A Logic of Proofs for Differential Dynamic Logic

Toward Independently Checkable Proof Certificates for Dynamic Logics

Nathan Fulton
Institute for Software Research
Carnegie Mellon University, Pittsburgh PA, USA
nathanfu@cs.cmu.edu

André Platzer
Computer Science Department
Carnegie Mellon University, Pittsburgh PA, USA
aplatzer@cs.cmu.edu

Abstract

Differential dynamic logic is a logic for specifying and verifying safety, liveness, and other properties about models of cyber-physical systems. Theorem provers based on differential dynamic logic have been used to verify safety properties for models of self-driving cars and collision avoidance protocols for aircraft. Unfortunately, these theorem provers do not have explicit proof terms, which makes the implementation of a number of important features unnecessarily complicated without soundness-critical and extra-logical extensions to the theorem prover. Examples include: an unambiguous separation between proof checking and proof search, the ability to extract program traces corresponding to counter-examples, and synthesis of surely-live deterministic programs from liveness proofs for nondeterministic programs.

This paper presents a differential dynamic logic with such an explicit representation of proofs. The resulting logic extends both the syntax and semantics of differential dynamic logic with proof terms – syntactic representations of logical deductions. To support axiomatic theorem proving, the logic allows equivalence rewriting deep within formulas and supports both uniform renaming and uniform substitutions.

Categories and Subject Descriptors F.4.1 [*Mechanical theorem proving*]; F.3.2 [*Denotational semantics*]

General Terms Verification

Keywords cyber-physical systems, differential dynamic logic, hybrid systems, proof terms

1. Introduction

Cyber-physical systems (CPS) are systems that combine computation with control of physical processes. Examples of CPS include self-driving cars, train control systems, and collision avoidance protocols for aircraft. Cyber-physical systems are an important domain in software verification because CPS are often safety-critical – a bug in the control software of a self-driving car or a train control system could lead to loss of human life. Unfortunately, many software verification techniques developed in the context of discrete dynamical systems are incapable of handling the infinite state space introduced by the presence of differential equations.

Hybrid systems are a mathematical model of cyber-physical systems that combine a model of discrete computation (imperative computation) with continuous dynamics (ordinary differential equations). Differential dynamic logic [20, 23] is a logic for specifying and verifying properties of hybrid systems. Recent work on theorem proving for cyber-physical systems demonstrates that dynamic logics are a powerful formalism for mechanizing proofs about many other types of dynamical systems. KeYmaera [27] is a theorem prover for differential dynamic logic that has been used to verify various properties of distributed adaptive cruise control for self-driving cars [13], the European Train Control System [28], and multiple collision avoidance protocols for aircraft [11, 14, 26]. KeYmaera X is a successor to KeYmaera that supports the same verification tasks, but features tactical theorem proving on top of a small soundness-critical core [7].

Unlike theorem provers based upon type-theoretic foundations, theorem provers in the dynamic logic tradition are not based upon logics with a formalized notion of explicit proof evidence. Like several other theorem provers, KeYmaera X ensures soundness by only allowing truth-preserving transformations on formulas, rather than by production of formally defined and independently checkable proof terms. The long list of successful theorem provers that are based on logics without proof terms demonstrates truth-preserving operations on formulas are enough to ensure the soundness of a theorem prover.

Although truth-preserving operations are sufficient for ensuring soundness, proof terms address a number of limitations that have arisen during the development and use of the KeYmaera and KeYmaera X theorem provers. KeYmaera and KeYmaera X do not:

- provide a clean separation between proof checking and proof search
- implement a mechanism for composing, reusing, or parameterizing *proofs* (merely mechanisms for composing provability); or
- take advantage of procedures that require interrogating or modifying the structure of a proof.

One advantage of the approach KeYmaera X takes is that there is never a need to re-check proofs obtained via proof search because search always proceeds via operations defined in the soundness-critical core of KeYmaera X. However, ensuring soundness is not the only motivation for separating searching from checking. KeYmaera X allows for parallel speculative proof search, so persisting the particular execution trace of a proof search procedure requires storing and merging proof state using extra-logical operations. Introducing an explicit notion of evidence into differential dynamic

logic is a more principled solution than post-hoc analysis of the execution of a search procedure.

The second challenge is surmountable within a single theorem proving session, but is problematic in cases where users collaborate on proofs. Proof terms provide a natural modularity mechanism and allow users to import proven lemmas from other users without re-executing an expensive proof search procedure or blindly trusting the source of the proof.

The significance of the final challenge extends beyond the specifics of implementations. Extant dynamic logics do not provide a compelling foundation for defining proof-preserving transformations; i.e., transformations to system models that are accompanied by a corresponding transformation on a proof. They are limited to truth-preserving transformations without preserving corresponding proofs syntactically.

This paper presents a *Logic of Proofs for Differential Dynamic Logic* (LPd$\mathcal{L}$). LPd$\mathcal{L}$ provides an explicit notion of evidence in the form of proof terms – syntactic objects that correspond to deductions in (the uniform substitution calculus of) differential dynamic logic (d$\mathcal{L}$). Concretely, we assign a syntactic term e to each derivation of ϕ in d$\mathcal{L}$ such that $e : \phi$ – read as "e is a proof of ϕ" – is a theorem of LPd$\mathcal{L}$. We provide a semantics and an axiomatization for this language of proof terms and establish some basic results about the logic and its relation to d$\mathcal{L}$. Although the primary topic of this paper is LPd$\mathcal{L}$ itself, potential applications are worth noting because they motivate the design of the logic.

One application – discussed in Section 5 – is an ongoing implementation of a proof term checker – a program that takes a formula of the form $e : \phi$ and checks that e is a proof of ϕ. Proof checkers are useful because they separate proof search from proof checking and provide obvious paths toward composition of proofs.

LPd$\mathcal{L}$ is designed to support other applications as well. A major goal for KeYmaera X is automatic transformation of a liveness proof for a non-deterministic model into a safety proof for a fully deterministic model. Understanding the motivation for this operation requires understanding the typical structure of a model specified in d$\mathcal{L}$. Models of cyber-physical systems are often stated as non-deterministic programs because a non-deterministic model can capture a variety of environmental conditions and a variety of control decisions. For example, a car's sensors might sample at non-deterministic points in time, and its control program might choose between acceleration and deceleration in ways that are not known a priori (or that are overly laborious to specify during verification). Synthesizing a most-conservative deterministic controller from these non-deterministic models plugs a major remaining gap between CPS models and control software implementations. Essentially, the key insight is that liveness proofs in d$\mathcal{L}$ contain enough information to construct the particular execution that witnesses liveness. The logic described in this paper directly supports this goal in a way that truth-preserving operations do not.

Summarily, our primary contributions are:

- We present a semantic model that extends the standard reachability relation semantics of differential dynamic logic with a notion of evidence (following Fitting [6]).
- We extend a differential dynamic logic with an explicit notion of evidence – a Logic of Proofs for Differential Dynamic Logic (LPd$\mathcal{L}$).
- We establish the correctness of this logic by proving that all pieces of evidence in LPd$\mathcal{L}$ correspond to a deduction in d$\mathcal{L}$.
- We explain how the results established in this paper can be used to construct a proof term checker for LPd$\mathcal{L}$ without extending the soundness-critical core of a theorem prover based on truth-preserving transformations to theorems, and discuss the details of an ongoing implementation of an LPd$\mathcal{L}$ proof checker as an extension to KeYmaera X.

These contributions constitute a logical foundation for hybrid systems with an explicit notion of evidence.

2. Background

This section presents necessary background for the remainder of the paper, including an introduction to cyber-physical systems and a discussion of the uniform substitution calculus of d$\mathcal{L}$.

2.1 Modeling Cyber-Physical Systems Using Hybrid Programs

Hybrid dynamical systems [2, 23] are mathematical models for analyzing the interaction between discrete and continuous dynamics. This section presents a semantic model of hybrid dynamical systems called hybrid programs, introduces the language of differential dynamic logic (d$\mathcal{L}$), and demonstrates how d$\mathcal{L}$ can be used to specify safety and liveness properties of hybrid programs.

Hybrid programs [21–23] are a programming language model of hybrid dynamics. Hybrid programs extend non-deterministic imperative programs (i.e., regular programs) with differential equations. A syntax and informal semantics of hybrid programs is given in Table 1.

Program Statement	Meaning
$\alpha;\beta$	Sequentially composes α and β.
$\alpha \cup \beta$	Executes either α or β.
α^*	Repeats α zero or more times.
$x := \theta$	Evaluates θ and assign result to x.
$x := *$	Assigns an arbitrary real value to x.
$\{x_1' = \theta_1, ..., x_n' = \theta_n \& F\}$	Continuous evolution[1].
$?F$	Aborts if F is not true.

Table 1. Hybrid Programs

Differential dynamic logic (d$\mathcal{L}$) is a modal logic for specifying and verifying reachability properties about hybrid programs. The formulas of d$\mathcal{L}$ contain the formulas of the First-order Logic of Real-Closed Fields (formulas of $\text{FOL}_{\mathbb{R}}$), the familiar logical connectives of propositional logic, and two modalities – $[\alpha]\phi$ and $\langle\alpha\rangle\phi$. These two modalities express reachability properties about the program α. The box modality ($[\alpha]\phi$) states that ϕ is true of *all* states that are reachable after executing the hybrid program α. The diamond modality ($\langle\alpha\rangle\phi$) is dual to the box, and states that ϕ is true of *some* state that is reachable after executing the hybrid program α.

A formal syntax and semantics for d$\mathcal{L}$ is given later in Def. 1 and Def. 3. For now, we provide examples of how d$\mathcal{L}$ can be used to model cyber-physical systems and specify properties of these models.

Example 1. *The following* d$\mathcal{L}$ *formula describes a safety property for a car model.*

$$\underbrace{v \geq 0 \wedge A > 0}_{\text{initial condition}} \rightarrow [\big(\underbrace{(a := A \cup a := 0)}_{\text{ctrl}} ; \underbrace{\{p' = v,\ v' = a\}}_{\text{plant}}\big)^*] \underbrace{v \geq 0}_{\text{postcondition}}$$

The hybrid program in this formula describes a car that chooses nondeterministically to accelerate with a maximum acceleration A or not accelerate, and then follows a differential equation. This process may repeat arbitrarily many times, and because there is no evolution domain constraint on plant, each continuous evolution

[1] A continuous evolution along the differential equation system $x_i' = \theta_i$ for an arbitrary duration within the region described by formula F.

has an non-negative duration $r \in \mathbb{R}_{\geq 0}$ The formula states that if the car begins with a non-negative velocity, then it will also have a non-negative velocity after choosing a new acceleration and moving for a nondeterministic period of time.

A tutorial with more examples of cyber-physical system models implemented in dℒ can be found in [30].

2.2 The Uniform Substitution Calculus of Differential Dynamic Logic

There are several formulations of differential dynamic logic. The earliest is a sequent calculus [20]. KeYmaera [27] is an implementation of the sequent calculus. In this paper, we augment the axiomatic formulation of dℒ [25] that is implemented in the KeYmaera X theorem prover.

Typical axiom systems contain a countably infinite number of axioms generated from a finite set of axiom schemata. For example, $\phi \wedge \psi \rightarrow \phi$ is an axiom schema, and $x = 1 \wedge x^2 > 0 \rightarrow x = 1$ is a concrete instance of the schema. The axiomatization of dℒ that we augment does not have axiom schemata; rather, it has a finite number of axioms, a finite number of proof rules (represented as sets of formulas), and a proof rule for performing soundness-preserving substitutions on these axioms.

The difference between axiom schemata and uniform substitutions is subtle, but is significant in the context of mechanized proofs. Moving from axiom schemata to concrete axioms isolates soundness-critical reasoning about binding structure into a very small soundness-critical core [7]. The uniform substitution calculus of dℒ provides locally sound axioms for differential equations by exploiting differential forms [25].

This section introduces the syntax, semantics, and axiomatization of dℒ and discusses its uniform substitution calculus. This logic is augmented in subsequent sections with an explicit notion of evidence for axiomatic deductions. Readers interested in further details about the uniform substitution calculus of dℒ are encouraged to read [25].

Definition 1 (Terms)**.** *Terms are defined by this grammar (with θ, η, θ_i as terms, x as variables, x' as differential symbols, and f as function symbols):*

$$\begin{array}{lll} \theta, \eta ::= x \mid x' & & \textit{Variables and Differential Symbols} \\ \mid f(\theta_1, \ldots, \theta_k) & & \textit{Function Application} \\ \mid \theta + \eta \mid \theta \cdot \eta & & \textit{Addition and Multiplication} \\ \mid (\theta)' & & \textit{Differentials} \end{array}$$

The variables x and x' are taken from finite sets of variables V and V' and real numbers are definable as function symbols without arguments.

Definition 2 (Hybrid Programs)**.** *Hybrid programs are defined with the following grammar (with α, β ranging over hybrid programs, a over program constants, x over variables, θ over terms possibly containing x, and ψ over formulas of first-order real arithmetic):*

$$\alpha, \beta ::= a \mid x := \theta \mid x' := \theta \mid ?\psi \mid x' = \theta \& \psi \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$$

Hybrid programs of differential-form dℒ extend the hybrid programs discussed in Table 1 with *differential assignments* ($x' := \theta$) and *program constants* a. The former are related to the differential form axiomatization of differential equations, and the latter are crucial to proofs involving uniform substitution.

Definition 3 (Formulas)**.** *The formulas of dℒ are defined as follows (with θ as terms, p as predicates, C as quantifier symbols, and ϕ, ψ ranging over dℒ formulas):*

$$\begin{array}{lll} \phi, \psi ::= \theta \geq \eta & & \textit{Comparisons} \\ \mid p(\theta_1, \ldots, \theta_k) & & \textit{Predicates} \\ \mid C(\phi) & & \textit{Quantifier Symbols} \\ \mid \neg\phi \mid \phi \wedge \psi \mid \forall x\ \phi \mid \exists x\ \psi & & \textit{First-order Logic} \\ \mid [\alpha]\phi \mid \langle\alpha\rangle\phi & & \textit{Modalities} \end{array}$$

$$\begin{array}{ll} \langle\cdot\rangle & \langle a\rangle p(\bar{x}) \leftrightarrow \neg[a]\neg p(\bar{x}) \\ [:=] & [x := f]p(x) \leftrightarrow p(f) \\ [?] & [?q]p \leftrightarrow (q \rightarrow p) \\ [\cup] & [a \cup b]p(\bar{x}) \leftrightarrow [a]p(\bar{x}) \wedge [b]p(\bar{x}) \\ [;] & [a; b]p(\bar{x}) \leftrightarrow [a][b]p(\bar{x}) \\ [^*] & [a^*]p(\bar{x}) \leftrightarrow p(\bar{x}) \wedge [a][a^*]p(\bar{x}) \\ \text{K} & [a](p(\bar{x}) \rightarrow q(\bar{x})) \rightarrow ([a]p(\bar{x}) \rightarrow [a]q(\bar{x})) \\ \text{I} & [a^*](p(\bar{x}) \rightarrow [a]p(\bar{x})) \rightarrow (p(\bar{x}) \rightarrow [a^*]p(\bar{x})) \\ \text{V} & p \rightarrow [a]p \\ \text{G} & \dfrac{p(\bar{x})}{[a]p(\bar{x})} \\ \forall & \dfrac{p(x)}{\forall x\, p(x)} \\ \text{MP} & \dfrac{p \rightarrow q \quad p}{q} \\ \text{CT} & \dfrac{f(\bar{x}) = g(\bar{x})}{c(f(\bar{x})) = c(g(\bar{x}))} \\ \text{CQ} & \dfrac{f(\bar{x}) = g(\bar{x})}{p(f(\bar{x})) \leftrightarrow p(g(\bar{x}))} \\ \text{CE} & \dfrac{p(\bar{x}) \leftrightarrow q(\bar{x})}{C(p(\bar{x})) \leftrightarrow C(q(\bar{x}))} \\ \text{US} & \dfrac{\varphi}{\sigma(\varphi)} \end{array}$$

Figure 1. Axioms and proof rules of differential dynamic logic; C is a quantifier symbol, p, q are predicate symbols, and c, f, g are function symbols.

2.2.1 Semantics of dℒ

States are mappings from variables and differential symbols to $\mathbb{R}$. The set S is the set of all states.

The semantics of dℒ terms is a mapping $[\![\cdot]\!]^I$ from terms to $\mathbb{R}$, where the interpretation I assigns to each n-ary function symbol f a smooth function $I(f) : \mathbb{R}^n \rightarrow \mathbb{R}$, to each n-ary predicate symbol p a relation $I(p) \subseteq \mathbb{R}^n$, and to each quantifier symbol C a functional $I(C)$ mapping a subsets $M \subseteq S$ to subsets $I(C)(M) \subseteq S$. Differential symbols and differentials are given local meaning by differential forms [25].

The semantics of a hybrid program $[\![\alpha]\!]^I \subseteq S \times S$ is a reachability relation on states defined for each interpretation I. The semantics of hybrid programs inductively define the transition behavior of each hybrid program. For example,

$$[\![x := \theta]\!]^I = \{(v, v_x^r) : r = [\![\theta]\!]^I v\}$$

where v_x^r is the state identical to v except that x maps to $r \in \mathbb{R}$.

The semantics of dℒ formulas is a mapping $[\![\cdot]\!]^I$ from formulas φ to the set of states where φ is true. E.g., $[\![\phi \wedge \psi]\!]^I = [\![\phi]\!]^I \cap [\![\psi]\!]^I$ and $[\![C(\phi)]\!]^I = I(C)([\![\phi]\!]^I)$ for quantifier symbols C.

The full inductive definition of $[\![\cdot]\!]^I$ for terms, programs, and formulas is given by Platzer in [25].

2.2.2 Axioms of dℒ

The axioms and proof rules of dℒ from [25] are enumerated in Figures 1 and 2.

In typical verification tasks, the axioms in Fig. 1 are used to symbolically decompose regular programs and the axioms in Fig. 2 enable various reasoning techniques for handling ordinary differen-

$$
\begin{array}{ll}
\text{DW} & [x' = f(x)\,\&\,q(x)]q(x) \\
\text{DC} & \big([x' = f(x)\,\&\,q(x)]p(x) \leftrightarrow [x' = f(x)\,\&\,q(x) \wedge r(x)]p(x)\big) \leftarrow [x' = f(x)\,\&\,q(x)]r(x) \\
\text{DE} & [x' = f(x)\,\&\,q(x)]p(x,x') \leftrightarrow [x' = f(x)\,\&\,q(x)][x' := f(x)]p(x,x') \\
\text{DI} & [x' = f(x)\,\&\,q(x)]p(x) \leftarrow \big(q(x) \rightarrow p(x) \wedge [x' = f(x)\,\&\,q(x)](p(x))'\big) \\
\text{DG} & [x' = f(x)\,\&\,q(x)]p(x) \leftrightarrow \exists y\,[x' = f(x), y' = a(x)y + b(x)\,\&\,q(x)]p(x) \\
\text{DS} & [x' = f\,\&\,q(x)]p(x) \leftrightarrow \forall t{\geq}0\,\big((\forall 0{\leq}s{\leq}t\,q(x+fs)) \rightarrow [x := x+ft]p(x)\big) \\
[':=] & [x' := f]p(x') \leftrightarrow p(f) \\
+' & (f(\bar{x}) + g(\bar{x}))' \;=\; (f(\bar{x}))' + (g(\bar{x}))' \\
\cdot' & (f(\bar{x}) \cdot g(\bar{x}))' \;=\; (f(\bar{x}))' \cdot g(\bar{x}) + f(\bar{x}) \cdot (g(\bar{x}))' \\
\circ' & [y := g(x)][y' := 1]\big((f(g(x)))' = (f(y))' \cdot (g(x))'\big)
\end{array}
$$

Figure 2. Differential equation axioms and differential axioms

tial equations. For example, the axioms in Fig. 2 have been used to implement an Ordinary Differential Equation solver based on logical deductions and have also been used to implement reasoning techniques based on differential invariants [7]. The CE proof rule allows for equational rewriting of equivalent subformulas, whereas CQ and CT allow for equational rewriting of equal terms.

2.3 Uniform Substitutions

Uniform substitutions are mappings from functions $f(\cdot)$ to terms, predicate symbols $p(\cdot)$ to formulas, quantifier symbols $C(\_)$ to formulas, and program constants a to programs where $\cdot$ is a reserved function symbol of arity zero and $\_$ a reserved quantifier symbol of arity zero. For example, $a \mapsto x := 0$ substitutes any occurrence of the program variable a with program $x := 0$. And $p(\cdot) \mapsto x \geq 0$ substitutes a predicate $p(\theta)$ with a formula $\theta \geq 0$ for any argument term θ. Logical deductions in dL may appeal to the truth-preserving nature of substitutions via the US proof rule (Fig. 1).

Example 2 (Admissible and Clashing Substitutions)**.** *Restricting the US proof rule to admissible uniform substitutions is necessary for preserving the soundness of the calculus. Consider the substitution and formula*

$$\sigma = \{a \mapsto x := x - 1, p \mapsto x \geq 0\}$$
$$\phi \equiv p \rightarrow [a]p.$$

If σ were admissible for ϕ (it is not!), then the US proof rule would allow a proof of $x \geq 0 \rightarrow [x := x - 1]x \geq 0$:

$$\frac{\dfrac{*}{p \rightarrow [a]p}}{x \geq 0 \rightarrow [x := x - 1]x \geq 0}$$

but this formula is clearly not valid. Conversely, consider the very similar substitution σ' and the formula φ:

$$\sigma' = \{a \mapsto x := x - 1, p(\cdot) \mapsto x \geq 0\}$$
$$\varphi \equiv [a]p(\bar{x})$$

for $\bar{x} = (x)$. Because σ' is φ-admissible, the US proof rule allows the deduction following

$$\frac{x \geq 0}{[x := x - 1]x \geq 0}$$

via a uniform substitution on the G proof rule.

Example 2 demonstrates that the US rule is not sound for arbitrary substitutions. A sound calculus must restrict uniform substitutions so that substitutions which introduce unsound deductions are not permitted. For this purpose, dL defines when a given substitution is *admissible* for a formula and restricts the US proof rule so that the rule is only applicable when the substitution σ is ϕ-admissible. The two cases in Example 2 demonstrate why admissibility of a substitution depends upon the formula to which a substitution is applied – a substitution may be sound for one formula and unsound for another.

The slight difference between the substitutions σ and σ' in Example 2 demonstrate the significance of the difference between p, $p(x)$, and $p(\bar{x})$. These three predicate symbols have different static semantics. The first symbol (p) has a nullary predicate symbol The second ($p(x)$) has a predicate symbol where the variable x may occur freely, and the third ($p(\bar{x})$) has a predicate symbol where any $x \in \bar{x}$ may occur freely. These free variables of p continue to be permitted in its replacement. Additional free variables are allowed by the US rule when admissible.

The definition of admissibility depends upon the static semantics of dL formulas, so this difference in the static semantics of p, $p(x)$, and $p(\bar{x})$ is crucial when determining whether a substitution is admissible.

The explication of admissibility for uniform substitutions in dL is critical for soundness but non-trivial (see [25] for details). Therefore, the results presented in this paper abstract over the particularities by simply assuming the existence of a mechanism for checking whether a given substitution is admissible for a given formula and assuming that there is therefore a sound implementation of the US proof rule. Readers interested in a constructive definition of admissibility for uniform substitutions in dL may consult Platzer (in particular, Fig. 1) [25].

Uniform substitutions map function, predicate, and quantifier symbols to terms and formulas, but do not map variables to variables. The KeYmaera X theorem prover implements both admissible uniform substitutions and uniform renaming.

2.4 Comparison with Other Approaches

There are many existing techniques for augmenting an existing logic with proof terms. This section discusses why we chose to design and implement a novel logic rather than some of the most prominent alternatives.

There are many reasons for implementing a new theorem prover – especially in the cyber-physical systems domain. KeYmaera X is designed as a platform for research on both automated and interactive theorem proving specifically for hybrid dynamical systems. Designing and implementing new tactics languages, proof construction GUIs, and other features is easier in a smaller system with significantly fewer lines of code, and KeYmaera X was specifically designed to support certain extensions (e.g., parallel proof search, control engineering-centric user interfaces) that Coq (for example) was not designed to support.

Proceeding from the premise that hybrid systems theorem proving benefits from a theorem proving system that is specifically tailored to differential dynamic logics, the primary benefit of the approach in this paper is that it is parsimonious with the metatheory of these logics. Both the syntax and semantics of LPdL are a straightforward extension of the semantics of differential dynamic logics.

The rationale for developing a custom theorem prover for differential dynamic logics apply equally to all of the alternatives discussed in this section. The following discussions of particular alternatives focus on more specific comparisons.

Encoding in a Proof Assistant. One alternative is encoding Fig. 1 and Fig. 2 in a proof assistant such as Coq [16] or Isabelle [17]. The Uniform Substitution algorithm implemented in KeYmaera X is constructive and is probably implementable in a proof assistant for a higher order logic, so this approach is certainly possible. If the proof assistant has proof terms, then those proof terms would serve our goal of adding proof terms to dL. Furthermore, this approach could be used to generate proof terms for proofs constructed in an independently implemented theorem prover such as KeYmaera X (e.g., by isolating a simulation of the operations in the KeYmaera X core using constructions in a hypothetical dL library for Coq or Isabelle).

The benefits of encoding dL in a proof assistant do not come for free. To achieve any benefit from this embedding, we would also need to formalize the soundness proof for dL within the proof assistant. Soundness proofs for hybrid systems are difficult, so a formalization of the soundness proof of dL would be greatly beneficial. However, this is almost certainly not the path of least resistance toward proof terms for dL because formalizing the soundness proof for dL would require considerable effort.

Even given a formalization of the soundness proof for dL, the benefit of a proof constructed in a proof assistant remains questionable because the KeYmaera X core is small. For example, although the Coq core is more thoroughly audited than the KeYmaera X core, it is also far larger (the Coq core is approx. 20000 lines of code and the KeYmaera X core is approx. 2000 lines).[2]

Logical Frameworks. Logical frameworks [9] provide a potential counter-point to the above observation that formalizing the soundness proof for dL would require considerable effort. Work toward a mechanization of Standard ML in Twelf [18] demonstrates that logical frameworks are particularly well-suited to reasoning about binding [12]; this strength is relevant in the context of dL because binding structure is at the heart of admissibility constraints on uniform substitutions. However, initial investigations suggest that the binding structure of hybrid programs is rich enough that encoding admissible uniform substitutions would require non-trivial effort. Furthermore, uniform substitution is only the first (and likely easiest) step of a mechanization of dL in Twelf, Beluga [19], etc. because obtaining soundness proofs would also require proving the local soundness of the axioms in Fig. 2.

3. The Logic of Proofs for Differential Dynamic Logic

This section presents the syntax and semantics for LPdL. Syntactically, the logic is the differential dynamic logic presented in [25], augmented with formulas of the form $e : \phi$ (where ϕ is a dL formula) whose intended meaning is that e serves as evidence for ϕ. Semantically, LPdL extends the semantics of dL with meanings for formulas of the form $e : \phi$.

[2] This argument is less strong for HOL Light [10] and Lean [5], both of which have implementations whose size and complexity is comparable to KeYmaera X.

The choice of proof terms presented in this section is motivated by the typical structure of proofs in dL. Proofs in dL combine equivalence/equational reasoning with uniform substitutions and uniform renamings. For example, consider the proof of $[x := 0 \cup x := 1]x \geq 0$ in Fig. 3. Each of the leafs of the proof is either an axiom of dL or else a tautology of $\text{FOL}_{\mathbb{R}}$. These leafs are obtained from the original problem by performing equivalence rewriting, modus ponens, and identifying uniform substitutions that translate the resulting subgoals into dL axioms. In this proof, uniform renaming is not necessary; however, renaming would be necessary for the formula $[y := 0 \cup y := 1]y \geq 0$ because the axiom for symbolically executing a discrete assignment mentions x instead of y.

3.1 Syntax

Definition 4 (Formulas)**.** *The formulas of* LPdL *are defined by extending the inductive definition of* dL *formulas given in Def. 3 with formulas of the form $e : \phi$, where ϕ is a formula of* dL *and e ranges over proof terms (defined below).*

Our definition of the grammar of LPdL formulas (e.g., the inclusion of dL formulas) is parsimonious with the Justification Logic tradition rather than the type theory tradition.

The formulas of LPdL as defined in Def. 4 augment the formulas of dL with an additional connective $e : \phi$. [3] This augmentation strictly extends the grammar of dL. Formulas such as $1 = 1 \wedge 2 = 2$ which do not contain proof terms remain formulas of LPdL. However, grammatical constructions of the form $e : e' : \phi$ (and $e : e' : e'' : \phi$, and so on) are *not* formulas of LPdL; i.e., proof terms provide evidence only for dL derivations – not for LPdL derivations. Although the authors are interested in extending LPdL to properly treat formulas of these forms, our immediate motivations for explicitly representing proofs do not require such a rich language.

Pure LPdL formulas are formulas that do not allow the use of dL connectives (such as $(e : \phi) \wedge (d : \phi)$). Pure LPdL formula either a formula of dL, or a formula of the form $e : \phi$ where e is a proof term and ϕ is a formula of dL.

Example 3 (LPdL formulas and non-formulas)**.** *The following are non-pure formulas of* LPdL *(where e, d are proof terms and ϕ, ψ are* dL *formulas):*

- $(e : \phi) \wedge (d : \psi)$
- $(e : \phi) \rightarrow (d : \psi)$
- $[x := 0](j_{1=1} : 1 = 1)$

whereas $e : (\phi \wedge \psi)$ is a pure formula of LPdL*:*

In most of this paper we are concerned only with pure LPdL formulas, because these are the formulas that correspond to judgements

$$e \text{ is a } \mathsf{dL} \text{ proof of } \phi$$

where ϕ is a formula of dL; i.e., pure LPdL formulas are *just* proof certificates for dL derivations. In particular, our axioms and proof rules focus only on the pure fragment of LPdL. It may be useful to axiomatize non-pure LPdL in the future; only application might be allowing the prover core to pass around multiple proven results directly instead of having to bundle up proven results using conjunctions. However, we leave these questions as future work and instead focus on parsimoniously extending dL with certificates for dL proofs.

A complete definition of the objects that may stand in for e occupies the remainder of this subsection.

[3] It is not misleading to think of _ : _ as a binary function mapping proofs terms and dL formulas to LPdL formulas.

$$\text{MP}\ \dfrac{\text{US}\ \dfrac{[\cup]\ \dfrac{}{[a\cup b]p(\bar{x})\leftrightarrow[a]p(\bar{x})\wedge[b]p(\bar{x})}}{[x:=0\cup x:=1]x\geq 0\leftrightarrow[x:=0]x\geq 0\wedge[x:=1]x\geq 0}\qquad \Delta}{[x:=0\cup x:=1]x\geq 0}$$

where $\Delta =$

$$\text{Prop}\ \dfrac{\text{Prop}\ \dfrac{\text{MP}\ \dfrac{\text{US}\ \dfrac{[:=]\ \dfrac{}{[x:=t]p(t)\leftrightarrow p(x)}}{[x:=0]x\geq 0\leftrightarrow 0\geq 0}\qquad \text{Prop}\ \dfrac{\mathbb{R}\ \dfrac{}{0\geq 0}}{[x:=0]x\geq 0\leftrightarrow 0\geq 0\rightarrow[x:=0]x\geq 0}}{[x:=0]x\geq 0}\qquad \text{MP, Prop, US}\ \dfrac{[:=]\ \dfrac{}{[x:=t]p(t)\leftrightarrow p(x)}\qquad \mathbb{R}\ \dfrac{}{1\geq 0}}{[x:=1]x\geq 0}}{[x:=0]x\geq 0\wedge[x:=1]x\geq 0}}{([x:=0\cup x:=1]x\geq 0\leftrightarrow[x:=0]x\geq 0\wedge[x:=1]x\geq 0)\rightarrow[x:=0\cup x:=1]x\geq 0}$$

Figure 3. A proof of $[x := 0 \cup x := 1]x \geq 0$ in the uniform substitution calculus of dL. The proof of Δ is slightly abbreviated for readability; the proof for the $x := 1$ case is very similar to the proof of the $x := 0$ case.

Definition 5 (Proof Terms)**.** *Proof terms are defined by this grammar (with e, d as proof terms, c ranging over sets of proof constants, σ as a uniform substitution, $\mathscr{B}$ as a uniform renaming, and ϕ as dL formulas as defined in Def. 3).*

$e, d ::= c_\phi$	*Proof Constants*
$\mid\ e \wedge d$	*Conjunctions*
$\mid\ e \bullet d$	*Implicative Application*
$\mid\ e \bullet_{\leftarrow} d \mid e \bullet_{\rightarrow} d$	*Directional Equivalence Application*
$\mid\ \sigma e \mid \mathscr{B} e$	*Uniform Substitutions and Renaming*
$\mid\ CT_\sigma e \mid CQ_\sigma e \mid CE_\sigma e$	*Equivalence/equational Reasoning*

Proof terms are the syntactic objects of LPdL corresponding to deductions in dL.

Atomic/Axiomatic Terms. Proof constants serve as evidence for dL axioms and $\text{FOL}_\mathbb{R}$ tautologies. In this paper, we consider two sets of proof constants – i_A where A is any dL axiom and j_T where T is any tautology of $\text{FOL}_\mathbb{R}$. We use c_ϕ whenever we mean to discuss both of these sets of proof constants.

The separation of atomic proof terms indexed by concrete axioms into disjoint sets is motivated by practical concerns that arise when implementing a theorem prover for hybrid systems.

The first benefit of separating atomic proof terms into sets is a clear separation between axiomatic and real arithmetic reasoning. Although the first-order theory of real arithmetic is decidable, the problem has extreme complexity. Furthermore, KeYmaera X (as well as other theorem provers) that utilize decision procedures for real arithmetic are typically sound only modulo the soundness of an external implementation of the decision procedure being used. Distinguishing computationally trivial appeals to axioms from possibly expensive appeals to arithmetic decision procedures isolates a natural extension point for incorporating certificates of arithmetic facts e.g., by extracting witnesses from an implementation of a Coq implementation of the Cylindrical Algebraic Decomposition algorithm [15] or by using approaches such as [29] that are amenable to certificate generation). Isolating real arithmetic facts from axiomatic facts also makes it very easy to identify appeals to $\text{FOL}_\mathbb{R}$ tautologies in proofs, which could be useful for identifying when the reproducibility of a proof is going to depend upon possibly expensive appeals to a decision procedure.

The second benefit of separating atomic proof terms into disjoint sets is that it enables code-reuse when implementing conservative extensions of an already supported logic but also disallows unsound combinations of logics. For example, dL contains axioms that are unsound for its game-theoretic variant dGL [24] so an implementation of a dGL theorem prover on top of KeYmaera X should ensure that dGL proofs only make use of dL axioms that are sound in dGL.[4]

Conjunctions. The $\wedge$ operator allows for the creation of evidence for conjunctive formulas. If $e : \phi$ and $d : \psi$ then $(e \wedge d) : \phi \wedge \psi$. This connective is also not strictly necessary if dL contains appropriate propositional axioms but is useful because many dL axioms contain conjunctions. Conjunctions represent the exact structure of a proof, so LPdL excludes the $+$ operator found in some Justification Logics ([3, Part II]) because we are interested only in single conclusion proof systems. From an implementation perspective, the most interesting multi-conclusion extensions are those that could serve as a category of values for a proof search specification language capable of describing decidable but non-deterministic forward proof search procedures.

Implications. The $\bullet$ operator allows the use of evidence of an implication, and corresponds to the modus ponens proof rule. For example, if $e : \psi \rightarrow \phi$ and $d : \psi$ then $e \bullet d : \phi$. This operator corresponds to the application operator of the Logic of Proofs and corresponds to application in the Simply Typed Lambda Calculus.

Equivalence Rewriting. The $\bullet_{\leftarrow}$ and $\bullet_{\rightarrow}$ operators are similar to the implication operator, but are used for equivalences instead of implications. The subscript on the operator indicates the direction in which the equivalence should be used. For example, if $e : \psi \leftrightarrow \phi$ and $d : \phi$ then $e \bullet_{\leftarrow} d : \psi$. The $\bullet_{\leftarrow}$ and $\bullet_{\rightarrow}$ operators are not strictly necessary because they can be replaced with axioms. If

$$i : (\phi \leftrightarrow \psi) \rightarrow (\phi \rightarrow \psi),$$
$$e : \phi \leftrightarrow \psi, \text{and}$$
$$d : \phi$$

then $(i \bullet e) \bullet d : \psi$. These operators are included because equivalence rewriting is a fundamental and pervasive operation in axiomatic proofs, so even the constant multiplier on the length of proof terms is enough to motivate the addition of operators.

Substitution and Renaming. Uniform substitution and renaming are essential parts of dL proofs and are witnessed by proof terms of the form σe and $\mathscr{B} e$, where σ and $\mathscr{B}$ are uniform substitutions and renamings respectively. Uniform substitutions do not map variables to variables, but variable renamings are necessary whenever a proof contains variables that do not occur in axioms. For example, a

[4] This motivation is informed by plans for future work; in this paper we present a logic of proof terms for only dL.

proof of $[a := 12]a = 12 \leftrightarrow 12 = 12$ is just a uniform renaming of x to a in the $[:=]$ axiom. KeYmaera X allows explicit uniform renamings during proving, and these explicit renamings are captured by the $\mathscr{B}$ proof terms.

Equivalence and Equational Reasoning. The CT_σ, CQ_σ, and CE_σ operators correspond to uniform substitution instances of the contextual equation and equivalence proof rules of d$\mathcal{L}$ (CT, CQ, and CE). For example, the proof term $\text{CT}_{\{c(\cdot)\mapsto\cdot^2, f(\cdot)\mapsto b, g(\cdot)\mapsto a\}}$ serves as evidence for the $\{c(\cdot) \mapsto \cdot^2, f(\cdot) \mapsto a, g(\cdot) \mapsto b\}$ uniform substitution instance of the CT proof rule:

$$\frac{a = b}{a^2 = b^2}$$

3.2 Semantics

The semantics of LPd$\mathcal{L}$ formulas extends the semantics of the uniform substitution calculus given in Section 2.2. As in d$\mathcal{L}$, interpretations I in LPd$\mathcal{L}$ give meaning to program constants, function, predicate and quantifier symbols [25].

Definition 6 (LPd$\mathcal{L}$ Semantics)**.** *The semantics of an* LPd$\mathcal{L}$ *formula χ is defined with respect to an interpretation I as the subset $[\![\chi]\!]^I \subseteq S$ of states in which χ is true and is defined inductively as follows (where i_A, j_T ranges over proof constants, e, d range over proof terms, and ϕ, ψ range over* d$\mathcal{L}$ *formulas):*

- $[\![\phi]\!]^I = [\![\phi]\!]^I_{\mathsf{d}\mathcal{L}}$ *where $[\![\cdot]\!]_{\mathsf{d}\mathcal{L}}$ is the denotation of* d$\mathcal{L}$ *given in [25]. The meaning of connectives $\wedge, \neg, \exists, [\cdot], \langle\cdot\rangle$ is also as in* d$\mathcal{L}$*, e.g., $[\![\varphi \wedge \chi]\!]^I = [\![\varphi]\!]^I \cap [\![\chi]\!]^I$*
- $[\![i_A : A]\!]^I = S$ *for* d$\mathcal{L}$ *axioms A*
- $[\![j_T : T]\!] = S$ *for FOL$_\mathbb{R}$ tautologies T*
- $[\![e \wedge d : \phi \wedge \psi]\!]^I = [\![e : \phi]\!]^I \cap [\![d : \psi]\!]^I = \{v \in S : v \in [\![e : \phi]\!]^I$ *and* $v \in [\![d : \psi]\!]^I\}$
- $[\![e \bullet d : \phi]\!]^I = \bigcup_\psi [\![e : (\psi \to \phi)]\!]^I \cap [\![d : \psi]\!]^I$
  $= \{v \in S : v \in [\![e : (\psi \to \phi)]\!]^I$ *and* $v \in [\![d : \psi]\!]^I$ *for some* $\psi\}$
- $[\![e \bullet_\leftarrow d : \phi]\!]^I = \bigcup_\psi [\![e : (\phi \leftrightarrow \psi)]\!]^I \cap [\![d : \psi]\!]^I$
  $= \{v \in S : v \in [\![e : (\phi \leftrightarrow \psi)]\!]^I$ *and* $v \in [\![d : \psi]\!]^I$ *for some* $\psi\}$
- $[\![e \bullet_\rightarrow d : \phi]\!]^I = \bigcup_\psi [\![e : (\psi \leftrightarrow \phi)]\!]^I \cap [\![d : \psi]\!]^I$
  $= \{v \in S : v \in [\![e : \psi \leftrightarrow \phi]\!]^I$ *and* $v \in [\![d : \psi]\!]^I$ *for some* $\psi\}$
- $[\![\sigma e : \sigma\phi]\!]^I = [\![e : \phi]\!]^I$ *if σ is admissible for ϕ*
  $= \{v \in S : v \in [\![e : \phi]\!]^I$ *and σ is admissible for ϕ*$\}$.
- $[\![\mathscr{B}e : \mathscr{B}\phi]\!]^I = [\![e : \phi]\!]^I$ *if $\mathscr{B}$ is a uniform renaming of ϕ*
  $= \{v \in S : v \in [\![e : \phi]\!]^I$ *and $\mathscr{B}$ is a uniform renaming of ϕ*$\}$
- $[\![\mathit{CT}_\sigma e : \sigma(c(f(\bar{x})) = c(g(\bar{x})))]\!]^I = [\![\sigma e : \sigma(f(\bar{x}) = g(\bar{x}))]\!]^I$
- $[\![\mathit{CQ}_\sigma e : \sigma(p(f(\bar{x})) \leftrightarrow p(g(\bar{x})))]\!]^I =$
  $[\![\sigma e : \sigma(f(\bar{x}) = g(\bar{x}))]\!]^I$
- $[\![\mathit{CE}_\sigma e : \sigma(C(p(\bar{x})) \leftrightarrow C(q(\bar{x})))]\!]^I =$
  $[\![\sigma e : \sigma(p(\bar{x}) \leftrightarrow q(\bar{x}))]\!]^I$

Undefined cases are empty.[5]

Note that the meaning of $e : \phi$ is always either S or $\emptyset$. Only LPd$\mathcal{L}$ formulas involving proper d$\mathcal{L}$ subformulas have state-dependent truth.

We do not prove soundness in this paper; instead, we establish a correctness result that is more useful in our context: whenever $e : \phi$ is a theorem of LPd$\mathcal{L}$, we can construct a d$\mathcal{L}$ proof of ϕ, which implies that ϕ is valid. (The advantages of this result are discussed in the introduction and in later sections.) In this section, we take a similar approach. Instead of establishing a direct connection between the semantics and axioms and proof rules of LPd$\mathcal{L}$, we instead establish a projection from the semantics of LPd$\mathcal{L}$ to the semantics of d$\mathcal{L}$.

Theorem 1 (Correctness of Proof Term Valuation)**.** *Consider any interpretation I, $v \in S$ and* d$\mathcal{L}$ *formula ϕ. If $v \in [\![e : \phi]\!]^I_{\mathsf{LPd}\mathcal{L}}$ then $v \in [\![\phi]\!]^I_{\mathsf{d}\mathcal{L}}$.*

Note that Theorem 1 pertains only to pure LPd$\mathcal{L}$ formulas; i.e., LPd$\mathcal{L}$ formulas of the form $e : \phi$ where e is a proof term and ϕ is a formula of d$\mathcal{L}$.

Proof. The proof proceeds by induction on the structure of e, simultaneously for all ϕ.

Axiomatic Terms. Suppose $v \in [\![i_\psi : \phi]\!]^I_{\mathsf{LPd}\mathcal{L}}$. By Def. 6, it must be that ϕ is ψ and ψ is an axiom of d$\mathcal{L}$. Therefore, ϕ is an axiom of d$\mathcal{L}$ so by soundness of d$\mathcal{L}$, $[\![\phi]\!]^I_{\mathsf{d}\mathcal{L}} = S$. Finally, $v \in S$.

FOL$_\mathbb{R}$ Tautology Terms. Suppose $v \in [\![j_\psi : \phi]\!]^I_{\mathsf{LPd}\mathcal{L}}$. By Def. 6, it must be that ϕ is ψ and ψ is a tautology of FOL$_\mathbb{R}$. Therefore, ϕ is a tautology of FOL$_\mathbb{R}$ so by soundness of d$\mathcal{L}$, $[\![\phi]\!]^I_{\mathsf{d}\mathcal{L}} = S$. Finally, $v \in S$.

Case $e \wedge d$. Suppose $v \in [\![e \wedge d : \phi]\!]^I_{\mathsf{LPd}\mathcal{L}}$. Inspecting the cases of Def. 6, it must be that

$$\phi = \varphi \wedge \psi$$

for some φ, ψ such that

$$v \in [\![e : \varphi]\!]^I_{\mathsf{LPd}\mathcal{L}} \tag{1}$$

$$v \in [\![d : \psi]\!]^I_{\mathsf{LPd}\mathcal{L}} \tag{2}$$

Applying the inductive hypothesis at (1) and (2), we have $v \in [\![\varphi]\!]^I_{\mathsf{d}\mathcal{L}}$ and $v \in [\![\psi]\!]^I_{\mathsf{d}\mathcal{L}}$. Therefore, $v \in [\![\varphi]\!]^I_{\mathsf{d}\mathcal{L}}$ and $v \in [\![\psi]\!]^I_{\mathsf{d}\mathcal{L}}$ from which it follows that

$$v \in [\![\varphi]\!]^I_{\mathsf{d}\mathcal{L}} \cap [\![\psi]\!]^I_{\mathsf{d}\mathcal{L}} = [\![\varphi \wedge \psi]\!]^I_{\mathsf{d}\mathcal{L}}$$

by the definition of the semantics of d$\mathcal{L}$ [25].

Case $e \bullet d$. Suppose $v \in [\![e \bullet d : \phi]\!]^I_{\mathsf{LPd}\mathcal{L}}$. By Def. 6 we know that

$$v \in [\![e : \psi \to \phi]\!]^I_{\mathsf{LPd}\mathcal{L}}$$

$$v \in [\![d : \psi]\!]^I_{\mathsf{LPd}\mathcal{L}}$$

for some ψ. Applying the inductive hypothesis to these facts establishes

$$v \in [\![\psi \to \phi]\!]^I_{\mathsf{d}\mathcal{L}}$$

$$v \in [\![\psi]\!]^I_{\mathsf{d}\mathcal{L}}$$

From these facts, a classical propositional encoding of $\psi \to \phi$, and elementary theorems of set theory, we obtain that

$$v \in ([\![\psi]\!]^I_{\mathsf{d}\mathcal{L}})^{\mathrm{C}} \cup [\![\phi]\!]^I_{\mathsf{d}\mathcal{L}}$$

(where X^{C} is the set complement $S \setminus X$ of X) which, because $v \in [\![\psi]\!]^I_{\mathsf{d}\mathcal{L}}$, implies $v \in [\![\phi]\!]^I_{\mathsf{d}\mathcal{L}}$.

Case $e \bullet_\leftarrow d$ and $e \bullet_\rightarrow d$. Similar to $e \bullet d$.

Case σe. Suppose that $v \in [\![\sigma e : \phi]\!]^I_{\mathsf{LPd}\mathcal{L}}$. Then by inspection of the cases of Def. 6, $\phi = \sigma(\phi')$ and $v \in [\![e : \phi']\!]^I_{\mathsf{LPd}\mathcal{L}}$. Applying the inductive hypothesis to this fact establishes $v \in [\![\phi']\!]^I_{\mathsf{d}\mathcal{L}}$. So because σ is, by Def. 6, an admissible substitution for ϕ' we have $v \in [\![\sigma(\phi')]\!]^I_{\mathsf{d}\mathcal{L}} = [\![\phi]\!]^I_{\mathsf{d}\mathcal{L}}$.

The remaining cases are similar. □

[5] E.g., $[\![(e \wedge d) : \phi]\!]^I = \emptyset$ whenever ϕ is not of the appropriate form. Likewise for the other cases.

3.3 Axioms and Proof Rules of the Logic of Proofs for Differential Dynamic Logic

Axioms governing the construction of proof terms allow for the derivation of proof terms that describe proofs by substitution, uniform renaming, uniform substitution, and appeals to axioms and tautologies. This is sufficient to describe proofs constructed by the uniform substitution calculus of dℒ, and by extension most proofs constructed by the KeYmaera X theorem prover. The KeYmaera X theorem prover also contains a propositional sequent calculus and skolemization, so in practice some proofs constructed by KeYmaera X may not have proof terms in LPdℒ. However, there exist proof term calculi for propositional sequent calculi, so this paper focuses on the portions of KeYmaera X proofs that do not yet have an easily adaptable proof term calculus.

After stating the axioms and proof rules of LPdℒ in Def. 7, we describe how each is used to construct proof terms for typical constructions.

Unlike dℒ, LPdℒ does not use uniform substitutions. Therefore, the objects described in the following definition are axiom schemata and proof rules – not just formulas or pairs of formulas.

Definition 7 (Axioms of LPdℒ)**.** *The following are axioms of* LPdℒ*, where φ, ψ range over* LPdℒ *formulas, and c, f, g are function symbols and p, q are predicate symbols, and C a quantifier symbol.*

$$\phi \qquad \text{(dℒ Axiom)}$$

$$i_A : A \qquad \text{(dℒ Constants)}$$

$$j_T : T \qquad \text{(FOL}_{\mathbb{R}}\text{ Constants)}$$

$$\frac{e : \phi \qquad d : \psi}{(e \wedge d) : (\phi \wedge \psi)} \qquad \text{(And)}$$

$$\frac{e : (\phi \to \psi) \qquad d : \phi}{e \bullet d : \psi} \qquad \text{(Application)}$$

$$\frac{e : (\phi \leftrightarrow \psi) \qquad d : \phi}{e \bullet_{\to} d : \psi} \qquad \text{(Right Equivalence)}$$

$$\frac{e : (\phi \leftrightarrow \psi) \qquad d : \psi}{e \bullet_{\leftarrow} d : \phi} \qquad \text{(Left Equivalence)}$$

$$\frac{e : \phi}{\sigma e : \sigma(\phi)} \qquad \text{(US Proof Term)}$$

$$\frac{e : \phi}{\mathscr{B}e : \mathscr{B}(\phi)} \qquad \text{(Renaming)}$$

$$\frac{\sigma e : \sigma(f(\bar{x}) = g(\bar{x}))}{CT_{\sigma} e : \sigma(c(f(\bar{x}) = c(g(\bar{x}))))} \qquad (\text{CT}_{\sigma})$$

$$\frac{\sigma e : \sigma(f(\bar{x}) = g(\bar{x}))}{CQ_{\sigma} e : \sigma(p(f(\bar{x}) \leftrightarrow p(g(\bar{x}))))} \qquad (\text{CQ}_{\sigma})$$

$$\frac{\sigma e : \sigma(p(\bar{x}) \leftrightarrow q(\bar{x}))}{CE_{\sigma} e : \sigma(C(p(\bar{x}) \leftrightarrow C(q(\bar{x}))))} \qquad (\text{CE}_{\sigma})$$

and where the rules US Proof Term, CT_{σ}, CQ_{σ}, and CE_{σ} are applicable only whenever σ is admissible for the dℒ formulas to which it is applied, and only whenever σ has no free variables. The set of free variables of a substitution is defined in [25]. The formula ϕ in rule dℒ Axiom needs to be a dℒ formula provable in dℒ.

The axioms in Def. 7 correspond to the intuitive meanings for proof terms given in Section 3.1.

Proof Constant Axioms. The axiomatization of dℒ is included in LPdℒ in the form of including all provable dℒ formulas (rule dℒ Axiom). Proof constants i_A and j_T internalize evidence for dℒ axioms and $\text{FOL}_{\mathbb{R}}$ tautologies. For example,

$$i_{[a;b]p(\bar{x}) \leftrightarrow [a][b]p(\bar{x})} : [a;b]p(\bar{x}) \leftrightarrow [a][b]p(\bar{x}), \text{ and}$$
$$j_{x \geq 0 \to x^2 \geq 0} : x \geq 0 \to x^2 \geq 0$$

are both axioms of LPdℒ. For brevity, we often use the names of axioms as subscripts instead of the axioms themselves. For example,

$$i_{[\cup]} : [a \cup b]p(\bar{x}) \leftrightarrow [a]p(\bar{x}) \wedge [b](\bar{x}).$$

Conjunction Proof Rule. The And proof rule enables construction of compound proof terms that serve as evidence for conjunctions. Constructing a proof term that allows for left and right projections of a conjunction is also possible using dℒ axioms and Application axiom, so these are not included as primitives. Unlike dℒ, proof term axioms and proof rules are schematic, so

$$\frac{d : x = y \qquad e : y = z}{(d \wedge e) : x = y \wedge y = z}$$

is a derivation in LPdℒ.

Application Proof Rules. The Application proof rule enables construction of proof terms that correspond to the use of the Modus Ponens rule in dℒ; for example,

$$\frac{d : p(x) \to q(x) \qquad e : p(x)}{e \bullet d : q(x)}$$

is a derivation in LPdℒ. The Left Equivalence and Right Equivalence rules are definable in terms of the Application rule at the expense of more verbose proof terms.

Uniform Substitution Proof Rule. The US Proof Term axiom allows the construction of evidence that appeals to uniform substitutions. Similarly, uniform renaming is evidenced by Renaming. A schematic sequent calculus for dℒ is definable using uniform substitutions [7] and proof terms can be assigned to each of these proof rules. For example, the proof terms for the sequent calculus proof rule

$$\frac{\vdash [\alpha]\varphi \qquad \vdash [\beta]\varphi}{\vdash [\alpha \cup \beta]\varphi}$$

are $\sigma i_{[\cup]} \bullet_{\to} e : [\alpha]\varphi \wedge [\beta]\varphi$ where $e : [\alpha \cup \beta]\varphi$ and $\sigma = \{a \mapsto \alpha, b \mapsto \beta, p(\cdot) \mapsto \varphi\}$.

Equivalence/Equational Proof Rules. The CT_{σ}, CQ_{σ}, and CE_{σ} proof rules combine uniform substitutions with the proof rules CT, CQ, and CE from dℒ.

Example 4 demonstrates how these axioms and proof rules are combined with the axioms and uniform substitutions of dℒ to construct witnesses for dℒ proofs by constructing a proof term corresponding to the previous example.

4. Converting LPdℒ Proof Terms into dℒ Proofs

We say that $\vdash_{\text{LPdℒ}} \phi$ whenever there is a proof of ϕ in LPdℒ, and we say that $\vdash_{\text{dℒ}} \phi$ whenever there is a proof of ϕ in dℒ.

Lemma 1 (Inversion)**.** *The following are facts about* LPdℒ*:*

- *If $\vdash_{\text{LPdℒ}} i_{\phi} : \psi$ then ϕ is ψ and ϕ is an axiom of dℒ.*
- *If $\vdash_{\text{LPdℒ}} j_{\phi} : \psi$ then ϕ is ψ and ϕ is a tautology of $FOL_{\mathbb{R}}$.*
- *If $\vdash_{\text{LPdℒ}} e \wedge d : \phi$ then ϕ is $(\chi \wedge \psi)$ where $\vdash_{\text{LPdℒ}} e : \chi$ and $\vdash_{\text{LPdℒ}} d : \psi$.*
- *If $\vdash_{\text{LPdℒ}} e \bullet d : \phi$ then $\vdash_{\text{LPdℒ}} e : \psi \to \phi$ and $\vdash_{\text{LPdℒ}} d : \psi$ for some ψ.*
- *If $\vdash_{\text{LPdℒ}} e \bullet_{\leftarrow} d : \phi$ then $\vdash_{\text{LPdℒ}} e : \phi \leftrightarrow \psi$ and $\vdash_{\text{LPdℒ}} d : \psi$ for some ψ.*
- *If $\vdash_{\text{LPdℒ}} e \bullet_{\to} d : \phi$ then $\vdash_{\text{LPdℒ}} e : \psi \leftrightarrow \phi$ and $\vdash_{\text{LPdℒ}} d : \psi$ for some ψ.*

$$\dfrac{\dfrac{\text{d}\mathcal{L}\text{ Constants}\ \dfrac{}{i_{[\cup]}\colon [a\cup b]p(\bar{x})\leftrightarrow [a]p(\bar{x})\wedge[b]p(\bar{x})}}{\text{US Proof Term}\ \dfrac{}{\sigma_1 i_{[\cup]}\colon [x:=0\cup x:=1]x\geq 0\leftrightarrow [x:=0]x\geq 0\wedge[x:=1]x\geq 0}}\quad \Delta_\wedge}{\text{Left Equivalence}\ \dfrac{}{\sigma_1 i_{[\cup]}\bullet\leftarrow \underbrace{((\sigma_2 i_{[:=]}\bullet\leftarrow j_{0\geq 0})\wedge(\sigma_3 i_{[:=]}\bullet\leftarrow j_{1\geq 0}))}_{e_\wedge}\colon [x:=0\cup x:=1]x\geq 0}\quad e_\wedge : [x:=0]x\geq 0\wedge[x:=1]x\geq 0}$$

where $\Delta_\wedge$ is

$$\dfrac{\dfrac{\dfrac{}{i_{[:=]}\colon [x:=t]p(x)\leftrightarrow p(t)}}{\sigma_2 i_{[:=]}\colon [x:=0]x\geq 0\leftrightarrow x\geq 0}\quad \dfrac{}{j_2 : 0\geq 0}}{\text{And}\ \dfrac{\sigma_2 i_{i[:=]}\bullet\leftarrow j_2 : [x:=0]x\geq 0}{\underbrace{((\sigma_2 i_{[:=]}\bullet\leftarrow j_{0\geq 0})\wedge(\sigma_3 i_{[:=]}\bullet\leftarrow j_{1\geq 0}))}_{e_\wedge} : [x:=0]x\geq 0\wedge[x:=1]x\geq 0}\quad \sigma_3 i_{[:=]}\bullet\leftarrow j_3 : [x:=1]x\geq 0}\quad \text{Left Equivalence}\ \dfrac{\text{US Proof Term}\ \dfrac{\text{d}\mathcal{L}\text{ Constants}\ \dfrac{}{i_{[:=]}\colon [x:=t]p(x)\leftrightarrow p(t)}}{\sigma_3 i_{[:=]}\colon [x:=1]x\geq 0\leftrightarrow x\geq 0}\quad \text{FOL}_{\mathbb{R}}\text{ Constants}\ \dfrac{}{j_3 : 1\geq 0}}{\sigma_3 i_{[:=]}\bullet\leftarrow j_3 : [x:=1]x\geq 0}}$$

Example 4 (A Simple Proof Term)**.** *A proof of*

$$(\sigma_1 i_{[\cup]}\bullet\leftarrow((\sigma_2 i_{[:=]}\bullet\leftarrow j_{0\geq 0})\wedge(\sigma_3 i_{[:=]}\bullet\leftarrow j_{1\geq 0}))) : [x:=0\cup x:=1]x\geq 0$$

where

$$\begin{aligned}\sigma_1 &\equiv \{a\mapsto x:=1, b\mapsto x:=1, p(\cdot)\mapsto x\geq 0\}\\ \sigma_2 &\equiv \{t\mapsto 0, p(\cdot)\mapsto \cdot\geq 0\}\\ \sigma_3 &\equiv \{t\mapsto 1, p(\cdot)\mapsto \cdot\geq 0\}\\ i_{[\cup]} &\equiv i_{[a\cup b]p(\bar{x})\leftrightarrow[a]p(\bar{x})\wedge[b]p(\bar{x})}\\ i_{[:=]} &\equiv i_{[x:=t]p(x)\leftrightarrow p(t)}\end{aligned}$$

is given above. Intuitively, this property states that if x nondeterministically takes on 0 or 1, then $x\geq 0$. The proof proceeds by symbolic decomposition of the hybrid program $x:=0\cup x:=1$ using axioms of d$\mathcal{L}$*. Uniform substitution instances of the relevant symbolic decomposition axioms are necessary in order to complete the proof. Labels on the left side of the proof of Δ are elided for readability, but exactly match the labels on the right side.*

- *If $\vdash_{\text{LPd}\mathcal{L}} CT_\sigma e : \phi$ then ϕ is $\sigma(c(f(\bar{x}))=c(g(\bar{x})))$, $\vdash_{\text{LPd}\mathcal{L}}\sigma e : \sigma(f(\bar{x})=g(\bar{x}))$, and σ is admissible on all formulas to which it is applied and $FV(\sigma)=\emptyset$.*[6]
- *If $\vdash_{\text{LPd}\mathcal{L}} CQ_\sigma e : \phi$ then ϕ is $\sigma(p(f(\bar{x}))\leftrightarrow p(g(\bar{x})))$, $\vdash_{\text{LPd}\mathcal{L}}\sigma e : \sigma(f(\bar{x})=g(\bar{x}))$, and σ is admissible on all formulas to which it is applied and $FV(\sigma)=\emptyset$.*
- *If $\vdash_{\text{LPd}\mathcal{L}} CE_\sigma e : \phi$ then ϕ is $\sigma(C(p(\bar{x}))\leftrightarrow C(q(\bar{x})))$, $\vdash_{\text{LPd}\mathcal{L}}\sigma e : \sigma(p(\bar{x})\leftrightarrow q(\bar{x}))$, and σ is admissible on all formulas to which it is applied and $FV(\sigma)=\emptyset$.*
- *If $\vdash_{\text{LPd}\mathcal{L}}\sigma e : \phi$ then $\vdash_{\text{LPd}\mathcal{L}} e : \phi'$ and $\sigma(\phi')=\phi$ for some ϕ' such that σ is admissible for ϕ'.*
- *If $\vdash_{\text{LPd}\mathcal{L}}\mathscr{B}e : \phi$ then $\vdash_{\text{LPd}\mathcal{L}} e : \phi'$ and $\mathscr{B}(\phi')=\phi$ for some ϕ'.*

Proof. The proof involves a straightforward induction involving inspection of the conclusions of LPd$\mathcal{L}$ axioms. □

Theorem 2 (Proof terms justify theorems)**.** *Let e be a proof term and ϕ a* d$\mathcal{L}$ *formula. If $\vdash_{\text{LPd}\mathcal{L}} e : \phi$ then $\vdash_{\text{d}\mathcal{L}}\phi$.*

Proof. The proof involves the construction of a d$\mathcal{L}$ proof corresponding to the proof term e. We proceed by induction on the structure of e.

Case i_A**.** Suppose that $\vdash_{\text{LPd}\mathcal{L}} i_A : \phi$. By Lemma 1, $\phi=A$ and is an axiom of d$\mathcal{L}$. Therefore, $\vdash_{\text{d}\mathcal{L}}\phi$.

Case j_T**.** Suppose that $\vdash_{\text{LPd}\mathcal{L}} i_A : \phi$. By Lemma 1, $\phi=A$ and is a tautology of $\text{FOL}_{\mathbb{R}}$. Therefore, $\vdash_{\text{d}\mathcal{L}}\phi$.

[6] The set, $FV(\sigma)$, of free variables of a substitution σ is defined in [25]

Case $e\wedge d$**.** Suppose that $e\wedge d : \phi$. By Lemma 1,

$$\phi=\chi\wedge\psi$$

and

$$\vdash_{\text{LPd}\mathcal{L}} e : \chi \tag{3}$$
$$\vdash_{\text{LPd}\mathcal{L}} d : \psi \tag{4}$$

Applying the inductive hypothesis to (3) and (4) establishes that

$$\vdash_{\text{d}\mathcal{L}}\chi \tag{5}$$
$$\vdash_{\text{d}\mathcal{L}}\psi \tag{6}$$

The schematic proof rule

$$(\wedge\text{R})\ \dfrac{\varphi\quad\Omega}{\varphi\wedge\Omega}$$

where φ and Ω are any d$\mathcal{L}$ formulas that are derivable in d$\mathcal{L}$ using the propositional tautology $\varphi\to\Omega\to\varphi\wedge\Omega$ and MP. From (5) and (6), andR derives $\vdash_{\text{d}\mathcal{L}}\chi\wedge\psi$.

Case $e\bullet d$**.** Suppose that $\vdash_{\text{LPd}\mathcal{L}} e\bullet d : \phi$. By Lemma 1,

$$\vdash_{\text{LPd}\mathcal{L}} e : \psi\to\phi \tag{7}$$
$$\vdash_{\text{LPd}\mathcal{L}} d : \psi \tag{8}$$

Applying the inductive hypothesis to (7) and (8) establishes that

$$\vdash_{\text{d}\mathcal{L}}\psi\to\phi \tag{9}$$
$$\vdash_{\text{d}\mathcal{L}}\psi \tag{10}$$

from which MP derives $\vdash_{\text{d}\mathcal{L}}\phi$.

Case $e \bullet_{\rightarrow} d$**.** Suppose $\vdash_{\mathsf{LPd}\mathcal{L}} e \bullet_{\rightarrow} d : \phi$. By Lemma 1,

$$\vdash_{\mathsf{LPd}\mathcal{L}} e : \psi \leftrightarrow \phi \tag{11}$$

$$\vdash_{\mathsf{LPd}\mathcal{L}} d : \psi \tag{12}$$

are provable in $\mathsf{LPd}\mathcal{L}$. Applying the inductive hypothesis to (11) and (12) establishes

$$\vdash_{\mathsf{d}\mathcal{L}} \psi \leftrightarrow \phi \tag{13}$$

$$\vdash_{\mathsf{d}\mathcal{L}} \psi \tag{14}$$

Note that

$$\vdash_{\mathsf{d}\mathcal{L}} (\psi \leftrightarrow \phi) \rightarrow (\psi \rightarrow \phi)$$

has a proof in $\mathsf{d}\mathcal{L}$. With (13), MP , thus, derives $\vdash_{\mathsf{d}\mathcal{L}} \psi \rightarrow \phi$. Applying MP once more to $\psi \rightarrow \phi$ with (14) establishes that $\vdash_{\mathsf{d}\mathcal{L}} \phi$.

Case $e \bullet_{\leftarrow} d$**.** Suppose $\vdash_{\mathsf{LPd}\mathcal{L}} e \bullet_{\leftarrow} d : \phi$. By Lemma 1,

$$\vdash_{\mathsf{LPd}\mathcal{L}} e : \phi \leftrightarrow \psi \tag{15}$$

$$\vdash_{\mathsf{LPd}\mathcal{L}} d : \psi \tag{16}$$

are provable in $\mathsf{LPd}\mathcal{L}$. Applying the inductive hypothesis to (15) and (16) establishes

$$\vdash_{\mathsf{d}\mathcal{L}} \phi \leftrightarrow \psi \tag{17}$$

$$\vdash_{\mathsf{d}\mathcal{L}} \psi \tag{18}$$

Note that

$$(\phi \leftrightarrow \psi) \rightarrow (\psi \rightarrow \phi)$$

has a proof in $\mathsf{d}\mathcal{L}$. From this fact and (17), it follows by the Modus Ponens proof rule that $\vdash_{\mathsf{d}\mathcal{L}} \psi \rightarrow \phi$. Applying Modus Ponens once more to this fact and (18) establishes that $\vdash_{\mathsf{d}\mathcal{L}} \phi$.

Case $\mathbf{CT}_{\sigma} e$**.** Suppose that $\vdash_{\mathsf{LPd}\mathcal{L}} \mathrm{CT}_{\sigma} e : \phi$. By Lemma 1,

$$\phi = \sigma(c(f(\bar{x})) = c(g(\bar{x})))$$

where

$$\vdash_{\mathsf{LPd}\mathcal{L}} e : \sigma(f(\bar{x}) = g(\bar{x})) \tag{19}$$

and σ is admissible for $f(\bar{x}) = g(\bar{x})$. Applying the inductive hypothesis to (19) establishes

$$\vdash_{\mathsf{d}\mathcal{L}} \sigma(f(\bar{x}) = g(\bar{x})) \tag{20}$$

Also by Lemma 1, σ is admissible on this formula and $FV(\sigma) = \emptyset$. Therefore, [25, Theorem 25] establishes that the σ uniform substitution instance of CT is sound in $\mathsf{d}\mathcal{L}$ and so $\vdash_{\mathsf{d}\mathcal{L}} \sigma(c(f(\bar{x})) = c(g(\bar{x})))$ by the σ uniform substitution instance of CT.

Case $\mathbf{CQ}_{\sigma} e$**.** Suppose that $\vdash_{\mathsf{LPd}\mathcal{L}} \mathrm{CQ}_{\sigma} e : \phi$. By Lemma 1,

$$\phi = \sigma(p(f(\bar{x})) \leftrightarrow p(g(\bar{x})))$$

where

$$\vdash_{\mathsf{LPd}\mathcal{L}} e : \sigma(f(\bar{x}) = g(\bar{x})) \tag{21}$$

and σ is admissible for $f(\bar{x}) = g(\bar{x})$. Applying the inductive hypothesis to (21) establishes

$$\vdash_{\mathsf{d}\mathcal{L}} \sigma(f(\bar{x}) = g(\bar{x})) \tag{22}$$

Also by Lemma 1, σ is admissible on this formula and $FV(\sigma) = \emptyset$. Therefore, [25, Theorem 25] establishes that the σ uniform substitution instance of CQ is sound in $\mathsf{d}\mathcal{L}$ and so $\vdash_{\mathsf{d}\mathcal{L}} \sigma(p(f(\bar{x})) \leftrightarrow p(g(\bar{x})))$ by the σ uniform substitution instance of CQ.

Case $\mathbf{CE}_{\sigma} e$**.** Suppose that $\vdash_{\mathsf{LPd}\mathcal{L}} \mathrm{CE}_{\sigma} e : \phi$. By Lemma 1,

$$\phi = \sigma(C(p(\bar{x})) \leftrightarrow C(q(\bar{x})))$$

where

$$\vdash_{\mathsf{LPd}\mathcal{L}} e : \sigma(p(\bar{x}) \leftrightarrow q(\bar{x})) \tag{23}$$

and σ is admissible for $p(\bar{x}) \leftrightarrow q(\bar{x})$. Applying the inductive hypothesis to (23) establishes

$$\vdash_{\mathsf{d}\mathcal{L}} \sigma(p(\bar{x}) \leftrightarrow q(\bar{x})) \tag{24}$$

Also by Lemma 1, σ is admissible on this formula and $FV(\sigma) = \emptyset$. Therefore, [25, Theorem 25] establishes that the σ uniform substitution instance of CE is sound in $\mathsf{d}\mathcal{L}$ and so $\vdash_{\mathsf{d}\mathcal{L}} \sigma(C(p(\bar{x})) \leftrightarrow C(q(\bar{x})))$ by the σ uniform substitution instance of CE.

Case σe**.** Suppose that $\vdash_{\mathsf{LPd}\mathcal{L}} \sigma e : \phi$. By Lemma 1, $\phi = \sigma(\phi')$ and $\vdash_{\mathsf{LPd}\mathcal{L}} e : \phi'$ for some ϕ'. The induction hypothesis for the smaller proof term e gives $\vdash_{\mathsf{d}\mathcal{L}} \phi'$. Therefore, $\vdash_{\mathsf{d}\mathcal{L}} \sigma(\phi')$ (i.e., ϕ) is provable by US.

Case $\mathscr{B} e$**.** Similar to the case for σe. □

The fact that $\mathsf{LPd}\mathcal{L}$ is sound with respect to the semantics of $\mathsf{d}\mathcal{L}$ under proof term erasure is a corollary of this theorem.

Corollary 1 (Validity of Evident Formulas)**.** *If* $\vdash_{\mathsf{LPd}\mathcal{L}} e : \phi$ *then* $[\![\phi]\!]^I_{\mathsf{d}\mathcal{L}} = S$ *where* S *is the set of all states.*

Proof. By Theorem 2, $\vdash_{\mathsf{LPd}\mathcal{L}} e : \phi$ implies $\vdash_{\mathsf{d}\mathcal{L}} \phi$ so ϕ is valid. Note that $\mathsf{d}\mathcal{L}$ is sound, so $[\![\phi]\!]^I_{\mathsf{d}\mathcal{L}} = S$. By Def. 6, $[\![\phi]\!]^I_{\mathsf{LPd}\mathcal{L}} = [\![\phi]\!]^I_{\mathsf{d}\mathcal{L}} = S$. □

5. Checking Proof Terms Using Truth-Preserving Transformations

KeYmaera X implements the uniform substitution calculus of differential dynamic logic. The soundness-critical core of KeYmaera X contains a set of truth-preserving operations on $\mathsf{d}\mathcal{L}$ formulas; these operations correspond to the axioms and proof rules of $\mathsf{d}\mathcal{L}$. Provable objects are the closest that KeYmaera X comes to proof certificates. A Provable is an object with a goal and a sequence of remaining subgoals, each of which is a sequent. A KeYmaera X proof certificate for a formula φ is a Provable object with no remaining subgoals and $\vdash \varphi$ as its goal. Provable objects may only be created by the soundness-critical core of KeYmaera X, so they are guaranteed to be constructed via a sequence of truth-preserving operations such as proof rules, axioms, or substitutions. However, a proof certificate does not record the actual sequence of truth-preserving operations through which it is produced. While memory-efficient, this state of affairs is less than ideal for reasons that were enumerated in the introduction.

Fortunately, adding proof terms to KeYmaera X is relatively simple[7] because $\mathsf{LPd}\mathcal{L}$ is in every way – syntactically, semantically, and axiomatically – parsimonious with $\mathsf{d}\mathcal{L}$. We are therefore able to augment KeYmaera X with a proof term checker without making any changes to the soundness-critical core.

The proof of Theorem 2 was written so that it suggests a procedure for proof term checking. The proof could have exploited completeness results at several points. Instead, we opted for explicitly constructing a syntactic $\mathsf{d}\mathcal{L}$ proof. For this reason, an $\mathsf{LPd}\mathcal{L}$ proof term checker can follow the structure of the proof of Theorem 2 – for each component of a proof term, the proof term checker constructs the sequence of truth-preserving operations described in the proof of Theorem 2. These truth-preserving operations are then executed by the KeYmaera X core. If each operation succeeds (e.g., no clashes occur during uniform substitutions), then the proof term checker returns true.

There are a few caveats. The inversion lemma relies on the existence of certain formulas; these formulas must be inferred automatically, or else proof terms must be augmented with additional anno-

[7] The proof term checker is implemented in KeYmaera X 4.0b2 in Scala in `edu.cmu.cs.ls.keymaerax.pt.ProofChecker`

tations. Our current ongoing implementation opts for the latter. Additionally, in the proof of Theorem 2, there are some points where the truth of a particular theorem is asserting (e.g., via soundness). In each of these cases, KeYmaera X has either a tactic or an extra proof rule that provides exactly the required truth-preserving transformation. For example, the `keymaerax.TacticLibrary.AndR` tactic of KeYmaera X performs the action of the AndR schema referenced in the $e \wedge d$ case. The σ instances of CT, CQ, and CE (which are guaranteed to be sound by [25, Theorem 25]) that we appeal to in the $\mathrm{CT}_\sigma e$, $\mathrm{CQ}_\sigma e$, and $\mathrm{CE}_\sigma e$ cases also have corresponding tactics in KeYmaera X.

5.1 Future Work

Although the proof term checker for KeYmaera X demonstrates the utility of LPdL, there are several avenues for future work. First, KeYmaera X does not currently provide a mechanism for *generating* proof terms from proof search procedures – users must manually write down proof terms to be checked. However, we believe it will be easy to argument the KeYmaera X tactic language interpreter with a mechanism that constructs proof terms in tandem with the truth-preserving operations it executes on Provables. This extension – which we leave as future work – would add *generation* of proof terms to KeYmaera X. Furthermore, the existence properties stated in the inversion lemma require inference that is not currently implemented; instead, users of the proof term checker must annotate implicational and equivalence rewriting.

6. Related Work

Logics containing explicit representations of proofs have a storied place in the history of mathematical logic and computer science The BHK semantics for intuitionistic logic is one early and prominent example. Type-theoretic theorem provers such as Coq [16] use proof terms as explicit notions of evidence. Conversely, differential dynamic logic has proved to be an excellent setting for verifying complex hybrid dynamical systems [30].

The approach taken in this paper is motivated primarily by pragmatic concerns related to the construction of certified software controllers for cyber-physical systems. We are particularly interested in developing a notion of evidence that is easy to add to existing theorem provers for differential dynamic logic (or other dynamic logics). For this reason, we take a logic with roots in the modal logic tradition – the Logic of Proofs [4] – as our point of departure with existing work.

The syntactic restriction placed on formulas containing proof terms is perhaps the most a significant difference between LPdL and modal logics with notions of evidence. In LPdL, it is not possible to construct a term of the form $e : e' : \phi$. For this reason, LPdL is – in a qualitative sense – considerably less expressive than what one might expect from a full logic of proofs for hybrid systems. However, our concern in this paper is with *modeling deductions in* dL, rather than with studying provability in the context of hybrid dynamical systems.

LPdL contains several mechanisms for performing contextual equivalence and equational rewriting. There exist many logics and calculi with primitives for this style of rewriting [1, 31]. Effortless rewriting of deeply nested formulas is a major benefit of Hilbert-style logics, but comes at the cost of less structured proofs.

7. Conclusions

Explicit notions of evidence provide a clean separation between proof checking and proof search and enable analyses that crucially depend upon an interrogation of the structure of proofs. The Logic of Proofs for Differential Dynamic Logic demonstrates that it is possible to construct a calculus of proof terms on top of an existing theorem prover. Our preliminary work on synthesizing certified fall-back controllers for safety-critical systems demonstrates that explicit representations of proofs enable principled solutions to problems that would otherwise require ad-hoc and soundness-critical analyses.

Acknowledgements. We thank the members of the Logical Systems Lab at Carnegie Mellon University for helpful discussions related to this work. We also thank the anonymous reviewers for their suggestions and comments.

This research was sponsored by the National Science Foundation under grant number CNS-1054246 and the Department of Transportation under grant number DTRT12GUTC11 and the Future of Life Institute (futureoflife.org) FLI-RFP-AI1 program, grant #2015-143867. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

References

[1] R. Alenda, N. Olivetti, and G. L. Pozzato. Nested Sequent Calculi for Conditional Logics. In L. Fariñas del Cerro, A. Herzig, and J. Mengin, editors, *Logics in Artificial Intelligence*, volume 7519 of *Lecture Notes in Computer Science*, pages 14–27. Springer-Verlag, 2012.

[2] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. spv, 1992.

[3] S. Artemov and L. Beklemishev. Provability Logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd Edition*, volume 13 of *Handbook of Philosophical Logic*, pages 189–360. Springer Netherlands, 2005.

[4] S. N. Artemov. Operational modal logic. Technical Report MSI 9529, Cornell University, 1995.

[5] L. M. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer. The Lean Theorem Prover (System Description). In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, pages 378–388, 2015.

[6] M. Fitting. The logic of proofs, semantically. *Annals of Pure and Applied Logic*, 132(1):1 – 25, 2005.

[7] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In A. P. Felty and A. Middeldorp, editors, *CADE*, volume 9195 of *LNCS*, pages 527–538. Springer, 2015.

[8] N. Fulton and A. Platzer. A logic of proofs for differential dynamic logic: Tech report. Technical Report CMU-CS-15-143, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 2015.

[9] R. Harper, F. Honsell, and G. Plotkin. A Framework for Defining Logics. *J. ACM*, 40(1):143–184, Jan. 1993.

[10] J. Harrison. HOL light: A tutorial introduction. In *Formal Methods in Computer-Aided Design, First International Conference, FMCAD '96, Palo Alto, California, USA, November 6-8, 1996, Proceedings*, pages 265–269, 1996.

[11] J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, and E. Z. A. Platzer. A formally verified hybrid system for the next-generation airborne collision avoidance system. In C. Baier and C. Tinelli, editors, *TACAS*, LNCS. Springer, 2015.

[12] D. K. Lee, K. Crary, and R. Harper. Towards a Mechanized Metatheory of Standard ML. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '07, pages 173–184, New York, NY, USA, 2007. ACM.

[13] S. M. Loos, A. Platzer, and L. Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In M. Butler and W. Schulte, editors, *FM*, volume 6664 of *LNCS*, pages 42–56. Springer, 2011.

[14] S. M. Loos, D. W. Renshaw, and A. Platzer. Formal verification of distributed aircraft controllers. In C. Belta and F. Ivancic, editors, *HSCC*, pages 125–130. ACM, 2013.

[15] A. Mahboubi. Programming and certifying the cad algorithm inside the coq system. In *Mathematics, Algorithms, Proofs, volume 05021 of Dagstuhl Seminar Proceedings, Schloss Dagstuhl*, 2005.

[16] The Coq development team. *The Coq proof assistant reference manual*, 2004. Version 8.0.

[17] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[18] F. Pfenning and C. Schürmann. System description: Twelf a meta-logical framework for deductive systems. In *Automated Deduction CADE-16*, volume 1632 of *Lecture Notes in Computer Science*, pages 202–206. Springer Berlin Heidelberg, 1999.

[19] B. Pientka and J. Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In *Int'l Joint Conference on Automated Reasoning (IJCAR 2010)*, pages 15–21, July 2010.

[20] A. Platzer. Differential dynamic logic for verifying parametric hybrid systems. In N. Olivetti, editor, *TABLEAUX*, volume 4548 of *LNCS*, pages 216–232. Springer, 2007.

[21] A. Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008.

[22] A. Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010.

[23] A. Platzer. Logics of dynamical systems. In *LICS*, pages 13–24. IEEE, 2012.

[24] A. Platzer. Differential game logic. *ACM Trans. Comput. Log.*, 17(1):1:1–1:51, 2015.

[25] A. Platzer. A uniform substitution calculus for differential dynamic logic. In A. P. Felty and A. Middeldorp, editors, *CADE*, volume 9195 of *LNCS*, pages 467–481. Springer, 2015.

[26] A. Platzer and E. M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In A. Cavalcanti and D. Dams, editors, *FM*, volume 5850 of *LNCS*, pages 547–562. Springer, 2009.

[27] A. Platzer and J.-D. Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In A. Armando, P. Baumgartner, and G. Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 171–178. Springer, 2008.

[28] A. Platzer and J.-D. Quesel. European Train Control System: A case study in formal verification. In K. Breitman and A. Cavalcanti, editors, *ICFEM*, volume 5885 of *LNCS*, pages 246–265. Springer, 2009.

[29] A. Platzer, J.-D. Quesel, and P. Rümmer. Real world verification. In R. A. Schmidt, editor, *CADE*, volume 5663 of *LNCS*, pages 485–501. Springer, 2009.

[30] J.-D. Quesel, S. Mitsch, S. Loos, N. Aréchiga, and A. Platzer. How to model and prove hybrid systems with KeYmaera: A tutorial on safety. 2015.

[31] B. Woltzenlogel Paleo. Contextual natural deduction. In S. Artemov and A. Nerode, editors, *Logical Foundations of Computer Science*, volume 7734 of *Lecture Notes in Computer Science*, pages 372–386. Springer Berlin Heidelberg, 2013.