems involving multiple programming languages is inherently challenging. We introduce *Heterogeneous Dynamic Logic* (HDL), a framework for combining reasoning principles from distinct (dynamic) program logics in a modular and compositional way. HDL mirrors the architecture of satisfiability modulo theories (SMT): Individual dynamic logics, along with their calculi, are treated as *dynamic theories* that can be combined to reason about heterogeneous systems whose components are verified using different program logics. HDL provides two key operations: *Lifting* extends an individual dynamic theory with new program constructs (e.g., the havoc operation or regular programs) and automatically augments its calculus with sound reasoning principles for the new constructs; and *Combination* enables cross-language reasoning in a *single modality* via *Heterogeneous Dynamic Theories*, facilitating the reuse of existing proof infrastructure. By lifting combined theories with regular programs, we obtain *heterogeneous* control structures that allow us to reason about intertwined cross-language behavior. We formalize dynamic theories, their lifting and combination, and prove the soundness of all proof rules in Isabelle. We also introduce a proof rule combining deductive DL-based reasoning with reasoning principles from Kleene Algebras with Tests. Furthermore, we prove *relative completeness* theorems for lifting and combination: Under usual assumptions, reasoning about lifted or combined theories is no harder than reasoning about the constituent dynamic theories and their common first-order structure (i.e., the "data theory"). We demonstrate HDL's value by verifying an automotive case study where a Java controller (formalized in Java dynamic logic) steers a plant model (formalized in differential dynamic logic).

CCS Concepts: • **Theory of computation → Program verification**; **Logic and verification**; **Modal and temporal logics**; • **Software and its engineering → Software verification**.

Additional Key Words and Phrases: Dynamic Logic, Theory Combination, Relative Completeness, Isabelle

ACM Reference Format:
Samuel Teuber, Mattias Ulbrich, André Platzer, and Bernhard Beckert. 2026. Heterogeneous Dynamic Logic: Provability Modulo Program Theories. *Proc. ACM Program. Lang.* 10, PLDI, Article 217 (June 2026), 24 pages. https://doi.org/10.1145/3808295

1 Introduction

Since its origins in Hoare logic [29] and Dijkstra's weakest precondition predicate transformers [18], formal verification has grown into a mature discipline, supporting a wide range of languages, from bytecode [42, 58], C [16, 34], and Java [2, 11], to modern languages like Rust [4, 37] and even to hybrid programs for cyber-physical systems [44]. While these individual "worlds" enjoy robust verification infrastructures, it remains challenging to *rigorously combine* verification results across languages. Yet, many modern systems, especially cyber-physical systems, are inherently heterogeneous and would benefit from verification methodology enabling *compositional* reasoning across program

Authors' Contact Information: Samuel Teuber, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, teuber@kit.edu; Mattias Ulbrich, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, ulbrich@kit.edu; André Platzer, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, platzer@kit.edu; Bernhard Beckert, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, beckert@kit.edu.

logics. Crucially, this is not merely a *tooling* challenge, but raises the fundamental questions of how semantics and proof principles of distinct program logics can be combined to *specify* and *verify* heterogeneous systems. In practice, heterogeneous verification today often resorts to encoding one system (e.g., a software controller) in the language of another (e.g., hybrid programs) [25, 33, 35, 54]. However, such encodings are neither seamless nor intuitive: They obscure structure due to incompatible programming paradigms and fail to reuse existing tools available for the individual languages.

Dynamic Logic. We argue that Dynamic Logic (DL) provides a strong foundation for reasoning about heterogeneous systems due to its expressiveness and its structural properties. DL generalizes traditional verification techniques [18, 29] by treating programs as first-class citizens of the logic embedded within modalities (rather than residing outside the logic). This enables DL to specify a large variety of systems including imperative languages [27] or hybrid systems [44]. Crucially, DL's treatment of programs gives rise to a fully compositional logic that naturally supports properties such as liveness [53] or incorrectness [50] as well as hyperproperties (e.g. refinement [38, 48, 59] or secure information flow [8]) *without necessitating specialized logics* (as necessary for, e.g., Hoare logic). However, prior DL frameworks are confined to a *single, fixed programming language*, and are thus ill-suited for verifying heterogeneous systems that combine components from multiple "worlds" (e.g. Java code and hybrid systems).

As a lighter-weight counterpart to (propositional) DL, Kleene Algebra with Tests [36] (KAT) is an established, effective methodology for (automated) program verification. We introduce a proof calculus capable of combining KAT reasoning with deductive DL reasoning. Section 7 demonstrates how this combination can be leveraged to ease heterogeneous systems verification.

Satisfiability Modulo Theories. The success of satisfiability modulo theories (SMT) has transformed first-order reasoning across a wide range of applications: By requiring a *unified set of assumptions* (e.g., stable infiniteness [41]) from first-order theories, SMT solvers can delegate theory-specific queries, e.g., involving integers or arrays, to specialized sub-solvers. Users can freely combine theories within a single formula while theory combination mechanisms ensure soundness and, under suitable assumptions, even completeness. The seminal results by Shostak, Nelson and Oppen, and their subsequent extensions [6, 40, 41, 52, 56, 57] thus paved the way for the modular combination of first-order theories. This enables a *separation of concerns* between theory-specific reasoning and the logical structure connecting the theories. Inspired by this design, we ask: Can a similar, modular structure be applied to program logics? That is, can we reason about heterogeneous programs while preserving the semantics and proof rules established for individual languages? This paper answers the question in the affirmative, building on the rigorous foundations of dynamic logic.

Contribution. This paper introduces *Heterogeneous Dynamic Logic* (HDL): The foundational formal notions that allow DL to serve as a generalized, SMT-inspired vehicle to combine different *program logics* (as *dynamic theories*), just like first-order logic serves as a vehicle to combine of different *data logics* (as first-order theories). Heterogeneous Dynamic Logic supports two key operations:

- *Lifting* (Section 6) extends a given *dynamic theory* Δ with additional program constructs, such as nondeterministic choice ($\Delta^{:=*}$, Section 6.1) or the closure over regular programs (Δ^{reg}, Section 6.2). Lifting also automatically extends the dynamic theory's calculus with sound proof rules to reason about the extended programming language.
- *Combination* (Section 7) merges two dynamic theories $\Delta^{(0)}$ and $\Delta^{(1)}$ (defined over distinct programming languages $\Lambda_P^{(0)}, \Lambda_P^{(1)}$) into a single *heterogeneous dynamic theory* $\hat{\Delta}$. This new dynamic theory supports *heterogeneous programs* in which programs from $\Lambda_P^{(0)}$ and $\Lambda_P^{(1)}$ can be composed freely using regular programs without the necessity of embeddings required in

prior approaches. Combination also merges and extends the calculi of the individual theories into a common, sound calculus for the heterogeneous dynamic theory.

This enables rigorous, modular reasoning over *heterogeneous programs* with deeply intertwined behavior of programs stemming from individual logics. Along with a relatively complete proof calculus for lifted/combined dynamic theories, we present an axiom leverages KAT-style equational rewriting – coupling KAT's reasoning principles with DL's proof system. We introduce a formalization of dynamic theories, their lifting and combination and their sound proof calculi in Isabelle in a reusable manner via its Locale infrastructure (formalized Lemmas/Theorems are in cyan, see **??**).

Overview. The remainder of this paper is structured as follows: Section 2 motivates the challenge of heterogeneous verification using a concrete case study. Section 3 surveys prior attempts to support general cross-language verification. Sections 4 and 5 introduce our notion of *dynamic theories*, a generalization of dynamic logic with reduced assumptions, and demonstrate that our assumptions suffice to recover well-known properties of "DL-like" logics. In this context, we also connect DL's proof calculus to equational reasoning via KAT. Section 6 presents *lifting*, an approach to extend a given dynamic theory with additional program featues such as regular programs along with deriving the necessary proof rules. To this end, we also prove that our regular program lifting results in a KAT which enables tool reuse. Section 7 proves that the *heterogeneous dynamic theory* over two dynamic theories is once again a *dynamic theory*. Corresponding additional proof rules are derived including additional KAT-style identities which ease verification in practice. Finally, Section 8 establishes conditions under which *relative completeness* transfers from a dynamic theory to its lifted and/or combined version. Section 9 showcases the utility of HDL on our automotive case study.

2 Motivation: Case Study

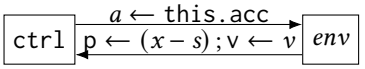
As an exemplary heterogeneous system we consider a car steered by a software controller written in Java while driving towards a stop sign. To effectively leverage existing proof infrastructures, we wish to model the time-continuous, physical system in Differential Dynamic Logic [44] ($d\mathcal{L}$) using KeYmaera X [22, 43, 44, 47]. In contrast, we wish to verify the controller using Java Dynamic Logic [7] (JavaDL) via the verification tool KeY.

```
1  int x1 = v+1;
2  int x2 = x1*x1;
3  if(10000*x2 + (amax-amin)*T*
4     (amax*T + 200*x1) > -20000*amin*(p-1)) {
5    this.acc = amin;
6  } else {
7    this.acc += (p>10000) ? 10 : -10;
8    this.acc = (this.acc > amax) ? amax :
9               (this.acc < amin) ? amin :
10              this.acc;  }
```

(a) Java Controller `ctrl` that can be analyzed using JavaDL.

$$env \equiv \Big(t := 0; (x' = v, v' = a, t' = 1 \;\&\; t \leq T)\Big)^*;$$

(b) Environment model that can be analyzed in Differential Dynamic Logic ($d\mathcal{L}$).



(c) The heterogeneous system (← denotes some kind of assignment; semantics see Section 9).

Fig. 1. Heterogeneous system case study

Java Dynamic Logic. JavaDL allows reasoning about Java programs [2, 7, 9] (including exceptions, heap state etc.). Consider the method in Figure 1a serving as controller for a car approaching a stop sign at distance `p` with velocity `v`: This method computes an acceleration via the *stateful* field `this.acc` and brakes in case the condition in line 3 is satisfied. The method is not idempotent due to `this.acc` and its computation also depends on static variables `amax`, `amin` and `T`. While JavaDL (and KeY) is capable of verifying Java programs, JavaDL cannot reason about physical dynamics, e.g. given as differential equations.

Differential Dynamic Logic. $d\mathcal{L}$ [44] allows reasoning about *hybrid programs* whose states evolve either along discrete state transitions (variable assignments) or continuously along differential equations. Consider the hybrid program env in Figure 1b: It sets a clock variable t to 0 and evolves x,

v and t along a differential equation for a maximum time of T. The program describes the evolution of a car's position under constant acceleration a. Using the proof calculus implemented in the theorem prover KeYmaera X [22, 43, 44, 47], we can (only) prove statements about hybrid programs.

Challenge. Both of the introduced verification methodologies come with strengths and weaknesses: While they excell at verifying properties for their "native" domain of computation, they lack support for their respective counterpart. However, to verify the safety of the entire heterogeneous system, we require a formalism that allows us to reason about the looped heterogeneous system sketched in Figure 1c. We propose a *general* framework that gives a formal semantic to this sketch and provides a proof calculus that emphasizes the *reusage* of proof infrastructure. Our framework is not tied to the particular combination of languages discussed in this case study.

3 Related Work

We emphasize differences in comparison to related work here while deferring a more detailed related work discussion to **??**. *Dynamic Logics* have been instantiated for a wide range of behavioral languages and properties [2, 3, 8, 10, 13, 17, 31, 38, 44, 45, 46, 50, 51, 53, 58, 59]. Unlike prior works, we study their common principles and how their semantics can be combined in a uniform fashion. Unlike *Parameterized Dynamic Logic* [60], we address the challenge of combining programming languages while retaining existing proof calculi and provide a formalization of our results. In contrast, literature on *Satisfiability Modulo Theories* extensively investigated theory combination [6, 40, 41, 56, 57], but is often limited to quantifier-free first-order segments and does not admit program modalities. While deductive verification systems with *intermediate languages* [5, 20, 39] or some *model checking based techniques* with a common exchange format [15, 32] enable the analysis of heterogeneous systems, we deliberately avoid the need for a common shared programming language. This, e.g., enables the integration of continuous program state transitions as observed in, e.g., differential dynamic logic [44, 45]. Unlike *Fibring* [23, 24], our combination of logics goes beyond the combination of "truth bearing" entities, by equally fibring programming languages and their semantics (see Section 7.1). While the *Unified Theories of Programming* [21, 30] provide a common framework for integrating diverse programming languages, they assume shared state and require a reimplementation of program semantics inside the common framework. In contrast, we emphasize reusage of proof infrastructure by reducing verification problems to the individual logics. In comparison to *contract-based* verification approaches (e.g. CONTRACT-LIB [19] or PolyVer [14]), we support a wider range of properties as over-approximating contracts limit application to safety properties. Additionally, we explicitly formalize the interaction between homogeneous components, i.e. we require no meta-level argument on the possibility or semantics of combination. Moreover, unlike other approaches, our framework comes with a relative completeness guarantee: Under reasonable assumptions any correct heterogeneous system can be proven correct. *Kleene Algebra Modulo Theories* [26] (KAMT) proposes an approach to turn a client theory into a Kleene Algebra with Tests. The objective of this framework is conceptually related to our regular program lifting and product-based combination approach. In contrast to KAMT, we derive a dynamic logic which admits quantifiers and hence provides us with a significantly more expressive logic that supports a wider range of (hyper-)properties due to dynamic logic's compositional nature. While this expressivity prohibits the derivation of decidability results, we are able to prove relative completeness of our calculus. As we show with axiom (E), where applicable, we can also integrate KAT/KAMT results into our reasoning while, in general, providing a more powerful logic.

4 Dynamic Theories

The foundation of our approach is the idea behind First-Order Dynamic Logic (FODL) [27, 28], which extends first-order logic to a multi-modal logic where programs parameterize modalities. For instance, for an integer first-order theory, the formula

$$\underbrace{0 \leq v}_{\text{atom}} \rightarrow \overbrace{[\underbrace{w \coloneqq v+1}_{\text{program}}]}^{\text{modal operator}}, \underbrace{1 \leq w}_{\text{atom}} \tag{1}$$

states that after all execution of $w \coloneqq v + 1$, w is positive if v was non-negative. Unlike Hoare triples [29] or predicate transformers [18], Dynamic Logic treats programs as first-class citizens of formulas, enabling compositional reasoning about richer properties such as reachability ($\langle w \coloneqq v + 1\rangle\ w > 0$) and hyper-properties ($[p]\ \langle q\rangle\ F$).

Our contribution generalizes this principle: not only the first-order theory but also the underlying program language can be chosen freely. We introduce *dynamic theories*: A minimal semantic interface capturing the assumptions on programs and first-order reasoning. Concrete program logics (existing or new) are instantiations of a dynamic theory; multiple such theories can be composed into a heterogeneous dynamic theory that still satisfies the same interface. This composability allows "stacking" theories while retaining derived proof rules. In Section 8 we show that this suffices for a relative completeness result: Under reasonable assumptions, two relatively complete calculi for two theories can be merged into one for their heterogeneous combination. By parameterizing over both programs and atoms, our framework unifies diverse program logics and provides a reusable foundation akin to SMT theory combination — ensuring that compliant dynamic logics can interoperate and inherit proof rules automatically.

4.1 Syntax

To focus the presentation on the modal aspects of the logic, we abstract away from the term structure of a Dynamic Logic's first-order fragment and instead begin our investigation directly at the level of first-order atoms. Consequently, the syntactic material of a Dynamic Logic is given via the set of variables Λ_V the set of its first-order atoms Λ_A and the set of its programs Λ_P. We call this the *dynamic signature* of a dynamic theory:

Definition 4.1 (Dynamic Signature)**.** *A* dynamic signature Λ *is given by a set of variables* Λ_V*, a set of first-order atoms* Λ_A*, and the set of programs* Λ_P *with all three sets pairwise disjoint. We denote this as* $\Lambda = (\Lambda_V, \Lambda_A, \Lambda_P)$

Given a signature Λ, we can define the structure of formulas that are part of a dynamic theory in a manner that is similar to classical FODL while constraining programs and variables to all syntactical constructs contained in Λ:

Definition 4.2 (Λ-formulas)**.** *Let* $\Lambda = (\Lambda_V, \Lambda_A, \Lambda_P)$ *be a dynamic signature. Given a variable* $v \in \Lambda_V$*, an atom* $a \in \Lambda_A$*, and a program* $p \in \Lambda_P$*,* Λ*-formulas are defined by the following grammar:* $F, G ::= a \mid \neg F \mid F \wedge G \mid \forall v\ F \mid [p]\ F$

The set of all Λ-Formulas is denoted as Fml_Λ. We denote the first-order fragment (i.e. all formulas without any box modality) as $\mathrm{FOL}_\Lambda \subset \mathrm{Fml}_\Lambda$. Further, we introduce syntactic sugar for the diamond modality, which we denote as $\langle p\rangle\ F \equiv \neg\,[p]\,\neg F$. Similarly, syntactic sugar can be introduced for existential quantifiers and other logical connectives ($\vee$, $\rightarrow$, ...).

Example 4.1 (Syntax for Ordered Semirings)**.** As a running example, consider an ordered semi-ring $(\mathcal{R}, +, \cdot, \leq)$ and an arbitrary set of variables $\Lambda_V^{\mathcal{R}} = \{v_1, v_2, \dots\}$: We define terms as literals $c \in L \subseteq \mathcal{R}$,

variables $v \in \Lambda_V^{\mathcal{R}}$ and any composition of terms with + and $\cdot$. Atoms are any formulas of the form $t_1 \leq t_2$ and programs have the form $v := t_1$ (for t_1, t_2 terms and $v \in \Lambda_V^{\mathcal{R}}$). Together, these components form a dynamic signature containing variables, atoms and programs.

4.2 Semantics

The previous section outlined the syntactic constructs necessary to define a dynamic theory. With formulas of our dynamic theory defined, we can now focus on the semantic constructs that are necessary to define a dynamic theory. To this end, we require five components: First, since we are considering a modal logic, we require a state space $\mathcal{S}$ over which we can evaluate Λ-formulas. Secondly, we require functions to evaluate the values of variables ($\mathcal{V}$) w.r.t. some universe ($\mathcal{U}$), the truth-value of atoms ($\mathcal{A}$), and the state-transition behavior of programs ($\mathcal{P}$). Each of these components must satisfy some conditions. As will be shown below, once these components are defined, we can define the evaluation of formulas. We now begin by defining state spaces and variable evaluations:

Definition 4.3 (Universes, State Spaces and Variable Evaluation)**.** *Given a dynamic signature Λ with variables Λ_V and a set $\mathcal{U} \neq \emptyset$ called* universe*, a* state space $\mathcal{S} \neq \emptyset$ *is defined as a set of states together with a* variable evaluation function $\mathcal{V} : \mathcal{S} \times \Lambda_V \to \mathcal{U}$ *such that $\mathcal{S}$ and $\mathcal{V}$ satisfy:*

($\mathcal{S}$ Independence) *For all $\mu, \sigma \in \mathcal{S}$ and $V \subseteq \Lambda_V$, there exists an $\omega \in \mathcal{S}$ such that for all $v \in \Lambda_V$:*
$$\mathcal{V}(\omega, v) = \mathcal{V}(\mu, v) \text{ if } v \in V \text{ and else } \mathcal{V}(\omega, v) = \mathcal{V}(\sigma, v)$$

This definition of state spaces slightly deviates from popular definitions of Dynamic Logic that often consider the state space as the set of all mappings from Λ_V to $\mathcal{U}$ which for us corresponds to defining $\mathcal{V}(\mu, v) = \mu(v)$. However, our set-based definition allows for more complex state spaces. For example, we can globally exclude the possibility of a variable v taking on certain values $\delta \in \mathcal{U}$, which can be leveraged for typed/sorted state spaces. Similarly, we can construct state spaces that are formed by composing states from other state spaces (an example of this can be found in Section 7). Importantly, the interpolation property retains a notion of well-formedness of the state space: If there exists a state where a variable $v \in \Lambda_V$ is assigned some value $\delta \in \mathcal{U}$, then any other state can also be modified so that v has that value δ. For example, this can ensure that a variable which (in a more fine-grained view of the dynamic theory) is an integer variable can indeed be assigned all integer values independently of the remaining state. The interpolation property is important for proving coincidence properties for formulas and programs that are crucial for the derivation of the proof rules of dynamic theories. Going forward, for states $\mu, \sigma \in \mathcal{S}$ we will say μ and σ are equal on $V \subseteq \Lambda_V$ (denoted $\mu \equiv_V \sigma$) whenever for all $v \in V$ it holds that $\mathcal{V}(\mu, v) = \mathcal{V}(\sigma, v)$.

With state spaces defined, we can now turn to the evaluation of atoms, which requires a function that ensures that the evaluation of atoms depends only on a finite set of variables:

Definition 4.4 (Atom Evaluation)**.** *Given a dynamic signature Λ with variables Λ_V and atoms Λ_A and a state space $\mathcal{S}$ we define a* free variable overapproximation $\mathbf{FV}_A : \Lambda_A \to 2^{\Lambda_V}$ *and an* atom evaluation function $\mathcal{A} : \Lambda_A \to 2^{\mathcal{S}}$ *as any functions satisfying the following properties:*

($\mathbf{FV}_A$ Coverage) *For any atom $a \in \Lambda_A$ and any states $\mu, \sigma \in \mathcal{S}$*
$$\text{if } \mu \equiv_{(\mathbf{FV}_A(a))} \sigma \text{ then } \mu \in \mathcal{A}(a) \text{ iff } \sigma \in \mathcal{A}(a)$$
($\mathbf{FV}_A$ Finite) *$\mathbf{FV}_A(a)$ is finite for all atoms $a \in \Lambda_A$.*

While the overapproximation property ensures that $\mathbf{FV}_A$ covers all variables that influence the evaluation of a given atom, the finiteness property ensures that atoms are well-behaved in a sense that is classically expected in first-order logic (where only variables that syntactically occur in the finite formula can have an impact). These two properties are once again of particular importance for coincidence properties essential for deriving the dynamic theory proof rules.

With the semantics of state spaces and atoms defined, we can now move on to the evaluation of programs. Similarly to the semantics of atoms, the semantics of programs also have side conditions on how their behavior is influenced by variables. To this end, we define the program evaluation function to satisfy the following constraints:

Definition 4.5 (Program Evaluation)**.** *Given a dynamic signature Λ with variables Λ_V and programs Λ_P and a state space $\mathcal{S}$ we define a* free variable overapproximation $\mathbf{FV}_P : \Lambda_P \to 2^{\Lambda_V}$ *and a* program evaluation function $\mathcal{P} : \Lambda_P \to 2^{\mathcal{S}^2}$ *as any functions satisfying the following properties:*

($\mathbf{FV}_P$ Coverage) *For all programs $p \in \Lambda_P$, all states $\mu, \sigma, \omega \in \mathcal{S}$, and any set $V \supseteq \mathbf{FV}_P(p)$:*
if $\mu \equiv_V \sigma$ and $(\mu, \omega) \in \mathcal{P}(p)$,
then there exists a state $\tilde{\omega}$ such that $(\sigma, \tilde{\omega}) \in \mathcal{P}(p)$ and $\omega \equiv_V \tilde{\omega}$

($\mathbf{FV}_P$ Finite) *For all programs $p \in \Lambda_P$ the result of $\mathbf{FV}_P(p)$ is finite.*

(Extensionality) *For all programs $p \in \Lambda_P$ and all states $\mu, \tilde{\mu}, \sigma, \tilde{\sigma} \in \mathcal{S}$:*
if $\mu \equiv_{\Lambda_V} \tilde{\mu}$ and $\sigma \equiv_{\Lambda_V} \tilde{\sigma}$, then $(\mu, \sigma) \in \mathcal{P}(p)$ iff $(\tilde{\mu}, \tilde{\sigma}) \in \mathcal{P}(p)$

The over-approximation property establishes that any variable possibly influencing the state transition of a program p must be part of $\mathbf{FV}_P(p)$. This is achieved by ensuring that if two states agree on a superset of $\mathbf{FV}_P(p)$, then there must exist "isomorphic" state transitions, i.e. state transitions in $\mathcal{P}(p)$ where the resulting states equally agree on this set of variables. For the remainder of this paper, it is paramount that this property is not just satisfied for $\mathbf{FV}_P(p)$, but also for its supersets to ensure equivalent effects on any variables written by p. The second condition ensures that this new set of free variables is finite, too, which is relevant for central coincidence properties. Finally, the extensionality property ensures that program behavior is only dependent on variable assignment and not intensional differences between states. This is necessary to prove (syntactic and semantic) coincidence properties. Based on these definitions, we can then define the *domain of computation*. Together with the underlying syntactic material, this makes up a *dynamic theory*:

Definition 4.6 (Domain of Computation)**.** *Given a dynamic signature Λ, a* domain of computation *of Λ (denoted as $\mathcal{D} = (\mathcal{U}, \mathcal{S}, \mathcal{V}, \mathcal{A}, \mathbf{FV}_A, \mathcal{P}, \mathbf{FV}_P)$) consists of:*

- *A universe $\mathcal{U}$ (see Definition 4.3)*
- *A corresponding state space $\mathcal{S}$ (see Definition 4.3)*
- *A variable evaluation function $\mathcal{V}$ (see Definition 4.3)*
- *An atom evaluation function $\mathcal{A}$ and free variable overapproximation $\mathbf{FV}_A$ (see Definition 4.4)*
- *A program evaluation function $\mathcal{P}$ and free variable overapproximation $\mathbf{FV}_P$ (see Definition 4.5)*

Definition 4.7 (Dynamic Theory)**.** *A* dynamic theory *is the combination of a dynamic signature Λ and a corresponding domain of computation $\mathcal{D}$. We denote a dynamic theory as $\Delta = (\Lambda, \mathcal{D})$.*

Given a dynamic theory $\Delta = (\Lambda, \mathcal{D})$, we can then define the semantics of the Λ-formulas described in Definition 4.2 as follows (for $V \subseteq \Lambda_V$ we denote by $V^{\complement}$ the complement set $\Lambda_V \setminus V$):

Definition 4.8 (Semantics of Formulas)**.** *Consider a dynamic theory $\Delta = (\Lambda, \mathcal{D})$. We define the semantics of Λ-formulas as follows:*

$$\mathcal{D}[\![a]\!] = \mathcal{A}(a) \qquad \mathcal{D}[\![\neg F]\!] = \mathcal{S} \setminus \mathcal{D}[\![F]\!] \qquad \mathcal{D}[\![F \wedge G]\!] = \mathcal{D}[\![F]\!] \cap \mathcal{D}[\![G]\!]$$

$$\mathcal{D}[\![\forall v\ F]\!] = \left\{ \mu \in \mathcal{S} \;\middle|\; \text{for all } \nu \in \mathcal{S} : \mu \equiv_{\{v\}^{\complement}} \sigma \text{ implies } \sigma \in \mathcal{D}[\![F]\!] \right\}$$

$$\mathcal{D}[\![[p]\ F]\!] = \left\{ \mu \in \mathcal{S} \;\middle|\; \text{for all } \nu \in \mathcal{S} : (\mu, \sigma) \in \mathcal{P}(p) \text{ implies } \sigma \in \mathcal{D}[\![F]\!] \right\}$$

We say F is Δ-*valid* (denoted as $\models_\Delta F$) iff every state in $\mathcal{S}$ satisfies F, i.e. $\mathcal{S} = \mathcal{D}[\![F]\!]$. If a particular state $\mu \in \mathcal{S}$ satisfies a formula F (i.e., $\mu \in \mathcal{D}[\![F]\!]$) we denote this as $\mu \models_\Delta F$ and say μ satisfies F

or that F is Δ-satisfiable. For example, the formula $F \wedge [p]\ G$ is true in a state μ if $\mu \models_\Delta F$ (i.e. μ satisfies F) and if for every $(\mu, \sigma) \in \mathcal{D}[\![p]\!]$ it holds that $\sigma \models_\Delta G$ (i.e. every such σ satisfies G).

Example 4.2 (Semantics of Ordered Semiring)**.** Recall the dynamic signature we constructed in Example 4.1 for the ring $(\mathcal{R}, +, \cdot, \leq)$: We define the universe $\mathcal{U} = \mathcal{R}$ and the state space as the set of all mappings from $\mu : \Lambda_V \to \mathcal{U}$. The variable evaluation function then is $\mathcal{V}^{\mathcal{R}}(\mu, v) = \mu(v)$. For example, consider the case of $\mathcal{R} = \mathbb{Z}$: In this case, a variable can take on any integer value. A concrete example of a formula in the dynamic theory over $(\mathbb{Z}, +, \cdot, \leq)$ is Equation (1). The atom evaluation function takes a state μ and evaluates an atomic formula (and its resp. terms) w.r.t. μ (i.e. $\mu \in \mathcal{A}(t_1 \leq t_2)$ iff $\mu(t_1) \leq \mu(t_2)$ in $\mathbb{Z}$, where $\mu(t)$ evaluates the variables in term t w.r.t. μ). For example, a state μ satisfies the atom $0 \leq v$ in Equation (1) iff $\mathcal{V}^{\mathbb{Z}}(\mu, v) = \mu(v) \geq 0$. The program evaluation function for $v := t_1$ induces a transition relation $(\mu, \sigma) \in \mathcal{P}^{\mathcal{R}}(v := t_1)$ iff $\sigma(v) = \mu(t_1)$. A concrete example of this is $w := v + 1$ in Equation (1): Here, $(\mu, \sigma) \in \mathcal{P}^{\mathbb{Z}}(w := v + 1)$ iff $\sigma(w) = \mu(v + 1) = \mu(v) + 1$ (and $\mu \equiv_{\{w\}^{\complement}} \sigma$). The definition of $\mathbf{FV}_A^{\mathcal{R}}, \mathbf{FV}_P^{\mathcal{R}}$ can then be given via straightforward syntactical analysis of terms and formulas. As discussed below in *Lemma* 4.2, this yields an instantiation of a dynamic theory $\Delta^{\mathcal{R}}$ which satisfies all properties of Definition 4.6. It is then easy to see that Equation (1) is indeed valid in $\Delta^{\mathbb{Z}}$: For any state with $\mu(v) \geq 0$ and any state transition to σ such that $\sigma(w) = \mu(v) + 1$ it holds that $\sigma(w) \geq 1$ (as required by the atom on the right of Equation (1)).

Relation to classical First-Order Dynamic Logic. The syntax and semantics of classical first-order dynamic logic [28], are more restrictive in the supported structure of atoms and state spaces: State spaces are assumed to be simple valuations [28, p. 293] (and not more complex constructs admitting the evaluation of variables and atoms), and atoms are defined as predicates over terms [28, p. 103]. We demonstrate that axioms and proof rules derivable for classical, interpreted first-order dynamic logic are equally derivable using our smaller set of assumptions; Moreover, this relaxation, in turn, has two advantages: (1) The designer of a program logic, formalized as dynamic theory, can focus on the particularities of the program logic at hand and obtain extensions like regular programs or havoc operators for free (lifting, see Section 6), (2) Program logics formalized as dynamic theories can be combined arbitrarily in a manner that admits compositional reasoning (heterogeneous dynamic theories, see Section 7).

4.3 Static Semantics

For deriving proof rules, we need a precise notion of semantically free and bound variables, i.e. those that affect or are affected by a formula or program. Definition 4.6 already assumes over-approximations $\mathbf{FV}_A/\mathbf{FV}_P$ of these sets that are typically defined syntactically: In programs over $\mathbb{Z}$ (Example 4.2), variables in expressions are considered free, and those on the left of assignments are bound. Such approximations are not exact: For instance, $x := x + y - y$ changes no state and hence has syntactically, but not semantically, free or bound variables. Exact computation of these sets is undecidable [49]. As we have no knowledge on atom and program structure, syntactic approximations are unavailable and we instead adopt the semantic approach from Platzer [43]:

Definition 4.9 (Free and Bound Variables). *Let p be some program and F be some formula over a dynamic theory $\Delta = (\Lambda, \mathcal{D})$. Free variables* FV_Δ *and bound variables* BV_Δ *are defined as follows:*

$$\mathrm{FV}_\Delta(F) = \left\{ v \in \Lambda_V \;\middle|\; \text{exists } \mu, \tilde{\mu} \in \mathcal{S} \text{ s.t. } \mu \equiv_{\{v\}^\complement} \tilde{\mu} \text{ and } \mu \in \mathcal{D}[\![F]\!] \not\ni \tilde{\mu} \right\}$$

$$\mathrm{FV}_\Delta(p) = \left\{ v \in \Lambda_V \;\middle|\; \begin{array}{l} \text{exists } \mu, \tilde{\mu}, \sigma \in \mathcal{S} \text{ s.t. } \mu \equiv_{\{v\}^\complement} \tilde{\mu} \text{ and } (\mu, \sigma) \in \mathcal{P}(p) \\ \qquad \text{but there is no } \tilde{\sigma} \text{ s.t. } \sigma \equiv_{\{v\}^\complement} \tilde{\sigma} \text{ and } (\tilde{\mu}, \tilde{\sigma}) \in \mathcal{P}(p) \end{array} \right\}$$

$$\mathrm{BV}_\Delta(p) = \{ v \in \Lambda_V \mid \text{exists } \mu, \tilde{\mu} \in \mathcal{S} \text{ s.t. } (\mu, \tilde{\mu}) \in \mathcal{P}(p) \text{ and } \mathcal{V}(\mu, v) \neq \mathcal{V}(\tilde{\mu}, v) \}$$

We denote $\mathrm{Vars}_\Delta(p) = \mathrm{FV}_\Delta(p) \cup \mathrm{BV}_\Delta(p)$. Going forward we will only use $\mathrm{BV}_\Delta(p)$ and $\mathrm{Vars}_\Delta(p)$ which combines both free and bound variables. The definition of free variables breaks down for atoms depending on an infinite set of variables (e.g., consider some atom that is true iff an infinite number of variables have value 1). However, we avoid this issue by assuming that $\mathbf{FV}_A$ is finite for all atoms. Note that $\mathbf{FV}_A(a)$ and $\mathbf{FV}_P(p)$ resp. over-approximate $\mathrm{FV}_\Delta(a)$ and $\mathrm{FV}_\Delta(p)$ (see **??**).

4.4 Instantiations of Dynamic Theories

Section 4 introduced an elaborate definition of dynamic theories. While the proof-theoretic value of these definitions will become clear in the subsequent sections through the derivability of numerous proof rules from the proposed assumptions, we should first examine whether these definitions are sensible. To this end, we prove that three Dynamic Logics are instantiations of our definitions:

***Lemma* 4.1** (PDL as Dynamic Theory). *Propositional Dynamic Logic over arbitrary atomic programs and propositional atoms is an instantiation of a dynamic theory.*

***Lemma* 4.2** (Semi-Ring First-Order Dynamic Logic as Dynamic Theory). *Consider first-order atoms (equality and less than) over terms of an ordered semi-ring (e.g. natural numbers, see Examples 4.1 and 4.2). The dynamic logic with programs of the form $v := t$ where t is a term of the semi-ring is an instantiation of a dynamic theory.*

Proof Sketch. To prove that Semi-Ring First-Order Dynamic Logic is a dynamic theory, we prove that the state space, variables, atom and program evaluation, and their respective free variable definitions (see Examples 4.1 and 4.2) satisfy the six requirements from Definitions 4.3 to 4.5. As this logic's state space is the set of all (unique) variable to universe mappings, $\mathcal{S}$ Independence and Extensionality is trivially satisfied. $\mathbf{FV}_A$ Coverage and $\mathbf{FV}_P$ Coverage can be achieved by constructing $\mathbf{FV}_A$ and $\mathbf{FV}_P$ as sound syntactical analyses. $\mathbf{FV}_A$ Finite and $\mathbf{FV}_P$ Finite are achieved through the syntactic construction of atoms/programs which only touch a finite set of variables. □

***Lemma* 4.3** (Differential Dynamic Logic as Dynamic Theory). *The variables, atoms, and programs of Differential Dynamic Logic with semantics as formalized in the AFP [12] form a dynamic theory.*

Consequently, all proof rules and results described below hold for these three dynamic theories. While the logic from *Lemma* 4.2 might at first seem overly simplistic, we show in Section 6 that we can lift this theory to support a wider range of programs. As discussed in Section 1, this paper is not only about existing dynamic theories, but also about the ability to construct *further* dynamic theories which are, by design, interoperable with the collection of existing dynamic theories.

4.5 Proof Calculi

From a program verification perspective, the objective of Dynamic Logics, and dynamic theories, is to check that a program p satisfies properties. For example, we might want to verify that for all inputs satisfying F, the program p satisfies the postcondition G. From a proof-theoretic perspective this

$$\text{(G)}\ \frac{\vdash_\Delta \phi}{\vdash_\Delta [\alpha]\ \phi} \qquad \text{(E)}\ [\alpha]\ \phi \leftrightarrow [\beta]\ \psi \quad (\alpha \cong \beta)$$

$$\text{(V)}\ \phi \to [\alpha]\ \phi \quad (\mathrm{FV}_\Delta(\phi) \cap \mathrm{BV}_\Delta(\alpha) = \emptyset)$$
$$\text{(B)}\ (\forall v\ [\alpha]\ \phi) \leftrightarrow ([\alpha]\ \forall v\ \phi) \quad (v \notin \mathrm{Vars}_\Delta(\alpha))$$
$$\text{(K)}\ ([\alpha]\ (\phi \to \psi)) \to ([\alpha]\ \phi \to [\alpha]\ \psi)$$

Fig. 2. Elementary axioms and proof rules for dynamic theories

corresponds to proving that dynamic theory formula $F \to [p]\ G$ is Δ-valid, i.e. that $\models_\Delta F \to [p]\ G$. This is usually achieved by *proof calculi* that, from axioms and proof rules, derive that a certain formula is valid.

For the purpose of this work, we consider a calculus to be a relation $\vdash_\Delta \in 2^{\mathrm{Fml}_\Delta} \times \mathrm{Fml}_\Delta$ where $(\Gamma, F) \in \vdash_\Delta$ (denoted $\Gamma \vdash_\Delta F$ or $\vdash_\Delta F$ iff $\Gamma = \emptyset$) iff F can be derived from Γ using the proof rules and axioms of the calculus $\vdash_\Delta$. We are only interested in *sound* proof calculi:

Definition 4.10 (Sound Calculus)**.** *For a dynamic theory Δ a proof calculus $\vdash_\Delta$ is called* sound *iff for all formulas $F \in \mathrm{Fml}_\Delta$ it holds that $\vdash_\Delta F$ implies $\models_\Delta F$.*

In the present context, soundness is a compositional property: A proof calculus is sound if all proof rules are sound and all instantiations of axioms are valid. Soundness is thus an inductive consequence of the soundness of individual calculus steps, because the calculus iteratively generates (valid) axiom instantiations and applies (sound) proof rules. For the remainder of this work, we assume that all (elementary) dynamic theories Δ are equipped with a sound proof calculus $\vdash_\Delta$ that is complete w.r.t. propositional and uninterpreted first-order reasoning (e.g., by extending the Hilbert or Sequent calculus). Section 5 extends this calculus with additional proof rules that hold independently of a particular dynamic theory. Sections 6 and 7 extend and combine dynamic theories and provide sound calculi building on the given calculus for Δ. This approach guarantees compositionality: Although our theories become more expressive as we lift and merge them with the proposed constructs, we can nonetheless reuse the calculus $\vdash_\Delta$ of the original theory.

Equational Reasoning. In addition to proof rule based reasoning, our approach also supports reasoning via a refinement [38] and an equivalence relation between programs:

Definition 4.11 (Program Refinement and Equivalence)**.** *We say a program p* refines *q (denoted as $p \lesssim q$) iff for all $(\mu, \sigma) \in \mathcal{P}(p)$ it holds that $(\mu, \sigma) \in \mathcal{P}(q)$. We say two programs are equal (denoted $q \cong q$) iff $p \lesssim q$ and $q \lesssim p$.*

***Lemma* 4.4** (Equivalence Relation)**.** *The relation $\cdot \cong \cdot$ is an equivalence relation on programs.*

Throughout this work we will introduce known, proven identities $\cdot \cong \cdot$ which can then be used to justify the application of axiom (E) introduced in the next section. Additionally, we show that our regular program lifting (Section 6.2) forms a Kleene Algebra with Tests [36] (KAT) over $\cdot \lesssim \cdot, \cdot \cong \cdot$ — making it ammenable to rewriting and normalization based tooling for KAT.

5 Elementary Results and Proof Rules for Dynamic Theories

Concerning the semantic definitions of free and bound variables, we observe that the definitions from Definition 4.9 exactly characterize the variables determining the valuation of a formula as well as the footprint of a program. Importantly, all three definitions come with a minimality guarantee that ensures we are not overly conservative in our estimation of which variables impact valuations:

***Lemma* 5.1** (Coincidence Lemma for Formulas & Minimality)**.** *For any dynamic theory Δ a formula F has the same truth value in any two states that agree on $\mathrm{FV}_\Delta(F)$ (i.e., for $V = \mathrm{FV}_\Delta(F)$, if $\mathcal{S} \ni \mu \equiv_V \tilde{\mu} \in \mathcal{S}$ then $\mu \in \mathcal{D}[\![F]\!]$ iff $\tilde{\mu} \in \mathcal{D}[\![F]\!]$). The set of semantically free variables (FV_Δ) is the smallest set with this* coincidence property.

***Lemma* 5.2** (Coincidence Lemma for Programs & Minimality). *For any dynamic theory Δ, consider states $\mu, \sigma, \tilde{\mu} \in \mathcal{S}$ such that μ and σ only differ on a finite set (i.e., $\{v \in \Lambda_V \mid \mathcal{V}(\mu, v) \neq \mathcal{V}(\sigma, v)\}$ is finite). If $\mu \equiv_V \sigma$ for $V \supseteq \mathrm{FV}_\Delta(p)$ and $(\mu, \tilde{\mu}) \in \mathcal{P}(p)$, then there exists a $\tilde{\sigma} \in \mathcal{S}$ such that $(\sigma, \tilde{\sigma}) \in \mathcal{P}(p)$ and $\tilde{\mu} \equiv_V \tilde{\sigma}$. The semantically free variables of a program (FV_Δ) are the smallest set with this property.*

***Lemma* 5.3** (Bounded Effect & Minimality). *For any dynamic theory Δ, the set $\mathrm{BV}_\Delta(p)$ is the minimal set with the* bounded effect *property: For all $(\mu, \sigma) \in \mathcal{P}(p)$ it holds that $\mu \equiv_{(\mathrm{BV}_\Delta(p))^\complement} \sigma$.*

Based on these results, we can now begin the derivation of a proof calculus for the validity of Λ-formulas. To this end, we propose a Hilbert-style calculus with the usual first-order axioms. Since we do not know the inner workings of programs $p \in \Lambda_P$ it is the obligation of the designer of a dynamic theory to provide a suitable calculus to decompose and prove statements about such atomic programs. However, we can nonetheless provide calculus rules that hold independently of the concrete programs. A first set of these axioms and rules can be found in Figure 2. These rules will be extended in the subsequent section by calculus rules for reasoning about the closure of regular programs over the programs in Λ_P. By providing our calculus rules, we propose a *compositional* calculus: The theory designer can focus on proof procedures for statements about (from our perspective) atomic programs, while our rules provide the general, compositional principles that are independent of the particular programming language at hand. To this end, we formalize the soundness of the described axioms:

***Theorem* 5.1** (Soundness of Elementary Proof Rules). *The proof rules and axioms in Figure 2 are sound w.r.t. to any dynamic theory Δ.*

The proof rule (G) and the axiom (V) are akin to first-order Hilbert-calculus generalization rules by allowing us to wrap formulas into modalities. Barcan's axiom (B) encodes a constant domain property for the dynamic theory's state space that emerges from the state space properties outlined in Section 4. (V) allows us to eliminate programs that do not affect the postcondition while axiom (K) allows distributing modalities across implications. Finally, (E) allows us to swap programs in modalities for equivalent programs. As described in *Theorem* 5.1, we can prove the soundness of these axioms independently of the considered state space, atomic formulas, and programs.

6 Lifting Dynamic Theories

In this section, we demonstrate, for the first time, the power of the concise formalization of necessary properties for the construction of a dynamic theory: Given a dynamic theory Δ, we can *extend* its functionality, e.g., by extending the structure of state spaces, atoms, or programs, with *common* constructs. By proving that this extension *again* satisfies the properties outlined in Section 4, all properties, proof rules, and axioms are lifted to the extended, more expressive, dynamic theory. Similarly to conservative extensions in first-order theory, our lifting ensures that previously valid axioms and formulas remain sound. Unlike conservative extensions, some liftings of dynamic theories meaningfully extend the expressiveness of the logic's programming language beyond the original theory's expressiveness by allowing new state transitions that were previously impossible. The most prominent example of this is the introduction of loops in Section 6.2. In Section 6.1, we begin by illustrating this principle for the havoc operator $v := *$ which assigns the variable v to an arbitrary value. Subsequently, in Section 6.2, we apply this approach to a more ambitious extension: The *closure* of regular programs over Λ_P: In this instance, we provide axioms for the decomposition of all regular program operators, including loop invariants and variants (the latter under minimal assumptions about the availability of natural number constraints).

6.1 Havoc Operator Lifting

On a syntactical level, lifting a dynamic theory Δ to a new dynamic theory $\Delta^{:=*}$ amounts to enhancing the original set of programs Λ_P with the havoc operator on all variables in Λ_V, i.e. $\Lambda_P^{:=*} = \Lambda_P \dot\cup \{v := * \mid v \in \Lambda_V\}$. We can similarly extend Δ's domain of computation $\mathcal{D}$. Overall, this amounts to the following lifting operation $(\cdot)^{:=*}$:

Definition 6.1 (Havoc Lift)**.** *Given a dynamic theory Δ we define $\Delta^{:=*} = (\Lambda^{:=*}, \mathcal{D}^{:=*})$ such that all components of Δ and $\Delta^{:=*}$ are equal except for:*

- $\Lambda_P^{:=*} = \Lambda_P \dot\cup \{v := * \mid v \in \Lambda_V\}$
- $\mathcal{P}^{:=*}(p) = \mathcal{P}(p)$ *(for $p \in \Lambda_P$)*
- $\mathcal{P}^{:=*}(v := *) = \left\{(\mu, \sigma) \in \mathcal{S}^2 \;\middle|\; \mu \equiv_{\{v\}^\complement} \sigma\right\}$ $\quad (v \in \Lambda_V)$
- $\mathbf{FV}_P^{:=*}(p) = \mathbf{FV}_P(p)$ *(for $p \in \Lambda_P$)*
- $\mathbf{FV}_P^{:=*}(v := *) = \emptyset$ *(for $v \in \Lambda_V$)*

For any dynamic theory Δ we can prove that its havoc lifting $\Delta^{:=*}$ again satisfies all properties of a dynamic theory and consequently all properties, axioms and proof rules carry over:

***Theorem* 6.1** (Havoc Lifted Dynamic Theories)**.** *The havoc lift $\Delta^{:=*}$ of any dynamic theory Δ is a dynamic theory.*

For first-order atoms (Λ_A) and the state space ($\mathcal{S}$), the required properties for a dynamic theory are still satisfied, because we did not modify these components. For programs, we ensured that our semantic definition of havoc satisfies the properties asserted in Definition 4.5. While it is advantageous that the properties proven for arbitrary dynamic theories carry over from Δ to $\Delta^{:=*}$, the havoc lifting would be useless if properties of the *specific* theory Δ under consideration would not carry over as well. If this were the case, all properties proven for a Δ under consideration would have to be reproven for $\Delta^{:=*}$ and the lifting operation would lose its utility. Fortunately, it turns out that, syntactically, our new set of formulas is a superset of the original formulas, i.e. $\mathrm{Fml}_\Lambda \subseteq \mathrm{Fml}_{\Lambda^{:=*}}$. This raises the question of whether their semantics also carry over. Indeed, we can prove a calculus rule which translates proofs about Δ into proofs about $\Delta^{:=*}$:

***Lemma* 6.1** (Havoc Reduction)**.** *For any dynamic theory Δ, the following proof rule is sound:*

$$\text{(HR)}\ \frac{\vdash_\Delta \phi}{\vdash_{\Delta^{:=*}} \phi}\ \textit{(assuming } \phi \in \mathrm{Fml}_\Lambda\textit{)}$$

Hence, we can reuse *any* decision procedure developed for determining the validity of formulas in Fml_Λ to determine the validity of formulas in the havoc-free fragment of $\mathrm{Fml}_{\Lambda^{:=*}}$ and carry these results over using (HR). Additionally, we prove the soundness of the well-known ($:=*$) axiom for the decomposition of the havoc operator:

***Lemma* 6.2** (Havoc Axiom)**.** *For any dynamic theory Δ, the following axiom is sound in $\Delta^{:=*}$:*

($:=*$) $([v := *]\ \phi) \leftrightarrow (\forall v\ \phi)$

Before we move to regular programs, let us recap what this section demonstrated: Given any dynamic theory Δ, we can create a new dynamic theory $\Delta^{:=*}$ which: (1) Has the additional program primitive $v := *$; (2) Inherits the dynamic theory axioms (G),(V),(B) and (K); (3) Allows reuse of existing proof infrastructure for Δ through (HR) (as a valid formula of Δ is also valid in $\Delta^{:=*}$). The remainder of this section, as well as Section 7, will demonstrate significantly more expressive lifting procedures with the same properties: Transfer of proof rules and validity preservation.

Example 6.1 (Havoc Lifting for Semiring)**.** Continuing the example of a dynamic theory over a semi-ring $\Delta^{\mathcal{R}}$ defined in Examples 4.1 and 4.2: Using the lifting procedure above, we can lift this dynamic theory to support the havoc operator resulting in the theory $\left(\Delta^{\mathcal{R}}\right)^{:=*}$. In addition to the proof principles from $\Delta^{\mathcal{R}}$, this theory also supports the proof rule (HR) and the axiom ($:=*$).

6.2 Regular Program Lifting

Given a set of (atomic) programs, it is often useful to consider their *regular programs closure*, i.e., any construction of programs obtained by sequential composition, nondeterministic choice, (first-order logic) checks, and nondeterministic loops. This is an interesting set of programs, because it allows constructing well-known imperative constructs such as `if` or `while`, from minimal set of operations. It is also isomorphic to the program constructs considered in Kleene Algebra with Tests [36]. We now show that any dynamic theory Δ can be lifted to this more expressive set of programs while retaining the properties established above. To this end, we begin by defining the regular closure of programs along with their semantics:

Definition 6.2 (Regular Closure)**.** *Given a dynamic theory Δ over programs Λ_P with program evaluation function $\mathcal{P}$ we define the regular closure of programs, denoted Λ_P^{reg} via the following grammar (where $r \in \Lambda_P$ and $F_{FOL} \in \text{FOL}_\Lambda$): $p, q ::= r \mid p; q \mid p \cup q \mid ?\,(F_{FOL}) \mid (p)^*$.*
We then lift the program evaluation function to $\mathcal{P}^{\text{reg}}$ as follows:

- $\mathcal{P}^{\text{reg}}(r) = \mathcal{P}(r)$ *(for $r \in \Lambda_P$)*
- $\mathcal{P}^{\text{reg}}(p; q) = \left\{(\mu, \omega) \in \mathcal{S}^2 \;\middle|\; \text{there exists } \sigma \in \mathcal{S} \text{ s.t. } (\mu, \sigma) \in \mathcal{P}^{\text{reg}}(p) \text{ and } (\sigma, \omega) \in \mathcal{P}^{\text{reg}}(q)\right\}$
- $\mathcal{P}^{\text{reg}}(p \cup q) = \mathcal{P}^{\text{reg}}(p) \cup \mathcal{P}^{\text{reg}}(q)$
- $\mathcal{P}^{\text{reg}}(?\,(F_{FOL})) = \left\{(\mu, \sigma) \in \mathcal{S} \;\middle|\; \mu \in \mathcal{D}[\![F_{FOL}]\!] \text{ and } \mu \equiv_{\Lambda_V} \sigma\right\}$
- $\mathcal{P}^{\text{reg}}\left((p)^*\right) = \bigcup_{n \in \mathbb{N}} \mathcal{P}^{\text{reg}}(p^n)$ *where* $p^0 \equiv ?\,(\top)$ *and* $p^{n+1} \equiv (p; p^n)$

To create a new dynamic theory, it remains to define the set of free variables $\mathbf{FV}_P^{\text{reg}}$. To this end, the free variables for $r \in \Lambda_P$ are defined as $\mathbf{FV}_P(r)$, and the free variables under sequential composition and nondeterministic choice are defined as the union of their components. The free variables of a looped program $(p)^*$ are the free variables of p. For the check $?\,(F_{\text{FOL}})$ we leverage our knowledge about the free variables of atoms (encoded in $\mathbf{FV}_A$) to construct the free variables of a first-order formula (see **??**). Note that popular definitions of $?\,(F_{\text{FOL}})$ often enforce semantics where it must be the case that $\mu = \nu$ while we allow transitions to equivalent states as the usual definition breaks our extensionality assumption in Definition 4.5 (since we allow distinct states $\mu, \sigma \in \mathcal{S}$ such that $\mu \equiv_{\Lambda_V} \sigma$). This particularity disappears if we assume $\mathcal{S}$ to be a "classical" mapping from variables to values. We can then formally define the regular closure lift and prove that it constructs once again a dynamic theory:

Definition 6.3 (Regular Closure Lift)**.** *For a dynamic theory Δ and its regular closure over programs $\Lambda_P^{\text{reg}}, \mathcal{P}^{\text{reg}}$ (see Definition 6.2) and $\mathbf{FV}_P^{\text{reg}}$ as defined in* **??***, the* regular closure lift *is defined as:*

$$\Delta^{\text{reg}} = \left(\left(\Lambda_V, \Lambda_A, \Lambda_P^{\text{reg}}\right), \left(\mathcal{U}, \mathcal{S}, \mathcal{V}, \mathcal{A}, \mathbf{FV}_A, \mathcal{P}^{\text{reg}}, \mathbf{FV}_P^{\text{reg}}\right)\right)$$

***Theorem* 6.2** (Regular Closure over Dynamic Theory)**.** *The regular closure lift Δ^{reg} of any dynamic theory Δ is a dynamic theory.*

Just as before, we can define a reduction rule which allows us to reuse proof results in Δ^{reg} that have already been shown for Δ. Additionally, our lifting comes with numerous axioms on the decomposition of regular programs which we summarize in Figure 3 and are provably sound:

***Lemma* 6.3** (Regular Reduction)**.** *For any dynamic theory Δ, the following proof rule is sound:*

$$\text{(RR)}\ \frac{\vdash_\Delta \phi}{\vdash_{\Delta^{\text{reg}}} \phi}\ \textit{(assuming } \phi \in \text{Fml}_\Lambda\textit{)}$$

***Theorem* 6.3** (Soundness of Proof Rules over Regular Closure)**.** *For any dynamic theory Δ the axioms in Figure 3 are sound in Δ^{reg}.*

(?)	$[?\,(\phi)]\ \psi \leftrightarrow (\phi \rightarrow \psi)$	(∪)	$[\alpha \cup \beta]\ \phi \leftrightarrow [\alpha]\ \phi \wedge [\beta]\ \phi$
(;)	$[\alpha;\beta]\ \phi \leftrightarrow [\alpha]\ [\beta]\ \phi$	(I)	$[\alpha^*]\ (\phi \rightarrow [\alpha]\ \phi) \rightarrow (\phi \rightarrow [\alpha^*]\ \phi)$
(*)	$[\alpha^*]\ \phi \leftrightarrow \phi \wedge [\alpha]\ [\alpha^*]\ \phi$		

Fig. 3. Axioms for Regular Programs

The proof rules provided in Figure 3 allow us to decompose checks, sequential compositions and nondeterminstic choice programs (see axioms (?),(;), (∪)). Moreover, they allow the iterative decomposition of loops as well as inductive invariant reasoning about loops (see axioms (*), (I)). However, the axioms are, yet, insufficient for loop termination.

Loop Convergence. A significant strength of dynamic logics, and hence dynamic theories, is their ability to reason about partial *and* total correctness via the dual diamond and box modalities. To this end, it is sometimes necessary to not only reason about loop invariants (via (I)), but also about loop *variants* to prove progress. To this end, consider a formula $\langle p^* \rangle\ F$: Here, we must prove that running program p in a loop *eventually* leads to a state satisfying F. This is usually achieved via well-founded induction proofs: Given a formula $\phi(N)$ that is satisfied for a sufficiently large $N \in \mathbb{N}$, one iteration of p ensures that $\phi(N-1)$ is satisfied in the post state. If $\phi(0)$ implies F, this proves that F becomes satisfied after iteratively running p for a sufficiently large number of steps. This requires *counting*. However, due to the generality of our assumed state space, we currently have no way of counting or performing well-founded induction. Hence, we now propose a set of additional assumptions that allow us to introduce a loop termination rule:

Definition 6.4 (Inductive Expressivity)**.** *Given some dynamic theory Δ, assume some function $\mathcal{N} : \mathcal{U} \rightarrow \mathbb{N}$ mapping elements from the universe to natural numbers. We say a variable $v \in \Lambda_V$ is* integer expressive *w.r.t. $\mathcal{N}$ iff for all $n \in \mathbb{N}$ there exists a state $\mu \in \mathcal{S}$ such that $\mathcal{N}(\mathcal{V}(\mu, v)) = n$. Let $\Lambda_{V\mathbb{N}}$ be a set of such variables. We say Δ has* inductive expressivity *iff for some given $\mathcal{N}$ there exist two functions $\mathrm{nat}_{+1}, \mathrm{nat}_{=} : \Lambda^2_{V\mathbb{N}} \rightarrow \mathrm{FOL}_\Lambda$ and one function $\mathrm{nat}_{>0} : \Lambda_{V\mathbb{N}} \rightarrow \mathrm{FOL}_\Lambda$ such that:*

(Positive Sound) *For any variable $v \in \Lambda_{V\mathbb{N}}$ and any state $\mu \in \mathcal{S}$:*
$\mu \in \mathcal{D}[\![\mathrm{nat}_{>0}(v)]\!]$ *iff* $\mathcal{N}(\mathcal{V}(\mu, v)) > 0$

(Equal Sound) *For variables $v, w \in \Lambda_{V\mathbb{N}}$ with $v \neq w$ and any state $\mu \in \mathcal{S}$:*
$\mu \in \mathcal{D}[\![\mathrm{nat}_{=}(v, w)]\!]$ *iff* $\mathcal{N}(\mathcal{V}(\mu, v)) = \mathcal{N}(\mathcal{V}(\mu, w))$

(Plus One Sound) *For variables $v, w \in \Lambda_{V\mathbb{N}}$ with $v \neq w$ and any state $\mu \in \mathcal{S}$:*
$\mu \in \mathcal{D}[\![\mathrm{nat}_{+1}(v, w)]\!]$ *iff* $(\mathcal{N}(\mathcal{V}(\mu, v)) + 1) = \mathcal{N}(\mathcal{V}(\mu, w))$

This inductive expressivity yields all that is required to prove loop convergence, which enables us to prove loop termination and reachability of certain states via loop iteration. To this end, dynamic logics usually are equipped with an axiom that looks something like this:

$$[p^*]\ (\forall v\ \ ((v > 0 \wedge \phi(v)) \rightarrow \langle p \rangle\ \phi(v-1))) \rightarrow \forall v\ \ (\phi(v) \rightarrow \langle p^* \rangle\ \phi(0)) \qquad (v \notin \mathbf{Vars}(p))$$

This axioms formalizes the well-founded induction argument outlined above: If $\phi(v)$ is satisfied and one iteration of p allows us to reach a state satisfying $\phi(v-1)$, then we can reach a state with $\phi(0)$ by iterating the execution of p (assuming v is a natural number). To prove loop reachability properties, we also want an axiom like this one in the regular closure over our dynamic theory; however, we lack any means to parameterize formulas with specific variables. After all, our theory so far makes no assumptions about how atomic formulas in Λ_A evaluate w.r.t. variables beyond the definition of $\mathbf{FV}_A$. Hence, we derive the following axiom which can be proven independently of the dynamic theory at hand so long as the properties from Definition 6.4 are given:

***Lemma* 6.4** (Loop Convergence Rule). *For any inductively expressive dynamic theory Δ, the following axiom is sound for its regular closure Δ^{reg} for any integer expressive distinct variables $v, w \in \Lambda_{V\mathbb{N}}$ with $w \notin \text{FV}_\Delta(\phi)$ and $v, w \notin \textbf{Vars}(p)$:*

$$\text{(C)} \quad \begin{array}{l}([p^*]\ (\forall v\ \ (\text{nat}_{>0}\,(v) \to \langle p\rangle\ (\forall w\ \ (\text{nat}_{+1}\,(w,v) \to \forall v\ \ (\text{nat}_{=}\,(v,w) \to \phi))))))\\ \to (\forall v\ \ (\phi \to \langle p^*\rangle\ (\exists v\ \ (\neg\text{nat}_{>0}\,(v) \wedge \phi))))\end{array}$$

Note that once we apply $\text{nat}_{+1}, \text{nat}_{=}$ or $\text{nat}_{>0}$ to concrete variables, they reduce to a concrete Δ-formula yielding a concrete instantiation of axiom (C). Importantly, we cannot subtract from v within a single predicate since any predicate (including nat_{+1}) requires a reference point from which to subtract 1. We mitigate this by employing the helper variable w. To demonstrate the utility of the additional assumptions, we have also proven that a concrete Dynamic Logic instantiates the properties defined in Definition 6.4:

***Lemma* 6.5** (Semi-Ring First-Order Dynamic Logic over Natural Numbers). *The semi-ring dynamic theory $\Delta^{\mathcal{R}}$ (from Lemma 4.2), instantiated for natural numbers or integers (resp. denoted $\Delta^{\mathbb{N}}$ or $\Delta^{\mathbb{Z}}$), is inductively expressive. Consequently, axiom (C) is a sound axiom for its havoc lifted, regular closure $\left(\left(\Delta^{\mathbb{N}}\right)^{:=*}\right)^{\text{reg}}$ (resp. $\left(\left(\Delta^{\mathbb{Z}}\right)^{:=*}\right)^{\text{reg}}$).*

Example 6.2 (Regular Program Lifting for Ordered Semiring)**.** We reconsider the example of a dynamic logic over an ordered semiring $(\mathcal{R}, +, \cdot, \leq)$ for $\mathcal{R} = \mathbb{Z}$: As seen in *Lemma* 6.5, this dynamic theory is inductively expressive. Hence, all axioms derived up to this point(Figures 2 and 3 as well as (HR),(RR),(:=*),(C)) apply to its havoc lifted, regular closure $\left(\left(\Delta^{\mathbb{Z}}\right)^{:=*}\right)^{\text{reg}}$. From a very simple definition of assignment, we have thus derived a dynamic theory with full support for regular programs and nondeterministic assignment. For example, the following program is part of this dynamic theory:

$$1 \leq n \to \left[x := 0; i := 0; \left(?\,(i \leq n)\,; x := x + i; i := i + 1\right)^*; ?\,(\neg\,(i \leq n))\right]\ 2 * x \leq n * (n+1) \qquad (2)$$

This formula asserts that x is no larger than the Gauss formula predicts after summing up the first n integers. Based on our simple definitions in Examples 4.1 and 4.2 we can now perform reasoning about programs like the one in Equation (2). In fact, $\left(\left(\Delta^{\mathbb{Z}}\right)^{:=*}\right)^{\text{reg}}$ provides the equivalent of a guarded command language over integers and, as will be shown in Section 8, a relatively complete proof calculus for it. In fact, our initial definition of $\Delta^{\mathbb{Z}}$ even could have omitted the assignment programs (by initializing $\Lambda_P^{\mathbb{Z}} = \emptyset$) by defining assignment as syntactic sugar for havoc and check [1]:

$$v := t \equiv (w := *; ?\,(t \leq w \wedge w \leq t)\,; v := *; ?\,(v \leq w \wedge w \leq v)) \qquad \text{for } w \in \Lambda_V^{\mathbb{Z}} \text{ fresh}$$

Kleene Algebra with Tests. In contrast to the formula-based approach presented in this section so far, Kleene Algebra with Tests, as an orthogonal direction of research, proposes to prove properties over regular programs of the kind discussed in this section using equational rewriting and normalization. Indeed, our definition of regular programs cannot only be used to derive standard axioms of dynamic logic, but it can equally be used to prove the axioms of a Kleene Algebra with Tests w.r.t. our refinement and equality relations $\cdot \lesssim \cdot$ and $\cdot \cong \cdot$:

***Theorem* 6.4** (Regular Closure forms KAT). *The program constructs of the regular closure lifting satisfy all axioms of a Kleene Algebra with Tests (see ??) where + is $\cup$, $\cdot$ is ;, 1 is $?\,(\top)$, $(\cdot)^*$ is the nondeterministic loop and the boolean algebra is constructed over checks $?\,(\cdot)$[1].*

[1]For negation of check constructs we introduce an additional operator which aggregates and negates a given program consisting purely of checks (see ??).

In combination with axiom (E), this insight allows us to mix-and-match proof tactics from Kleene Algebra with Tests with proof tactics from dynamic logic.

7 Heterogeneous Dynamic Theories

We have shown how any dynamic theory Δ can be lifted for more complex behavior—either via havoc ($\Delta^{:=*}$) or via the regular closure (Δ^{reg}). In both cases, the developer of a logic can focus on axiomatizing their concrete program constructs while reusing existing components to increase expressiveness.

We now enhance dynamic theories further by *combining* multiple theories to reason about *heterogeneous systems*, whose components operate on different domains of computation, but may interact. Heterogeneous theories serve two purposes: (1) they provide a unified framework that explicitly models both systems and their interactions, and (2) they enable rigorous reasoning about the overall system by *reusing* results and proof calculi from the *homogeneous* component theories. The heterogeneous proof calculus decomposes obligations to the individual theories, analogous to how reasoning over regular programs reduces to reasoning over atomic programs.

Given two dynamic theories $\Delta^{(0)}, \Delta^{(1)}$, we first construct a *simple heterogeneous theory* $\Delta^{(01)}$ over $\Lambda_P^{(0)} \cup \Lambda_P^{(1)}$, then lift it with havoc and regular closure. The resulting *fully heterogeneous dynamic theory* $\hat{\Delta}$ provides an expressive logic combining $\Delta^{(0)}$ and $\Delta^{(1)}$ with all axioms from Sections 5 and 6 included.

7.1 Simple Heterogeneous Dynamic Theories

We assume we are given two dynamic theories $\Delta^{(0)}, \Delta^{(1)}$ and begin by defining the state space of our new dynamic theory:

Definition 7.1 (Heterogeneous State Space)**.** *Given two dynamic theories $\Delta^{(0)}, \Delta^{(1)}$ and their resp. state spaces $\mathcal{S}^{(0)}, \mathcal{S}^{(1)}$ we define the* heterogeneous state space *as $\hat{\mathcal{S}} = \mathcal{S}^{(0)} \times \mathcal{S}^{(1)}$. For a given state $\left(\mu^{(0)}, \mu^{(1)}\right) \in \hat{\mathcal{S}}$ variable evaluation is defined as follows:*

$$\hat{\mathcal{V}}\left(\left(\mu^{(0)}, \mu^{(1)}\right), v\right) = \begin{cases} \mathcal{V}^{(0)}\left(\mu^{(0)}, v\right) & v \in \Lambda_V^{(0)} \\ \mathcal{V}^{(1)}\left(\mu^{(1)}, v\right) & v \in \Lambda_V^{(1)} \end{cases}$$

A priori, the behavior modeled in $\Delta^{(0)}$ and $\Delta^{(1)}$ hence happens in isolation. However, similarly to reasoning about concurrent systems, the appeal of reasoning about heterogeneous systems lies precisely in the *interaction* of these a priori independent behaviors. Hence, to relate the behavior of these two independent state space components, we introduce a new set of atoms (denoted $\Lambda_A^{\cap}$) to reason about and relate $\mu^{(0)}$ and $\mu^{(1)}$:

Definition 7.2 (Heterogeneous Atoms)**.** *Given two dynamic theories $\Delta^{(0)}, \Delta^{(1)}$ and their heterogeneous state space $\hat{\mathcal{S}}$ we define a new set of* heterogeneous atoms *$\Lambda_A^{\cap}$ over variables $\Lambda_V^{\cap} = \Lambda_V^{(0)} \cup \Lambda_V^{(1)}$ with an evaluation function $\mathcal{A}^{\cap} : \Lambda_A^{\cap} \to 2^{\hat{\mathcal{S}}}$ and a free variable function $\mathbf{FV}_A^{\cap} : \Lambda_A^{\cap} \to 2^{\Lambda_V^{\cap}}$ such that the constructs satisfy the atom evaluation requirements from Definition 4.4.*

Heterogeneous Atoms can be considered the dynamic theory counterpart to mixed terms in the first-order theory combination. Just like in the case of theory combination, these terms allow us to constrain the relation between values in both worlds. Based on these definitions, we can now proceed to define the syntactic materials of our simple heterogeneous dynamic theory:

Definition 7.3 (Simple Heterogeneous Dynamic Signature)**.** *Given two dynamic theories* $\Delta^{(0)}, \Delta^{(1)}$ *with resp. signatures* $\Lambda^{(0)}, \Lambda^{(1)}$ *and a set of heterogeneous atoms* $\Lambda_A^{\cap}$ *over* $\Delta^{(0)}, \Delta^{(1)}$ *(see Definition 7.2) we define the* simple heterogeneous dynamic signature *as* $\Lambda^{(01)} = \left(\hat{\Lambda_V}, \hat{\Lambda_A}, \Lambda_P^{(01)}\right)$ *with:*

$$\hat{\Lambda_V} = \Lambda_V^{(0)} \dot{\cup} \Lambda_V^{(1)} \qquad \hat{\Lambda_A} = \Lambda_A^{(0)} \dot{\cup} \Lambda_A^{(1)} \dot{\cup} \Lambda_A^{\cap} \qquad \Lambda_P^{(01)} = \Lambda_P^{(0)} \dot{\cup} \Lambda_P^{(1)}$$

The domain of computation, i.e. the semantical definition of our new dynamic theory then naturally follows:

Definition 7.4 (Simple Heterogeneous Domain of Computation)**.** *Given two dynamic theories* $\Delta^{(0)}, \Delta^{(1)}$ *and notation as in Definition 7.3 we define the simple heterogeneous domain of computation* $\mathcal{D}^{(01)}$ *as follows:*

- *The universe is the combination of individual universes:* $\hat{\mathcal{U}} = \mathcal{U}^{(0)} \cup \mathcal{U}^{(1)}$
- *The state space is the heterogeneous state space* $\hat{\mathcal{S}}$ *over* $\Delta^{(0)}, \Delta^{(1)}$ *with the corresponding variable evaluation function* $\hat{\mathcal{V}}$ *(see Definition 7.1)*
- *The atom evaluation function* $\hat{\mathcal{A}}$ *and program evaluation function* $\mathcal{P}^{(01)}$ *is given as below (with corresponding free variable functions* $\hat{\mathbf{FV}}_A, \mathbf{FV}_P^{(01)}$ *as defined in* **??***).*

$$\hat{\mathcal{A}}(a) = \begin{cases} \mathcal{A}^{(0)}(a) \times \mathcal{S}^{(1)} & a \in \Lambda_A^{(0)} \\ \mathcal{S}^{(0)} \times \mathcal{A}^{(1)}(a) & a \in \Lambda_A^{(1)} \\ \mathcal{A}^{\cap}(a) & a \in \Lambda_A^{\cap} \end{cases} \qquad \mathcal{P}^{(01)}(p) = \begin{cases} \mathcal{P}^{(0)}(p) & p \in \Lambda_P^{(0)} \\ \mathcal{P}^{(1)}(p) & p \in \Lambda_P^{(1)} \end{cases}$$

The notation of our simple heterogeneous signature $\Lambda^{(01)}$ and the simple heterogeneous dynamic theory $\mathcal{D}^{(01)}$ is already giving a hint at the components of the dynamic theory which are here to stay for the fully heterogeneous version (denoted with $\hat{(\cdot)}$) vs. components of the simple heterogeneous theory that still require further refinement (denoted with $(\cdot)^{(01)}$): While the universe, state space, variables and atoms remain as-is, the range of admissible programs will still be extended. Before we lift $\Delta^{(01)}$ to a more expressive set of programs, we begin by showing that our simple heterogeneous dynamic signature and domain of computation indeed form a dynamic theory:

***Theorem* 7.1** (Simple Heterogeneous Dynamic Theories)**.** *Assume two dynamic theories* $\Delta^{(0)}, \Delta^{(1)}$ *and a corresponding set of heterogeneous atoms* $\Lambda_A^{\cap}$*, then* $\Lambda^{(01)}$ *and* $\mathcal{D}^{(01)}$ *as defined in Definitions 7.3 and 7.4 form a dynamic theory* $\Delta^{(01)}$ *as defined in Definition 4.7. All axioms and proof rules from Figure 2 carry over.*

This result unlocks the first four proof rules for $\Delta^{(01)}$. However, the range of properties we will be able to formalize and prove with $\Delta^{(01)}$ are still underwhelming: Each modality in $\Delta^{(01)}$ can only contain a program from $\Delta^{(0)}$ *or* a program from $\Delta^{(1)}$, but *not both*. Before we *lift* this limitation in the next section, we first consider how this logic relates to another technique for combining logics, namely fibring.

Relation to Fibring. By explicitly labeling the modalities w.r.t. the contained program (i.e. $[p]\ F$ becomes $[p]^{(0)}\ F$ for $p \in \Lambda_P^{(0)}$ and $[p]^{(1)}\ F$ otherwise), we can interpret this logic as a "multi-dynamic logic" with two distinct modal operators. This approximately corresponds to what could be achived via fibring [24] of dynamic logics where we could combine the truth bearing entities of $\Delta^{(0)}$ and $\Delta^{(1)}$. While fibring allows for the combination of truth-bearing entities from $\Delta^{(0)}$ and $\Delta^{(1)}$, it does not recognize programs as first-class citizes. Hence, it cannot integrate their respective program constructs which restricts the expressive power of analyzable systems. In particular, it cannot capture systems involving, e.g., intricate control structures such as loops. Instead, fibring is

confined to modeling systems that can be decomposed into a (nondeterministic) finite, sequential compositions of homogeneous subsystems. Importantly, our dynamic theory does *not* have two distinct modalities. This key difference allows us to apply the lifting procedures devised in Section 6 which naturally yields a significantly more expressive program logic.

7.2 Fully Heterogeneous Dynamic Theories

Using the lifting procedures from Section 6 we can define the fully heterogeneous dynamic theory, supporting the composition of homogeneous programs using all regular program constructs, in a straightforward manner:

Definition 7.5 (Fully Heterogeneous Dynamic Theories)**.** *Given two dynamic theories $\Delta^{(0)}, \Delta^{(1)}$ and a corresponding set of heterogeneous atoms $\Lambda_A^{\cap}$ forming the simple heterogeneous dynamic theory $\Delta^{(01)}$, we define the* fully heterogeneous dynamic theory *over $\Delta^{(0)}, \Delta^{(1)}$ as $\hat{\Delta} = \left(\left(\Delta^{(01)}\right)^{:=*}\right)^{\text{reg}}$ with $(\cdot)^{\text{reg}}$ and $(\cdot)^{:=*}$ as given in Theorem 6.1 and Theorem 6.2.*

As planned, we can recover all axioms established above (with the exception of (C) for which the requirements will be discussed below):

***Theorem* 7.2** (Fully Heterogeneous Dynamic Theories)**.** *Given two dynamic theories $\Delta^{(0)}, \Delta^{(1)}$ and a corresponding set of heterogeneous atoms $\Lambda_A^{\cap}$ the fully heterogeneous dynamic theory $\hat{\Delta}$ is a dynamic theory. Proof rules and axioms from Figure 3 as well as axioms (HR),(:=*), and (RR) carry over to $\hat{\Delta}$.*

Just as for the regular closure over programs of *one* dynamic theory, the construction of a heterogeneous dynamic theory over *two* theories would be entirely devoid of purpose if validity wouldn't carry over from the individual logics to the combined one. Fortunately, we can show that for any *pure* $\Lambda^{(0)}/\Lambda^{(1)}$-formula the validity w.r.t. $\Delta^{(0)}/\Delta^{(1)}$ entails the validity w.r.t. $\hat{\Delta}$:

***Lemma* 7.1** (Heterogeneous Reduction)**.** *For any fully heterogeneous dynamic theory $\hat{\Delta}$ over arbitrary $\Delta^{(0)}, \Delta^{(1)}$ the following proof rules are sound:*

$$\text{(HR0)}\ \frac{\vdash_{\Delta^{(0)}} \phi}{\vdash_{\hat{\Delta}} \phi}\ (\text{assuming } \phi \in \text{Fml}_{\Lambda^{(0)}}) \qquad \text{(HR1)}\ \frac{\vdash_{\Delta^{(1)}} \phi}{\vdash_{\hat{\Delta}} \phi}\ (\text{assuming } \phi \in \text{Fml}_{\Lambda^{(1)}})$$

Loop Convergence. In order for the loop convergence rule to carry over as well, we require that our heterogeneous dynamic theory $\hat{\Delta}$ is inductively expressive. To this end, we note that whenever one of the homogeneous dynamic theories $\Delta^{(0)}, \Delta^{(1)}$ is inductive, then the heterogeneous dynamic theory is inductive as well:

***Lemma* 7.2** (Inductive Expressivity of $\hat{\Delta}$)**.** *Let $\Delta^{(0)}$ be inductively expressive, then there exists a (constructive) adjusted function $\hat{\mathcal{N}}$ such that $\hat{\Delta}$ is inductively expressive w.r.t. the variables $\Lambda^{(0)}_{V\mathbb{N}}$. The same holds in case $\Delta^{(1)}$ is inductively expressive.*

Depending on whether we use a loop convergence rule inherited from $\Delta^{(0)}$ or from $\Delta^{(1)}$, we denote it as $(\text{C}^{(0)})$ or $(\text{C}^{(1)})$, respectively.

Program Rewriting. While not strictly necessary for relative completeness, it turns out that axiom (E) is of particular appeal for the verification of heterogeneous systems in cases where our heterogeneous dynamic theory contains (at least) one homogeneous dynamic theory which is lifted with regular programs. To demonstrate the utility of equational program rewriting, consider the program $\alpha_{\text{mixed}} \equiv \left(p_1; p_2; (q_1 \cup q_2)^*\right)^*$.where we have marked *homogeneous program constructs* in blue and orange while *fully heterogeneous program constructs* are marked in magenta. In this instance, we would now have to take account of program compositions in three different proof

calculi: The proof calculus of homogeneous dynamic theory 1 for the sequential composition ; and the proof calculus of homogeneous dynamic theory 2 for $\cup$ and $(\cdot)^*$. Additionally. we need the proof calculus of the heterogeneous dynamic theory for ; and the outer loop $(\cdot)^*$. Likely, there are cases where a more elegant proof could be achieved by reasoning about all regular programs at the *heterogeneous* level. To this end, we prove an additional set of identities:

***Theorem* 7.3** (Heterogeneous Rewriting). *Let $\Delta^{(0)}, \Delta^{(1)}$ be two dynamic theories and let $\hat{\Delta}$ be the fully heterogeneous dynamic theory over $\left(\Delta^{(0)}\right)^{\text{reg}}$ and $\Delta^{(1)}$. Then the following identities hold where programs of $\hat{\Delta}$ are in magenta and programs of $\left(\Delta^{(0)}\right)^{\text{reg}}$ are in blue:*

$$?(\phi) \cong ?(\phi) \qquad \alpha \cup \beta \cong \alpha \cup \beta \qquad \alpha;\beta \cong \alpha;\beta \qquad (\alpha)^* \cong (\alpha)^*$$

Using these identities, we can derive that $\alpha_{\text{mixed}} \cong \alpha_{\text{unmixed}} \equiv \left(p_1; p_2; (q_1 \cup q_2)^*\right)^*$. This allows us to *atomatize* the appearances of homogeneous programs in the case where logics with regular program support are turned into a heterogeneous theory: Any appearance of α_{mixed} in some formula can then be replaced by an appearance of α_{unmixed} using axiom (E). While the idea that a homogeneous loop may be replaced by a heterogeneous loop (and vice-versa) may seem obvious, *Theorem* 7.3 along with axiom (E) turns this knowledge into a set of proof rules that may be applied in practice.

8 Relative Completeness

The expressivity of heterogeneous dynamic logic, enabling us to reason about intertwined heterogeneous dynamics, would equally be its downfall if the expressivity were to keep us from verifying properties in practice. To this end, we usually desire to formalize dynamic logics in a way that guarantees *relative completeness*, i.e. the property that any valid formula can also been proven valid if we assume the availability of an oracle for first-order validity. This property is interesting, because it guarantees that, using the given calculus, proving properties about the dynamic logic is "no harder" than proving properties about its underlying (first-order) data theory. In **??** we present a calculus of meta properties which allows us to reason about relative completeness and related properties for lifted and combined dynamic logics. Due to space constraints, we focus the exposition of the paper to one central result, namely the preservation of relative completeness under combination in heterogeneous dynamic theories: Under common, reasonable assumptions, for two dynamic theories with relatively complete proof calculi their combination with a fully heterogeneous dynamic theory's proof calculus is once again relatively complete:

Theorem 8.1 (Relative Completeness of Fully Heterogeneous Dynamic Theories). *Let $\Delta^{(0)}, \Delta^{(1)}$ be two relatively complete (**??**), FOL expressive (**??**), Gödel Expressive (**??**) dynamic theories with finite support (**??**) that are communicating in $\Delta^{(01)}$ (**??**). Then the fully heterogeneous dynamic theory $\hat{\Delta}$ with its proof calculus $\vdash_{\hat{\Delta}}$ is relatively complete.*

9 Case Study

To address the case study presented in Section 2, we first derive results about the individual two components. Subsequently, we merge these results into a system level gurantee.

Java. For the Java component `ctrl` we wish to verify that once the relative distance to the stop sign (given as `p`) becomes too small it must brake by choosing an appropriate acceleration. Using JavaDL and the proof calculus implemented in the interactive theorem prover KeY [2, 7], we can then derive the result below. We do not only prove properties on the result of `this.acc`, but also on the preservation of the heap structure and the field values of `A`, `B`, and `T` (mechanized proof in KeY):

Lemma 9.1 (Correctness of `ctrl`). *The following* JavaDL *formula is valid*[2]*:*

$$\underbrace{\left(\begin{array}{l}\mathsf{A} > 0 \land \mathsf{A}^- \doteq \mathsf{A} \land \\ \mathsf{B} < 0 \land \mathsf{B}^- \doteq \mathsf{B} \land \\ \mathsf{T} > 0 \land \mathsf{T}^- \doteq \mathsf{T} \land \\ \texttt{heap\_assumptions}\,()\end{array}\right)}_{\texttt{ctrlPre}} \rightarrow [\texttt{ctrl}] \underbrace{\left(\begin{array}{l}\mathsf{B} \leq \texttt{this.acc} \leq \mathsf{A} \land \\ \texttt{heap\_assumptions}\,() \land \\ (\texttt{brake\_cond} \rightarrow \texttt{this.acc} \doteq \mathsf{B}) \land \\ \mathsf{A}^- \doteq \mathsf{A} \land \mathsf{B}^- \doteq \mathsf{B} \land \mathsf{T}^- \doteq \mathsf{T}\end{array}\right)}_{\texttt{ctrlPost}}$$

with `brake_cond` *abbreviating* $10^4(\mathsf{v}+1)^2 + (\mathsf{A}-\mathsf{B})\mathsf{T}(\mathsf{AT}+200(\mathsf{v}+1)) > -2\cdot 10^4\mathsf{B}(\mathsf{p}-1)$.

Differential Dynamic Logic. Using the proof calculus implemented in the theorem prover KeYmaera X, we can prove statements about hybrid programs. For example, we can prove the following formula valid which shows (under some assumptions) that x will always be at most s (the stop sign's position) and an additional invariant after execution of *env* (proof mechanized in KeYmaera X):

Lemma 9.2 (Evolution of *env*). *The following formula is valid in* $d\mathcal{L}$*:*

$$\underbrace{\left(\begin{array}{l}A > 0 \land B > 0 \land \\ T > 0 \land \mathit{acc\_assumptions} \land \\ x + v^2/(2B) \leq s\end{array}\right)}_{\mathit{envPre}} \rightarrow [\mathit{env}] \underbrace{\left(\overbrace{(x \leq s \land x + v^2/(2B) \leq s}^{\mathit{invariant}}\right)}_{\mathit{envPost}}$$

with the following abbreviation for acc_assumptions:

$$\left(-B \leq a <= A \land \left(\left(x + \frac{v^2}{2B} + \left(\frac{A}{B}+1\right)\left(\frac{AT^2}{2} + Tv\right) > s\right) \rightarrow a \doteq -B\right)\right) \lor (v \doteq 0 \land a \doteq 0)$$

Heterogeneous Safety. Both $d\mathcal{L}$ and JavaDL are instantiations of dynamic theories and we can hence construct a fully heterogeneous dynamic theory $\Delta^{(d\texttt{Java}\mathcal{L})}$, called differential JavaDL, over $\Delta^{(d\mathcal{L})}$ and $\Delta^{(\texttt{JavaDL})}$. To this end, we add heterogeneous atoms to $\Lambda_A^\cap$ with respective evaluation $\mathcal{A}^\cap$ for any integer variable v in JavaDL and any real arithmetic variable w in $d\mathcal{L}$:

- int2real (v, w) with $\mu \in \mathcal{A}^\cap(\text{int2real}\,(\mathsf{v}, w))$ iff $\mathbb{Z} \ni \hat{\mathcal{V}}\,(\mu, \mathsf{v}) = \hat{\mathcal{V}}\,(\mu, w)$
- round (v, w)
  with $\mu \in \mathcal{A}^\cap(\text{round}\,(\mathsf{v}, w))$ iff $\hat{\mathcal{V}}\,(\mu, \mathsf{v}) \in \mathbb{Z}$ and $\hat{\mathcal{V}}\,(\mu, \mathsf{v}) - 0.5 < \hat{\mathcal{V}}\,(\mu, w) \leq \hat{\mathcal{V}}\,(\mu, \mathsf{v}) + 0.5$

These atoms allow us to explicitly model the communication between $\Delta^{(d\mathcal{L})}$ and $\Delta^{(\texttt{JavaDL})}$ which is all we need to model the full heterogeneous system in $\Delta^{(d\texttt{Java}\mathcal{L})}$. After executing the controller we anonymize the $d\mathcal{L}$ variable a and set it to the value of the Java variable in `this.acc` via a check using the atomic formula int2real. Based on the updated acceleration we execute the environment *env* and similarly retrieve the new position and velocity. This can be formalized as

$$\texttt{ctrl};\ \hat{\alpha} \equiv \texttt{ctrl};\ (a := *;\ ?\,(\text{int2real}\,(\texttt{this.acc}, a * 100))\,;\ \mathit{env};\ \mathsf{p} := *;\ \mathsf{v} := *;\ ?\,(\text{coupling}))$$

where we abbreviate coupling $\equiv$ $(\text{round}\,(\mathsf{p}, 100\,(s - x)) \land \text{round}\,(\mathsf{v}, 100v))$. Using our heterogeneous calculus we can then prove that our heterogeneous system never overshoots the stop sign:

Lemma 9.3 (Safety of the Heterogeneous System). *The following formula is valid in* $\Delta^{(d\texttt{Java}\mathcal{L})}$*:*

$$\text{coupledPre} \rightarrow \left[(\texttt{ctrl};\ \hat{\alpha})^*\right] x \leq s \tag{3}$$

Proof Sketch. We provide a sketch how our heterogeneous calculus can be leveraged to prove Lemma 9.3 via Lemmas 9.1 and 9.2 (▷ additional proof branches see **??**):

[2]For the actual proof we require further assumptions which we summarize as `heap_assumptions`. This concerns the correct initialization of the heap, the correct instantiation of Java objects, etc. While this paper denotes object fields as variables (e.g. A), these values are technically obtained via a rigid function which we omit for clarity [2].

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\phi_1 \to [\texttt{ctrl}]\ \texttt{ctrlPost}}\,(\text{HR0})\,{*}\,(\text{KeY: Lemma 9.1})}{\phi_1 \wedge \phi_2 \to [\texttt{ctrl}]\ (\texttt{ctrlPost} \wedge \phi_2)}\,??}{\texttt{coupledPre} \to [\texttt{ctrl}]\ ((\texttt{ctrlPost} \wedge \phi_2) \wedge \phi_3)}\,??}{\texttt{coupledPre} \equiv (\texttt{ctrlPost} \wedge \phi_2) \wedge \phi_3}\qquad \dfrac{\dfrac{\dfrac{}{\texttt{coupledPre} \to \phi_3}\,{*}\,(\text{FOL})}{\texttt{coupledPre} \to [\texttt{ctrl}]\ \phi_3}\,(\text{V})}{((\texttt{ctrlPost} \wedge \phi_2) \wedge \phi_3) \to [\hat{\alpha}]\ \texttt{coupledPre}}\,\begin{array}{l}\textit{using Lemma 9.2}\\ \textit{see ??}\end{array}$$

$$\dfrac{\dfrac{\dfrac{\texttt{coupledPre} \to [\texttt{ctrl}]\ [\hat{\alpha}]\ \texttt{coupledPre}}{\texttt{coupledPre} \to [\texttt{ctrl};\ \hat{\alpha}]\ \texttt{coupledPre}}\,(;)}{\texttt{coupledPre} \to \left[(\texttt{ctrl};\ \hat{\alpha})^*\right] \texttt{coupledPre}}\,??\ \triangleright}{\texttt{coupledPre} \to \left[(\texttt{ctrl};\ \hat{\alpha})^*\right] x \leq s}\,??\ \triangleright$$

The validity of this formula guarantees safety ($x \leq s$) for any number of control-environment loop iterations assuming coupledPre (see **??**). First, note that there is no straight forward way to even specify this desired behavior without HDL as it represents a system level guarantee only emerging from the looped interaction of Java and the hybrid program. Secondly, note that our calculus proves this result by *decomposing* proof obligations until the JavaDL and $d\mathcal{L}$ lemmas above are applicable. This underscores the power of our logic to state and prove statements about systems that do not neatly live in the world of a single dynamic theory, but require their interaction.

10 Conclusion

This work introduces *Heterogeneous Dynamic Logic*, a compositional framework for the verification of heterogeneous programs. Similarly to how Satisfiability Modulo Theories modularly combines data logics to construct more expressive, combined first-order theories, Heterogeneous Dynamic Logic modularly combines program logics to construct more expressive, combined heterogeneous program logics. The resulting heterogeneous dynamic theories enable not only the *specification* of otherwise hard to formalize properties (see Sections 2 and 9), but also their *verification* by providing a proof calculus that decomposes problems in a manner that is compatible with existing homogeneous verification infrastructure – including, both, classical dynamic logic calculi and KAT-based equational reasoning. Section 8 shows, under common assumptions, that heterogeneous proof calculi inherit not just rules and axioms but also relative completeness. This proves that verifying heterogeneous programs is no harder than verifying properties about the homogeneous programs and their shared first-order structure (i.e., their data logic). Our proof theory of heterogeneous systems will not only enable the combination of existing verification methodologies, but also serves as a vehicle for future dynamic logic or KAT-based verification techniques, which can seamlessly integrate with the existing ecosystem through HDL.

Data Availability

We provide our extensible Isabelle formalization on Zenodo [55].

Acknowledgments

This work was supported by funding from the pilot program Core-Informatics of the Helmholtz Association (HGF) and by an Alexander von Humboldt Professorship.

References

[1] Noah Abou El Wafa and André Platzer. 2025. Complete first-order game logic. *CoRR*, abs/2504.03495. arXiv: 2504.03495.

[2] Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, Reiner Hähnle, Peter H. Schmitt, and Mattias Ulbrich, (Eds.) 2016. *Deductive Software Verification - The KeY Book - From Theory to Practice. LNCS*. Vol. 10001. Springer. ISBN: 978-3-319-49811-9. DOI: 10.1007/978-3-319-49812-6.

[3] Gidon Ernst, Matthias Güdemann, Alexander Knapp, Florian Nafz, Frank Ortmeier, Hella Ponsar, Gerhard Schellhorn, and Alexander Schiendorfer, (Eds.) 2025. *The many uses of dynamic logic. Go Where the Bugs Are - Essays Dedicated to Wolfgang Reif on the Occasion of His 65th Birthday*. LNCS. Springer, 56–82. DOI: 10.1007/978-3-031-92196-4_4.

[4] Vytautas Astrauskas, Peter Müller, Federico Poli, and Alexander J. Summers. 2019. Leveraging rust types for modular specification and verification. *Proc. ACM Program. Lang.*, 3, OOPSLA, 147:1–147:30. DOI: 10.1145/3360573.

[5] Michael Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs, and K. Rustan M. Leino. 2005. Boogie: A modular reusable verifier for object-oriented programs. In *Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures* (LNCS). Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, (Eds.) Springer, 364–387. DOI: 10.1007/11804192_17.

[6] Clark W. Barrett and Cesare Tinelli. 2018. Satisfiability modulo theories. In *Handbook of Model Checking*. Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, (Eds.) Springer, 305–343. DOI: 10.1007/978-3-319-10575-8_11.

[7] Bernhard Beckert. 2000. A dynamic logic for the formal verification of java card programs. In *Java on Smart Cards: Programming and Security, First International Workshop, JavaCard 2000, Cannes, France, September 14, 2000, Revised Papers* (LNCS). Isabelle Attali and Thomas P. Jensen, (Eds.) Springer, 6–24. DOI: 10.1007/3-540-45165-X_2.

[8] Bernhard Beckert, Daniel Bruns, Vladimir Klebanov, Christoph Scheben, Peter H. Schmitt, and Mattias Ulbrich. 2013. Information flow in object-oriented software. In *Logic-Based Program Synthesis and Transformation, 23rd International Symposium, LOPSTR 2013, Madrid, Spain, September 18-19, 2013, Revised Selected Papers* (LNCS). Gopal Gupta and Ricardo Peña, (Eds.) Springer, 19–37. DOI: 10.1007/978-3-319-14125-1_2.

[9] Bernhard Beckert and André Platzer. 2006. Dynamic logic with non-rigid functions. In *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings* (LNCS). Ulrich Furbach and Natarajan Shankar, (Eds.) Springer, 266–280. DOI: 10.1007/11814771_23.

[10] Bernhard Beckert, Peter Sanders, Mattias Ulbrich, Julian Wiesler, and Sascha Witt. 2024. Formally verifying an efficient sorter. In *Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I* (LNCS). Bernd Finkbeiner and Laura Kovács, (Eds.) Springer, 268–287. DOI: 10.1007/978-3-031-57246-3_15.

[11] Stefan Blom and Marieke Huisman. 2014. The vercors tool for verification of concurrent programs. In *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings* (LNCS). Cliff B. Jones, Pekka Pihlajasaari, and Jun Sun, (Eds.) Springer, 127–131. DOI: 10.1007/978-3-319-06410-9_9.

[12] Rose Bohrer. 2017. Differential dynamic logic. *Archive of Formal Proofs*, (Feb. 2017). https://isa-afp.org/entries/Differential_Dynamic_Logic.html, Formal proof development.

[13] Rose Bohrer and André Platzer. 2018. A hybrid, dynamic logic for hybrid-dynamic information flow. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*. Anuj Dawar and Erich Grädel, (Eds.) ACM, 115–124. DOI: 10.1145/3209108.3209151.

[14] Pei-Wei Chen, Shaokai Lin, Adwait Godbole, Ramneet Singh, Elizabeth Polgreen, Edward A. Lee, and Sanjit A. Seshia. 2025. Polyver: A compositional approach for polyglot system modeling and verification. In *Proceedings of the 25th Conference on Formal Methods in Computer-Aided Design, FMCAD 2025, Menlo Park, CA, USA, October 6-10, 2025*. Ahmed Irfan and Daniela Kaufmann, (Eds.) TU Wien Academic Press. DOI: 10.34727/2025/ISBN.978-3-85448-084-6_10.

[15] Alessandro Cimatti, Alberto Griggio, Sergio Mover, Marco Roveri, and Stefano Tonetta. 2022. Verification modulo theories. *Formal Methods Syst. Des.*, 60, 3, 452–481. DOI: 10.1007/S10703-023-00434-X.

[16] Edmund M. Clarke, Daniel Kroening, and Flavio Lerda. 2004. A tool for checking ANSI-C programs. In *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings* (LNCS). Kurt Jensen and Andreas Podelski, (Eds.) Springer, 168–176. DOI: 10.1007/978-3-540-24730-2_15.

[17] Martin de Boer, Stijn de Gouw, Jonas Klamroth, Christian Jung, Mattias Ulbrich, and Alexander Weigl. 2023. Formal specification and verification of jdk's identity hash map implementation. *Formal Aspects Comput.*, 35, 3, 18:1–18:26. DOI: 10.1145/3594729.

[18] Edsger W. Dijkstra. 1975. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18, 8, 453–457. DOI: 10.1145/360933.360975.

[19] Gidon Ernst, Wolfram Pfeifer, and Mattias Ulbrich. 2024. Contract-lib: A proposal for a common interchange format for software system specification. In *Leveraging Applications of Formal Methods, Verification and Validation. Specification and Verification - 12th International Symposium, ISoLA 2024, Crete, Greece, October 27-31, 2024, Proceedings, Part III* (LNCS). Tiziana Margaria and Bernhard Steffen, (Eds.) Springer, 79–105. DOI: 10.1007/978-3-031-75380-0_6.

[20] Jean-Christophe Filliâtre and Andrei Paskevich. 2013. Why3 - where programs meet provers. In *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Rome, Italy, March 16-24, 2013. Proceedings* (LNCS). Matthias Felleisen and Philippa Gardner, (Eds.) Springer, 125–128. DOI: 10.1007/978-3-642-37036-6_8.

[21] Simon Foster, Frank Zeyda, and Jim Woodcock. 2014. Isabelle/utp: A mechanised theory engineering framework. In *Unifying Theories of Programming - 5th International Symposium, UTP 2014, Singapore, May 13, 2014, Revised Selected Papers* (LNCS). David A. Naumann, (Ed.) Springer, 21–41. DOI: 10.1007/978-3-319-14806-9_2.

[22] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. 2015. Keymaera X: an axiomatic tactical theorem prover for hybrid systems. In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings* (LNCS). Amy P. Felty and Aart Middeldorp, (Eds.) Springer, 527–538. DOI: 10.1007/978-3-319-21401-6_36.

[23] Dov M. Gabbay. 1996. An overview of fibred semantics and the combination of logics. In *Frontiers of Combining Systems, First International Workshop FroCoS 1996, Munich, Germany, March 26-29, 1996, Proceedings*. Applied Logic Series. Franz Baader and Klaus U. Schulz, (Eds.) Kluwer Academic Publishers, 1–55. DOI: 10.1007/978-94-009-0349-4_1.

[24] Dov M. Gabbay. 1999. *Fibring Logics*. Clarendon Press, New York.

[25] Luis Garcia, Stefan Mitsch, and André Platzer. 2019. Hyplc: hybrid programmable logic controller program translation for verification. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2019, Montreal, QC, Canada, April 16-18, 2019*. Xue Liu, Paulo Tabuada, Miroslav Pajic, and Linda Bushnell, (Eds.) ACM, 47–56. DOI: 10.1145/3302509.3311036.

[26] Michael Greenberg, Ryan Beckett, and Eric Hayden Campbell. 2022. Kleene algebra modulo theories: a framework for concrete kats. In *PLDI '22: 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, San Diego, CA, USA, June 13 - 17, 2022*. Ranjit Jhala and Isil Dillig, (Eds.) ACM, 594–608. DOI: 10.1145/3519939.3523722.

[27] David Harel. 1979. *First-Order Dynamic Logic. LNCS*. Vol. 68. Springer. ISBN: 3-540-09237-4. DOI: 10.1007/3-540-09237-4.

[28] David Harel, Dexter Kozen, and Jerzy Tiuryn. 2000. *Dynamic Logic*. The MIT Press, (Sept. 2000). ISBN: 978-0-262-27495-1. DOI: 10.7551/mitpress/2516.001.0001.

[29] C. A. R. Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM*, 12, 10, 576–580. DOI: 10.1145/363235.363259.

[30] C. A. R. Hoare. 1997. Unified theories of programming. In *Mathematical Methods in Program Development*. Manfred Broy and Birgit Schieder, (Eds.) Springer Berlin Heidelberg, Berlin, Heidelberg, 313–367. ISBN: 978-3-642-60858-2. DOI: 10.1007/978-3-642-60858-2_21.

[31] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Aurora C. Schmidt, Ryan W. Gardner, Stefan Mitsch, and André Platzer. 2017. A formally verified hybrid system for safe advisories in the next-generation airborne collision avoidance system. *Int. J. Softw. Tools Technol. Transf.*, 19, 6, 717–741. DOI: 10.1007/S10009-016-0434-1.

[32] Chris Johannsen, Karthik Nukala, Rohit Dureja, Ahmed Irfan, Natarajan Shankar, Cesare Tinelli, Moshe Y. Vardi, and Kristin Yvonne Rozier. 2024. The moxi model exchange tool suite. In *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I* (LNCS). Arie Gurfinkel and Vijay Ganesh, (Eds.) Springer, 203–218. DOI: 10.1007/978-3-031-65627-9_10.

[33] Eduard Kamburjan, Stefan Mitsch, and Reiner Hähnle. 2022. A hybrid programming language for formal modeling and verification of hybrid systems. *Leibniz Trans. Embed. Syst.*, 8, 2, 04:1–04:34. DOI: 10.4230/LITES.8.2.4.

[34] Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. 2015. Frama-c: A software analysis perspective. *Formal Aspects Comput.*, 27, 3, 573–609. DOI: 10.1007/S00165-014-0326-7.

[35] Jonas Klamroth, Bernhard Beckert, Max Scheerer, and Oliver Denninger. 2023. Qin: enabling formal methods to deal with quantum circuits. In *IEEE International Conference on Quantum Software, QSW 2023, Chicago, IL, USA, July 2-8, 2023*. Shaukat Ali et al., (Eds.) IEEE, 175–185. DOI: 10.1109/QSW59989.2023.00029.

[36] Dexter Kozen. 1997. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19, 3, 427–443. DOI: 10.1145/256167.256195.

[37] Andrea Lattuada, Travis Hance, Chanhee Cho, Matthias Brun, Isitha Subasinghe, Yi Zhou, Jon Howell, Bryan Parno, and Chris Hawblitzel. 2023. Verus: verifying rust programs using linear ghost types. *Proc. ACM Program. Lang.*, 7, OOPSLA1, 286–315. DOI: 10.1145/3586037.

[38] Sarah M. Loos and André Platzer. 2016. Differential refinement logic. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*. Martin Grohe, Eric Koskinen, and Natarajan Shankar, (Eds.) ACM, 505–514. DOI: 10.1145/2933575.2934555.

[39] Peter Müller, Malte Schwerhoff, and Alexander J. Summers. 2016. Viper: A verification infrastructure for permission-based reasoning. In *Verification, Model Checking, and Abstract Interpretation - 17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016. Proceedings* (LNCS). Barbara Jobstmann and K. Rustan M. Leino, (Eds.) Springer, 41–62. DOI: 10.1007/978-3-662-49122-5_2.

[40] Greg Nelson and Derek C. Oppen. 1979. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1, 2, 245–257. DOI: 10.1145/357073.357079.

[41] Derek C. Oppen. 1980. Complexity, convexity and combinations of theories. *Theor. Comput. Sci.*, 12, 291–302. DOI: 10.1016/0304-3975(80)90059-6.

[42] Marco Paganoni and Carlo A. Furia. 2023. Verifying functional correctness properties at the level of java bytecode. In *Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings* (LNCS).

Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker, (Eds.) Springer, 343–363. DOI: 10.1007/978-3-031-27481-7_20.

[43] André Platzer. 2017. A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reason.*, 59, 2, 219–265. DOI: 10.1007/S10817-016-9385-1.

[44] André Platzer. 2008. Differential dynamic logic for hybrid systems. *J. Autom. Reason.*, 41, 2, 143–189. DOI: 10.1007/S10817-008-9103-8.

[45] André Platzer. 2012. The complete proof theory of hybrid systems. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*. IEEE Computer Society, 541–550. DOI: 10.1109/LICS.2012.64.

[46] André Platzer and Jan-David Quesel. 2009. European train control system: A case study in formal verification. In *Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM 2009, Rio de Janeiro, Brazil, December 9-12, 2009. Proceedings* (LNCS). Karin K. Breitman and Ana Cavalcanti, (Eds.) Springer, 246–265. DOI: 10.1007/978-3-642-10373-5_13.

[47] André Platzer and Yong Kiam Tan. 2020. Differential equation invariance axiomatization. *J. ACM*, 67, 1, 6:1–6:66. DOI: 10.1145/3380825.

[48] Enguerrand Prebet and André Platzer. 2024. Uniform substitution for differential refinement logic. In *Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part II* (LNCS). Christoph Benzmüller, Marijn J. H. Heule, and Renate A. Schmidt, (Eds.) Springer, 196–215. DOI: 10.1007/978-3-031-63501-4_11.

[49] H. G. Rice. 1953. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74, 2, 358–366. DOI: 10.2307/1990888.

[50] Philipp Rümmer and Muhammad Ali Shah. 2007. Proving programs incorrect using a sequent calculus for java dynamic logic. In *Tests and Proofs - 1st International Conference, TAP 2007, Zurich, Switzerland, February 12-13, 2007. Revised Papers* (LNCS). Yuri Gurevich and Bertrand Meyer, (Eds.) Springer, 41–60. DOI: 10.1007/978-3-540-73770-4_3.

[51] Gerhard Schellhorn and Wolfgang Ahrendt. 1997. Reasoning about abstract state machines: the WAM case study. *J. Univers. Comput. Sci.*, 3, 4, 377–413. DOI: 10.3217/JUCS-003-04-0377.

[52] Robert E. Shostak. 1978. An algorithm for reasoning about equality. *Commun. ACM*, 21, 7, 583–585. DOI: 10.1145/359545.359570.

[53] Yong Kiam Tan and André Platzer. 2021. An axiomatic approach to existence and liveness for differential equations. *Formal Aspects Comput.*, 33, 4-5, 461–518. DOI: 10.1007/S00165-020-00525-0.

[54] Samuel Teuber, Stefan Mitsch, and André Platzer. 2024. Provably safe neural network controllers via differential dynamic logic. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*. Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, (Eds.)

[55] [SW] Samuel Teuber, Mattias Ulbrich, André Platzer, and Bernhard Beckert, Artifact for "Heterogeneous Dynamic Logic: Provability Modulo Program Theories" Mar. 2026. DOI: 10.5281/zenodo.19551156.

[56] Cesare Tinelli and Mehdi T. Harandi. 1996. A new correctness proof of the Nelson-Oppen combination procedure. In *Frontiers of Combining Systems, First International Workshop FroCoS 1996, Munich, Germany, March 26-29, 1996, Proceedings* (Applied Logic Series). Franz Baader and Klaus U. Schulz, (Eds.) Kluwer Academic Publishers, 103–119. DOI: 10.1007/978-94-009-0349-4_5.

[57] Cesare Tinelli and Calogero G. Zarba. 2004. Combining decision procedures for sorted theories. In *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004, Proceedings* (LNCS). José Júlio Alferes and João Alexandre Leite, (Eds.) Springer, 641–653. DOI: 10.1007/978-3-540-30227-8_53.

[58] Mattias Ulbrich. 2010. A dynamic logic for unstructured programs with embedded assertions. In *Formal Verification of Object-Oriented Software - International Conference, FoVeOOS 2010, Paris, France, June 28-30, 2010, Revised Selected Papers* (LNCS). Bernhard Beckert and Claude Marché, (Eds.) Springer, 168–182. DOI: 10.1007/978-3-642-18070-5_12.

[59] Mattias Ulbrich. 2013. *Dynamic Logic for an Intermediate Language: Verification, Interaction and Refinement*. Ph.D. Dissertation. Karlsruhe Institute of Technology. DOI: 10.5445/IR/1000041169.

[60] Yuanrui Zhang. 2024. Parameterized dynamic logic - towards A cyclic logical framework for program verification via operational semantics. *CoRR*, abs/2404.18098. arXiv: 2404.18098.

Received 2025-11-13; accepted 2026-04-03