A Hybrid, Dynamic Logic for Hybrid-Dynamic Information Flow

Brandon Bohrer  
Carnegie Mellon University  
Pittsburgh, PA  
bbohrer@cs.cmu.edu

Abstract

Information-flow security is important to the safety and privacy of cyber-physical systems (CPSs) across many domains: information leakage can both violate user privacy and provide information that supports further attacks. CPSs face the challenge that information can flow both in discrete cyber channels and in continuous real-valued physical channels ranging from time to physical flow of resources. We call these hybrid information flows and introduce dHL, the first logic for verifying these flows in hybrid-dynamical models of CPSs. We achieve verification of hybrid information flows by extending differential dynamic logic (dL) for hybrid-dynamical systems with hybrid-logical features for explicit representation and relation of program states. By verifying hybrid information flows, we ensure security even under a strong attacker model wherein an attacker can observe time and physical values continuously. We present a Hilbert-style proof calculus for dHL, prove it sound, and compare the expressive power of dHL with dL. We demonstrate dHL’s abilities by developing a hybrid system model of the smart electrical grid FREEDM. We verify that the naïve model has a previously-unknown information flow vulnerability and verify that a revised model resolves the vulnerability. To the best of our knowledge, this is both the first information flow proof for hybrid information flows and the first for a hybrid-dynamical model. We discuss applications of hybrid information flow to a range of critical systems.

Keywords  dynamic logic, hybrid logic, hybrid systems, information flow, cyber-physical systems, formal verification, smart grid

1 Introduction

Cyber-physical systems (CPSs), which feature discrete computer control interacting with a continuous physical environment, are ubiquitous. They include critical infrastructure such as electricity, natural gas, and petroleum transportation grids, medical devices such as pacemakers and insulin pumps, and transportation systems including aircraft, trains, and automobiles. Because of their criticality, it is essential to ensure their safe, correct operation, and formal methods for safety (e.g. collision-freedom) in CPS have had important successes [11, 15, 24, 26, 28].

In contrast, there is relatively little work on formal methods for security of CPSs, which is often as critical as physical safety. Because it has an especially critical impact across a variety of systems, this paper focuses on non-deducibility information-flow security, which captures the notion that an attacker cannot infer private information for certain in a system with non-observable non-determinism. As consumer-facing infrastructure including electrical and telecommunication networks is increasingly computerized, it faces new potential to leak confidential customer information. Beyond leaking customer information, it has been suggested [3] that information flow leaks have the potential to aid attackers in identifying vulnerable infrastructure targets. Computerized medical devices risk leaking private (HIPAA-protected) medical records, which can enable attacks that can have life-threatening results, such as ventricular fibrillation [14]. Information leakage concerns are also significant in the transportation domain, e.g. position-spoofing attacks in aircraft have been proposed that could cause major disruptions to air-traffic control [31].

The common feature in all these applications is that information can flow not only through computer communication channels, but physical channels as well: transmission lines, pipelines, the human body1, and roadways. Because information flows through both computation and physics, CPSs demand a hybrid notion of flow combining discrete and continuous flows. While several approaches on CPS information flow have been explored [1, 3, 13, 17, 34], all the work we are aware of employs a coarse discrete abstraction of physical dynamics and time. This is limiting because a real attacker can record continuous signals and make observations at any of uncountably-many moments in time, so discrete model cannot represent the full abilities of an attacker and thus cannot verify security against a real adversary. In contrast, hybrid information flows allow us to verify security even when an attacker can observe continuously-changing variables in continuous time. Furthermore, even if an attacker can only make discrete observations, discretizing a naturally-hybrid model can make it needlessly complex and thus more difficult to verify: for example, simple differential equations used in physical dynamics often have complicated solutions, so modeling a differential equation by implementing its solution as a discrete action makes verification much harder.

We provide the first approach for verifying security of hybrid information flows by introducing the logic dHL. It extends differential dynamic logic2 dL for reachability of hybrid-dynamical systems with first-order hybrid logic [6], which provides first-class representation of program states. This combination of hybrid dynamics with first-class program states enables us to verify hybrid information flow security. In capturing physical and temporal phenomena, hybrid dynamics also provide a flexible framework for expressing side-channels, which are then verified with the same techniques as any other cyber-physical channel.

The distinguishing feature of information flow (vs. safety and liveness proofs in dL) is that it is not a trace property, but a 2-trace

---

1While the products on the market today communicate through cyber means, research prototypes [29] use living tissue as a medium for exchanging cryptographic keys.

2And the proposed extension dLh [23], which is not strong enough for information flow.
hyperproperty [10] (i.e. a property of pairs of program traces). This poses a hurdle for program verification calculi: Hoare calculi, for example, cannot verify hyper-properties without significant source-level transformations such as self-composition [5]. The situation is similar [32] for dynamic logics, although the exact translation details differ. These source transformations obscure the true verification task at hand. The impact of this obscurance is visible in practical proving, where self-composition has been noted [33] to make verification needlessly difficult.

Our use of hybrid logic overcomes this challenge in a novel way by providing first-class representation of program states, which makes direct statements and proofs of information-flow properties straightforward. In doing so, we put light on the henceforth-unseen connection between hybrid logic and hyper-properties: any hyperproperty has a natural statement in hybrid dynamic logic. Beyond its aesthetic appeal, this generality promises to enable verification of numerous related hyper-properties in one common logic, without having to adapt the logic to: (a) additional notions of information flow such as non-interference (b) additional hyper-properties such as robustness and service-level objective properties [10].

We introduce a Hilbert-style proof calculus for dHL and prove it sound. We relate the complexity of proofs in dHL vs. dL: a reduction is possible for a significant fragment of dHL, but impractical: (a) it has limitations when applied to the general case, interfering with advanced proof techniques such as refinement [18] for modular verification, (b) the reduction causes quadratic formula size blowup in the worst case, and (c) the reduction is surprisingly subtle, suggesting that a proof by reduction to dL would be not only needlessly long, but unintuitive.

The dHL calculus, as with modern implementations [12] and machine-checked correctness proofs [7] for dL, is based on uniform substitution [28][9, §35], which improves the ease with which dHL could be implemented and its soundness proof checked by computer. From this calculus founded in hybrid logic, we show that intuitive high-level bisimulation rules for information-flow are easily derived.

As an example application, we provide the first hybrid-dynamical model of a smart-grid controller with concrete dynamics for distributed energy generation/storage and load-balancing, in contrast to prior models [3] of the FREEDM [16] smart-grid, which consider only the high-level structure of message-passing interactions between components. Our precise dynamics reveal a previously unknown vulnerability in the deterministic control algorithm [2] used by FREEDM. We then prove that adding nondeterminism solves the bug even in presence hybrid information flow. The non-deducibility [4] formulation of information-flow we use commonly arises in CPS where nondeterminism is used to model flexibility in a controller and where, e.g. attackers typically do not have direct access to our instruction pointer. Our hybrid model of FREEDM captures its essential hybrid dynamics and our proof demonstrates important features of proofs in dHL: (1) The well-understood principle of proof by bisimulation translates naturally to dHL proofs, (2) dHL provides an effective mechanism to tease apart the interactions between discrete transitions and continuous flow, enabling verification to scale to the complex interactions found in CPSs, and (3) Typical CPSs have sufficiently complex information flows to warrant a manual deductive approach such as ours.

These traits are typical across different domains of CPS, showing that our approach holds promise for verifying information flow of applications in various domains beyond smart-grids.

### 2 The Logic dHL

We present the complete syntax and semantics of the logic dHL, extending the dynamic logic dL with hybrid logic’s explicit representation of program states. This presentation is based on the formalism of uniform substitution [28]: symbols ranging over predicates, programs, etc. are explicitly represented in the syntax. This formalism has been used in modern theorem-prover implementations [12] and machine-checked soundness proofs [7] for dL, simplifying the future work of implementing and mechanizing dHL.

#### 2.1 Syntax

The expressions of dHL consist of real-valued terms θ, world-valued terms w, programs α, and formulas ϕ. We write Θ for an arbitrary term θ or w, and write e when an expression can be either a term θ or a formula ϕ.

**Definition 1** (Real-valued terms of dHL).

\[ θ ::= c \mid x \mid f(\overline{θ}) \mid F \mid θ + θ \mid θ \cdot θ \mid @\overline{θ} θ \]

Here \( c \in \mathbb{Q} \) are literals and \( x \in \mathcal{V} \) are real-valued *program variables*, which are said to be *flexible* because they can be bound in quantifiers. Their rigid counterparts are *nullary function symbols f(\overline{θ})* that cannot be bound. The meaning of a function symbol \( f(\overline{θ}) \) depends on a fixed, arbitrary number of real-valued arguments. Functional F are a generalization of functions whose meaning depends on all flexible symbols. Functions and functionals are used to express axioms in Section 5. The terms of dHL extend dL with at-terms @\overline{θ} θ denoting the value of term θ in the state denoted by the world-valued term w.

**Definition 2** (World-valued terms of dHL).

\[ w ::= s \mid \overline{s} \]

The language of world-valued terms w is simple, consisting only of *world variables* s, t and *nominals* π, ρ, which differ only in that world variables are flexible while nominals are rigid.

**Definition 3** (Programs of dHL).

\[ α, β ::= ?(ϕ) \mid x := θ \mid x := s \mid x' = θ \\& ψ \mid α \cup β \mid α; β \mid α^* \mid α \]

Programs in dHL are identical to the hybrid programs of dL. Hybrid programs combine discrete programming constructs with differential equations to provide a program representation of hybrid systems. The atomic dL programs are tests ?(ϕ) that abort execution if ϕ is false, assignments x := θ and x := s which update program variable x to the value of θ or a nondeterministic value, differential equation evolution x' = θ \& ψ, and object-level program constants a which range over fixed, arbitrary programs and should not be confused with program metavariabes α used in schemas and theorems. Differential equations are the defining feature of dL: their effect is to evolve the differential equation x' = θ nondeterministically for any duration, but only so long as the formula ψ is always true. They are composed with nondeterministic choice α \cup β running exactly one of α or β, sequential composition α; β, and nondeterministic iteration α^* running α any finite number of times sequentially. Traditional deterministic programming constructs can be derived from the nondeterministic hybrid program constructs,
A Hybrid, Dynamic Logic for Hybrid-Dynamic Information Flow

e.g. if $(\phi) [\alpha]$ else $[\beta] \equiv (?) (\phi); \alpha \cup (?) (\neg \phi); \beta)$ and while $(\phi) [\alpha] \equiv [(? (\phi); \alpha^*)] (\neg \phi)$. 

\textbf{Definition 4} (Formulas of dHL).

$$\phi, \psi ::= \phi \land \psi \mid \neg \phi \mid \exists s: R. \phi \mid \theta_1 \geq \theta_2 \mid (\alpha) \phi$$

$$\mid \exists s: W. \phi \mid w \mid @w \phi \mid \downarrow s \phi \mid p(\Theta) \mid P$$

The operators $\land, \neg, \exists s: R. \phi$, $\theta_1 \geq \theta_2$ are as in first-order logic. As in dL, the dynamic logic modality $(\alpha) \phi$ says there exists an execution of the (nondeterministic) program $\alpha$ where $\phi$ holds in the ending state.

The following features are added from first-order hybrid logic. The quantifier $\exists s: W. \phi$ says that there exists a (program state) $s$ for which $\phi$ holds, where $\phi$ mentions the world variable $s$. We will also use the universal quantifier $\forall s: W. \phi$, which is a derived construct by the equivalence $\forall s: W. \phi \equiv \neg \exists s: W. \neg \phi$. Nominal formulas $w$ are true in the one state denoted by the world-valued term $w$. Note the same syntax is used regardless whether $w$ appears as a term or formula; these usages are distinguished by syntactic context. The hybrid satisfaction modality @w$ϕ$ says that $ϕ$ is true at the unique state named by w. In addition to the typical existential and universal quantifiers, hybrid logic features the local quantifier $\downarrow s \phi$ which binds the current state to the world variable $s$ within the formula $\phi$, whereas the universal quantifier $\forall s: W. \phi$ binds an arbitrary state to $s$. The local quantifier $\downarrow s \phi$ can actually be derived as $\downarrow s \phi \leftrightarrow \exists s: W. (s \land \phi)$ or equivalently $\forall s: W. (s \rightarrow \phi)$. We present this quantifier in its entirety regardless, because it is important to information-flow applications and may be unfamiliar to the reader. The connectives $\downarrow s \phi$ and @w$ϕ$ can be understood by computational intuition as well: storing or loading the current state to $s$/from $w$, respectively.

The predicate symbols $p(\Theta)$, can range over both real-valued terms $\theta$ and nominal expressions $w$, and are used in axioms to stand for propositions. Beyond axioms, predicates will be used widely in bisimulation arguments for information-flow: $R(i,j)$ denotes a binary predicate over nominals. Predicational $P$ simply stand for arbitrary formulas and are used in axioms in Section 5.

3 Information-Flow Example: FREEDM Smart-Grid

In this section, we introduce two variants of a model for the NSF FREEDM [16] smart-grid, a microgrid which controls a local section of the power grid and interacts with the surrounding macrogrid. Our model is the first hybrid-systems model of FREEDM and closely follows the published algorithm [2], incorporating detailed dynamics not present in prior models [3]. We show how to state information-flow security properties and their negations in dHL. We will then prove the properties in Sections 8 and 9 once the proof calculus is introduced.

Smart-grids like FREEDM use computer control to make electrical grids more robust, efficient, and cost-effective in face of increasingly diverse power loads and supplies. Computer control in grids makes joint cyber-physical security of this critical national infrastructure essential. Not only can information flow violations compromise private consumer information, but it has been suggested they can aid attackers [3] in identifying potential targets for physical attacks.

\begin{figure}[t]
\centering
\includegraphics[width=0.8\textwidth]{figure1.png}
\caption{FREEDM Load Balancing}
\end{figure}

\textbf{Scenario} We look at an exchange (depicted in Figure 1) that migrates power between two neighboring transformers $T_1$ and $T_2$ connected to a macrogrid $G$ over a shared line. Each transformer $T_i$ carries power $P_i$ and is connected to a renewable energy resource $R_i$; household demanding power $D_i$ and energy storage device $B_i$. The transformers are connected by a communication Link. While a real instance of the FREEDM grid has more than two transformers, any individual migration takes place between exactly two transformers, so the two-transformer case provides important insight into the general case.

Each transformer can be in one of three demand states: Low Demand, Normal Demand, or High Demand. The load balancing algorithm [2] states that:

- Net demand $N_i$ is the difference of gross power demand $D_i$ and the sum of power draw $P_i$ and generation $R_i$.
- A transformer is in Low Demand if it has net demand $N_i < 0$; High Demand if net demand exceeds a provided threshold $N_i \geq \text{Thresh} > 0$ or Normal otherwise.
- If any transformer i is Low (has excess power) while its opposite i is High, it migrates at a provided constant rate $\alpha$; we stop until one of them is Normal.
- Any excess power supply $-N_i > 0$ not used in migration is accumulated as energy $0 \leq B_i \leq B_{\text{max}}$ in battery $i$.
- Any excess power demand $N_i > 0$ not met by migration is drawn from energy in battery $B_i$ with power $b_i$ and migration rate $b_{\text{m}}$, or sold to the grid if the battery is full ($B_i = B_{\text{max}}$).
- If $T_i$’s corresponding battery $B_i$ is empty, it draws power $G$ (with migration rate $G = G_{\text{m}}$) from the macrogrid instead.

In Figure 2, we express the algorithm (and the accompanying physics) as a hybrid program $\pi_\theta$. We separate out the battery controller $bat$ in order to provide two implementations: a deterministic implementation $bat_{\theta}$ of the above algorithm, which will be shown insecure, and a nondeterministic version $bat_{\theta}$, which will be shown secure. We abbreviate $\pi_\theta \equiv \pi_{bat_{\theta}}$ and $\pi_{bat} \equiv \pi_{bat_{\theta}}$ for the full program using the insecure or secure battery, respectively.

Our treatment of $D_i$ and $R_i$ is general, assuming only that they are non-negative and can change countably often. Time $t' = 1$ is not used in our controller, but will factor into our proofs because it is observable by attackers and we wish to know that, e.g. observing the duration of the ODE does not leak the continuous variables $P_i, B_i, b_i$.

\textbf{Defining Information Flow} The general formulation of non-deducibility information flow is:
\( a_F \equiv (\text{ctrl; plant})^* \)

\[
\begin{align*}
\text{ctrl} &\equiv \{ D_1 := \epsilon; (D_1 \geq 0) ; \ R_1 := \epsilon; (R_1 \geq 0) ; \ N_1 := D_1 - (R_1 + P_1) ; \\
&\quad \text{if}(N_1 \geq \text{Thresh} \land N_1 < 0) \{ m := M \cdot (-1)^i \} \\
&\quad \text{else} \{ m := 0 \} ; \\
&\quad \text{bat} \} \\
\text{plant} &\equiv \{ P'_f = m \cdot (-1)^i , B'_f = b_i , B'_t = b_{m_t} , G' = Gm , t' = 1 \land B_i \geq 0 \}
\end{align*}
\]

\( \text{bat}_f \equiv \left\{ G := 0 ; \ b_{m_i} := 0 ; \ Gm := 0 ; \right\} \)

\[
\begin{align*}
&\quad \text{if}((N_i \leq 0 \land B_i < B_{\text{max}}) \lor (N_i > 0 \land B_i > 0)) \{ \\
&\quad \quad b_i := -N_i ; \ b_{m_i} := b_{m_i} + m \cdot (-1)^{i+1} \\
&\quad \quad \text{else} \{ b_i := 0 ; G := G + N_i ; Gm := Gm + m \cdot (-1)^{i+1} \} \\
&\quad \} \right)
\]

\( \text{bat}_S \equiv \left\{ G := 0 ; \ b_{m_i} := 0 ; \ Gm := 0 ; \right\} \)

\[
\begin{align*}
&\quad \text{if}(B_i < B_{\text{max}}) \lor (N_i > 0 \land B_i > 0) ; \\
&\quad b_i := -N_i ; b_{m_i} := b_{m_i} + m \cdot (-1)^{i+1} \\
&\quad \text{else} \{ b_i := 0 ; G := G + N_i ; Gm := Gm + m \cdot (-1)^{i+1} \} \\
&\right)
\]

**Figure 2.** FREEDM Model with insecure battery \( \text{bat}_f \) and secure battery \( \text{bat}_S \)

**Definition 5** (Non-deducibility Information-Flow Security). Let \( \alpha \) be a program and let \( L \) be the set of publicly observable expressions, e.g. \( L = \{ G,t \} \) for FREEDM with publicly-known time \( t \) and macrogrid flow \( G \). Predicate \( \text{pre}(i) \) is problem-specific and says world \( i \) satisfies additional preconditions. The binary relation \( R(i,j) \) says \( i \) and \( j \) agree on all public expressions \( L \), and is defined by:

\[
R(i,j) \equiv \left( \bigwedge_{\theta \in L} \theta(i) = \theta(j) \right) \land \left( \bigwedge_{\phi \in L} \phi(i) \leftrightarrow \phi(j) \right)
\]

We then define non-deducibility information flow:

\[
\forall i_1,i_2,o_1 : \mathcal{W} \left( \oplus_{\iota_1} (\alpha)_o \land R(i_1,i_2) \land \text{pre}(i_1) \land \text{pre}(i_2) \right) \\
\rightarrow \oplus_{\iota_2} (\alpha)_o \phi R(o_1,o_2)
\]

Non-deducibility means an observer can never deduce anything about the input value \( \theta_x \) of a private variable \( x \notin L \) from the final values of public expressions \( \oplus_\theta (\theta \lor \phi) \), when the observer does not directly observe how nondeterminism is resolved. This is the case when Definition 5 holds because it says every input state \( i_2 \) that agrees on public terms \( R(i_1,i_2) \) has at least one program path \( @_{\iota_2} (\alpha)_o \phi \) (chosen by resolving nondeterministic choices appropriately) that would explain the final values of public expressions at \( o_1 \). Because all inputs would have made the output possible, it is impossible to deduce anything about the input state. As we will see in Section 9, the core challenge in proving this property is identifying which program path explains any given final public value.

In Section 8 we will also prove \( \text{bat}_f \) is non-deducibility insecure, by proving the negation of non-deducibility security, i.e.:

\[
\exists i_1,i_2,o_1 : \mathcal{W} \left( R(i_1,i_2) \land @_{\iota_1} (\alpha)_o \land @_{\iota_2} (\alpha)_o \neg R(o_1,o_2) \right)
\]

4 **Semantics**

We return to developing dHL. Its semantic development begins with preliminaries on states and interpretations.
Equation 12 says differential equations evolve according to the solution of the ODE for any nondeterministic duration \( t \geq 0 \), but must stop while the formula \( \psi \) still holds. In Equation 14, \( \circ \) is the composition operator on relations. In Equation 15, \( \llbracket [\sigma] \rrbracket \) is the reflexive, transitive closure of relation \( \llbracket [\sigma] \rrbracket \). Equation 16 says program constants receive their meaning from the interpretation (and galaxy, because programs can mention nominals via formulas).

**Definition 9 (Formula semantics).**

\[
\omega \in \llbracket [\phi \land \psi] \rrbracket I g \text{ iff } \omega \in \llbracket [\phi] \rrbracket I g \text{ and } \omega \in \llbracket [\psi] \rrbracket I g \\
\omega \in \llbracket [\neg \phi] \rrbracket I g \text{ iff } \omega \notin \llbracket [\phi] \rrbracket I g \\
\omega \in \llbracket [\exists \theta : R. \phi] \rrbracket I g \text{ iff } \exists \theta \in R. \omega \in \llbracket [\phi] \rrbracket I g \text{ for some } \theta \in R \\
\omega \in \llbracket [\theta_1 \geq \theta_2] \rrbracket I g \text{ iff } \llbracket [\theta_1] \rrbracket I g \omega \geq \llbracket [\theta_2] \rrbracket I g \omega \\
\omega \in \llbracket [\sigma(x)] \rrbracket I g \text{ iff } \forall v \in \llbracket [x] \rrbracket I g \text{ for some } v \in \mathcal{W} \\
\omega \in \llbracket [\nu \phi] \rrbracket I g \text{ iff } \forall v \in \llbracket [\nu \phi] \rrbracket I g \text{ for } v = \llbracket [w] \rrbracket I g \\
\omega \in \llbracket [\exists \nu \phi] \rrbracket I g \text{ iff } \exists \nu \in \llbracket [\nu \phi] \rrbracket I g \omega \\
\omega \in \llbracket [\exists \nu \phi] \rrbracket I g \text{ iff } \exists \nu \in \llbracket [\nu \phi] \rrbracket I g \omega \\
\omega \in \llbracket [\nu \phi] \rrbracket I g \text{ iff } \llbracket [\nu \phi] \rrbracket I g \omega \\
\omega \in \llbracket [\phi(\Theta)] \rrbracket I (g \circ (I))(p) \\
\omega \in \llbracket [\phi] \rrbracket I g \text{ iff } \omega \in (I)(g)(p) 
\]  

Equations (1-4) are a standard definition of first-order logic connectives, wherein \( \llbracket [\theta_1] \rrbracket I g : \mathbb{R} \) is the denotation of real-valued term \( \theta_1 \) (Definition 6). In Equation 19, \( \llbracket [\phi] \rrbracket I g \) denotes the result of replacing the value of program variable \( x \) with \( r \in \mathbb{R} \) in state \( \omega \). Equation 5 defines the reachability modality \( (\alpha)\phi \): we employ a Kripke semantics where possible worlds are program states (Definition 8). Equations (6-9) define the hybrid-logical operators, where \( \llbracket [w] \rrbracket I g : \mathcal{W} \) (Definition 7) is the denotation of a world term. Equation (10-11) say predicates and predicationals derive their meaning from the interpretation \( I \), with the difference that predicates depend only on their arguments while predicationals depend on the entire state. We say a formula \( \phi \) is valid when the relation \( \omega \in \llbracket [\phi] \rrbracket I g \) holds for all states \( \omega \), galaxies \( g \), and interpretations \( I \).

### 5 Axiomatization

We present a sound axiomatization of dHL, which is used for deductive verification. The axiomatization is given in Hilbert style, i.e. axioms are used wherever possible, with a minimal number of proof rules. Axioms are instantiated with a uniform substitution [28][9, §35] rule: from the validity of \( \phi \) we can derive validity of \( \sigma(\phi) \) where substitution \( \sigma \) specifies concrete replacements for some or all rigid symbols in a formula \( \phi \):

\[
\text{US} \quad \phi \quad \sigma(\phi) 
\]  

The side-conditions determining which substitutions \( \sigma \) are sound are non-trivial, and constitute much of the soundness proof in Section 6, with the benefit that soundness arguments and implementation for individual axioms are greatly simplified.

**Program Axioms**

The axioms for programs in diamond modalities in Figure 3 are as in dl [24]. Rules for box modalities can be derived by duality (axiom \( \langle \cdot \rangle \), Figure 3). With the exception of the loop rules, these rules can be read off directly from the denotational semantics of hybrid programs. The axiom \( \langle \cdot \rangle \) replaces a differential equation with a global solution, represented here by the expression \( y(t) \). Loops can be finitely unfolded with the axiom \( (\ast) \). More often, we reason by induction using axiom \( I \).

**Modal Axioms and Hilbert Rules**

Generic modal axioms and Hilbert rules are as in dl [24] and are listed in Figure 3. The axiom \( \langle \cdot \rangle \) relates the diamond and box modalities, and is used to derive axioms for box modalities \( [\sigma] \). The Hilbert rules G, US, and MP are used to compose axioms into proofs. Axiom V says nullary predicates \( p() \) are preserved under program execution because they depend on no program variables.

**Hybrid Rules + Axioms**

Our hybrid modality and quantifier axioms come from first-order hybrid logic [6] and Combinatory Dynamic Logic [21] and are listed in Figure 4. Axiom \( @id \) says every nominal constant formula \( \exists \) is satisfied at the state named by \( \exists \). Axiom \( @\circ \) says nested @ modalities collapse to the inner modality. Axiom \( I \circ \) introduces an @ modality when \( I \) is the current state. Axiom \( @\ast \) replaces equal states in context. Axiom \( \langle \ast \rangle \) introduces an @ modality for any state \( I \) reachable by a program \( a \). Axioms \( \forall \Theta \circ \) and \( \forall \Theta \) are the typical elimination and Skolemization rules for universal quantifiers over worlds. Axiom \( \Downarrow \) reduces the local quantifier to its definition in terms of the existential operator. Axiom \( \exists W \) says a name may be introduced for the current state. Homomorphism axiom family \( @i \) completely captures the meaning of at-terms. Axiom schema \( BW \) is the Barcan formula for world variables. If we prohibit nominals from occurring in programs \( a \), then \( BW \) simplifies to a concrete axiom with program constant \( a \) in place of schematic program \( a \).

### 6 Theory

We now develop the theory of dHL, showing that the proof calculus is sound and comparing the expressiveness of dHL with other logics.\(^{2}\)

\(^{2}\)The presentation of this axiom is simplified for clarity. In reality, differential equation solving is implemented with the combination of several axioms [28].
6.1 Soundness

We show soundness of dHL by extending the soundness proof for the uniform substitution calculus for dL [28]. Uniform substitution allows for a modular soundness proof: the soundness proof is separated into proving a finite list of axioms are valid and that uniform substitution (along with the remaining Hilbert rules) preserves validity.

\textbf{Substitution} The US rule in dHL is analogous to that in dL:

\[(\text{US}) \quad \frac{\phi}{\sigma(\phi)}\]

In dL, the US rule is sound when the substitution \(\sigma\) does not introduce free references to bound variables. Such substitutions are called \textit{admissible}, a condition which can be checked syntactically.

In dHL we generalize the notion of admissibility: Admissible substitutions do not introduce free references to any bound program variable or \textit{world variable}.

Admissibility conditions are checked recursively during the substitution algorithm proper (Figure 5). We give the new cases and their admissibility conditions (if any), with the full algorithm in Appendix A. The free-variable function FV(\(e\)) computes all flexible symbols that might influence \(e\) in the obvious way (Appendix A).

The admissibility conditions in Figure 5 are just instantiations of the principle that substitution should not introduce new free references under a binder. However, soundness demands strong admissibility conditions in the \(\sigma(\phi)\) and \(\sigma(\theta)\) cases because they bind the entire state; thus they deserve attention. With such strong admissibility conditions, one might wonder how an axiom like \(\theta\) (which features \(\phi\) and \(\theta\) terms) can ever be instantiated. This is exactly where the distinction between, e.g. functions \(f(\theta)\) and functionals \(F\) becomes essential. Because a functional \(F\) depends on every variable, an axiom like \(\phi\) can use them to express that the axiom supports any term that might be substituted for \(F\). For example, the admissibility check for \(\sigma(\phi)\) would fail only if \(\sigma(\phi)\) depends on \textit{strictly more} variables than \(\phi\). Because \(\phi\), by definition, may depend on all variables, this is not possible and thus all instantiations \(\sigma(\phi)\) for \(F\) are sound when applying uniform substitution to \(\theta\).

Figures 4 and 5. Uniform Substitution Algorithm (New Cases)

<table>
<thead>
<tr>
<th>Case Replacement</th>
<th>Admissible when:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sigma(\omega \phi) = \sigma(\omega \sigma(\phi)))</td>
<td>No new (x \in \mathcal{V}) in FV((\sigma(\theta)))</td>
</tr>
<tr>
<td>(\sigma(\omega \theta) = \sigma(\omega (\sigma(\theta))))</td>
<td>No new (x \in \mathcal{V}) in FV((\sigma(\theta)))</td>
</tr>
<tr>
<td>(\sigma(\mathcal{V}: \omega \phi) = \mathcal{V}: \omega \sigma(\phi))</td>
<td>No new (s \in \mathcal{V}) in FV((\sigma(\theta)))</td>
</tr>
<tr>
<td>(\sigma(\mathcal{V}: \omega \phi) = \mathcal{V}: \omega \sigma(\phi))</td>
<td>No new (s \in \mathcal{V}) in FV((\sigma(\theta)))</td>
</tr>
<tr>
<td>(\sigma(\mathcal{V}: \omega \phi) = \mathcal{V}: \omega \sigma(\phi))</td>
<td>No new (s \in \mathcal{V}) in FV((\sigma(\theta)))</td>
</tr>
<tr>
<td>(\sigma(\mathcal{V}: \omega \phi) = \mathcal{V}: \omega \sigma(\phi))</td>
<td>No new (s \in \mathcal{V}) in FV((\sigma(\theta)))</td>
</tr>
</tbody>
</table>

\[\text{Proof.} \text{ (In Appendix D.) Soundness of US follows the same structure as previous work [28]. We begin with lemmas on the correctness of free variable and signature computations, where the signature } \Sigma(e) \text{ is the analog of FV(}e\text{) for rigid symbols. The coincidence lemmas say that expressions depend only on their signature and free variables. We extend coincidence for terms and formulas with new cases for hybrid-logical constructs:}

\textbf{Lemma 1 (Coincidence).}

1. If \(\omega, g = \widetilde{\omega}, h \text{ on } FV(\theta)\) and \(I = J \text{ on } \Sigma(\theta)\), then \(\llbracket \sigma(\theta) \rrbracket \omega \theta = \llbracket \sigma(\theta) \rrbracket \widetilde{\omega} \theta \rrbracket \).
2. If \(\omega, g = \widetilde{\omega}, h \text{ on } FV(\phi)\) and \(I = J \text{ on } \Sigma(\phi)\), then \(\omega \in \sigma(\phi) \text{ iff } \widetilde{\omega} \in \sigma(\phi)\).
3. If \(\omega, g = \widetilde{\omega}, h \text{ on } V \supseteq FV(\alpha)\) and \(I = J \text{ on } \Sigma(\alpha)\) and \(\omega, \nu \in \sigma(\alpha) \text{ then exists } \nu \text{ such that } \sigma(\nu, \nu) \in \sigma(\alpha) \text{ and } \omega, \nu \in \sigma(\phi)\).

\textbf{Theorem 2 (dHL Soundness). All dHL rules are sound and all axioms valid, thus all provable formulas are valid.}

\textit{Proof.} (In Appendix D.) Soundness of US is proven inductively, appealing to Lemma 1. Validity of axioms is by direct proof. \(\square\)

6.2 Reducibility

We compare the expressive power of dHL to that of dL. The comparison is surprisingly subtle, and finds that while dHL is reducible to dL, its specialized hybrid-logical rules make direct proof in dHL preferable for practical purposes. The core idea is to emulate each world variable from dHL with a finite number of program variables in dL, resulting in an equivalent, finite dL formula. The complication is that a dHL world contains infinitely many variables:

1. Infinite worlds ensure every state always contains a fresh variable \(r\). Without freshness, the semantics would be undesirable; consider \(\phi \equiv \forall s: \mathcal{W}(x := s): \text{This should be invalid; e.g. program } x := * \text{ cannot transition from } \omega = \{x \mapsto 0, r \mapsto 1\} \text{ to } (\nu \subset\{x \mapsto 0, r \mapsto 2\}). \text{ However, allowing arbitrary finite state domains (specifically } \mathcal{V} \equiv \{x\}) \text{ would make this valid.}
2. Program constants (and predications) depend on infinitely many variables. This is because program constants stand for arbitrary programs, which should naturally be able to use
arbitrary variables in their implementation, of which there are infinitely many.

We show that for formulas without program constants and predications (called concrete formulas), infinite worlds are no obstacle, while for formulas with program constants (called abstract formulas) they are. Concrete formulas are reducible because even though we demand each state contains fresh variables, it suffices to use a single fresh variable \( r \) (as in example \( \phi \) above) instead of infinitely many. To make this claim formally, we introduce a notion of finite domains for states, galaxies and interpretations.

**Definition 10** (Finite states). We say a state \( \omega \) has finite-domain \( S \subseteq \mathcal{V} \) if \( S \) is finite and \( \omega(x) = 0 \) for all \( x \notin S \), with galaxies and interpretations analogous. A formula \( \phi \) is finite-domain valid with domain \( S \) if \( \omega \in \text{[[}\phi\text{]]}_S \) for all \( \omega, l, g \) with finite domain \( S \).

We outline the reduction proof here, with proofs in Appendix B. The proof proceeds by showing that validity and finite-domain validity agree for each of concrete \( dHL \) and concrete \( dl \), and that our reduction preserves finite-domain validity. Thus our reduction preserves validity for concrete formulas. The converse also holds because \( dl \) is a fragment of \( dHL \).

**Lemma 3** (Finitization). Let \( \phi \) be a concrete \( dHL \) formula. Let \( r \notin \text{FV}(\phi) \cup \text{BV}(\phi) \) (where \( \text{BV}(\phi) \) is the set of variables bound in \( \phi \)). Then \( \phi \) is valid iff \( \phi \) is finite-domain valid with domain \( \{ r \} \cup \)(\( \text{FV}(\phi) \cup \text{BV}(\phi) \)).

**Lemma 4** (Finite-Domain Reducibility). There exists a computable reduction \( T(\phi) \) such every \( dHL \) formula \( \phi \) is finite-domain valid iff the \( dl \) formula \( T(\phi) \) is finite-domain valid.

**Proof.** We present the translation \( T(\phi) \). In this translation we write \( \tilde{x} \) for the vector of all variables \( x_1, \ldots, x_n \) in the domain \( S \), and \( e \tilde{x} \) for the vectorial substitution of all \( \tilde{x} \) for the corresponding \( x_i \) in \( e \). For each of the finitely-many world terms \( \omega \) in \( \phi \), let \( \tilde{x}^\omega \) be a vector of \( |S| \) fresh symbols implementing \( @_w \tilde{x} \).

\[
T(@_w \phi) = [\tilde{x} := \tilde{x}^\omega]T(\phi) \quad (28)
\]

\[
T(@_w \theta) = T(\theta)\tilde{x}^\omega \quad (29)
\]

\[
T(w) = (\tilde{x} = \tilde{x}^\omega) \quad (30)
\]

\[
T(V \psi : W \phi) = \forall \tilde{x}^W : R T(\phi) \quad (31)
\]

\[
T(\exists x : W \phi) = \exists \tilde{x}^W : R T(\phi) \quad (32)
\]

\[
T(\land \psi) = [\tilde{x}^W := \tilde{x}^\omega]T(\phi) \quad (33)
\]

\[
T(\bigoplus (e_1, \ldots, e_2)) = \bigoplus (T(e_1), \ldots, T(e_2)) \quad (34)
\]

In Equation 34, the notation \( \bigoplus (e_1, \ldots, e_2) \) stands for an arbitrary \( dl \) connective. In Equation 30, vectorial equality \( \tilde{x} = \tilde{y} \) is shorthand for the finite conjunction \( \bigwedge_i x_i = y_i \). The result proves by induction. \( \Box \)

**Lemma 5** (De-finitization). A concrete \( dl \) formula \( \phi \) is valid iff it is finite-domain valid over domain \( \text{FV}(\phi) \cup \text{BV}(\phi) \).

**Theorem 6** (Concrete Reducibility). Concrete \( dHL \) (i.e. with no rigid symbols) reduces to \( dl \).

**Theorem 7** (\( dHL \) contains \( dl \)). For all \( dl \) formulas \( \phi \), all galaxies \( g \), and all states \( \omega \), we have \( \omega \in \text{[[}\phi\text{]]}_g \) in \( dHL \) iff \( \omega \in \text{[[}\phi\text{]]}_g \) in \( dl \).

We note these theorems do not entail (infinite) validity reduction for abstract \( dHL \) formulas. Lemma 4 does however preserve finite validity even in the presence of abstract formulas. Phrased differently, reduction fails if and only if abstract constants are allowed to introduce arbitrary new variables.

**Corollary 8** (Relative Semi-Decidability). Concrete \( dHL \) is semi-decidable relative to properties of differential equations.

**Proof.** By relative semi-decidability [24] of concrete \( dl \) and Theorem 6. \( \Box \)

**Why \( dHL ?** What are the practical implications of the reducibility results? While a reduction exists, it blows up theorems and proofs by the size of the variable domain. In the abstract case, the natural variable domain is infinite for compositionality reasons, and we obtain a reduction only for finite domains. This means verification by reduction is especially ill-suited for proofs using advanced proof techniques like refinement [18] which rely on program constants. Additionally, most \( dHL \) axioms use these constants, and thus cannot be translated to concrete \( dl \) axioms! Even on concrete formulas, this blowup is significant (quadratic) and obscures the more convenient proof techniques available in \( dHL \). Thus, direct proof in \( dHL \) is strongly preferable to \( dl \) reduction for practical purposes.

7 Derived Rules for Bisimulation

The proof calculus of Section 5 provides a hybrid-logical core for hyper-property verification. In moving toward our smart-grid information flow example, we build a small library of derived rules for information-flow proofs which, being derived, lie outside the core calculus. Our derived rules show that bisimulation, the core proof technique for information flow, derives from nominals in hybrid logic. Because information flow arguments specifically equate values from initial and ending states, we also derive rules for equalities over alternating terms. As we will see in Sections 8 and 9, these derived rules raise the level of abstraction. Derivations are given in Appendix D.

**Bisimulation Rules** In Figure 6, \( R \) is a relation over world expressions, i.e. \( R(i_1, i_2) \) means that worlds \( i_1 \) and \( i_2 \) are related in \( R \). Rule \( @Ind \) is an auxiliary rule for loop induction with nominals, derived from loop induction axiom I. It is in turn used to derive Rule BS*, which says any relation \( R \) is a bisimulation for loop \( \alpha \) any time it is (in every state) a bisimulation for \( \alpha \). Rule BS is Hoare-style composition reasoning raised to the bisimulation level: we can reason about \( \alpha; R \) by establishing a relation \( R_m \) that holds in the intermediate state. Rule BSU says a nondeterministic choice maintains a bisimulation if each branch does. In rule BS*, ODE is a differential equation of form \( x^\prime = f(x), x^\prime = g(x) \) (any model can be trivially extended to this form by adding a fresh variable \( t \) and ASGN \( \equiv (t \leftarrow o_t, x := y(t \leftarrow o_t)) \) implements the solution of ODE, plugging in the same duration \( o_t \) at \( x \). Rule NT\( V \) says a term \( \theta \) is unchanged if its variables never appear bound in \( \alpha \). The remaining rules are derived from the program axioms and capture the effect of each program on a term. In rule NT\( V \), \( y(t) \) is a global solution to \( x^\prime = f(x) \).

**At-Terms** Figure 7 derives rules for at-term equalities. Rule NTV says a term \( \theta \) is unchanged if its variables never appear bound in \( \alpha \). The remaining rules are derived from the program axioms and capture the effect of each program on a term. In rule NT\( V \), \( y(t) \) is a global solution to \( x^\prime = f(x) \).

8 FREEDM: Proving Vulnerability Existence

We now prove that the naive deterministic controller \( bat_1 \) for FREEDM based on the published algorithm [2] is insecure: our quantitative dynamical model reveals a bug obscured by the finite
event-based abstraction in previous models [3]. Information leaks because when \( G > 0 \) we can infer some battery \( B_i \) must be full, in which case we have leaked the information \( B_i = B_{\text{max}} \), which might indicate a battery is vulnerable to attack.

To prove that a system is non-deducibility-secure, we prove the negation of non-deducibility security, i.e. we prove:

**Proposition 9** (bat1 is insecure). The formula \( \exists i_1, i_2, o_1 : W \left( R(i_1, i_2) \land \neg R(o_1, o_2) \right) \) is valid, where \( R(i, j) \equiv \neg \theta \) and \( \theta \) is the insecure version of the grid \( G_F \).

**Proof.** We give the proof outline here and the full proof in Appendix F. First we construct the states \( i_1, i_2, o_1 \). Let \( i \) be an arbitrary state. Let \( i_1, i_2 \) be the unique states such that \( @i(B_i := B_{\text{max}}; t := 0; G := 0)i_1 \) and \( @i(\neg B_i := B_{\text{max}}/2; t := 0; G := 0)i_2 \). Let \( o_1 \) be any state such that \( @o_1(\alpha_1) \land @o_1(\theta) \) and \( @o_1(\theta) \) is valid over \( G = 0 \). We know such a state exists by running \( \theta \) for exactly one iteration, setting \( R = \text{max}(0, -P_i) \) and \( D_i = 1 + \max(0, P_i) \) which always results in \( N_i = 1 \). Thus after evolving the differential equation for time 0 we arrive at \( G > 0 \). By induction we show that all traces of \( \alpha_1 \) maintain the invariant \( J \equiv (t \geq 0 \land G > 0) \) can be proven by mechanically applying program axioms, then checking first-order real arithmetic at the leaves. Lastly, note \( @o_1(\theta) \) by real arithmetic, and the proof is complete. □

9  FREEDM: Ensuring and Proving Security

We learned that bat1 leaks \( B_i = B_{\text{max}} \), ultimately because it is too deterministic: If \( G > 0 \) we learn for a fact some \( B_i = B_{\text{max}} \). The simplest solution, as taken in bat5 of Figure 2, is to add the non-deterministic option to use the macrogrid even when a battery has capacity. Now that the macrogrid is always an option, an attacker who observes \( G > 0 \) cannot infer \( B_i = B_{\text{max}} \) for certain.

We now prove bat5 non-deducibility secure, i.e. an attacker observing only \( G \) and \( t \) deduces nothing else. We instantiate Definition 5 to arrive at the theorem statement:

**Proposition 10** (Non-deducibility for FREEDM). The dHL formula

\[ \forall i_1, i_2, o_1 : W \left( @i_1(\alpha_1) \land R(i_1, i_2) \land @i_1.pre \land @i_2.pre \rightarrow @i_2(\alpha_2) \land R(o_1, o_2) \right) \]

is valid, where we define \( \text{pre} \equiv M > 0 \land B_{\text{max}} \land \text{Thresh} > 0 \) and \( R(i, j) \equiv \neg \theta \). This shows the secure version of the grid \( G_F \).

**Proof.** The full proof is given in Appendix E; we give an outline here. Recall from Section 3 that the heart of the proof is choosing a trace \( @i_1(\alpha_1)\alpha_2 \) which shows the public outputs \( o_1, G \) and \( o_2, t \) of trace

\[ @i_2(\alpha_2) \]

are possible from every related input state \( i_2 \). The proof is by loop bisimulation (rule BS\3) with the relation \( R(i, j) \equiv \neg \theta \). The key proof observation is that for the final values of \( G \) to agree, it suffices that the values agree for both \( G \) and \( G_m \) at the start of the ODE. We perform this reasoning formally using the bisimulation composition rule BS; with the relation \( R_p(i, j) \equiv R(i, j) \land @G = @G \lor @t = @t \). This gives two proof obligations: one for the control and one for the physics. We split into four cases for the controller using Lemma 11.

**Lemma 11** (Control Inversion). The dHL formula \( @m(B_m = G \land m = 0) \rightarrow @m(G = 0 \land G_m = 0) \lor (G = N_i \land G_m = m) \lor (G = N_i \land G_m = m) \lor (G = N_i + N_j = 0) \) is valid, where \( ctrl \) is as in Figure 2.

The second case is representative, the rest are in Appendix E.

**Case.** \( @m(B_m = G \land m = 0) \) by inspection, suffice to set \( N_i = @m, N_j = 0, m = 0 \). If \( @m_1 N_i = @m, P_1 \geq 0 \) then set \( R_1 = @m_1 N_i - @m_1 P_1 \) and \( D_1 = 0 \). Else set \( R_1 = 0, D_1 = @m_1 P_1 + @m_1 N_i \). By algebra, either case \( @m_1 N_i = @m_1 N_i \). Then for \( i = 2 \) if \( @m_2 P_2 \geq 0 \) then set \( R_2 = @m_2 P_2 \) and \( D_2 = 0 \) else set \( R_2 = 0, D_2 = @m_2 P_2 \). Then by algebra \( @m_2 N_2 = 0 \). Executing the load balancer we get \( m = 0 \) because \( N_1 = 0 \) (thus \( T_2 \) is Normal). We take the second branch of the battery controller both times and get \( G = N_1 + N_2 = N_1 \) and \( G = m = -1^2 + m = -1^2 = 0 \). Desired.

\[ \square \]

**Implications for Implementation.** We have shown how to use dHL to identify and resolve non-deducibility information flow vulnerabilities in hybrid systems models of CPs. We reflect on how verified safety of a model can contribute to the safety of real-world implementations as well.

Our first model was insecure because a deterministic branch in the battery controller leaked information. This was fixed by introducing a nondeterministic branch. Implementations can simulate the same effect, e.g. with randomized branching. Our models taught
us, for example, that the battery controller needs such measures, but the load balancing controller, in contrast, is secure even with deterministic control. This knowledge is helpful in practice because measures like randomization typically reduce operational efficiency of a controller, so verifying that a deterministic controller is secure enables using the efficient deterministic controller with confidence.

In comparison with many other formal security models, our approach is especially well-suited to verifying security in the presence of side-channels. Many would-be side-channels for cyber systems (time, electrical flow, etc.) are primary channels in CPS and, as shown in our models, are modeled naturally as hybrid systems. Thus, these channels and more can be incorporated without fundamentally changing the verification approach — they need only be added to the hybrid system model!

10 Related Work

**Dynamic Logics and Hybrid Logics** The logic dHL is a hybrid version of the dynamic logic dL, adding the ability to verify hyper-properties in addition to safety and liveness properties. The logic dL_0 [23] is an early proposal for a hybrid version of dL, but lacks world quantifiers and at-terms, which are essential for information flow. Dynamic logic and first-order hybrid logic have been combined in Combinatory PDL [20], which extends Propositional Dynamic Logic (PDL) with additional set-theoretic program combinators, but has neither at-terms nor differential equations. Hybrid logic has been used in distributed systems [22] reasoning as well. While many CPSs are distributed systems, distributed systems reasoning alone does not suffice to verify hybrid discrete and continuous dynamics. The logic QdL [25] allows verification of distributed hybrid dynamics, but is not a hybrid logic, and faces the same challenges with hyper-properties as dL does. First-order hybrid logic [6] (without dynamic-logical or continuous features) and its proof theory [8] have been studied in detail.

**Static Information Flow Security** Logics and type systems for information-flow security have been widely studied for discrete programs. Sebelfeld and Myers [30] provide a survey of language-based security approaches. Approaches can be broadly categorized into automatic vs. interactive (or manual) approaches. Automation increases the potential user base, typically at the cost of greatly reduced completeness. When the proof is done automatically, the simplicity of the proof is of little practical concern, and the self-composition [5] approach can be used to reduce information-flow proofs to a safety property suitable for Hoare and dynamic logics.

For interactive use, self-composition has been noted [33] to make proofs awkward because it reduces locality: bisimulation techniques consider the local effect of each statement σ on two traces, but in a self-composed program, the original statement σ may be far away from its copy. For a human user, a usable calculus as provided by dHL is far more important. As our smart-grid example demonstrates, typical CPS information flow properties rely on fine-grained interactions between discrete computation and differential equations within system loops, suggesting that automated approaches would struggle, meriting our approach, which is amenable to interactive proof. Scheben and Schmitt [32] have implemented an approach for Java dynamic logic in the theorem prover KeY (which supports both automatic and interactive proof) analogous to proof by reducing concrete dHL to dL. Their logic does not support nominals. Due to the awkwardness of reductions, calculi meant for interactive use, such as Relational Hoare Type Theory (RHTT) [19], typically do not reduce information-flow to a safety property, but build special-purpose information flow predicates into the calculus. The disadvantage of such calculi is that baking in these predicates prevents generalizing to other hyper-properties.

We strike a middle ground with dHL: The proof techniques we would expect of a dedicated calculus are easily implemented as derived rules, yet we maintain the generality to express arbitrary safety and liveness properties and hyper-properties as well. In the process, we identify a previously-unexplored connection between hybrid logic and hyper-properties.

While we present the first information flow result we know of for a hybrid system model of a CPS, information flow has been verified for other models via model-checking; e.g. Akella [3] has verified process algebra models of FREEDM and Wang [34] has verified a Petri-net model of a pipeline network. In the FREEDM example, our detailed dynamics revealed a bug previously obscured by the finite abstractions in Akella [3]. More generally, the crucial advantages of verifying a hybrid model are (a) that we show security even against attackers who make continuous observations (b) direct proofs of a hybrid model are typically simpler than proofs after discretization.

11 Conclusion and Future Work

We introduced dHL, a hybrid logic for verifying information-flow security properties of hybrid dynamical systems in order to ensure the security of critical cyber-physical systems (CPS). In contrast with previous approaches, it allows verifying cyber-physical hybrid information flows, communicating information through both discrete computation and physical dynamics, so security is ensured even when an attacker can observe continuously-changing values in continuous time. It achieves this by combining dL, a logic for hybrid dynamical systems, with hybrid-logical features enabling explicit reference to program states. This provides a novel way to verify information flow: information flow properties are hyper-properties, which are expressed naturally in hybrid logic via its ability to refer freely to states from multiple traces simultaneously. The foundation of hybrid logic allows falsification and verification of security in a common system at no added complexity, and we expect the same system can support additional notions of information flow such as non-interference, as well as arbitrary hyper-properties. We introduced a calculus for dHL, proved it sound, and derived high-level bisimulation rules for information flow proofs. Our choice of proof calculus provides a clear path for implementing dHL in the theorem-prover KeYmaera X.

We showed that dHL is capable of verifying the presence or absence of information-flow vulnerabilities in realistic hybrid models. As an example, we debugged and then verified a hybrid model of the FREEDM [16] smart-grid controller based on the published algorithm [2] with load-balancing and distributed energy generation and storage, all important features for practical grids. Moreover, the proof demonstrates both (a) the close correspondence between dHL information-flow proofs and natural-language proofs and (b) the non-trivial proof arguments that quickly arise when mixing cyber and physical dynamics. The main places where dHL proofs require more effort than a paper proof were in introducing names for intermediate states and observing the effect of a program on an individual term. We wish to design a proof format in the KeYmaera X implementation that allows us to automate the majority
of these low-level steps, making proofs efficiently match to human intuition. Our information-flow arguments depend closely on the exact semantics of the program and do not follow from, e.g. simple syntactic checks on variable dependencies, meaning the expressive power provided by a deductive calculus is essential for verifying realistic CPS information flow problems. A major novelty in both the logic dHl and our model of FREEDM is the presence of hybrid information flows that mix discrete cyber and continuous physical channels. These cyber-physical flows arise naturally in many other critical applications, such as oil and natural gas networks, canals, smart homes, medical devices, and vehicles, which deserve exploration in the future.

References


\[
\begin{align*}
  \text{FV}(c) &= \{\} & (35) \\
  \text{FV}(x) &= \{x\} & (36) \\
  \text{FV}(\theta_1 + \theta_2) &= \text{FV}(\theta_1) \cup \text{FV}(\theta_2) & (37) \\
  \text{FV}(\theta_1 \cdot \theta_2) &= \text{FV}(\theta_1) \cup \text{FV}(\theta_2) & (38) \\
  \text{FV}(f(\theta)) &= \text{FV}(\theta) & (39) \\
  \text{FV}(f(w)) &= \text{FV}(w) & (40) \\
  \text{FV}(F) &= \mathcal{V}_R \cup \mathcal{V}_W & (41) \\
  \text{FV}(\phi \land \psi) &= \text{FV}(\phi) \cup \text{FV}(\psi) & (42) \\
  \text{FV}(\neg \phi) &= \text{FV}(\phi) & (43) \\
  \text{FV}(\exists x: \mathbb{R} \phi) &= \text{FV}(\phi) \setminus \{x\} & (44) \\
  \text{FV}(\theta_1 \geq \theta_2) &= \text{FV}(\theta_1) \cup \text{FV}(\theta_2) & (45) \\
  \text{FV}(\langle \alpha \rangle \phi) &= \text{FV}(\alpha) \cup (\text{FV}(\phi) \setminus \text{MBV}(\alpha)) & (46) \\
  \text{FV}(\exists s: \mathcal{W} \phi, \forall s: \mathcal{W} \phi) &= \text{FV}(\phi) \setminus \{s\} & (47) \\
  \text{FV}(\exists s \phi) &= (\text{FV}(\phi) \cup \mathcal{V}_R) \setminus \{s\} & (48) \\
  \text{FV}(\langle \alpha \rangle \phi) &= \text{FV}(\langle \alpha \rangle \theta) = \text{FV}(\phi \text{ or } \theta) |_W \cup \{s\} & (49) \\
  \text{FV}(\langle \alpha \rangle \phi) &= \text{FV}(\langle \alpha \rangle \theta) = \text{FV}(\phi \text{ or } \theta) |_W & (50) \\
  \text{FV}(s) &= \mathcal{V}_R \cup \{s\} & (51) \\
  \text{FV}(\bar{\phi}) &= \mathcal{V}_R & (52) \\
  \text{FV}(\phi) &= \text{FV}(\phi) & (53) \\
  \text{FV}(\langle \phi \rangle) &= \text{FV}(\phi) & (54) \\
  \text{FV}(x := \theta) &= \text{FV}(\theta) & (55) \\
  \text{FV}(x := \ast) &= \{\} & (56) \\
  \text{FV}(x' = \theta \land \psi) &= \text{FV}(\theta) \cup \text{FV}(H) & (57) \\
  \text{FV}(\alpha \cup \beta) &= \text{FV}(\alpha) \cup \text{FV}(\beta) & (58) \\
  \text{FV}(\alpha; \beta) &= \text{FV}(\alpha) \cup (\text{FV}(\beta) \setminus \text{MBV}(\alpha)) & (59) \\
  \text{FV}(\alpha^*) &= \text{FV}(\alpha) & (60) \\
  \text{FV}(\phi) &= \mathcal{V}_R \cup \mathcal{V}_W & (61)
\end{align*}
\]

**Figure 8.** Free variable computation

A Uniform Substitution Algorithm

We give the complete presentation of the uniform substitution algorithm, i.e.

- The FV(e) function computing flexible symbols which can influence expression e
- The BV(α) function computing flexibles which can change during program α (unchanged from prior work)
- The MBV(α) function computing flexibles which are necessarily bound on all execution paths of α
- The signature Σ(ε) of rigid symbols in expression ε.
- The substitution algorithm σ(ε) proper.

Here \(U_R\) and \(U_W\) denote the restriction of set \(U\) to only program variables or world variables, respectively. Equations (35-46) are as in previous work [28]. In Equation 46, MBV(α) is the set of variables that are bound on all executions of α. Equation 47 says quantifiers remove the quantified world variable from the free variable set because references to \(s\) in \(\phi\) refer to the value bound by the quantifier. Equations 49 and 50 say the only free variables are the world variables of \(\theta\) or \(\phi\) (and \(s\) in the world variable case). It may be surprising that the free program variables of \(\phi\) and \(\theta\) make no appearance. The reason is this: program variable references \(x\) within \@wϕ or \@wθ refer to \@wxs, which is encapsulated by the single dependency on \(s^4\), or to \@wxs which is rigid due to the rigidity of \(s\) and thus incurs no dependencies on

---

*One might be tempted to increase the precision of admissibility by distinguishing, e.g. dependency on \@wxs from \@wxs. This would have no benefit because all *binders* of states bind them in their entirety, in which case introducing free reference to any \@wxs violates admissibility. We thus lose nothing by using the simpler dependency on \(s^4\).*
\[\Sigma(\pi \phi) = \Sigma(\pi \theta) = \Sigma(\phi \text{ or } \theta) \cup \{\pi\}\]
\[\Sigma(\exists s, \phi) = \Sigma(\exists s, \theta) = \Sigma(\phi \text{ or } \theta)\]
\[\Sigma(\exists x : \exists W, \phi) = \Sigma(\exists x : \exists W, \phi) = \Sigma(\exists x \phi) = \Sigma(\phi)\]
\[\Sigma(sym \in f, F, p, P) = \{sym\}\]
\[\Sigma(\exists(e_1, \ldots, e_n)) = \Sigma(e_1) \cup \cdots \cup \Sigma(e_n)\]

Figure 9. Signature computation (sym is an arbitrary rigid)

a flexible symbol such as \(x\).

\[
\begin{align*}
BV(?\phi) &= \{\} \\
BV(x := \theta) &= \{x\} \\
BV(x := \ast) &= \{x\} \\
BV(x' = \theta & \psi) &= \{x, x'\} \\
BV(\alpha \cup \beta) &= BV(\alpha) \cup BV(\beta) \\
BV(\alpha; \beta) &= BV(\alpha) \cup BV(\beta) \\
BV(\alpha^+) &= BV(\alpha) \\
BV(\alpha) &= V_{\forall} \cup V_{\mathbb{R}} \\
MBV(\alpha \cup \beta) &= MBV(\alpha) \cap MBV(\beta) \\
MBV(\alpha; \beta) &= MBV(\alpha) \cup MBV(\beta) \\
MBV(\alpha) &= MBV(\alpha^+) = \{\} \\
MBV(\alpha) &= BV(\alpha)
\end{align*}
\]

Analogously to \(FV(e)\), the signature \(\Sigma(e)\) indicates all rigid symbols which influence the meaning of \(e\). Note that since rigid symbols, by definition, are not bound by the \(\@\theta\) modality, they are always counted in the signature:

Admissibility conditions are checked recursively during the substitution algorithm proper (Figure 10). These checks use an auxiliary notion called \(U\)-admissibility:

**Definition 11** (\(U\)-admissibility). We say a substitution \(\sigma\) is \(U\)-admissible for an expression \(e\) with a flexible symbol set \(U \cup \bigcup_{sym \in \Sigma(e)} FV(\sigma, sym)\) where \(\sigma|_{\Sigma(e)}\) is the restriction of \(\sigma\) that replaces only symbols occurring in \(e\).

This makes the admissibility conditions as expressed in the main paper precise. Note also in Figure 10 that the symbol \(\cdot\) is a reserved function (or nominal) symbol standing for the argument.
<table>
<thead>
<tr>
<th>Case (α) = ε</th>
<th>(61)</th>
</tr>
</thead>
<tbody>
<tr>
<td>σ(x) = x</td>
<td>(62)</td>
</tr>
<tr>
<td>σ(θ₁ + θ₂) = σ(θ₁) + σ(θ₂)</td>
<td>(63)</td>
</tr>
<tr>
<td>σ(θ₁ · θ₂) = σ(θ₁) · σ(θ₂)</td>
<td>(64)</td>
</tr>
<tr>
<td>σ(f(θ)) = {· → σ(θ)}(σf), f ∈ σ</td>
<td>(65)</td>
</tr>
<tr>
<td>σ(f(θ)) = f(σ(θ)), f ∉ σ</td>
<td>(66)</td>
</tr>
<tr>
<td>σ(f(ω)) = {· → σ(ω)}(σf), f ∈ σ</td>
<td>(67)</td>
</tr>
<tr>
<td>σ(f(ω)) = f(σ(ω)), f ∉ σ</td>
<td>(68)</td>
</tr>
<tr>
<td>σ(F) = σF, F ∈ σ</td>
<td>(69)</td>
</tr>
<tr>
<td>σ(F) = F, F ∉ σ</td>
<td>(70)</td>
</tr>
<tr>
<td>σ(a) = a, a ∈ σ</td>
<td>(71)</td>
</tr>
<tr>
<td>σ(α) = α</td>
<td>(72)</td>
</tr>
<tr>
<td>σ(x := θ) = x := σ(θ)</td>
<td>(73)</td>
</tr>
<tr>
<td>σ(x := α) = x := α</td>
<td>(74)</td>
</tr>
<tr>
<td>σ(?ϕ) = σ(ϕ)</td>
<td>(75)</td>
</tr>
<tr>
<td>σ(x = x') = σ(x) &amp; σ(ϕ)</td>
<td>(76)</td>
</tr>
<tr>
<td>(a)β = σ(a); σ(β)</td>
<td>(77)</td>
</tr>
<tr>
<td>σ(α ∪ β) = σ(α) ∪ σ(β)</td>
<td>(78)</td>
</tr>
<tr>
<td>σ(α') = σ(α)'</td>
<td>(79)</td>
</tr>
<tr>
<td>σ(θ₁ ≥ θ₂) = σ(θ₁) ≥ σ(θ₂)</td>
<td>(80)</td>
</tr>
<tr>
<td>σ(p(θ)) = {· → σ(θ)}(σp), p ∈ σ</td>
<td>(81)</td>
</tr>
<tr>
<td>σ(p(θ)) = p(σ(θ)), p ∉ σ</td>
<td>(82)</td>
</tr>
<tr>
<td>σ(p(ω)) = {· → σ(ω)}(σp), p ∈ σ</td>
<td>(83)</td>
</tr>
<tr>
<td>σ(p(ω)) = p(σ(ω)), p ∉ σ</td>
<td>(84)</td>
</tr>
<tr>
<td>σ(σF) = σF, p ∈ σ</td>
<td>(85)</td>
</tr>
<tr>
<td>σ(σF) = F, p ∉ σ</td>
<td>(86)</td>
</tr>
<tr>
<td>σ(¬ϕ) = ¬σ(ϕ)</td>
<td>(87)</td>
</tr>
<tr>
<td>σ(ϕ ∧ ψ) = σ(ϕ) ∧ σ(ψ)</td>
<td>(88)</td>
</tr>
<tr>
<td>σ(∃x : ϕ) = ∃x : σ(ϕ)</td>
<td>(89)</td>
</tr>
<tr>
<td>σ(ϕ) = (σ(ϕ))</td>
<td>(90)</td>
</tr>
<tr>
<td>σ(κϕ) = κ(σ(ϕ))</td>
<td>(91)</td>
</tr>
<tr>
<td>σ(κϕ) = κ(σ(ϕ))</td>
<td>(92)</td>
</tr>
<tr>
<td>σ(κϕ) = κ(σ(ϕ))</td>
<td>(93)</td>
</tr>
<tr>
<td>σ(κϕ) = κ(σ(ϕ))</td>
<td>(94)</td>
</tr>
<tr>
<td>σ(κϕ) = κ(σ(ϕ))</td>
<td>(95)</td>
</tr>
<tr>
<td>σ(κϕ) = κ(σ(ϕ))</td>
<td>(96)</td>
</tr>
<tr>
<td>σ(κϕ) = κ(σ(ϕ))</td>
<td>(97)</td>
</tr>
<tr>
<td>σ(κϕ) = κ(σ(ϕ))</td>
<td>(98)</td>
</tr>
</tbody>
</table>

**Figure 10.** Uniform Substitution Algorithm
B Reducibility Proofs

We begin with the inclusion of $\mathit{dL}$ into $\mathit{dHL}$. This is intuitively obvious because $\mathit{dL}$ is a fragment of $\mathit{dHL}$, but the technical details are finicky. The main finicky detail is that states and interpretations in $\mathit{dHL}$ are larger than in $\mathit{dL}$. We write $\omega_h = \omega_d \cup S$ to say that $\mathit{dL}$ state $\omega_h$ is an extension of the $\mathit{dL}$ state $\omega_d$, where the set $S$ contains all the additional mappings of $\omega_h$. Likewise we say $I_d \subseteq I_h$ when $I_d$ is an extension of $I_h$. The $\subseteq$ relation is not quite the subset relation but closely related. We write $D(\omega_h)$ for the $\mathit{dL}$ part of a state and $H(\omega_h)$ for the hybrid part. It holds when:

- $I_d(\omega_d, \omega_h) = (D(\omega_h), D(\omega_d)) \in I_d(a) \land H(\omega_h) = H(\omega_d)$
- $I_d(f) = I_h(f)(r) = I_d(f)(r)$
- $I_d(\omega_d, \omega_h) = \omega_h = \omega_d \cup S \ni f h(f)(r)$
- $I_d(\omega_d, \omega_h) = \omega_h = \omega_d \cup S \ni f h(p)(r)$
- $I_d(\omega_d, \omega_h) = \omega_h = \omega_d \cup S \ni f h(P)(\omega_d) = I_h(P)(\omega_d)$

We now establish a series of lemmas leading up to the reduction.

**Lemma 12** (Surjectivity). For all $\omega_h, I_h$ exist $\omega_d, S, I_d \subseteq I_h$ such that $\omega_h = \omega_d \cup S$ and $I_d \subseteq I_h$.

*Proof.* By straightforward construction, let $\omega_d = D(\omega_h), S = \omega_h \setminus I_d$, $I_d = D(I_h)$.

**Lemma 13** (Term inclusion). For all $\omega_h = \omega_d \cup S$ and $I_d \subseteq I_h$ and all $\omega, \theta \vdash \omega_d \models \theta$, $\theta \vdash \omega_h$.

*Proof.* By induction on the term $\theta$.

- **Case** $c \in C : [c] \models \omega_d \models c = [c] \models \omega_h I_h$.
- **Case** $x \vdash \theta : [x] \models \omega_d \models x = \omega_h(x) = [x] \models \omega_h I_h$ because $\omega_d \subseteq \omega_h$.
- **Case** $I_d = I_h \models [\theta] \models \omega_d \models I_d = I_h \models [\theta] \models \omega_h I_h$.
- **Case** $I_d = I_h \models \omega_d \models I_d = I_h \models \omega_h I_h$.
- **Case** $h(f) = h(f)(r)$.
- **Case** $h(f)(r) = h(f)(r)$.
- **Case** $h(P)(\omega_d) = I_h(P)(\omega_d)$.

**Lemma 14** (Program inclusion). For all $\omega_d, \omega_h, S, I_d, I_h$ if $(\omega_d, \omega_h) \in [\alpha] I_d$ and $I_d \subseteq I_h$ then $(\omega_h, \omega_h) \in [\alpha] I_h$.

*Proof.* By induction on $\alpha$, in simultaneous induction with Lemma 15.

- **Case** $x : \theta : [x] \models \theta \models \omega_d \models I_d$ implies $\omega_d \models [x] \models \omega_d$. By Lemma 13. By semantics, $[\omega_d, \omega_h, x \models \theta \models \omega_h] \in [x : \theta \models \omega_h].$ Then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $x : \theta : [x] \models \theta \models \omega_d \models I_d$ implies $\omega_d \models [x] \models \omega_d$. By semantics, $[\omega_d, \omega_h, x \models \theta \models \omega_h] \in [x : \theta \models \omega_h], x \models \theta \models \omega_h$, completing the case.
- **Case** $h(f) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(P) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(f) \models [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(P) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(f) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(P) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(f) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(P) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(f) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(P) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(f) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(P) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(f) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.
- **Case** $h(P) : [\theta] \models \omega_d \models I_d$ implies $\omega_d \models [\theta] \models \omega_d$. By then note by assumptions and set arithmetic, $\omega_h \models [\theta] \models \omega_h$, completing the case.

Having completed the lemmas we can complete the main proof of reducibility:

**Theorem 16** ($\mathit{dHL}$ contains $\mathit{dL}$). For all $\phi \in \mathit{dL}$, $\phi$ is valid in $\mathit{dL}$ iff $\phi$ is valid in $\mathit{dHL}$.

*Proof.* To show $\phi$ is valid in $\mathit{dHL}$, fix an interpretation $I_h$ and state $\omega_h$ and show $I_h \models [\phi] \models \omega_h$. By Lemma 12, exist $I_d$ and $\omega_d$ such that $I_d \subseteq I_h$ and $I_d = I_h \cup S$. By validity of $\phi$ in $\mathit{dL}$, have $I_d \models [\phi] \models \omega_d$. Then by Lemma 15, $I_h \models [\phi] \models \omega_h$. 

Brandon Bohrer and André Platzer
C Concrete Reducibility

There are two intuitions behind the concrete reduction:

1. World variables can be simulated with program variables, with nominal constants likewise simulated by constant functions.
2. Such a simulation can be done finitely despite the infinity of states because (a) dl constructs see only explicitly-mentioned variables and (b) while hybrid constructs see the unmentioned variables, they see no difference between finitely or infinitely-many unmentioned variables.

Point 2 is shown formally by showing that in both dl and dHL, validity for concrete formulas agrees with finite-domain validity where states, galaxies and interpretations are non-zero on only finitely many variables. Then point 1 is shown by induction because the translation preserves finite-domain validity.

In what follows we fix arbitrary bijections to \( N \) and \( (\mathbb{N}^N)^{|N|} \) respectively. We also let \( r \) always refer some canonical variable that is fresh in the translated formula \( \phi \). Furthermore \( V(\phi) = FV(\phi) \cup BV(\phi) \) refers to all variables mentioned in \( \phi \). As in Appendix A we use \( S_R \) for the restriction of flexible symbol set \( S \) to just program variables and \( S^{|W|} \) for the restriction to just world variables.

Definition 12 (Finite-domain world). A world \( \omega \) is finite-domain with domain \( V(\phi)|_W \) iff \( \{ x | \omega(x) \neq 0 \} \) is finite.

Definition 13 (Finite-domain galaxy). A galaxy \( g \) is finite-domain with domain \( V(\phi)|_W \) iff \( \{ s | g(s) \neq \omega_0 \} \) is finite, where \( \omega_0 \) is \( \{(x,0) | x \in V \} \).

Definition 14 (Finite-domain interpretation). An interpretation \( I \) is finite-domain (with domain \( S \)) iff (1) for all \( \pi \), \( I(n) \) has domain \( S_R \) and (2) For all \( p,f,w_1,w_2 \) if \( w_1 \) and \( w_2 \) agree on \( S_R \), then \( I(p)(w_1) = I(p)(w_2) \) and \( I(f)(w_1) = I(f)(w_2) \).

Definition 15 (Finite validity). A formula \( \phi \) is finitely-valid iff for all finite-domain \( I,g,\omega, I_g \in \mathbb{F}_\phi \omega\).

We define a bijection \( \tilde{\omega} \) between dHL worlds and finite-domain dHL worlds. We write the inverse direction of the bijection with inverse notation \( \tilde{\omega} \).

**World translation in dHL:**

\[
\tilde{\omega} = \{(x,\omega(x)) | x \in V(\phi)|_W \} \cup \{(x,0) | x \in V(\phi)|_W \}
\]

The fact that \( \tilde{\omega} \) is a bijection follows directly from \( P_{\omega} \) being a bijection.

The bijection \( \tilde{\omega} \) for galaxies is a simple extension:

**Galaxy translation in dHL:**

\[
\tilde{\phi} = \{(s,\tilde{\phi}(s)) | s \in V(\phi)|_W \}
\]

The bijection \( \tilde{\omega} \) for interpretations proceeds by cases on rigid symbols. We write the inverse direction of the bijection with inverse notation \( \tilde{\omega} \). The rigid symbols that can appear in a concrete formula are nominals \( \pi \) and untyped predicates \( p \) and untyped functions \( f \).

**Definition 16** (Interpretation translation in dHL).

\[
\begin{align*}
\tilde{f}(f)(x) &= I(f)(x) \\
\tilde{f}(p)(x) &= I(p)(x) \\
\tilde{f}(\pi) &= (\tilde{\pi}) \\
\tilde{f}(w) &= I(f)(w)
\end{align*}
\]

We next define translation from finite dHL to finite dl.

**World+galaxy finitization**

\[
(\omega|_W) = \omega \cup \{(x, g(n)(x)) | x \in S_{|R}, n \in S^{|W|} \}
\]

**Definition 17** (Interpretation translation in dl).

\[
\begin{align*}
\hat{f}(f)(x) &= I(f)(x) \\
\hat{f}(p)(x) &= I(p)(x) \\
\hat{f}(\pi) &= (\tilde{\pi}) \\
\hat{f}(w) &= I(f)(w)
\end{align*}
\]

Having defined the key concepts, we can state the lemmas that contain all the work of the proof:

**Lemma 17** (Concrete dHL Finitization - World terms). \( \langle \langle w \rangle \rangle \omega lg = \langle \langle w \rangle \rangle \tilde{\omega} \hat{\phi} \).

**Proof.** By cases on \( w \).

- **Case** 1: Then \( \langle \langle \pi \rangle \rangle \omega lg = (\tilde{\pi})(n) = \tilde{\pi}(n) = \tilde{\pi} \).
- **Case** 2: Then \( \langle \langle s \rangle \rangle \omega lg = (g(s)) = g(s) = \langle \langle s \rangle \rangle \tilde{\phi} \).

**Lemma 18** (Concrete dHL Finitization - Real terms). \( \langle \theta \rangle \omega lg = \langle \theta \rangle \tilde{\phi} \).

**Proof.** By induction on \( \theta \).

- **Case** 1: \( \langle c \rangle \omega lg = c = \langle c \rangle \tilde{\phi} \).
- **Case** 2: \( \langle x \rangle \omega lg = (\omega(x)) = (\omega(x)) \).
- **Case** 3: \( \langle \theta_1 \theta_2 \rangle \omega lg = \langle \theta_1 \rangle \omega lg + \langle \theta_2 \rangle \omega lg = \langle \theta_1 \rangle \tilde{\phi} \).
- **Case** 4: \( \langle \theta_1 + \theta_2 \rangle \omega lg = \langle \theta_1 \rangle \omega lg + \langle \theta_2 \rangle \omega lg = \langle \theta_1 \rangle \tilde{\phi} + \langle \theta_2 \rangle \tilde{\phi} \).
- **Case** 5: \( \langle \theta_1 \cdot \theta_2 \rangle \omega lg = \langle \theta_1 \rangle \omega lg \cdot \langle \theta_2 \rangle \omega lg = \langle \theta_1 \rangle \tilde{\phi} \cdot \langle \theta_2 \rangle \tilde{\phi} \).

\[ \boxed{15} \]
• Case $f(\theta)$ : $\|f(\theta)\|_{\omega} = \lambda(I(f))[\theta]_{\omega} = (I)(f)[\|\theta\|_{\omega}] = (I)(f)[\|\theta\|_{\omega}](I)(\bar{g}) = \|\theta\|_{\omega}(I)(\bar{g})(I)(\bar{g})$ by definition of translation for $f(x) : \mathbb{R}$.

• Case $f(\omega) : \|f(\omega)\|_{\omega} = \lambda(I(f))[\|\omega\|_{\omega}]_{\omega}$ and

$$
\|f(\omega)\| \subseteq (I)(f)[\|\omega\|](I)(\bar{g}) = (I)(f)[(I)(f)[\|\omega\|]]_{\omega} = (I)(f)(\|\omega\|)(I)(\bar{g}) = (I)(f)(\|f(\omega)\|)(I)(\bar{g}) = (I)(f)(\|\omega\|)(I)(\bar{g})
$$

by IH and Lemma 17 so both sides are equal.

• Case $\oplus : \|\oplus\|_{\omega} = \|\oplus\|_{\omega} = \|\oplus\|_{\omega}$ iff $(I)(\bar{g})$ is $(I)(\bar{g})$ for $(I)(\bar{g})$ and Lemma 17.

$\square$

Lemma 19 (Concrete dHLP Finitization - Formulas and Programs).

A concrete dHLP formula is valid iff it is finitely valid. Specifically, $Ig \in \|\psi\|_{\omega}$ iff $(I)(\bar{g}) \in (I)(\bar{g})$. By simultaneous induction we also show finitization for programs: $(\omega, \nu) \in [\|\nu\|_{\omega}]_{\omega}$ if $Ig(\|\psi\|_{\omega})$ iff $(I)(\bar{g}) \in (I)(\bar{g})$. By simultaneous induction.

Proof.

• Case $\exists x : \|\exists x \|_{\omega} : \oplus \in \|\exists x \|_{\omega}$ iff $\{\exists x \} \subseteq \{\exists x \}$. $Ig(\|\psi\|_{\omega})$ iff $(I)(\bar{g}) \in (I)(\bar{g})$ for $(I)(\bar{g})$ and Lemma 17.

• Case $\lambda(x) : \|\lambda(x)\|_{\omega} : \oplus \in \|\lambda(x)\|_{\omega}$ iff $\{\exists x \} \subseteq \{\exists x \}$. $Ig(\|\psi\|_{\omega})$ iff $(I)(\bar{g}) \in (I)(\bar{g})$ for $(I)(\bar{g})$ and Lemma 17.

$\square$

Lemma 20 (Finite Translatability). A dHLP formula $\phi$ is finitely-valid iff its translation $(\bar{\phi})$ is finitely-valid. Specifically, for $I, \nu, \omega, \nu$ with finite domain $S$ we have:

$$
\|\theta\|_{\omega} = \{\bar{\theta}\}_{\omega} \rightarrow \{\bar{\theta}\}_{\omega} \rightarrow \{\bar{\theta}\}_{\omega}
$$

Proof.

• Case $\exists x : \|\exists x \|_{\omega} : \oplus \in \|\exists x \|_{\omega}$ iff $\{\exists x \} \subseteq \{\exists x \}$. $Ig(\|\psi\|_{\omega})$ iff $(I)(\bar{g}) \in (I)(\bar{g})$ for $(I)(\bar{g})$ and Lemma 17.

$\square$
Proof. Fix $\phi \in \text{dHL}$. By Lemma 19, $\phi$ is valid in $\text{dIHF}$ iff it is finitely-valid. Then $\phi$ is finitely-valid iff $\bar{\phi}$ is finitely valid in $\text{dI}$ by Lemma 20. Then $\bar{\phi}$ is finitely-valid in $\text{dI}$ if it is valid by Lemma 21, so $\phi$ is valid iff $\bar{\phi}$ is valid. \hfill $\square$

\textbf{Theorem 22 (dHL reduces to dI).} There exists a computable reduction $\delta$ such that for all concrete $\phi \in \text{dHL}$, $\phi$ is valid $\text{dHL}$, iff $\delta \phi$ is valid in $\text{dI}$.

\textbf{Proof.} Fix $\phi \in \text{dHL}$. By Lemma 19, $\phi$ is valid in $\text{dIHF}$ iff it is finitely-valid. Then $\phi$ is finitely-valid iff $\bar{\phi}$ is finitely valid in $\text{dI}$ by Lemma 20. Then $\bar{\phi}$ is finitely-valid in $\text{dI}$ iff it is valid by Lemma 21, so $\phi$ is valid iff $\delta \phi$ is valid. \hfill $\square$

\textbf{Lemma 21 (Concrete dI Finitization).} A concrete $\text{dI}$ formula is valid iff it is finitely valid. Specifically, $I \in \llbracket \phi \rrbracket$ iff $I \in \llbracket \delta \phi \rrbracket$.

\textbf{Proof.} Trivially by coincidence: every concrete $\text{dI}$ formula has finitely-many free variables, so every state agrees with some finite state on its domain, and thus satisfies $\phi$ iff the corresponding finite state does. \hfill $\square$
D Soundness Proofs

We begin with extending the soundness proofs of uniform substitution. When proofs build on prior work [28] we present only the new cases. We show a series of coincidence lemmas.

Lemma 23 (Coincidence for Terms). If $\omega \cup \eta = \omega \cup h$ on $\text{FV}(\theta)$ and $I = J$ on $\Sigma(\theta)$ then $[\omega][\eta]_I = \{[\eta]\omega\}_J$.

Proof. Induction on $\theta$.
- Case $\varnothing$, $\theta$ have $\text{FV}(\varnothing, \theta) = \{s \cup \{t \in \text{FV}(\theta)\}$ then $[\varnothing][\eta]_I = \{[\eta]\varnothing\}_J$ by IH since $\omega(\varnothing) = \omega(s)$ on $\text{FV}(\theta)$ from assumption equal on $\{t \in \text{FV}(\theta)\}$.
- Case $\pi \theta$ have $\text{FV}(\pi \theta) = \{s \cup \{t \in \text{FV}(\theta)\}$ and $\Sigma(\pi \theta) = \{\eta, \pi\}$ then $[\pi \theta][\eta]_I = \{[\eta]\pi\}_J$ by IH since $I(n) = J(n)$ on $\text{FV}(\theta)$ from assumption equal on $\{t \in \text{FV}(\theta)\}$ and agree on all program variables since $I(n) = J(n)$ by assumption.

The remaining cases are as in prior work. □

Lemma 24 (Coincidence for Formulas). If $\omega = \omega$ on $\text{FV}(\phi)$ and $I = J$ on $\Sigma(\phi)$ then $\lambda g \in \phi[\omega]_I$ iff $\lambda g \in \phi[\omega]_J$.

Proof. Induction on $\phi$ (and simultaneous induction on $\alpha$ for coincidence for programs), but we show only the new cases.
- Case $s \omega \in \phi[s]_I$ iff $g(s) \in \phi$ (because $\forall \text{V}$ and $s$ in (FV)) $h(s) = \omega s$ iff $\omega \in \{\phi\}_I$.
- Case $\bigwedge\phi \in \phi[\pi]_I$ iff $I(n) = \omega$ (iff (because $\forall \text{V}$ in $\text{FV}(\pi)$ and $n \in \Sigma(\pi)$) $J(n) = \omega$ iff $\omega \in \{\phi\}_I$.
- Case $\varnothing \phi \in \{\phi\}_I$ iff $g(s) \in \phi[\omega]_I$ (iff (because $t \in \text{FV}(\phi)$) in free vars and $s$ in free vars) iff $h(s) \in \phi[\omega]_I$ iff $\omega \in \{\phi\}_I$.
- Case $\pi \theta \in \{\phi\}_I$ iff $I(n) \in \phi[\omega]_I$ (iff (because $t \in \text{FV}(\phi)$) in free vars and $n$ in signature) iff $J(n) \in \phi[\omega]_I$ iff $\omega \in \{\phi\}_I$.
- Case $\forall s : \phi \in \{\forall s : \phi\}_I$ if for all worlds $\omega \in \{\phi\}_I$ iff (since free vars are $\text{FV}(\phi) \setminus \{s\}$) for all value $\omega \in \{\phi\}_I$ for $s$.
- Case $\exists s : \phi \in \{\exists s : \phi\}_I$ if for some world $\omega \in \{\phi\}_I$ for $s$ iff (since free vars are $\text{FV}(\phi) \setminus \{s\}$) for some value $\omega \in \{\phi\}_I$ for $s$.
- Case $\bigwedge \phi \in \{\phi\}_I$ iff $\omega \in \{\phi\}_I$ iff $\omega \in \{\phi\}_I$.
- Case $\bigvee \phi \in \{\phi\}_I$ iff $\omega \in \{\phi\}_I$ iff $\omega \in \{\phi\}_I$.

□

Lemma 25 (Coincidence for adjoints). Adjoint interpretations $\sigma^*_\omega$ update the interpretation $I$ to reflect the effect of a substitution $\sigma$; the meaning of every symbol substituted by $\sigma$ is updated to the meaning of its replacement in state $\omega$. Adjoints are analogous to prior work [28].

The notion of $U$-admissibility used here is as in Appendix A and as in prior work [28]:

Definition 18 (U-admissibility). We say a substitution $\sigma$ is $U$-admissible for an expression $e$ with a flexible symbol set $U$ iff $\bigcup_{\omega \epsilon \Sigma(e)} \text{FV}(\sigma \text{sym}) \where \sigma^*_\omega$ is the restriction of $\sigma$ that replaces only symbols occurring in $e$ and where sym is an arbitrary rigid.

If $\omega = \nu$ on $\text{FV}(\sigma)$, then $\sigma^*_\omega = \sigma^*_\nu$. If $\sigma$ is $U$-admissible for $e$ and $\omega = \nu$ on $U^C$ then $\{e[\sigma]_\omega = \{e[\sigma]_\nu$.

Proof. From prior work [28]. □

Lemma 26 (Term Substitution). $\llbracket \sigma(\theta) \rrbracket_I = \llbracket \theta \rrbracket_I \sigma^*_\omega$.\llbracket I.

Proof. By induction on $\theta$. We include just the new cases.

- Case $\varnothing \pi$, $n \epsilon \{e[\pi]_\theta \} = \{e[\pi]_\sigma \}$ then $\llbracket e[\pi]_\theta \rrbracket_I = \{e[\pi]_\sigma \}$ by IH, then by Lemma 25 and admissibility assumption, $\llbracket e[\pi]_\sigma \rrbracket_I = \{e[\pi]_\sigma \}$ since $\llbracket e[\pi]_\sigma \rrbracket_I$ by definition of adjoints.

- Case $\varnothing \pi$, $n \epsilon \{e[\pi]_\theta \} = \{e[\pi]_\sigma \}$ then $\llbracket e[\pi]_\sigma \rrbracket_I = \{e[\pi]_\sigma \}$ by IH, then by Lemma 25 and admissibility assumption, $\llbracket e[\pi]_\sigma \rrbracket_I = \{e[\pi]_\sigma \}$ by definition of adjoints.

- Case $\varnothing \pi$, $\llbracket e[\pi]_\sigma \rrbracket_I = \{e[\pi]_\sigma \}$ then $\llbracket e[\pi]_\sigma \rrbracket_I = \{e[\pi]_\sigma \}$ by IH, then by Lemma 25 and admissibility assumption, $\llbracket e[\pi]_\sigma \rrbracket_I = \{e[\pi]_\sigma \}$ by definition of adjoints.

□

Lemma 27 (Formula Substitution). $\omega \in \llbracket \sigma(\phi) \rrbracket_I \iff \omega \in \llbracket \sigma(\phi) \rrbracket_I$.

Proof. By induction on $\phi$ with simultaneous induction on programs $\alpha$. We present only the new cases.

- Case $\varnothing \pi$, $\pi \epsilon \{e[\pi]_\theta \}$ then $\llbracket e[\pi]_\sigma \rrbracket_I$ iff $\llbracket e[\pi]_\sigma \rrbracket_I$ by IH again $\llbracket e[\pi]_\sigma \rrbracket_I$ by admissibility assumption $\llbracket e[\pi]_\sigma \rrbracket_I$ (since $\llbracket e[\pi]_\sigma \rrbracket_I$ by IH again $\llbracket e[\pi]_\sigma \rrbracket_I$ as desired.

- Case $\varnothing \pi$, $\pi \epsilon \{e[\pi]_\theta \}$ then $\llbracket e[\pi]_\sigma \rrbracket_I$ iff $\llbracket e[\pi]_\sigma \rrbracket_I$ by IH again $\llbracket e[\pi]_\sigma \rrbracket_I$ by admissibility assumption $\llbracket e[\pi]_\sigma \rrbracket_I$ (since $\llbracket e[\pi]_\sigma \rrbracket_I$ by IH again $\llbracket e[\pi]_\sigma \rrbracket_I$ as desired.

□

Soundness of the uniform substitution rule follows immediately by transposing over the same proof from DL [28]. Next, note by
Theorem 16 we get validity of all dL axioms for free, so it suffices to show soundness for the new axioms of dHL.

**Theorem 28 (Hybrid Axiom Soundness).** The hybrid axioms are valid.

**Proof.** We show the axioms are sound one at a time.

- **Axiom K@, (P → Q) → @, P → @, Q** is valid. Fix I and ω, g. Let μ = I(c). Assume ω ∈ [[@[c] (P → Q)]]Ig, then (a) v ∈ [[P → Q]]Ig. Assume ω ∈ [[@[c] P]]Ig, then (b) v ∈ [[P]]Ig. By (a), (b) and modus ponens, (c) v ∈ [[Q]]Ig so ω ∈ [[@[c] Q]]Ig.

- **Axiom @id is valid.** Fix I and ω, g. Assume ω ∈ [[@a @, P]]Ig iff I(a) ∈ [[@a P]]Ig iff I(b) ∈ [[P]]Ig iff ω ∈ [[@a P]]Ig.

- **Axiom @I a ∧ P → @, P** is valid. Fix I and ω, g. Assume ω ∈ [[a ∧ P]]Ig so (a) I(a) = ω and (b) ω ∈ I(P). Then I(a) = ω by (a) then I(a) ∈ [[P]]Ig by (b) and ω ∈ [[@a P]]Ig.

- **Axiom @c → (p(a) ↔ p(c))** is valid. Fix I and ω, g. Assume (a) ω ∈ [[@c c]]Ig, thus I(a) = I(c) so (b) I(a) = I(c). Then ω = I(p(I(a))) iff ω = I(p(I(b))) so ω ∈ [[p(a) ↔ p(c)]]Ig.

- **Axiom (Æ) [α] s p(s) ∧ (α)c → p(c)** is valid. Fix I and ω, g. Assume (a) ω ∈ [[α] s p(s)]Ig and (b) ω ∈ [[(α)c]]Ig. By (a), for all world v if (ω, v) ∈ I(α) then v ∈ I(p). By (b), there exists world µ where (ω, v) ∈ I(α) and µ = I(c). Instantiating (a), have µ ∈ I(p) Combined with (c), have ω ∈ [[p(c)]]Ig and thus ω ∈ [[p]P]]Ig.

- **Axiom ∀Æ y: W p(s) → p(Æ) Fix I and ω, g. Assume (a) for all world v, have ω ∈ [[p(s)]]Ig so (b) I(p(v)). Pick v = I(n) then have I(p(I(n))) and ω ∈ [[p(Æ)]]Ig.**

- **Rule ∀Æ [q](y) Fix x. I and ω. Assuming for all y have (a) I(q(µ(y))) for all states µ. By (a) have (b) ω ∈ [[q]]]Ig for all µ. By (b) have ω ∈ [[Æ y: W q(s)]]Ig.**

- **Axiom Æ (Æ) s p(s) ⊨ Æ s: W p(s))** is valid. Fix I and ω. Then ω ∈ [[Æ s p(s)]]Ig iff ω ∈ [[p(s)]]Ig so I(p(ω)) so I(p(ω)) iff ∃ s.t. v = ω and I(p(v)) iff ω ∈ [[Æ s: W s ∧ p(s)]]Ig.

- **Axiom 3Æ Æ s: W s. Fix I and ω. Suffices to show exists world v such that v = v. Pick v = ω (state) by reflexivity.**

- **Axiom schema BW ⟨α⟩3Æ: W P ↔ Æ s: W ⟨α⟩P** is valid. Fix I, g, and ω.

\[
\begin{align*}
\omega &\in [[⟨α⟩]s: ‘W P]Ig \\
\Rightarrow& v \in [[s: ‘W P]Ig, \text{ for some } (α, v) \in [α]Ig \\
\Rightarrow& v \in [[P]Ig^μ, \text{ for some } (α, v) \in [α]Ig, μ \in ‘W \\
\Rightarrow& v \in [[P]Ig^μ, \text{ for some } μ \in ‘W, (α, v) \in [α]Ig \\
\Rightarrow& v \in [[P]Ig^μ, \text{ for some } μ \in ‘W, (α, v) \in [α]Ig \\
\Rightarrow& v \in [[(α)P]Ig^μ, \text{ for some } μ \in ‘W \\
\Rightarrow& ν \in [[s: ‘W (α)P]Ig \\
\end{align*}
\]

Where the starred step holds by the coincidence lemma due to the assumption s ∈ FV(α).

- **Axiom G@ φ @φ is sound.** Fix I, g, ω. Assume for all v ∈ ‘W, all h ∈ g, have v ∈ [[φ]]Ig. Then instantiate v = g(i), h = g and have g(i) ∈ [[φ]]Ig and thus ω ∈ [[@φ]]Ig. Since this held for all g and ω and I the conclusion is valid, i.e. the rule is sound. 

□
**Theorem 29** (At-Term soundness). The at-term axioms are valid.

Proof. We begin with the non-derived axiom \( \@i 0 \).

**Lemma 30.** Formula \( \@i p(F_1, \ldots, F_n) \leftrightarrow p(\@i F_1, \ldots, \@i F_n) \) is valid.

Proof. Semantic proof. Fix \( k \in \mathbb{N} \), interpretation \( I \) and state \( \omega \). Then the sides have the same semantics:

\[
\begin{align*}
\omega &\in [p(\@i F_1, \ldots, \@i F_k)] I g \\
= & I(p)([\@i F_1] I g \omega, \ldots, [\@i F_k] I g \omega) \\
= & I(p)([F_1] I g (i), \ldots, [F_k] I g (i)) \\
= & I(i) \in [p(F_1, \ldots, F_k)] I g \\
= & \omega \in [\@i p(F_1, \ldots, F_k)] I g
\end{align*}
\]

\( \Box \)

Combining axiom \( \@i 0 \) with existing axioms and sequent rules (which are derivable from typical hilbert axioms), we derive the remaining axioms.

- Axiom NT:= is valid:

\[
\begin{align*}
\text{refl} & \quad \begin{array}{c}
\frac{f() = f()}{x : f() \vdash (x := f()) x}
\end{array} \\
\text{id} & \quad \begin{array}{c}
\frac{(x := f()) y + (x := f()) y}{x : f() \vdash (x := f()) y}
\end{array} \\
\text{cut} & \quad \begin{array}{c}
\frac{\exists (x := f) \vdash \exists (x := f) \vdash F = \@j x}{\exists (x := f) \vdash \exists (x := f) \vdash F = \@j x}
\end{array}
\end{align*}
\]

\( \Box \)

- Axiom NT; is valid:

\[
\begin{align*}
\text{cut} & \quad \begin{array}{c}
\frac{\exists (x := f) \vdash \exists (x := f) \vdash F = \@j x}{\exists (x := f) \vdash \exists (x := f) \vdash F = \@j x}
\end{array}
\end{align*}
\]

\( \Box \)
A Hybrid, Dynamic Logic for Hybrid-Dynamic Information Flow

LICS’18, Jul 9-12, 2018, Oxford, UK

• Axiom NT is valid:

\[ \forall (a) \land \forall (b) \vdash \neg (a) \land \neg (b) \]

Proof.

\[ \forall L \vdash (a) \land (b) \vdash \neg (a) \land \neg (b) \]

\[ (U) \vdash (a) \land (b) \vdash \neg (a) \land \neg (b) \]

• Axiom NT’ is valid:

\[ \neg (x = y) \vdash \theta (a) \land \theta (b) \]

Proof.

\[ \neg (x = y) \vdash \theta (a) \land \theta (b) \]

\[ \neg (x = y) \vdash \theta (a) \land \theta (b) \]

\[ \theta (a) \land \theta (b) \vdash \neg (x = y) \]

• Axiom NT* is valid:

\[ \neg (x = y) \vdash \theta (a) \land \theta (b) \]

Proof. Recall in this proof Rule FP is the fixpoint rule for loops, interderivable from loop induction I.

\[ \forall (a) \land \forall (b) \vdash \neg (a) \land \neg (b) \]

Proof.

\[ \neg (x = y) \vdash \theta (a) \land \theta (b) \]

\[ \neg (x = y) \vdash \theta (a) \land \theta (b) \]

\[ \theta (a) \land \theta (b) \vdash \neg (x = y) \]

• Axiom NTV is valid:

\[ \neg (x = y) \vdash \theta (a) \land \theta (b) \]

Proof.
Theorem 31 (Bisimulation Soundness). The bisimulation (derived) rules are sound.

Proof. • The auxiliary loop elimination rule @Ind is derived: \[\begin{array}{c}
@i, o_1 \rightarrow p(o_1) \\
@o, (a^*)o_1 \land p(m_1) \rightarrow p(o_1)
\end{array}\]  

\[\begin{array}{c}
\vdash i, \alpha \rightarrow F \\
\vdash j, \alpha \rightarrow F
\end{array}\]

\[\vdash j, \alpha \rightarrow F \]

Proof. • Loop bisimulation rule BS* is derived: \[\begin{array}{c}
\vdash i, (a^*)o_1 \land R(i_1, i_2) \rightarrow @i, (a^*)o_2 R(o_1, o_2)
\end{array}\]

Proof. Below, @ is the open goal.
\[ \begin{align*}
\@i_{\alpha}(a)m_1 \land R_{i_1,i_2} &\rightarrow \@i_{\alpha}(a)m_2 R_{m_1,m_2} \\
\@m_1(a)\alpha_1 \land R_m(m_1,m_2) &\rightarrow \@m_2(a)\alpha_2 R_o(m_1,m_2)
\end{align*} \]

- Sequential composition bisimulation rule BS is derived: \[ \@i_{\alpha}(a)\beta\alpha_1 \land R_{i_1,i_2} \rightarrow \@i_{\alpha}(a)\beta\alpha_2 \land R_{o}(o_1,o_2) \]

**Proof.** Let \( \Gamma \equiv \@i_{\alpha}(a)m_1, @m_1(\beta)\alpha_0, R(i_1,i_2) \)

- Choice composition bisimulation rule BS|U is derived: \[ \@i_{\alpha}(a)\alpha_0 \land R_{i_1,i_2} \rightarrow \@i_{\alpha}(a)\alpha_0 \land R_{o}(o_1,o_2) \]

**Proof.**
\[
\begin{align*}
(\land) &\quad \@i_{\alpha}(a)\alpha_0 \land R_{i_1,i_2} \rightarrow \@i_{\alpha}(a)\alpha_0 \land R_{o}(o_1,o_2) \\
(\lor) &\quad \@i_{\alpha}(a)\beta_0 \land R_{i_1,i_2} \rightarrow \@i_{\alpha}(a)\beta_0 \land R_{o}(o_1,o_2) \\
(\land) &\quad \@i_{\alpha}(a)\alpha_0 \land R_{i_1,i_2} \rightarrow \@i_{\alpha}(a)\alpha_0 \land R_{o}(o_1,o_2) \\
(\lor) &\quad \@i_{\alpha}(a)\beta_0 \land R_{i_1,i_2} \rightarrow \@i_{\alpha}(a)\beta_0 \land R_{o}(o_1,o_2)
\end{align*}
\]
E  Full proof of secure smart grid model

Additional Notations We use some additional concepts and notations in the proof appendices, in order to make proofs simpler.

While the presentation given in the body of the paper is Hilbert-style, this is easily extended to a sequent-style calculus by adding standard propositional sequent calculus rules and contextual equivalence rules which carry over directly from the base uniform substitution calculus for $\mathcal{L}$ [28]. We write sequents $\Gamma \vdash \Delta$ where $\Gamma$ (the antecedent) and $\Delta$ (the succedent) are both lists of formulas, and then sequent $\Gamma \vdash \Delta$ is true in exactly the same states as the formula $\prod_{i \in I} \Gamma_i \rightarrow \bigvee_{j \leq J} \Delta_j$.

We mark steps that close a proof branch with an asterisk (*). When a derivation is too large to fit in its entirety, we introduce variables standing for unfinished branches of the derivation which are then proved separately. These variables typically start with $\Xi$ for derivation.

We use triple notation for conciseness, e.g., $\theta(x, y) = \theta_j(x, y)$ is shorthand for $\theta(q_x = x \land \theta_j y = \theta_j y)$. Likewise, $x, y = \theta_k, \theta_l$ is shorthand for the programs $x = \theta_k, y = \theta_l$. For long sequences of assignments, we elide irrelevant assignments with dots (..). We also write transitive equalities $x \equiv y \equiv z$ with the typical meaning $x \equiv y \equiv z$.

As in the main paper we use $\mathcal{E}_x^\Theta$ for the substitution of $\theta$ for $x$ in $\mathcal{E}$. We will also sometimes construct named substitutions $\sigma$ and substitute with the notation $\sigma(e)$ when convenient, as in the uniform substitution algorithm.

Lastly, we use straightforward derived constructs for clarity, specifically truth $\top$ and falsehood $\bot$, which are trivially derived as $0 < 1$ and $0 < 0$, respectively.

Derived Rules In this proof we will use a few derived rules which were ignored in the main text for clarity of presentation.

**M (\top)**

\[
\frac{P \Rightarrow Q}{[\top] P \Rightarrow [\top] Q}
\]

\[
\frac{(P \vee (\alpha) Q) \Rightarrow Q}{(\alpha^* P) \Rightarrow \Delta}
\]

Rule M (\top) derives trivially from K, G, and MP and is the box analog of M. Rule FP has been shown inter-derivable with loop induction axiom 1 in prior work [27].

Proof. In this proof we decompose the model into pieces, letting $\sigma_{\mathcal{N}}$ stand for the nondeterministic assignments

\[
\begin{align*}
&\sigma_t \; \text{stand for the load balancers}
&\sigma_{\mathcal{N}_t} \; \text{stand for the battery controller}
&\sigma_{\mathcal{B}} \; \text{stand for the differential equation}
&\sigma_{\mathcal{L}} \; \text{stand for the controller correctness lemma}
\end{align*}
\]

and combine letters in subscripts to denote compositions, e.g., $\sigma_{\mathcal{NLBP}} \equiv (\alpha_{\mathcal{NL}}; \alpha_{\mathcal{LB}}; \alpha_{\mathcal{P}})$. Here we define the relations $R(i, f) \equiv \oplus \iota \equiv \oplus t \land \oplus G = \oplus f \land \text{Thresh} \geq 0$ and $R_{\mathcal{L}}(i, f) \equiv R(i, f) \land \oplus g m = \oplus g m \land m > 0 \land B_{\text{max}} > 0$. The main proof, where $\mathcal{D}_\sigma$ stands for the controller correctness lemma:

\[
\begin{align*}
&\text{Rassumption } R_m(m, m_2), \sigma_{\mathcal{NL}}, \sigma_{\mathcal{LB}}, \sigma_{\mathcal{P}} \vdash \sigma_{\mathcal{L}} \text{[\top]} \quad \text{NT}^* \quad \text{R〗} \\
&\text{[\top]} \quad \text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \\
&\text{R〗} \
\end{align*}
\]

next we prove the lemma $\mathcal{D}_\sigma$ for controller safety, which itself uses a lemma $\mathcal{D}_\sigma$ for controller inversion. The lemma application immediately splits us into four branches $B_1$ to $B_4$.
Below we abbreviate $\Gamma \equiv \Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4$.

The controller inversion lemma. Here we abbreviate

$\phi_1 \equiv (G = \alpha \wedge \text{gm} = 0)$

$\phi_2 \equiv (G = N_2 \wedge \text{gm} = m)$

$\phi_3 \equiv (G = N_2 \wedge \text{gm} = -m)$

$\phi_4 \equiv (G = N_1 + N_2 \wedge \text{gm} = 0)$
for formulas and

\[ y = G = 0, b_m = 0, g_m = 0 \]
\[ \beta_{ia} \equiv (B_1 < B_{\max}) \lor (N_1 > 0 \land B_1 > 0), b_1 = -N_1; b_{ma} = b_{ma} + m \cdot (-1)^3 \]
\[ \beta_{ib} \equiv b_1 = 0, G = G + N_1; g_m = g_m + m \cdot (-1)^3 \]
\[ \beta_{ia} \equiv (B_2 < B_{\max}) \lor (N_2 > 0 \land B_2 > 0), b_2 = -N_2; b_{ma} = b_{ma} + m \cdot (-1)^3 \]
\[ \beta_{ib} \equiv b_2 = 0, G = G + N_2; g_m = g_m + m \cdot (-1)^3 \]

Then the lemma is: \( \varrho_{b_1} (\sigma_{NLB}) m_1 \Rightarrow \varrho_{m_1} (\sigma_1 \lor \phi_2 \lor \phi_3 \lor \phi_4) \)

Here \( b_1, \ldots, b_4 \) are branches, proved after the main lemma.

\[
\begin{align*}
(\text{b}_1) & \quad (\Rightarrow, \forall R) \\
(\text{b}_2) & \quad (\Leftarrow, \Rightarrow) \\
(\text{b}_3) & \quad (\Leftarrow, \Rightarrow) \\
(\text{b}_4) & \quad (\Leftarrow, \Rightarrow)
\end{align*}
\]

We also define \( \sigma = [G \mapsto 0, b_m \mapsto 0, g_m \mapsto 0] \)

\( b_1 \)
F  Full proof of insecure smart grid model

See note at the start of Appendix E on extra notations used in the appendixes.

Lemma 32. Formula $\exists_1 \cdot W \mathrel{\uparrow}_{\iota_1}(B_i = B_{\text{max}}; t = 0, G = 0)\iota_1$ is valid.

Proof.

\[ \exists_1 \cdot W \mathrel{\uparrow}_{\iota_1}(B_i = B_{\text{max}}; t = 0, G = 0)\iota_1 \]

Lemma 33. Formula $\exists_2 \cdot W \mathrel{\uparrow}_{\iota_2}(B_i = B_{\text{max}}; t = 0, G = 0)\iota_2$ is valid.

Proof.

\[ \exists_2 \cdot W \mathrel{\uparrow}_{\iota_2}(B_i = B_{\text{max}}; t = 0, G = 0)\iota_2 \]

Lemma 34. Formula $\exists_2 \cdot W \mathrel{\uparrow}_{\iota_2}(B_i = B_{\text{max}}/2; t = 0, G = 0)\iota_2$ is valid.

Proof.

\[ \exists_2 \cdot W \mathrel{\uparrow}_{\iota_2}(B_i = B_{\text{max}}/2; t = 0, G = 0)\iota_2 \]

Lemma 35. Formula $\exists_2 \cdot W \mathrel{\uparrow}_{\iota_2}(B_i = B_{\text{max}}/2; t = 0, G = 0)\iota_2$ is valid.

Proof.

\[ \exists_2 \cdot W \mathrel{\uparrow}_{\iota_2}(B_i = B_{\text{max}}/2; t = 0, G = 0)\iota_2 \]
Lemma 37. Formula $\mathcal{J}_{\mathcal{L}}[\alpha](t \geq 0 \land (t = 0 \rightarrow (G = 0 \land B_1 = B_2 = B_{max}))$ is valid.

Proof. Define $f \equiv t \geq 0 \land (t = 0 \rightarrow (G = 0 \land B_1 = B_2 = B_{max}))$. Define $\mathcal{C} = (N_1 \leq 0 \land B_i < B_{max}) \lor (N_1 > 0 \land B_i > 0)$. Define branches $\mathcal{B}_1 \equiv \{ b_1 \equiv N_1; b_{m_i} = bm_i + m \cdot (-1)^{i+1} \}_i \land \mathcal{B}_2 \equiv \{ b_2 \equiv G = G + N_i; g_{m} = g_{m} + m \cdot (-1)^{i+1} \}_i$.

Theorem 38 (Insecurity). Formula $\exists \mathcal{I}_1, \mathcal{I}_2, \mathcal{W} R(\mathcal{I}_1, \mathcal{I}_2) \land \mathcal{J}_{\mathcal{J}_{\mathcal{L}}}[\alpha_1, \alpha_2] [\alpha_1 \rightarrow \mathcal{R}(\mathcal{I}_1, \mathcal{I}_2)]$ is valid.

Proof. We abbreviate.
\[\phi_1 = @\ell(B_1 = B_{\text{max}}; t = 0, G = 0)\]
\[\phi_2 = @\ell(B_2 = B_{\text{max}}/2; t = 0, G = 0)\]
\[\phi_3 = @\ell_2(\ell_2)\eta_1 \land @\ell_2 t = 0 \land G > 0\]
\[P_1(t_1) = @\ell(B_1 = B_{\text{max}} \land t = 0 \land G = 0)\]
\[P_2(t_2) = @\ell_2(B_2 = B_{\text{max}}/2 \land t = 0 \land G = 0)\]

\[
\begin{array}{c}
\phi_1, \phi_2, \phi_3, P_1(t_1), P_2(t_2), t \rightarrow 3_1, 2_1, 1_1: W(R(t_1, t_2) \land @\ell_1(\ell_1)\eta_1 \land @\ell_2(\ell_2)\eta_1 \land @\ell_2(\ell_2)\eta_2 \land @\ell_2(\ell_2)\eta_1 \omega_1 \rightarrow R(0_1, 0_2)) \\
\phi_1, \phi_2, \phi_3, t \rightarrow 3_1, 2_1, 1_1: W(R(t_1, t_2) \land @\ell_1(\ell_1)\eta_1 \land @\ell_2(\ell_2)\eta_1 \land @\ell_2(\ell_2)\eta_2 \land @\ell_2(\ell_2)\eta_1 \omega_1 \rightarrow R(0_1, 0_2)) \\
\end{array}
\]