






dL_{ℓ} : Definite Descriptions in Differential Dynamic Logic

Rose Bohrer¹ , Manuel Fernández¹ , and André Platzer^{1,2} 

¹ Computer Science Department, Carnegie Mellon University,
Pittsburgh, USA

rose.bohrer.cs@gmail.com, {manuel.f,aplatzer}@andrew.cmu.edu

² Fakultät für Informatik, Technische Universität München,
Munich, Germany

Abstract. We introduce dL_{ℓ} , which extends differential dynamic logic (dL) for hybrid systems with definite descriptions and tuples, thus enabling its theoretical foundations to catch up with its implementation in the theorem prover KeYmaera X. Definite descriptions enable partial, nondifferentiable, and discontinuous terms, which have many examples in applications, such as divisions, n th roots, and absolute values. Tuples enable systems of multiple differential equations, arising in almost every application. Together, definite description and tuples combine to support long-desired features such as vector arithmetic.

We overcome the unique challenges posed by extending dL with these features. Unlike in dL , definite descriptions enable non-locally-Lipschitz terms, so our differential equation (ODE) axioms now make their continuity requirements explicit. Tuples are simple when considered in isolation, but in the context of hybrid systems they demand that differentials are treated in full generality. The addition of definite descriptions also makes dL_{ℓ} a free logic; we investigate the interaction of free logic and the ODEs of dL , showing that this combination is sound, and characterize its expressivity. We give an example system that can be defined and verified using these extensions.

Keywords: Dynamic logic · Definite description · Hybrid systems · Theorem proving · Uniform substitution · Partial functions

1 Introduction

Cyber-physical systems (CPSs) such as self-driving cars, trains, and airplanes combine discrete control and continuous physical dynamics and are often safety-critical because they operate around humans. Thus, it is essential to achieve the highest possible confidence in their correctness, e.g., using formal methods with strong theoretical foundations. Differential dynamic logic (dL) [18, 22, 23] is a

This research was sponsored by NDSEG, the AFOSR under grant number FA9550-16-1-0288, and the Alexander von Humboldt Foundation.

logic for formal verification of *hybrid systems* [10], widely-used models of CPSs that incorporate both their discrete and continuous behaviors. Among formal methods for CPSs, dL is notable both for its case studies [12, 15, 16] using the KeYmaera X [9] theorem prover, and for its strong foundations, as evidenced by its completeness results [18, 22, 23, 25] and a formal proof of soundness in both Isabelle/HOL and Coq [4].

However, there is a tension between the goals of practical applicability and rigorous foundations. In practice, theorem prover implementations often demand new features which were not anticipated in theory. Formalizations of KeYmaera X [5], Coq [2], and NuPRL [1] all omit or simplify whichever practical features are most theoretically challenging for their specific logic: discontinuous and partial terms in KeYmaera X, termination-checking in Coq, or context management in NuPRL. When formalizations of theorem provers *do* succeed in reflecting the implementation [13], they owe a credit to the generality of the underlying theory: it is much more feasible to formalize a general base theory than to formalize multiple ad-hoc extensions as they arise.

This paper introduces dL_ι, a new, generalized foundation for dL where *definite description* $\iota x \phi$ denotes the unique x for which ϕ holds, enabling practical extensions like divisions θ_1/θ_2 , roots $\sqrt[n]{\theta}$, and the functions $\min(\theta_1, \theta_2)$, $\max(\theta_1, \theta_2)$, and $|\theta|$, while pairs (θ_1, θ_2) enable differential equation (ODE) *systems*. Useful new features like trigonometric functions and vectors are also definable, and existing features like differentials $(\theta)'$ have elegant new axiomatizations in dL_ι.

The term $\iota x \phi$ is the definite (i.e., requiring unique existence) counterpart of Hilbert’s choice $\varepsilon x \phi$; both have seen success in HOL-style theorem provers [17, 26]. We chose definite $\iota x \phi$ over $\varepsilon x \phi$ because uniqueness significantly simplifies continuity and differential reasoning. In adopting definite descriptions and tuples in dL, we solve the novel challenges of integrating them with differential equations, dL’s distinguishing feature. Definite descriptions allow partiality, discontinuity, and nondifferentiability, all of which interact subtly with sound ODE reasoning. Multidimensional systems, enabled by tuples, demand a general treatment of differentials and expose subtle variable dependencies in some advanced ODE reasoning principles.

An example demonstrates the power of definite description: definite descriptions allow non-polynomial terms and thus non-polynomial ODEs, which need not have unique solutions. While non-polynomial ODEs (and all of dL_ι) are reducible to dL in theory, the reduction of $\iota x \phi$ is completely impractical [3]. Expressivity comes with deep semantic changes: supporting partiality makes dL_ι a free logic, for which we adopt a 3-valued Łukasiewicz semantics. We show this profound change in foundations needs only small changes to the proof calculus with additional definedness conditions. We develop the theory of dL_ι, show that the proof calculus is sound and show the nontrivial reduction from dL_ι to dL.

2 Syntax

We present the core syntax of dL_ι, which extends dL with definite descriptions and tuples. We describe the constructs informally here, deferring formal seman-

tics to Sect. 3. As a free logic [8], \mathbf{dL}_ι contains terms that do not denote and formulas whose truth values are unknown (truth is indicated \oplus , falsehood by \ominus , and unknown by \odot), a major point of difference between our semantics and proof calculus vs. those of \mathbf{dL} . Our calculus uses uniform substitution [6, §35, §40], where symbols ranging over predicates, programs, etc. are explicitly represented in the syntax, because it has simplified the construction of \mathbf{dL} calculi [23], implementations [9], and machine-checked correctness proofs [4]. This will ease implementing \mathbf{dL}_ι and mechanizing the soundness proof in future work. The syntax of \mathbf{dL}_ι is divided into terms, programs, and formulas, whose definitions, unlike in \mathbf{dL} , are all mutually recursive. The terms θ of \mathbf{dL}_ι extend the terms of \mathbf{dL} with definite descriptions, pairs, and reductions:

$$\theta ::= q \mid x \mid f(\theta) \mid \theta + \theta \mid \theta \cdot \theta \mid (\theta)' \mid \iota x \phi \mid (\theta, \theta) \mid \text{red}(\theta, s \theta, lr \theta)$$

for literal $q \in \mathbb{Q}$ and variable $x \in \mathcal{V}$, where \mathcal{V} is the set of all variable names, f is a function symbol, and ϕ is a formula. The first six cases, polynomials, differentials, and function symbols, are as in \mathbf{dL} . Variables are *flexible*: they are modified by quantifiers and programs. Variables x always denote some value and so assignments succeed only when the RHS denotes a value. In contrast, $f(\theta)$ is an *uninterpreted function* f applied to term θ , but both θ and $f(\theta)$ are allowed to be non-denoting. While function symbols f rarely appear in theorem statements, they are essential for the axioms of Sect. 5. The definite description $\iota x \phi$ denotes the *unique* value of x that makes formula ϕ true, if exactly one such value exists, else it does not denote (since description is definite). Pairs (θ_1, θ_2) can be nested to arbitrary finite depth, so their eliminator is primitive recursion on binary trees with values at the leaves. Reduction $\text{red}(\theta_1, s \theta_2, lr \theta_3)$ reduces every leaf $t \in \mathbb{R}$ to $\theta_2^t_s$ and reduces every pair a, b of recursive results to $\theta_3^a_b$, where e^y_x is the capture-avoiding substitution of y for every x in e . For example, if $\theta_1 = ((-1, 2), -3)$, then the reduction $\text{red}(\theta_1, s s^2, lr (r, l))$ is the elementwise square of the reverse tree, $(9, (4, 1))$.

The programs α, β of \mathbf{dL}_ι are *hybrid programs*, a program syntax for *hybrid systems* combining discrete and continuous dynamics. Hybrid programs of \mathbf{dL}_ι are identical to those of \mathbf{dL} with the exception that any formula or term contained therein is again any formula or term of \mathbf{dL}_ι , not necessarily just \mathbf{dL} . For any starting state, a program α might transition to zero, one, or many final states. Whenever a program transitions to zero states, we say it *aborts*.

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

Assignments $x := \theta$ assign the value of term θ to variable x , if θ denotes a value, else they abort. Tests $?\phi$ abort execution if formula ϕ is not true, else they are no-ops. Nondeterministic choices $\alpha \cup \beta$ behave as either α or β , nondeterministically. Sequential composition $\alpha; \beta$ performs β in any state resulting from α . Loops α^* repeat α sequentially any number of times, nondeterministically. The defining construct of hybrid programs are the differential equations $x' = \theta \& \psi$,

which continuously evolve x according to the differential equation $x' = \theta$ for any duration such that that term θ denotes and formula ψ is true throughout. Note the core syntax of dL_ι need only contain systems of a single variable x : in Sect. 4 we will derive systems with multiple variables from systems of one variable. Uninterpreted program constants a range over programs. We parenthesize programs α as $\{\alpha\}$ with braces for disambiguation and readability. The formulas ϕ, ψ of dL_ι are defined inductively:

$$\phi, \psi ::= \phi \wedge \psi \mid \neg\phi \mid \forall x \phi \mid \theta_1 \geq \theta_2 \mid [\alpha]\phi \mid p(\theta)$$

Conjunctions $\phi \wedge \psi$, negations $\neg\phi$, and quantifiers $\forall x \phi$ are as is standard in first-order Lukasiewicz [14] logic. The quantifier $\exists x \phi$ is also as in first-order Lukasiewicz logic and can be derived $\exists x \phi \equiv \neg\forall x \neg\phi$. In comparing $\theta_1 \geq \theta_2$, if terms θ_1 and θ_2 both denote reals, those reals are compared, if they both denote tuples they are compared elementwise, in all other cases the result is unknown (\odot). The defining construct of dynamic logics is $[\alpha]\phi$, which says ϕ holds in all states reachable by running α . Its dual, $\langle\alpha\rangle\phi$, says there exists a state reachable by running α where ϕ holds, and can be derived by the equivalence $\langle\alpha\rangle\phi \equiv \neg[\alpha]\neg\phi$. Uninterpreted predicate symbols p expect terms θ , which are also allowed not to denote, as arguments, and are allowed truth value unknown (\odot). We write P, Q for predicates which take *all* variables as arguments. We sometimes write the implication $\phi \rightarrow \psi$ as $\psi \leftarrow \phi$ for emphasis on ψ .

Example 1 (Robot Water Cooler). The textbook examples of non-Lipschitz ODEs are those of form $h' = k \cdot \sqrt{h}$ for constant k . In dL_ι , in contrast to dL , non-Lipschitz terms simplify describing a hybrid system with such ODEs, which we base on Hubbard’s leaky bucket [11, §4.2]. Consider a water cooler of height h and an opening of surface area a in its bottom of surface area A , where g is acceleration due to gravity. Suppose an enterprising student has equipped the cooler’s valve with robotic control. We could then model the cooler as:

$$\alpha_B \equiv \left\{ \left\{ \{?h > 0; a := 1\} \cup a := 0 \right\}; h' = -\sqrt{2gh} \frac{a}{A} \ \& \ h \geq 0 \right\}^*$$

This says that so long as there is water in the cooler ($?h > 0$) we can choose to open the valve ($a := 1$), but we can always close the valve ($a := 0$). Then the water drains out the cooler at a rate proportional to the square root of the current volume by Torricelli’s Law [7], or rate 0 if the valve is closed. This control process repeats arbitrarily often. The constructs $\sqrt{2gh}$ (root) and $\frac{a}{A}$ (division) are not core dL , but we can rewrite α_B using definite descriptions:

$$\left\{ \left\{ \{?h > 0; a := 1\} \cup a := 0 \right\}; h' = -(v y y^2 = 2gh \wedge y \geq 0)(v z z A = a) \ \& \ h \geq 0 \right\}^*$$

This example is representative because the ODE is non-Lipschitz: the solution is unique at $h = 0$ *only* within the constraint $h \geq 0$. The terms $\sqrt{2gh}$ and $\frac{a}{A}$ are also both *partial*: defined only assuming $gh \geq 0$ and $A \neq 0$, respectively. The interactions between partiality, uniqueness, and the constraint combine to make the proof subtle, even if short.

Common \mathbf{dL} (and likewise, \mathbf{dL}_l) theorems include *safety assertions* of the form $\phi \rightarrow [\alpha]\psi$ which say that if ϕ holds initially, then ψ will necessarily hold after α . For example, we might wish to prove the final water height of α_B never exceeds the initial height, so it is actually *leaky* (or at least is not filling up):

Proposition 1 (Leakiness). *This is valid (definitely true \oplus in all states):*

$$g > 0 \wedge h = h_0 \wedge h_0 > 0 \wedge A > 0 \rightarrow [\alpha_B](h \leq h_0)$$

We will prove Proposition 1 after we have introduced a proof calculus for \mathbf{dL}_l in Sect. 5.

3 Denotational Semantics

We now formally define the semantics of \mathbf{dL}_l terms, formulas, and programs. Due to the presence of definite descriptions $\iota x \phi(x)$, not every \mathbf{dL}_l term denotes in every state, i.e., \mathbf{dL}_l is a *free logic* [8]. We write \perp for the interpretation of a term that does not denote any value. When a term denotes, it denotes a *finite, binary* tree with real values at the leaves: a scalar denotes a singleton tree, while (arbitrarily nested) pairs denote non-singleton trees. We refer to the set of all real trees as $\mathbf{Tree}(\mathbb{R})$, where for any S , $\mathbf{Tree}(S)$ is the smallest set such that: (i) $S \subseteq \mathbf{Tree}(S)$, and (ii) for any l and $r \in \mathbf{Tree}(S)$, $(l, r) \in \mathbf{Tree}(S)$. Typing is extrinsic, i.e., we do not make typing distinctions between \mathbb{R} and $\mathbf{Tree}(\mathbb{R})$ in the semantics; typing constraints will be expressed explicitly as predicates. To account for non-denoting terms, formulas can take on three truth values: \oplus (definitely true), \circlearrowleft (unknown), and \ominus (definitely false). Thus \mathbf{dL}_l is a *3-valued* logic, and first-order connectives use the Łukasiewicz [14] interpretation.

The interpretation functions are parameterized by state $\omega : \mathcal{V} \rightarrow \mathbf{Tree}(\mathbb{R})$ mapping variables to values, and by an interpretation I mapping function symbols, predicate symbols, and program constants to their interpretation, including the possibility of not denoting a value. Writing \mathcal{S} for the set of all states, we have $I(f) : (\mathbf{Tree}(\mathbb{R}) \cup \perp) \rightarrow (\mathbf{Tree}(\mathbb{R}) \cup \perp)$, $I(p) : (\mathbf{Tree}(\mathbb{R}) \cup \perp) \rightarrow \{\oplus, \circlearrowleft, \ominus\}$, and $I(a) : \wp(\mathcal{S} \times \mathcal{S})$, where $\wp(U)$ is the power set of a set U . Below, ω_x^t is the state that is equal to ω except at x , where $\omega_x^t(x) = t$.

Definition 1 (Term semantics). *The denotation of a term is either a tree or undefined, i.e. $I\omega[\theta] : \mathbf{Tree}(\mathbb{R}) \cup \{\perp\}$, and is inductively defined as:*

$$\begin{aligned} I\omega[q] &= q & I\omega[x] &= \omega(x) & I\omega[f(\theta)] &= I(f)(I\omega[\theta]) \\ I\omega[\theta_1 + \theta_2] &= I\omega[\theta_1] + I\omega[\theta_2] & \text{if } I\omega[\theta_1], I\omega[\theta_2] &\in \mathbb{R} \\ I\omega[\theta_1 \cdot \theta_2] &= I\omega[\theta_1] \cdot I\omega[\theta_2] & \text{if } I\omega[\theta_1], I\omega[\theta_2] &\in \mathbb{R} \end{aligned}$$

$$\begin{aligned}
 I\omega[\iota x \phi] &= \begin{cases} t & \text{if a unique } t \in \mathbf{Tree}(\mathbb{R}) \text{ has } I\omega_x^t[\phi] = \oplus \\ \perp & \text{otherwise} \end{cases} \\
 I\omega[(\theta_1, \theta_2)] &= (I\omega[\theta_1], I\omega[\theta_2]) \text{ if } I\omega[\theta_1], I\omega[\theta_2] \neq \perp \\
 I\omega[\text{red}(\theta_1, s \theta_2, \text{lr } \theta_3)] &= \text{Fold}(I\omega[\theta_1], s \theta_2, \text{lr } \theta_3, I\omega) \text{ if } I\omega[\theta_1] \neq \perp \\
 I\omega[(\theta)'] &= \sum_{x \in \mathcal{V}} \omega(x') \frac{\partial I\omega[\theta]}{\partial x} \text{ if } I[\theta] \text{ totally differentiable at } \omega \\
 I\omega[\theta] &= \perp \text{ in all other cases}
 \end{aligned}$$

where $\omega(x') \frac{\partial I\omega[\theta]}{\partial x}$ abuses notation: when $\omega(x)$ is a tuple, the partial is taken w.r.t. each leaf in x , scaled by the corresponding component of x' ; the subtleties of semantics for differentials are discussed at greater length in the companion report [3]. In previous formalisms for dL [23] the semantics of $(\theta)'$ do not explicitly require that θ is totally differentiable because all pure dL terms are already smooth, thus totally differentiable. In contrast, not all dL_i terms are smooth, thus we require total differentiability explicitly, as it is required for soundness (specifically of DI_≥, Sect. 5). If differentiability conditions are not met, then $I\omega[(\theta)'] = \perp$. Reductions $\text{Fold}(t, s \theta_R, \text{lr } \theta_T, I\omega)$ recurse on t :

$$\begin{aligned}
 \text{Fold}(t, s \theta_R, \text{lr } \theta_T, I\omega) &= I\omega_s^t[\theta_R] \text{ when } t \in \mathbb{R} \\
 \text{Fold}((L, R), s \theta_R, \text{lr } \theta_T, I\omega) &= I\omega_l^K \overset{S}{r}[\theta_T] \text{ where} \\
 &K = \text{Fold}(L, s \theta_R, \text{lr } \theta_T, I\omega), S = \text{Fold}(R, s \theta_R, \text{lr } \theta_T, I\omega)
 \end{aligned}$$

That is, they reduce singletons t by binding s to t in θ_R , and reduce node (L, R) by binding l, r to the reductions of the respective branches in θ_T .

Definition 2 (Formula semantics). *The formula semantics are 3-valued:*

$$\begin{aligned}
 I\omega[\phi \wedge \psi] &= I\omega[\phi] \sqcap I\omega[\psi] & I\omega[\neg\phi] &= \overline{I\omega[\phi]} \\
 I\omega[\forall x \phi] &= \bigsqcap_{t \in \mathbf{Tree}(\mathbb{R})} I\omega_x^t[\phi] & I\omega[[\alpha]\phi] &= \bigsqcap_{(\omega, \nu) \in I[\alpha]} I\nu[\phi] \\
 I\omega[\theta_1 \geq \theta_2] &= \text{Geq}(I\omega[\theta_1], I\omega[\theta_2]) & I\omega[p(\theta)] &= I(p)(I\omega[\theta]) \\
 & & \text{Geq}(r_1, r_2) &= r_1 \geq r_2 \text{ if } r_1, r_2 \in \mathbb{R} \\
 & & \text{Geq}((l_1, r_1), (l_2, r_2)) &= \text{Geq}(l_1, l_2) \sqcap \text{Geq}(r_1, r_2) \\
 & & \text{Geq}(v_1, v_2) &= \circ \text{ otherwise}
 \end{aligned}$$

$p \sqcap q$	$q = \oplus \otimes \ominus$	\bar{p}	$p = \oplus \otimes \ominus$	$p \rightarrow q$	$q = \oplus \otimes \ominus$	$p \leftrightarrow q$	$q = \oplus \otimes \ominus$
$p = \oplus$	$\oplus \otimes \ominus$		$\ominus \otimes \oplus$	$p = \oplus$	$\oplus \otimes \ominus$	$p = \oplus$	$\oplus \otimes \ominus$
$p = \otimes$	$\otimes \otimes \ominus$			$p = \otimes$	$\oplus \oplus \otimes$	$p = \otimes$	$\otimes \oplus \otimes$
$p = \ominus$	$\ominus \otimes \ominus$			$p = \ominus$	$\oplus \oplus \oplus$	$p = \ominus$	$\ominus \otimes \oplus$

Implication $p \rightarrow q$ can be intuited as $p \leq q$, (where $\ominus < \otimes < \oplus$) so $(p \rightarrow q)$ is \oplus even when $p = q = \otimes$. Conjunction $p \sqcap q$ takes the minimum value of the arguments, and is unknown \otimes when the least conjunct is \otimes . Equivalence $p \leftrightarrow q$ is reflexive (even $(\otimes \leftrightarrow \otimes)$ is \oplus), but is \otimes in all other cases where some argument is \otimes . We say a formula ϕ is *valid* if $I\omega[\phi] = \oplus$ for all ω and I . Comparisons $\theta_1 \geq \theta_2$ are taken elementwise and are unknown (\otimes) for differing shapes. Predicates p are interpreted by the interpretation I . The meaning of quantifiers $\forall x \phi$ and $[\alpha]\phi$ are taken as conjunctions \sqcap_S over potentially-uncountable index sets S . The value of \sqcap_S is the least value of any conjunct, one of $\{\ominus, \otimes, \oplus\}$.

Definition 3 (Program semantics). *Program semantics generalize those of dL as conservatively as possible so that verification finds as many bugs as possible: e.g. assignments of non-denoting terms and tests of unknown formulas abort. The denotation of a program α is a relation $I[\alpha]$ where $(\omega, \nu) \in I[\alpha]$ whenever final state ν is reachable from initial state ω by running α .*

$$\begin{aligned}
I[x := \theta] &= \{(\omega, \omega_x^{I\omega[\theta]}) \mid I\omega[\theta] \neq \perp\} & I[?\phi] &= \{(\omega, \omega) \mid I\omega[\phi] = \oplus\} \\
I[\alpha \cup \beta] &= I[\alpha] \cup I[\beta] & I[\alpha; \beta] &= I[\alpha] \circ I[\beta] \\
I[\alpha^*] &= I[\alpha]^* = \bigcup_{n \in \mathbb{N}} \underbrace{I[\alpha; \dots; \alpha]}_{n \text{ times}}
\end{aligned}$$

$$\begin{aligned}
I[x' = \theta \& \psi] &= \{(\omega, \nu) \mid \omega = \varphi(0) \text{ on } \{x'\}^{\mathbb{G}} \text{ and } \nu = \varphi(r) \text{ for some } \varphi : [0, r] \rightarrow \mathcal{S} \\
&\text{which solves } x' = \theta \& \psi, \text{ i.e., for } s \in [0, r], \frac{\partial \varphi(t)(x)}{\partial t}(s) = \varphi(s)(x') \\
&\text{and } I\varphi(s)[x' = \theta \wedge \psi] = \oplus \text{ and } \varphi(s) = \varphi(0) \text{ on } \{x, x'\}^{\mathbb{G}}\}
\end{aligned}$$

where $X^{\mathbb{G}}$ is the complement of set X . ODEs $x' = \theta \& \psi$ are initial value problems: $(\omega, \nu) \in I[x' = \theta \& \psi]$ if some solution φ of some duration $r \in \mathbb{R}_{\geq 0}$ takes ω to ν while satisfying ψ throughout. A solution φ must satisfy $x' = \theta$ as an equation, satisfy constraint ψ , and assign the time-derivative of each x to each x' . The initial value of x' is overwritten and variables except x, x' are not changed. Assignments $x := \theta$ are strict: they store the value of θ in variable x , or abort if θ does not denote a value. Tests $?\phi$ succeed if ϕ is definitely true (\oplus); both the unknown (\otimes) and definitely false (\ominus) cases abort execution. Likewise, the domain constraint ψ of a differential equation $x' = \theta \& \psi$ must be definitely-true (\oplus) throughout the entire evolution and the term θ implicitly must denote values throughout the evolution, since $I\varphi(s)[x' = \theta \wedge \psi] = \oplus$.

4 Derived Constructs

A key benefit of dL_ℓ is extensibility: Many term constructs can be defined with definite descriptions $\iota x \phi$ and tuples which otherwise require unwieldy encodings as formulas. In this section we reap the benefits of extensibility by defining such new term constructs.

Arithmetic Operations. In practice, we often wish to use arithmetic operations beyond the core dL operations. Figure 1 demonstrates basic arithmetic operations which have simple definitions in dL_ℓ but not as terms in dL: Of these, max, min, and $|\cdot|$ preserve Lipschitz-continuity but not differentiability. Roots $\sqrt{\theta}$ can violate even Lipschitz-continuity and both roots and divisions are non-total. In practice (as in Example 1), these operators are used in ODE models, making their continuity properties essential. Since pure dL requires smooth terms [23], even *functions* max and min would be encoded as formulas in pure dL.

$$\begin{aligned}
 (\text{if}(\phi)(\theta_1)\text{else}(\theta_2)) &= \iota x (\phi \wedge x = \theta_1) \vee (-\phi \wedge x = \theta_2) \\
 \max(\theta_1, \theta_2) &= \iota x (\theta_1 \geq \theta_2 \wedge x = \theta_1) \vee (\theta_2 \geq \theta_1 \wedge x = \theta_2) \\
 \min(\theta_1, \theta_2) &= \iota x (\theta_1 \geq \theta_2 \wedge x = \theta_2) \vee (\theta_2 \geq \theta_1 \wedge x = \theta_1) \\
 |\theta| &= \max(\theta, -\theta) \quad \sqrt{\theta} = \iota x (x^2 = \theta \wedge x \geq 0) \quad \theta_1/\theta_2 = \iota x (x \cdot \theta_2 = \theta_1) \\
 (\sin \theta, \cos \theta) &= \iota z [t := 0; s := 0; c := 1; s' = c, c' = -s, t' = 1; ?t = \theta]z = (s, c)
 \end{aligned}$$

Fig. 1. Derived arithmetic operations (for fresh x, t, c, s, z)

Tuples. We make tuples first-class in dL_ℓ to simultaneously simplify the treatment of ODEs compared to prior work [18] and provide support for data structures such as vectors, widely used in physical computations. In contrast to the flexible function symbols (think: unbounded arrays) of QdL [20], they are equipped with a primitive recursion operator, making it easier to write sophisticated functional computations. These structures can be used in systems with non-scalar inputs, for example a robot which avoids a list of obstacles [16].

While pairs (θ_1, θ_2) are core dL_ℓ constructs, the left and right projections $\pi_1\theta$ and $\pi_2\theta$ are derivable, as are convenience predicates $\text{inR}(\theta)$ and $\text{isT}(\theta)$ which hold exactly for scalars and tuples, respectively:

$$\begin{aligned}
 \pi_1\theta &\equiv \iota l \exists r (\theta = (l, r)) & \pi_2\theta &\equiv \iota r \exists l (\theta = (l, r)) \\
 \text{inR}(\theta) &\equiv (\text{red}(\theta, s \ 1, lr \ 0) = 1) & \text{isT}(\theta) &\equiv (\text{red}(\theta, s \ 1, lr \ 0) = 0)
 \end{aligned}$$

When combined with the reduce operation on trees, these operations can be used to implement a variety of data structures. Figure 2 shows an example library of operations on lists. Lists are represented as nested pairs, with no special terminator. We name an argument L to indicate its intended use as a list rather than an arbitrary tree. Lists are trees whose left-projections are never pairs. Additional data structures are shown in the report [3].

Systems of ODEs. Tuples reduce ODE systems to individual ODEs, e.g.:

$$\{x'_1=\theta_1, x'_2=\theta_2\} \equiv (z := (x_1, x_2); \{z' = (\theta_1^{\pi_j z}, \theta_2^{\pi_j z})\}; x_1 := \pi_1 z; x_2 := \pi_2 z)$$

While this encoding is simple, it will enable us in Sect. 5 to support systems of any finite dimension in axiom **DG**, which implementation experience [9] has shown challenging due to the variable dependencies involved.

$$\begin{aligned} \text{map2}(T, f(x, y)) &= \text{red}(T, s \ s, lr \ \text{if}(\text{in}\mathbb{R}(r))(f(l, r))\text{else}\{(f(\pi_1 l, \pi_2 l), r)\}) \\ \text{snoc}(L, x) &= \text{red}(L, s \ (s, x), lr \ (\pi_1 l, r)) \quad \text{rev}(L) = \text{red}(L, s \ s, lr \ \text{snoc}(r, l)) \\ \text{zip}(L_1, L_2) &= \pi_1 \text{red}(\text{rev}(L_1), s \ ((s, \pi_1 L_2), \pi_2 L_2), lr \ (((\pi_1 \pi_1 l, \pi_1 \pi_2 r), \pi_1 r), \pi_2 \pi_2 r)) \\ (L_1 + L_2) &= \text{map2}(\text{zip}(L_1, L_2), x + y) \\ L_1 \cdot L_2 &= \text{red}(\text{map2}(\text{zip}(L_1, L_2), x \cdot y), s \ s, lr \ l + r) \end{aligned}$$

Fig. 2. Example vector functions

Types and Definedness. Many of the operations in dL_t expect, for example, reals or terms that denote values. For simplicity, we make these type distinctions extrinsically: core dL terms are untyped, and proposition $\text{in}\mathbb{R}(\theta)$ says θ belongs to type \mathbb{R} . Typed quantifiers are definable, e.g., $\forall x:\mathbb{R} \ \phi \equiv \forall x(\text{in}\mathbb{R}(x) \rightarrow \phi)$. Whether a term denotes is also treated extrinsically. Formula $\text{E}(\theta) \equiv \text{D}(\theta = \theta)$ only holds for terms that denote, where $\text{D}(\phi)$ says ϕ is *definitely true*, which has truth value \oplus when ϕ has truth value \oplus and has value \ominus otherwise. We give its truth table and a definition:

$$\frac{p}{\text{D}(p)} \left| \begin{array}{c} \oplus \ \ominus \ \ominus \\ \oplus \ \oplus \ \ominus \end{array} \right. \quad \text{D}(\phi) \equiv \neg(\phi \rightarrow \neg\phi)$$

That is, $\text{D}(\phi)$ collapses \ominus into \ominus . These constructs are used in the axioms of Sect. 5. In the same spirit, we sometimes need to know that a function $f(x)$ (of any dimension) is continuous, but derive this notion. We write $\text{Con}(f(x))$ to say that $f(x)$ is continuous as x varies around its current value:

$$\text{Con}(f(x)) \equiv \text{D}(\forall \xi \exists \delta \forall y (0 < \|y - x\| < \delta \rightarrow \|f(y) - f(x)\| < \xi))$$

Note that when $\text{Con}(f(x))$ holds, the shape of $f(x)$ is constant in a neighborhood of x , since the Euclidean norm $\|f(y) - f(x)\|$ does not exist when $f(y)$ and $f(x)$ differ in shape. Likewise, $\text{Con}(f(x))$ requires only continuity on y whose shape agrees with that of x , since the Euclidean norm $\|y - x\|$ does not otherwise exist.

5 dL_t Axioms

Our proof system is given in the Hilbert style, with a minimum number of proof rules and larger number of axioms, each of which is an individual concrete formula. The core proof rule is uniform substitution [23][6, §35,§40]: from the

validity of ϕ we can conclude validity of $\sigma(\phi)$ where the uniform substitution σ specifies concrete replacements for some or all predicates, functions, and program constants in a formula ϕ :

$$(\text{US}) \frac{\phi}{\sigma(\phi)}$$

The soundness side-conditions to **US** about σ are non-trivial, and make up much of its soundness proof in Sect. 6. The payoff is that uniform substitution enables a modular design where such subtle arguments need only be done once in the soundness proof of the **US** rule, and every axiom, which is now an individual concrete dL_ℓ formula, is significantly simpler to prove valid and to implement.

$[\cdot] \langle a \rangle P \leftrightarrow \neg[a]\neg P$	$\text{K } [a] (P \rightarrow Q) \rightarrow ([a]P \rightarrow [a]Q)$
$[:=] ([x := f]p(x) \leftrightarrow p(f)) \leftarrow \mathbf{E}(f)$	$\text{I } [a^*]\mathbf{D}(P \rightarrow [a]P) \rightarrow \mathbf{D}(P \rightarrow [a^*]P)$
$[?] [?Q]P \leftrightarrow (\mathbf{D}(Q) \rightarrow P)$	$\forall p \rightarrow [a]p$
$[\cup] [a \cup b]P \leftrightarrow [a]P \wedge [b]P$	$\text{G } \frac{P}{[a]P}$
$[;] [a; b]P \leftrightarrow [a][b]P$	$\forall \frac{p(x)}{\forall x p(x)}$
$[*] [a^*]P \leftrightarrow P \wedge [a][a^*]P$	$\text{MP } \frac{P \rightarrow Q \quad P}{Q}$
$\forall i (\forall x p(x)) \rightarrow (\mathbf{E}(f) \rightarrow p(f))$	$\forall \forall p \rightarrow \forall x p$
$\forall \rightarrow \forall x (p(x) \rightarrow q(x)) \rightarrow \forall x p(x) \rightarrow \forall x q(x)$	

Fig. 3. Discrete dL axioms

Figure 3 gives axioms and rules for the discrete programming constructs, which are generalizations of corresponding axioms [23] for dL to account for non-denoting terms and unknown formulas. Axioms are augmented with definedness conditions whenever multiple occurrences of terms or formulas differ in their tolerance for partiality. The conclusion (in canonical usage) of each axiom is highlighted in blue, while any difference from the dL axioms is highlighted in red. Recall the operator $\mathbf{D}(\phi)$ says ϕ is *definitely* true. For example, axiom [?] says that a test $?Q$ succeeds when Q is definitely true. The induction axiom **I** requires the inductive step proved definitely true, but concludes definite truth. The other axioms for program constructs ($[\cdot], [\cup], [;], [*]$) carry over from dL without modification, since partiality primarily demands changes when mediating between formulas and programs or between terms and program variables. As is standard in free logics, axiom $\forall i$ says that since quantifiers range over values, they must be instantiated only to terms that denote values. Assignments $[:=]$ require the assigned term to denote a value, since program variables x range over values.

Figure 4 gives the dL_ℓ generalizations of dL’s axioms for reasoning about differential equations: **DC** is generalized by analogy to [?] to require definite truth

$$\begin{aligned}
& \text{DW } [x' = f(x) \& q(x)]q(x) \\
& \text{DC } ([x' = f(x) \& q(x)]p(x) \leftrightarrow [x' = f(x) \& q(x) \wedge r(x)]p(x)) \leftarrow \mathbf{D}([x' = f(x) \& q(x)]r(x)) \\
& \text{DE } [x' = f(x) \& q(x)][x' := f(x)]p(x, x') \leftrightarrow [x' = f(x) \& q(x)]p(x, x') \\
& \text{DI}_{\geq} \quad \left([x' = h(x) \& q(x)]f(x) \geq g(x) \leftrightarrow [?q(x)]f(x) \geq g(x) \right. \\
& \quad \left. \leftarrow [x' = h(x) \& q(x)](f(x))' \geq (g(x))' \right. \\
& \quad \quad \left. \forall x (q(x) \rightarrow \mathbf{Con}(a(x)) \wedge \mathbf{Con}(b(x))) \right) \\
& \text{DG } \rightarrow ([x' = f(x) \& q(x)]p(x) \leftrightarrow \exists y : \mathbb{R} [x' = f(x), y' = a(x)y + b(x) \& q(x)]p(x)) \\
& \text{DS } (\forall t : \mathbb{R} ((\forall 0 \leq s \leq t) q(x + fs)) \rightarrow [x := x + ft]p(x)) \rightarrow [x' = f \& q(x)]p(x) \\
& (\theta)' \quad (f(x))' = x' \cdot \iota M \forall \xi > 0 \exists \delta \forall y \mathbf{D}(0 < \|y - x\| < \delta \rightarrow f(y) - f(x) - M(y - x) < \xi \|y - x\|) \\
& \quad \leftarrow \mathbf{E}((f(x))') \\
& \mathbf{E}' \quad \mathbf{E}((f(x))') \leftarrow \mathbf{E}(\iota M \forall \xi > 0 \exists \delta \forall y \mathbf{D}(0 < \|y - x\| < \delta \rightarrow f(y) - f(x) - M(y - x) < \xi \|y - x\|))
\end{aligned}$$

Fig. 4. Differential equation axioms and differential axioms

and **DG** is generalized to require continuity, otherwise the axioms carry over unchanged. **DW** says the constraint of an ODE always holds as a postcondition. **DC** says any postcondition which is proven (definitely) true may be added to the constraint. **DE** says the ODE holds as an equation in the postcondition. **DI**_≥ is the *differential induction* [19] axiom for proving nonstrict inequalities $f(x) \geq g(x)$ follow from their *differential formula* $(f(x))' \geq (g(x))'$. The strict case $f(x) > g(x)$ is analogous; axioms for equality, inequality, conjunction, and disjunction can be derived from these. Note the assumptions in **DI**_≥ hold only when $f(x)$ and $g(x)$ are *totally* differentiable within the constraint, as required for soundness. **DG** allows extending a system with an additional ghost dimension, and is used for everything from solving systems to reasoning about exponentially-decaying systems [25]. The new dimension is required to be Lipschitz so that solutions exist and is required to be linear in the new variables so that the solutions of the extended system exist as long as those of the initial system. **DS** says the solution of a constant ODE system is linear. To solve multidimensional systems with **DS**, interpret $x + fs$ and $x + ft$ as pairwise vector sums per Fig. 2. Axiom $(\theta)'$ expands a differential $(f(x))'$ according to the definition of total differential. It assumes $\mathbf{E}((f(x))')$ because equalities are not allowed to hold between non-denoting terms; proving these assumptions is enabled by axiom \mathbf{E}' . In practice, axioms are derived from \mathbf{E}' for each case and applied recursively to automatically prove existence, for example:

$$\mathbf{E}((f(x))') \wedge \mathbf{E}((g(x))') \rightarrow \mathbf{E}(((f + g)(x))')$$

is used to show differentials of sums exist. Likewise, axiom $(\theta)'$ is long-winded for practical proving, so we will use it to implement simpler special-case axioms in Example 2. The definition of $(\theta)'$ above only supports real-valued x and $f(x)$,

because scalar differences $f(y) - f(x)$ and $y - x$ only denote a value when $x, y, f(x)$, and $f(y)$ are reals. The report [3] discusses its generalization to tree-valued functions of tree-valued arguments.

$$\begin{array}{l}
 \iota \ p(\iota z p(z)) \leftrightarrow \exists x (p(x) \wedge \forall y (p(y) \rightarrow y = x)) \quad =\text{T } l_1=l_2 \wedge r_1=r_2 \leftrightarrow (l_1, r_1)=(l_2, r_2) \\
 \text{QE } \frac{*}{\left(\bigwedge_{x \in V(\phi)} \text{in}\mathbb{R}(x)\right) \rightarrow \phi} \quad (\text{where } \phi \text{ is valid in first-order real arithmetic)} \\
 \text{redT } \text{red}((L, R), s f(s), lr g(l, r)) = g(\text{red}(L, s f(s), lr g(l, r)), \text{red}(R, s f(s), lr g(l, r))) \\
 \text{redR } \text{in}\mathbb{R}(r) \rightarrow \text{red}(r, s f(s), lr g(l, r)) = f(r) \\
 \text{TreeI } \text{D}\left(p(\iota x 0 = 1) \wedge \forall s (\text{in}\mathbb{R}(s) \rightarrow p(s)) \wedge \forall lr (p(l) \wedge p(r) \rightarrow p((l, r)))\right) \rightarrow \text{D}(p(t))
 \end{array}$$

Fig. 5. Axioms for datatypes

Figure 5 gives axioms for definite descriptions and tuples. Axiom ι fully characterizes definite descriptions, and it is used to derive axioms for defined term constructs like those in Example 2. Axiom $=\text{T}$ enables comparisons on tuples. Quantifier elimination rule QE uses that first-order real arithmetic, a fragment, is decidable [27]. Since variables of dL_ℓ may range over tuples, which are not part of first-order arithmetic, it must first check that all variables of the formula (written $V(\phi)$) are indeed real-valued. Axioms redT and redR evaluate reductions when their shape is known, and axiom TreeI allows proving a property of an arbitrary value by induction on its shape, including a second base case $p(\iota x 0 = 1)$ where the argument to p does not denote.

Example 2 (Derived axioms). The following are examples of derived axiom schemata that have been proved from those above. Proofs are in the report [3].

$$\begin{array}{l}
 \pi_1(l, r) = l \quad \pi_2(l, r) = r \quad \text{in}\mathbb{R}(f) \vee \text{is}\mathbb{T}(f) \leftarrow \text{E}(f) \quad (x)' = x' \\
 (f(x) + g(x))' = (f(x))' + (g(x))' \leftarrow \text{E}((f(x))') \wedge \text{E}((g(x))') \quad (f)' = 0 \leftarrow \text{E}(f) \\
 (f(x) \cdot g(x))' = (f(x))' \cdot g(x) + (g(x))' \cdot f(x) \leftarrow \text{E}((f(x))') \wedge \text{E}((g(x))')
 \end{array}$$

It is significant that the differential axioms of Example 2 are *derived*: when new term constructs are added in the future, we expect to derive their differential axioms as well, so that these extensions lie entirely *outside* the core dL_ℓ calculus. Note that these axioms also conclude (by applying axiom $\text{E}'()$) that the differential of the larger term exists, because it equals something. Thus, these axioms are suitable both for showing differentials exist and what form differentials take.

Example 3 (Proof of leakiness). Proposition 1 of Sect. 2 is provable in dL .

Proof (Sketch). By axiom **I** with loop invariant $P \equiv (g > 0 \wedge A > 0 \wedge 0 \leq h \leq h_0)$. The first two conditions are trivially invariant by axiom **V** because g and A are constant throughout α_B . Proceed by cases with axiom **[U]**. In each case, show $h \leq h_0$ to be an invariant of the ODE by **DI $_{\geq}$** . Because $h \leq h_0$ holds initially and the ODE is locally Lipschitz-continuous given constraint $h \geq 0$, it suffices to show $(h)' \leq (h_0)' = 0$ throughout. Then $(h)' \leq 0 \iff -\sqrt{2gh} \frac{g}{A} \leq 0 \iff \sqrt{h} \geq 0$ by algebra and **DE**, which is true by **DW**, showing $h \leq h_0$. \square

6 Theory

Proofchecking is decidable, and provable formulas are valid.

Theorem 1 (Proofchecking decidability). *There exists an algorithm which decides whether a derivation \mathcal{D} is a proof of a given \mathbf{dL}_l formula ϕ .*

Theorem 2 (Soundness of \mathbf{dL}_l). *If ϕ is provable in \mathbf{dL}_l , then ϕ is valid.*

The proof of soundness proceeds by induction on the structure of derivations. That is, we prove each axiom (which is an individual formula) to be *valid* and prove every proof rule to be *sound* (producing valid conclusions from valid premisses). Because \mathbf{dL}_l supports the formula and program connectives of \mathbf{dL} , many of the axioms are extensions of corresponding \mathbf{dL} axioms. The axiom validity proofs also have a similar flavor to those of \mathbf{dL} : each axiom is proven valid by direct proof, showing truth of the axiom according to the denotational semantics in an arbitrary state. The full proofs for each axiom and rule are given in the report [3]; Lemma 1 gives an example.

Lemma 1 (Assignment axiom is valid). *The following formula is valid:*

$$([x := f]p(x) \leftrightarrow p(f)) \leftarrow E(f)$$

Proof. Assume (1) $I\omega[E(f)] = \oplus$ for some state ω and interpretation I , then observe $I\omega[[x := f]p(x)] = I\omega[p(f)]$ by the chain of equalities $I\omega[[x := f]p(x)] = \prod_{\nu \mid (\omega, \nu) \in \{(\omega, \omega_x^{I\omega[f]})\}, I\omega[f] \neq \perp} I\nu[p(x)] = I\omega_x^{I\omega[f]}[p(x)] = I(p)(I(f)) = I\omega[p(f)]$ \square

6.1 Uniform Substitution

The uniform substitution proof rule in \mathbf{dL}_l is analogous to that in \mathbf{dL} :

$$(\mathbf{US}) \frac{\phi}{\sigma(\phi)}$$

In \mathbf{dL} , the **US** rule is sound when the substitution σ does not introduce free references to bound variables, in a sense made precise elsewhere [23]. Such substitutions are called *admissible*, a condition which can be checked syntactically.

We show that the same holds of \mathbf{dL}_l when adding terms $\iota x \phi, (\theta_1, \theta_2)$ and $\text{red}(\theta_1, s \theta_2, \text{lr } \theta_3)$ and generalizing \mathbf{dL} to a three-valued semantics. As in \mathbf{dL} , we formulate admissibility in terms of U -admissibility (Definition 4) checks.

Definition 4 (Admissible uniform substitution). A substitution σ is *U-admissible* for ϕ (or θ or α) with respect to a set $U \subseteq \mathcal{V} \cup \mathcal{V}'$ iff $FV(\sigma|_{\Sigma(\phi)}) \cap U = \emptyset$ where $\sigma|_{\Sigma(\phi)}$ is the restriction of σ that only replaces symbols that occur in ϕ and $FV(\sigma) = \bigcup_{f \in \sigma} FV(\sigma f(\cdot)) \cup \bigcup_{p \in \sigma} FV(\sigma p(\cdot))$ are the free variables that σ introduces, and where $\mathcal{V}' = \{x' \mid x \in \mathcal{V}\}$. The substitution σ is *admissible* for ϕ (or θ or α) if all such checks during its applications hold, per Fig. 6.

Case	Replacement	Admissible when
	$\sigma((\theta_1, \theta_2)) = (\sigma(\theta_1), \sigma(\theta_2))$	
	$\sigma(\text{red}(\theta_1, s \theta_2, \text{lr } \theta_3)) = \text{red}(\sigma(\theta_1), s \sigma(\theta_2), \text{lr } \sigma(\theta_3))$	σ is $\{s\}$ -admissible for θ_2 σ is $\{l, r\}$ -admissible for θ_3
	$\sigma(\iota x \phi) = \iota x \sigma(\phi)$	σ is $\{x\}$ -admissible for ϕ
	$\sigma(\forall x \phi) = \forall x \sigma(\phi)$	σ is $\{x\}$ -admissible for ϕ
	$\sigma([\alpha]\phi) = [\sigma(\alpha)]\sigma(\phi)$	σ is $BV(\sigma(\alpha))$ -admissible for ϕ
	$\sigma(f(\theta)) = f(\sigma(\theta))$, if $f \notin \sigma$, else $\sigma f(\sigma(\theta))$	

Fig. 6. Uniform substitution algorithm (selected cases)

In Fig. 6, σf denotes the replacement for symbol f provided by σ . We give the new cases of $FV(\cdot)$ here and the full static semantics in the report [3]:

$$\begin{aligned} FV((\theta_1, \theta_2)) &= FV(\theta_1) \cup FV(\theta_2) & FV(\iota x \phi) &= FV(\phi) \setminus \{x\} \\ FV(\text{red}(\theta_1, s \theta_2, \text{lr } \theta_1)) &= FV(\theta_1) \cup (FV(\theta_2) \setminus \{s\}) \cup (FV(\theta_3) \setminus \{l, r\}) \end{aligned}$$

Admissibility checks employ static semantics consisting of free-variable ($FV(\cdot)$), may-bound-variable ($BV(\cdot)$), and must-bound-variable ($MBV(\cdot)$) computations. Generally speaking, the free variables of a compound expression θ are the free variables of its immediate subexpressions, minus any variables that it binds. Formally, $FV(\theta)$ (or ϕ, α) contains all variables that influence meaning:

Lemma 2 (Coincidence). *The interpretation of an expression depends only on the values of its free variables and constants, e.g. for any term θ , any interpretations I and J that agree on the signature (mentioned predicate symbols, function symbols, and program constants) $\Sigma(\theta)$ of θ , and any states ω and $\tilde{\omega}$ that agree on $FV(\theta)$, we have $I\omega \llbracket \theta \rrbracket = J\tilde{\omega} \llbracket \theta \rrbracket$.*

The substitution result for a compound expression is found by substituting in each immediate subexpression, and is defined so long as all admissibility checks hold recursively. In general, the admissibility check for each constructor says that the substitution result must not contain any new occurrences of the variables bound at that constructor.

Theorem 3 (Uniform substitution). *Rule US is sound.*

Soundness of the proof system then follows from validity of the axioms and soundness of US and of the other proof rules.

6.2 Expressive Power

After showing soundness of \mathbf{dL}_l , we explore its expressive power: can \mathbf{dL}_l express formulas that are inexpressible in \mathbf{dL} , or is its advantage the ease with which certain formulas are expressed? Conversely, are all \mathbf{dL} formulas expressible in \mathbf{dL}_l ? Because \mathbf{dL}_l is an extension of \mathbf{dL} , it is unsurprising that it can express all \mathbf{dL} formulas. However, a valid \mathbf{dL} formula ϕ is not always valid in \mathbf{dL}_l .

Remark 1 (Conservativity counterexample). There exist valid formulas of \mathbf{dL} that are not valid formulas of \mathbf{dL}_l .

Proof. The formula $\phi \equiv (x \cdot x \geq 0)$ is not conserved, because it is true for all real values of x , but fails when x is a tuple such as $(0, 0)$, outside the domain of multiplication. This is why rule **QE** requires $\text{in}\mathbb{R}(x)$ for each mentioned x . \square

We transform \mathbf{dL} quantifiers to real-valued \mathbf{dL}_l quantifiers to close the gap:

Theorem 4 (Converse reducibility). *There exists a linear-time transformation T such that for all ϕ in \mathbf{dL} , $T(\phi)$ is valid in \mathbf{dL}_l iff ϕ is valid in \mathbf{dL} .*

The greater challenge is to show that \mathbf{dL} also suffices to express all \mathbf{dL}_l formulas and thus \mathbf{dL} and \mathbf{dL}_l are equiexpressive:

Theorem 5 (Reducibility). *There is a computable T s.t. for all formulas ϕ , interpretations I , and states ω in \mathbf{dL}_l , $I\omega[\phi] = \oplus$ in \mathbf{dL}_l iff $I\omega[T(\phi)] = \oplus$ in \mathbf{dL} .*

While this result might be misread to suggest that \mathbf{dL}_l is not truly necessary, definite descriptions enable us to define constructs that have no description as terms in \mathbf{dL} , even if they can be expressed through a sufficiently complex formula translation. The key is that the reduction from \mathbf{dL}_l to \mathbf{dL} is indeed complex, exploiting for example Gödel encodings for tuples and continuous functions [21, 24]. On the contrary, the complexity of the reduction shows that native support for definite descriptions is essential for practical proving. The equiexpressiveness result is of theoretical interest because it allows us to inherit results from \mathbf{dL} [18]:

Theorem 6 (Completeness and decidability). *\mathbf{dL}_l is reducible to \mathbf{dL} , and therefore semidecidable relative to properties of differential equations.*

While the reduction gives a semi-decision procedure for \mathbf{dL}_l in principle, it is infeasible for implementation, especially since deciding even core \mathbf{dL} is hard in practice. Moreover, this would defeat our purpose: easing implementation of practical term language extensions in \mathbf{dL} , where interactive proof is common.

7 Conclusion and Future Work

In this paper we developed \mathbf{dL}_l , an extension to differential dynamic logic (\mathbf{dL}) for formal verification of hybrid systems models of safety-critical cyber-physical systems. The key feature of \mathbf{dL}_l is definite description $\iota x \phi$, which provides a foundation for defining new term language constructs from their characteristic

formulas. We develop the theory of dL_ℓ, including semantics, a proof calculus, and soundness and expressiveness proofs. We apply dL_ℓ to verify a classic example of a non-Lipschitz ODE, which could not be directly verified in dL.

In particular, we give a novel axiomatization that accounts for the interactions between non-differentiable and partially defined operators with systems of differential equations, an interaction which does not occur for dL's simpler language where all terms are smooth. More generally, example applications abound: almost every serious case study of dL employs these constructs in practice; we give a fully rigorous foundation to these case studies. In future work, implementing dL_ℓ in KeYmaera X would enable case studies to soundly employ the constructs given herein and to define their own. We expect few core changes would be needed, thanks to our use of uniform substitution, rather the challenge is to efficiently prove and track the new assumptions on existence and continuity.

Acknowledgments. We thank Martin Giese for discussions on the use of definite descriptions in theorem provers and the referees for their thoughtful feedback.

References

1. Anand, A., Rahli, V.: Towards a formally verified proof assistant. In: Klein, G., Gamboa, R. (eds.) ITP 2014. LNCS, vol. 8558, pp. 27–44. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08970-6_3
2. Barras, B.: Sets in Coq, Coq in sets. *J. Formaliz. Reason.* **3**(1), 29–48 (2010). <https://doi.org/10.6092/issn.1972-5787/1695>
3. Bohrer, R., Fernández, M., Platzer, A.: dL_ℓ: definite descriptions in differential dynamic logic. Technical report. CMU-CS-19-111, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA (2019)
4. Bohrer, R., Rahli, V., Vukotic, I., Völpl, M., Platzer, A.: Formally verified differential dynamic logic. In: Bertot, Y., Vafeiadis, V. (eds.) CPP, pp. 208–221. ACM (2017). <https://doi.org/10.1145/3018610.3018616>
5. Bohrer, R., Tan, Y.K., Mitsch, S., Myreen, M.O., Platzer, A.: VeriPhy: verified controller executables from verified cyber-physical system models. In: Grossman, D. (ed.) PLDI, pp. 617–630. ACM (2018). <https://doi.org/10.1145/3192366.3192406>
6. Church, A.: Introduction to Mathematical Logic. Princeton University Press, Princeton (1956)
7. Driver, R.: Torricelli's law: an ideal example of an elementary ODE. *Am. Math. Mon.* **105**(5), 453–455 (1998)
8. Fitting, M., Mendelsohn, R.L.: First-Order Modal Logic. Kluwer, Norwell (1999)
9. Fulton, N., Mitsch, S., Quesel, J.-D., Völpl, M., Platzer, A.: KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 527–538. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_36
10. Henzinger, T.A.: The theory of hybrid automata. In: LICS. IEEE (1996). <https://doi.org/10.1109/LICS.1996.561342>
11. Hubbard, J.H., West, B.H.: Differential Equations: A Dynamical Systems Approach. Springer, Heidelberg (1991). <https://doi.org/10.1007/978-1-4612-4192-8>
12. Jeannin, J., et al.: A formally verified hybrid system for safe advisories in the next-generation airborne collision avoidance system. *STTT* **19**(6), 717–741 (2017). <https://doi.org/10.1007/s10009-016-0434-1>

13. Kumar, R., Arthan, R., Myreen, M.O., Owens, S.: Self-formalisation of higher-order logic: semantics, soundness, and a verified implementation. *J. Autom. Reason.* **56**(3), 221–259 (2016). <https://doi.org/10.1007/s10817-015-9357-x>
14. Lukasiewicz, J.: O logice trójwartościowej (on 3-valued logic). *Ruch Filozoficzny* **5**, 169–171 (1920)
15. Mitsch, S., Gario, M., Budnik, C.J., Golm, M., Platzer, A.: Formal verification of train control with air pressure brakes. In: Fantechi, A., Lecomte, T., Romanovsky, A. (eds.) *RSSRail. LNCS*, vol. 10598, pp. 173–191. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68499-4_12
16. Mitsch, S., Ghorbal, K., Vogelbacher, D., Platzer, A.: Formal verification of obstacle avoidance and navigation of ground robots. *Int. J. Robot. Res.* **36**(12), 1312–1340 (2017). <https://doi.org/10.1177/0278364917733549>
17. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic. *LNCS*, vol. 2283. Springer, Heidelberg (2002). <https://doi.org/10.1007/3-540-45949-9>
18. Platzer, A.: Differential dynamic logic for hybrid systems. *J. Autom. Reason.* **41**(2), 143–189 (2008). <https://doi.org/10.1007/s10817-008-9103-8>
19. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.* **20**(1), 309–352 (2010). <https://doi.org/10.1093/logcom/exn070>
20. Platzer, A.: A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *Log. Method Comput. Sci.* **8**(4), 1–44 (2012). [https://doi.org/10.2168/LMCS-8\(4:17\)2012](https://doi.org/10.2168/LMCS-8(4:17)2012). Special issue for selected papers from CSL2010
21. Platzer, A.: The complete proof theory of hybrid systems. In: *LICS*, pp. 541–550. IEEE (2012). <https://doi.org/10.1109/LICS.2012.64>
22. Platzer, A.: Logics of dynamical systems. In: *LICS*, pp. 13–24. IEEE (2012). <https://doi.org/10.1109/LICS.2012.13>
23. Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reason.* **59**(2), 219–265 (2017). <https://doi.org/10.1007/s10817-016-9385-1>
24. Platzer, A.: Differential hybrid games. *ACM Trans. Comput. Log.* **18**(3), 19:1–19:44 (2017). <https://doi.org/10.1145/3091123>
25. Platzer, A., Tan, Y.K.: Differential equation axiomatization: the impressive power of differential ghosts. In: Dawar, A., Grädel, E. (eds.) *LICS*, pp. 819–828. ACM, New York (2018). <https://doi.org/10.1145/3209108.3209147>
26. Slind, K., Norrish, M.: A brief overview of HOL4. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) *TPHOLs 2008. LNCS*, vol. 5170, pp. 28–32. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71067-7_6
27. Tarski, A.: A decision method for elementary algebra and geometry. In: Caviiness, B.F., Johnson, J.R. (eds.) *Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts and Monographs in Symbolic Computation (A Series of the Research Institute for Symbolic Computation, Johannes-Kepler-University, Linz, Austria)*, pp. 24–84. Springer, Vienna (1998). https://doi.org/10.1007/978-3-7091-9459-1_3