A Temporal Differential Dynamic Logic Formal Embedding

Lauren White
NASA
Hampton, VA, USA
lauren.m.white@nasa.gov

Laura Titolo
AMA Inc.
Hampton, VA, USA
laura.titolo@nasa.gov

J. Tanner Slagel
NASA
Hampton, VA, USA
j.tanner.slagel@nasa.gov

César A. Muñoz
NASA
Hampton, VA, USA
cesar.a.munoz@nasa.gov

Abstract

Differential temporal dynamic logic dTL^2 is a logic to specify and verify temporal properties of hybrid systems. It extends differential dynamic logic (dL) with temporal operators that enable reasoning on intermediate states in both discrete and continuous dynamics. This paper presents an embedding of dTL^2 in the Prototype Verification System (PVS). The embedding includes the formalization of a trace semantics as well as the logic and proof calculus of dTL^2, which have been enhanced to support the verification of universally quantified reachability properties. The embedding is fully functional and can be used to interactively verify hybrid programs in PVS using a combination of PVS proof commands and specialized proof strategies.

CCS Concepts: • **Theory of computation** → *Logic and verification.*

Keywords: Temporal Logic, Differential Dynamic Logic, Hybrid Systems, Formal Proofs, PVS

ACM Reference Format:
Lauren White, Laura Titolo, J. Tanner Slagel, and César A. Muñoz. 2024. A Temporal Differential Dynamic Logic Formal Embedding. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '24), January 15–16, 2024, London, UK*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3636501.3636943

1 Introduction

Differential dynamic logic (dL) [37, 39, 45, 48] is a framework for specifying and reasoning about hybrid programs, i.e., programs that exhibit both discrete and continuous dynamics. The core of dL is a proof calculus that contains a collection of axioms and rules for the rigorous verification of properties of hybrid programs. This calculus is implemented in the KeYmaera X[1] theorem prover, which is built up from a small trusted core that assumes the axioms of dL. KeYmaera X has been used in the formal verification of several cyber-physical systems [7, 8, 16, 20, 22–24, 28]. Recently, dL has been embedded within the theorem prover Prototype Verification System (PVS) [36] resulting in the tool Plaidypvs[2] [51]. This embedding expands the expressive and deductive power of dL with PVS's capabilities. For example, PVS's defined functions, or external library functions such as trigonometric and other transcendental functions, can be used within the embedding. Furthermore, meta-reasoning about hybrid programs, including reasoning about entire classes of hybrid programs, could be performed in PVS.

One limitation of dL, KeYmaera X, and Plaidypvs is that they can only reason on the input/output semantics of hybrid programs. Nevertheless, it is often the case that the correctness of a hybrid program is expressed in terms of the intermediate states that are reached during its execution not only on its final states. For example, guaranteeing that the position of an aircraft stays within a geofenced region or ensuring that eventually in the future an unmanned vehicle reaches and maintains a certain velocity are properties that cannot be expressed directly in dL.

Differential temporal dynamic logic (dTL^2) was introduced in [21] to extend dL with temporal logic operators and to reason about all the states reachable during the execution of a hybrid program. dTL^2 supports existential and universal quantifiers on execution paths through the temporal operators *globally* and *eventually*.

This paper presents an embedding of dTL^2 in PVS as an extension of Plaidypvs. The original specification in [21] has been modified to provide a compact denotation for the semantics that is more amenable to formal verification. In

CPP '24, January 15–16, 2024, London, UK
2024. ACM ISBN 979-8-4007-0488-8/24/01
https://doi.org/10.1145/3636501.3636943

[1] https://keymaerax.org
[2] Plaidypvs with the temporal extension presented in this paper is available as part of the NASA PVS library at https://github.com/nasa/pvslib/tree/master/dL.

addition, the interpretation of temporal operators over traces and part of the calculus have been refined to allow for a more natural interpretation of reachability properties for hybrid programs.

Specifically, the contributions of this paper are

- a formalization of dTL^2 trace semantics for hybrid programs;
- a formalization of dTL^2 temporal formulas that includes the temporal operators *eventually* and *globally* and the universal and existential quantifications over traces;
- formal verification of dTL^2 axioms and deduction rules in PVS;
- implementation of the proof calculus for dTL^2 as proof rules in PVS.

The embedding is fully functional and supports the formal verification of hybrid programs in PVS through the use of PVS proof commands and specialized proof rules, which are based on the dTL^2 proof calculus. To the authors' knowledge, this is the first formal specification and verification of a temporal extension of dL.

The rest of this paper is organized as follows. Section 2 introduces the basic notions related to hybrid programs, dL, and Plaidypvs. Section 3 introduces dTL^2 and its trace semantics. Section 4 presents the proof calculus and its embedding in PVS. Section 5 provides some examples on how to prove dTL^2 programs in PVS. Finally, Section 6 outlines the related work, and Section 7 concludes the paper.

2 Hybrid Programs and Differential Dynamic Logic

This section presents background information on hybrid programs and their input/output semantics as defined in [51] and implemented in Plaidypvs.

In the following, $\mathbb{V}$ denotes an infinite, but enumerable set of real-valued variables. More precisely, in Plaidypvs, variables are represented by indices. The state of a hybrid program at a moment in time is given by an *environment* of type $\mathcal{E} \triangleq [\mathbb{V} \to \mathbb{R}]$ that maps the program variables to their respective values. Real $\mathcal{R}$ and Boolean $\mathcal{B}$ expressions are represented by their evaluation functions of type $[\mathcal{E} \to \mathbb{R}]$ and $[\mathcal{E} \to \mathbb{B}]$, respectively, where $\mathbb{B}$ is the Boolean domain. For instance, $cnst(c) \triangleq \lambda(e : \mathcal{E}).c$ represents the hybrid program constant that returns the value $c \in \mathbb{R}$ in any environment. Similarly, $\top \triangleq \lambda(e : \mathcal{E}).TRUE$ and $\bot \triangleq \lambda(e : \mathcal{E}).FALSE$ represent the Boolean hybrid program constants that always return $TRUE \in \mathbb{B}$ and $FALSE \in \mathbb{B}$, respectively. While real and Boolean expressions can be arbitrary functions, Plaidypvs provides support for standard arithmetic and Boolean operators by lifting them to the domain of $\mathcal{R}$ and $\mathcal{B}$. Given $r_1, r_2 \in \mathcal{R}$ and $n \in \mathbb{N}$ the following are recognized to be of type $\mathcal{R}$ in Plaidypvs: $r_1 + r_2$, $r_1 - r_2$, r_1/r_2, $r_1 \cdot r_2$, $-r_1$, $\sqrt{r_1}$, and r_1^n. It is important to notice that, for instance, in the real expression $r_1 + r_2$, the operator + is not the arithmetic addition, but it is of type $\mathcal{R} \times \mathcal{R} \to \mathcal{R}$. Furthermore, the equality $r_1 = r_2$ is defined as the extensional equality on real number functions, i.e., $\forall e \in \mathcal{E}.r_1(e) = r_2(e)$. Similarly, given Boolean expressions $b_1, b_2 \in \mathcal{B}$, the following are recognized to be of type $\mathcal{B}$ in Plaidypvs: $b_1 \wedge b_2$, $b_1 \vee b_2$, $b_1 \to b_2$, $b_1 \leftrightarrow b_2$, $b_1 = b_2$, and $\neg b_1$.

Hybrid programs are syntactically defined as a datatype $\mathcal{H}$ in PVS according to the following grammar

$$\alpha ::= \bar{x} := \bar{\theta} \mid \bar{x}' := \bar{\theta} \,\&\, \chi \mid ?\,\chi \mid \alpha; \alpha \mid \alpha \cup \alpha \mid \alpha^*$$

where $\bar{x}, \bar{x}' \in \mathbb{V}^k$, $\bar{\theta} \in \mathcal{R}^k$, $k \in \mathbb{N}$, and $\chi \in \mathcal{B}$. The statement $\bar{x} := \bar{\theta}$ denotes a list of simultaneous discrete assignments. The statement $\bar{x}' := \bar{\theta} \,\&\, \chi$ models the continuous evolution of the variables in $\bar{x}'$ according to the first order differential equations described by $\bar{\theta}$, while guaranteeing that the solution satisfies χ along the evolution. The test $?\chi$ checks if χ is satisfied. Hybrid programs can be combined through sequential execution ($\alpha_1; \alpha_2$), nondeterministic choice ($\alpha_1 \cup \alpha_2$), and nondeterministic finite repetition (α^*).

The input/output semantics $[\![\cdot]\!]_{io}$ maps a hybrid program into a set of environment pairs $\langle e_i, e_o \rangle \in \mathcal{E} \times \mathcal{E}$. Given a vector $\bar{v}$, the notation v_i denotes the i-th element of $\bar{v}$.

$$[\![\bar{x} := \bar{\theta}]\!]_{io} \triangleq \{\langle e_i, e_o \rangle \mid \forall x \in \mathbb{V}.\, x \in \bar{x} \to e_o(x) = \bar{\theta}_{idx(x,\bar{x})}(e_i) \wedge x \notin \bar{x} \to e_i(x) = e_o(x)\}$$

$$[\![\bar{x}' := \bar{\theta} \,\&\, \chi]\!]_{io} \triangleq \{\langle e_i, e_o \rangle \mid e_i = e_o \vee \exists D \in \mathbb{D}.\, \delta(D, \bar{x}', \bar{\theta}, \chi, e_i, e_o)\}$$

$$[\![?\,\chi]\!]_{io} \triangleq \{\langle e, e \rangle \mid \chi(e)\}$$

$$[\![\alpha_1; \alpha_2]\!]_{io} \triangleq \{\langle e_i, e_o \rangle \mid \exists e \in \mathcal{E}.\, \langle e_i, e \rangle \in [\![\alpha_1]\!]_{io} \wedge \langle e, e_o \rangle \in [\![\alpha_2]\!]_{io}\}$$

$$[\![\alpha_1 \cup \alpha_2]\!]_{io} \triangleq [\![\alpha_1]\!]_{io} \cup [\![\alpha_2]\!]_{io}$$

$$[\![\alpha^*]\!]_{io} \triangleq \bigcup_{n \in \mathbb{N}} [\![\alpha^n]\!]_{io} \text{ where } \alpha^0 \triangleq ?\,true,\ \alpha^1 \triangleq \alpha, \text{ and } \alpha^{n+1} \triangleq \alpha; \alpha^n \text{ for } n \geq 1$$

where $idx(x, \bar{x})$ is the index for variable x in the vector $\bar{x}$ and $\mathbb{D} ::= \{[0, b] \mid b \in \mathbb{R}\} \cup \{\mathbb{R}_{\geq 0}\}$ denotes the ordinary differential expression domain. The predicate $\delta : \mathbb{D} \times \mathbb{V}^k \times \mathcal{R}^k \times \mathcal{B} \times \mathcal{E} \times \mathcal{E}$ holds when there exists a continuous evolution from environment e_i to environment e_o following the ordinary differential equations $\bar{x}' = \bar{\theta}$.

$$\delta(D, \bar{x}', \bar{\theta}, \chi, e_i, e_o) \triangleq \exists r \in D.\, \exists! \bar{f} \in [D \to \mathbb{R}]^k.\ sol(D, \bar{x}', \bar{\theta}, e_i, \bar{f}) \wedge e_o = env(\bar{x}', \bar{f}, e_i, r) \wedge \forall t \in \mathbb{R}.\, (D(t) \wedge t \leq r) \to \chi(env(\bar{x}', \bar{f}, e_i, t)).$$

Unpacking this further, the following function characterizes the environment e_i, with the continuously evolving variables x' replaced by values from a function $\bar{f} : [D \to \mathbb{R}]^k$.

$$env(\bar{x}', \bar{f}, e_i, r) \triangleq \lambda(x' : \mathbb{V}).\ \begin{cases} \bar{f}_{idx(x', \bar{x}')}(r) & \text{if } x' \in \bar{x}' \\ e_i(x') & \text{if } x' \notin \bar{x}' \end{cases}$$

The predicate *sol* ensures that $\bar{f}$ is the solution to the k-dimensional differential equation $\bar{x}' = \bar{\theta}$ throughout the domain D. Note in the definition of δ this solution is further assumed to be unique on the domain D.

$$sol(D, \bar{x}', \bar{\theta}, e_i, \bar{f}) \triangleq \forall t \in D.\, \bar{f}'(t) = \bar{\theta}(env(\bar{x}', \bar{f}, e_i, t))$$

The choice of a shallow embedding of real and Boolean expressions within hybrid programs allows for their interpretation directly in the logic of PVS, facilitating the task of verifying hybrid programs. Moreover, it provides the ability to extend the set of hybrid program expressions with user-defined functions. This capability is one of the unique features of the Plaidypvs implementation of dL.

Example 2.1 The following example shows the functionality of Plaidypvs in allowing user-defined functions in a hybrid program. Consider the hybrid program given by

$$x' := 6; x := \lfloor x \rfloor$$

This program progresses x according to the differential equation $x' = 6$ and then assigns to x the floor of its value. This floor function is a user-defined function that takes an input of type $\mathcal{R}$ and returns a value of type $\mathcal{R}$ that denotes the floor of its input. It can be shown that

$$\forall \langle e_i, e_o \rangle \in [\![\alpha]\!]_{io}.\, e_o(x) \in \mathbb{Z}.$$

However, it should be noted that there are intermediate states, attained by the differential part of the program, where x stores non-integer values. ■

3 Temporal Differential Dynamic Logic

The semantics in Section 2 only allows for reasoning on the input and output environments of a hybrid program. In order to reason about the intermediate steps, a more expressive semantics is needed. In [21], the dTL^2 logic is proposed as an extension of dL to specify temporal properties of hybrid programs. This section presents an alternative formalization of the trace semantics and logic of dTL^2, which has been developed in PVS. Several notions presented here differ from the ones in [21], especially in the denotation of continuous evolutions. This was a design choice to make the semantics more compact and amenable to formal verification.

A *state* can be either a *discrete* step represented by an environment in $\mathcal{E}$, a continuous step representing a continuous evolution, or an error Λ. It is useful to distinguish two kinds of continuous steps: finite (*diff*) and infinite (*inf*). A finite continuous step $diff_b(e_0, \iota_b)$ models a continuous evolution in the range $[0, b]$ from an initial environment e_0 following a dynamics $\iota_b \in [[0, b] \to \mathcal{E}]$ such that $\iota_b(0) = e_0$. Similarly, an infinite continuous state $inf(e, \iota_\infty)$ models a (possibly) infinite continuous dynamics stating from e_0 and following the dynamics $\iota_\infty \in [\mathbb{R}_{\geq 0} \to \mathcal{E}]$ such that $\iota_\infty(0) = e_0$.

Definition 3.1 (State) *A* state *is defined by the following grammar.*

$$\sigma ::= e \mid diff_b(e, \iota_b) \mid inf(e, \iota_\infty) \mid \Lambda$$

where $e \in \mathcal{E}$, $b \in \mathbb{R}_{\geq 0}$, $\iota_b \in [[0, b] \to \mathcal{E}]$, *and* $\iota_\infty \in [\mathbb{R}_{\geq 0} \to \mathcal{E}]$.

The behavior of a hybrid program can be modeled as a sequence of discrete and continuous states called a *trace*.

Definition 3.2 (Trace) *A trace is a finite and non-empty sequence of states in the form* $\bar{\sigma} = \sigma_0 \cdot \sigma_1 \ldots \sigma_n$ *such that for all* $i < n$ *either* $\sigma_i \in \mathcal{E}$ *or* $\sigma_i = diff_b(e, \iota_b)$ *for some* $e \in \mathcal{E}$, $b \in \mathbb{R}_{\geq 0}$, *and* $\iota_b \in [[0, b] \to \mathcal{E}]$, *where* σ_i *denotes the* i*-th element of the trace. The set of all traces is denoted by* $\mathbb{T}$.

From the above definition, it follows that error and infinite continuous steps can only occur at the end of a trace. A trace is said to be *finite* if it terminates with a discrete step or a finite continuous step, and it is said to be *valid* when it does not end with an error. Notice that infinite behaviors can only occur for hybrid programs with infinite continuous dynamics and all discrete traces are finite.

Definition 3.3 (Initial & Final Environments) *The initial and final environments of a trace* $\bar{\sigma}$ *are defined as follows.*

$$init(\bar{\sigma}) \triangleq \begin{cases} \sigma_0 & \text{if } \sigma_0 \in \mathcal{E} \\ \iota_b(0) & \text{if } \sigma_0 = diff_b(e, \iota_b) \\ \iota_\infty(0) & \text{if } \sigma_0 = inf(e, \iota_\infty) \end{cases}$$

$$final(\bar{\sigma}) \triangleq \begin{cases} \sigma_{|\sigma|-1} & \text{if } \sigma_{|\sigma|-1} \in \mathcal{E} \\ \iota_b(b) & \text{if } \sigma_{|\sigma|-1} = diff_b(e, \iota_b). \end{cases}$$

Definition 3.4 (Trace Semantics) *The trace semantics* $[\![\cdot]\!]_{\mathbb{T}}$ *maps a hybrid program to a set of traces and is defined inductively as follows.*

$$[\![\bar{x} := \bar{\theta}]\!]_{\mathbb{T}} \triangleq \{e_i \cdot e_o \mid \forall x \in \mathbb{V}.\, x \in \bar{x} \to e_o(x) = \bar{\theta}_{idx(x,\bar{x})}(e_i) \wedge x \notin \bar{x} \to e_i(x) = e_o(x)\}$$

$$[\![x' := \bar{\theta}\ \&\ \chi]\!]_{\mathbb{T}} \triangleq \{e \in \mathcal{E} \mid \chi(e)\} \cup \{e \cdot \Lambda \mid \neg\chi(e)\} \cup \{diff_b(e, \iota_b) \mid \exists! \bar{f} \in [[0, b] \to \mathbb{R}]^k .\, sol([0, b], \bar{x}, \bar{\theta}, e_i, \bar{f}) \wedge \iota_b = \lambda(r : [0, b]).env(\bar{x}, \bar{f}, e_i, r) \wedge \forall t \in [0, b].\chi(\iota_b(t))\} \cup \{inf(e, \iota_\infty) \mid \exists! \bar{f} \in [\mathbb{R}_{\geq 0} \to \mathbb{R}]^k .\, sol(\mathbb{R}_{\geq 0}, \bar{x}, \bar{\theta}, e_i, \bar{f}) \wedge \iota_\infty = \lambda(r : \mathbb{R}_{\geq 0}).env(\bar{x}, \bar{f}, e_i, r) \wedge \forall t \in \mathbb{R}_{\geq 0}.\chi(\iota_\infty(t))\}$$

$$[\![?\,\chi]\!]_{\mathbb{T}} \triangleq \{e \in \mathcal{E} \mid \chi(e)\} \cup \{e \cdot \Lambda \mid \neg\chi(e)\}$$

$$[\![\alpha_1; \alpha_2]\!]_{\mathbb{T}} \triangleq \{\bar{\sigma}_1 \cdot \bar{\sigma}_2 \mid \bar{\sigma}_1 \in [\![\alpha_1]\!]_{\mathbb{T}} \wedge \bar{\sigma}_2 \in [\![\alpha_2]\!]_{\mathbb{T}} \wedge \bar{\sigma}_1 \text{ is finite} \wedge final(\bar{\sigma}_1) = init(\bar{\sigma}_2)\} \cup \{\bar{\sigma}_1 \mid \bar{\sigma}_1 \in [\![\alpha_1]\!]_{\mathbb{T}} \wedge \bar{\sigma}_1 \text{ is not finite}\}$$

$$[\![\alpha_1 \cup \alpha_2]\!]_{\mathbb{T}} \triangleq [\![\alpha_1]\!]_{\mathbb{T}} \cup [\![\alpha_2]\!]_{\mathbb{T}}$$

$$[\![\alpha^*]\!]_{\mathbb{T}} \triangleq \bigcup_{n \in \mathbb{N}} [\![\alpha^n]\!]_{\mathbb{T}} \text{ where } \alpha^0 \triangleq ?\,true,\ \alpha^1 \triangleq \alpha, \text{ and } \alpha^{n+1} \triangleq \alpha; \alpha^n \text{ for } n \geq 1$$

The semantics of the continuous evolution comprises four different cases. The case where the hybrid program does not continuously evolve and stops at time 0 is modeled by the set of all environments e that satisfy χ. The case where e does not satisfy χ is modeled by a trace consisting of the environment e followed by an error. Finite and infinite continuous evolutions are modeled by the states *diff* and *inf*, respectively. Their behaviors are required to follow the specification of the differential equations $\bar{\theta}$. The semantics of the test produces an error when the guard χ is not satisfied. In the semantics of the sequential operator $\alpha_1;\alpha_2$, it is required that, if the trace associated with α_1 is finite, its final environment has to match with the initial environment of the trace in the semantics of α_2. In addition, all infinite traces in the semantics of α_1 also belong to the semantics of $\alpha_1;\alpha_2$. For both the continuous evolution and test semantics, it can be noticed that an error is produced when the property χ is not satisfied.

The following results about the correctness, completeness, and soundness of the trace semantics w.r.t. the input-output semantics defined in Plaidypvs have been formally proven in PVS. These results are fundamental to ensuring that the temporal extension provided by dTL^2 is compatible with Plaidypvs.

Theorem 3.5 (Completeness) *For all hybrid programs α, for all $\langle e_i, e_o\rangle \in [\![\alpha]\!]_{io}$, there exists a finite trace $\bar{\sigma} \in [\![\alpha]\!]_{\mathbb{T}}$ such that $init(\bar{\sigma}) = e_i$ and $final(\bar{\sigma}) = e_o$.*

Thus, the trace semantics has a representative finite trace for each pair in the input-output semantics.

Theorem 3.6 (Correctness) *Given a hybrid program α, for all finite trace $\bar{\sigma} \in [\![\alpha]\!]_{\mathbb{T}}$, $\langle init(\bar{\sigma}), final(\bar{\sigma})\rangle \in [\![\alpha]\!]_{io}$.*

This means that the trace semantics does not introduce any finite trace whose initial and final environments are not in the input-output semantics.

Corollary 3.7 (Soundness) *For all hybrid programs α_1 and α_2, $[\![\alpha_1]\!]_{\mathbb{T}} = [\![\alpha_2]\!]_{\mathbb{T}} \Rightarrow [\![\alpha_1]\!]_{io} = [\![\alpha_2]\!]_{io}$.*

Thus, as expected, the trace semantics distinguishes more programs than the input-output one, but all programs that are different under the input-output semantics are also distinguishable under the trace semantics.

To reason about temporal properties of hybrid programs, the notion of formula is introduced. The following grammars define the two kinds of formulas presented in dTL^2: *state formulas*, which are interpreted over a single state, and *trace formulas*, which are interpreted over a trace.

$$\begin{aligned}\phi ::= {} & \theta_1 \prec \theta_2 \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \forall x\ \phi \mid \exists x\ \phi \\ & \mid [\alpha]_{st}\phi \mid [\alpha]_{tr}\pi \mid \langle\alpha\rangle_{st}\phi \mid \langle\alpha\rangle_{tr}\pi \quad \text{(state formula)} \\ \pi ::= {} & \phi \mid \neg\pi \mid \Box\pi \mid \Diamond\pi \quad \text{(trace formula)}\end{aligned}$$

where $\alpha \in \mathcal{H}$, $\theta_1, \theta_2 \in \mathcal{R}$ and $\prec \in \{<, >, =, \neq, \leq, \geq\}$. Besides the standard Boolean operators, dTL^2 supports the temporal operators $\Box$ (globally) and $\Diamond$ (eventually). The modal operators $[\cdot]$ (allruns) and $\langle\cdot\rangle$ (someruns) quantify over the runs of a hybrid program. Differently from [21], two versions of these modal operators are defined here. In fact, every state formula ϕ has both a state and a trace interpretation. It is, therefore, essential to specify both versions of these operators to obtain a rigorous formalization in a strongly typed language such as PVS.

State and trace formulas are shallowly embedded in PVS meaning they are represented by their evaluation functions. Therefore, state formulas have type $[\mathcal{E} \to \mathbb{B}]$ and trace formulas have type $[\mathbb{T} \to \mathbb{B}]$. Given a state formula $\phi \in [\mathcal{E} \to \mathbb{B}]$, it is possible to lift it to a trace formula $\phi_{tr} \in [\mathbb{T} \to \mathbb{B}]$ in the following way: $\phi_{tr} = \lambda\bar{\sigma}.\, \phi(init(\bar{\sigma}))$.

Definition 3.8 (Satisfaction of dTL^2 formulas) *The satisfaction of a state formula ϕ in a given state σ, i.e., checking if $\phi(\sigma)$ holds, is denoted as $\sigma \models \phi$ and defined as follows:*

$$\begin{aligned}
&\sigma \models \theta_1 \prec \theta_2 \iff \theta_1(\sigma) \prec \theta_2(\sigma)\\
&\sigma \models \neg\phi \iff \sigma \not\models \phi\\
&\sigma \models \phi_1 \wedge \phi_2 \iff \sigma \models \phi_1 \text{ and } \sigma \models \phi_2\\
&\sigma \models \phi_1 \vee \phi_2 \iff \sigma \models \phi_1 \text{ or } \sigma \models \phi_2\\
&\sigma \models \forall x\ \phi \iff \text{for all } d \in \mathbb{R}, \sigma[x/d] \models \phi\\
&\sigma \models \exists x\ \phi \iff \text{exists } d \in \mathbb{R}, \sigma[x/d] \models \phi\\
&\sigma \models [\alpha]_{st}\phi \iff \text{for each trace } \bar{\sigma} \in [\![\alpha]\!]_{\mathbb{T}} \text{ such that}\\
&\qquad init(\bar{\sigma}) = \sigma,\ \text{if } \bar{\sigma} \text{ terminates, then } final(\bar{\sigma}) \models \phi\\
&\sigma \models \langle\alpha\rangle_{st}\phi \iff \text{exists a trace } \bar{\sigma} \in [\![\alpha]\!]_{\mathbb{T}} \text{ such that}\\
&\qquad init(\bar{\sigma}) = \sigma,\ \bar{\sigma} \text{ terminates, and } final(\bar{\sigma}) \models \phi\\
&\sigma \models [\alpha]_{tr}\pi \iff \bar{\sigma} \models \pi \text{ for each valid trace } \bar{\sigma} \in [\![\alpha]\!]_{\mathbb{T}}\\
&\qquad \text{such that } init(\bar{\sigma}) = \sigma\\
&\sigma \models \langle\alpha\rangle_{tr}\pi \iff \text{it exists a valid trace } \bar{\sigma} \in [\![\alpha]\!]_{\mathbb{T}} \text{ such that}\\
&\qquad \bar{\sigma} \models \pi \text{ and } init(\bar{\sigma}) = \sigma
\end{aligned}$$

The satisfaction of a trace formula π in a given trace $\bar{\sigma}$, i.e., checking if $\pi(\bar{\sigma})$ holds, is denoted as $\bar{\sigma} \models \pi$ and defined as follows:

$$\begin{aligned}
&\bar{\sigma} \models \phi_{tr} \iff \phi(init(\bar{\sigma}))\\
&\bar{\sigma} \models \neg\pi \iff \bar{\sigma} \not\models \pi\\
&\bar{\sigma} \models \Box\pi \iff \text{for all valid traces } \bar{\sigma}' \in suffix_{\mathbb{T}}(\bar{\sigma}),\ \bar{\sigma}' \models \pi\\
&\bar{\sigma} \models \Diamond\pi \iff \text{it exists a valid trace } \bar{\sigma}' \in suffix_{\mathbb{T}}(\bar{\sigma})\\
&\qquad \text{such that } \bar{\sigma}' \models \pi
\end{aligned}$$

where the set of suffixes of a dTL^2 trace is defined in the following and suffix denotes the standard list suffix operator.

$$\begin{aligned}
suffix_{\mathbb{T}}(\bar{\sigma}) \triangleq {} & \{s \mid s_0 \in \mathcal{E} \wedge s \in suffix(\bar{\sigma})\}\\
\cup {} & \{diff_{b-t}(\iota_b(t), \lambda s.\iota_b(t+s)) \cdot \bar{\sigma}' \mid diff_b(e, \iota_b) \cdot \bar{\sigma}'\\
& \qquad \in suffix(\bar{\sigma}) \wedge t < b\}\\
\cup {} & \{\iota_b(b) \cdot \bar{\sigma}' \mid diff_b(e, \iota_b) \cdot \bar{\sigma}' \in suffix(\bar{\sigma})\}
\end{aligned}$$

$$\cup \{inf(\iota_\infty(t), \lambda s.\iota_\infty(t+s)) \mid inf(e, \iota_\infty) \in suffix(\bar{\sigma}) \wedge t \geq 0\}$$

Note that the lifting function from a state formula to a trace formula is designed to work with the globally and eventually temporal operators. Since a suffix for a trace can begin at any point within that trace, the ability to check if the first value of a suffix satisfies some state formula is essential.

The formula satisfaction rules of all state formulas except the run quantifications are inherited from Plaidypvs. The satisfaction of a state formula lifted to traces, is interpreted over the first element of the trace. The temporal operator globally $\Box\pi$ is satisfied in a trace $\bar{\sigma}$ when all valid suffixes of $\bar{\sigma}$ satisfy π. Similarly, eventually $\Diamond\pi$ is satisfied in a trace $\bar{\sigma}$ when there exists a valid suffix of $\bar{\sigma}$ that satisfies π. Differently from [21], the satisfiability of temporal operators and run quantifications, i.e., someruns and allruns, is restricted to valid traces, i.e., traces that do not terminate in error. This was a specific design choice to allow the verification of reachability properties for all runs of a trace. For instance, consider the encoding of the loop $while\ (x \leq 10)\{x := x+1\}$ starting with $x = 0$ in dL. This can be specified by the hybrid program $\alpha_{while} \triangleq x := 0; (?\, x < 10; x := x+1)^*; ?\, x \geq 10$. The last test $?x \geq 10$ is necessary to ensure that the execution does not stop when $x < 10$. The trace semantics of α is defined as follows.

$$\begin{aligned}[\![\alpha_{while}]\!]_{\mathbb{T}} \triangleq \{&(x=0)\cdot\Lambda, \ldots\\ &(x=0)\cdot(x=1)\cdot\cdots\cdot(x=9)\cdot\Lambda, \ldots\\ &(x=0)\cdot(x=1)\cdot\cdots\cdot(x=10)\}\end{aligned}$$

In [21], $[\alpha_{while}]_{tr}\Diamond(x=10)$ cannot be proven since the definition of $[\cdot]$ quantifies over all traces, even non-valid ones. In the formalization presented here, this property is satisfied since non-valid traces, i.e., those ending in error, are discarded.

The formal specification developed in this work helped uncover other interesting details hidden in the original paper on dTL^2. As already mentioned, PVS is a strongly typed system, thus two versions of the run quantifiers have been introduced. In [21], just one operator is defined whose behavior is discriminated by the type of its argument (state or trace formula) and the lifting of state formulas is not rigorously defined. This leads to some confusion when these quantification operators are applied to state formulas, which can be interpreted over both states and traces. Notice that the behavior of a state formula is different when lifted to the trace domain, i.e.,

$$[\alpha]_{st}\phi \neq [\alpha]_{tr}\phi_{tr} \quad \text{and} \quad \langle\alpha\rangle_{st}\phi \neq \langle\alpha\rangle_{tr}\phi_{tr}.$$

The state interpretation has been defined to be compatible with dL. Thus, the formula is checked on the last environment of the trace (the output environment) and the trace is assumed to be finite. However, the trace interpretation is compatible with the classic temporal logic interpretation, where the formula is checked on the initial environment of the trace. Therefore, it is key to ensure the correct quantification operator is used in the dTL^2 proof rules to obtain a sound calculus.

The following interesting equivalence between the trace allruns and the state allruns has been formally proven in PVS for finite traces, which incidentally may lead to simpler specifications.

Lemma 3.9 *For all hybrid program α and state formula ϕ:*

$$[\alpha]_{st}\phi \iff [\alpha]_{tr}\Box\Diamond\phi_{tr}.$$

Proof. ($\rightarrow$). Suppose $[\alpha]_{st}\phi$, let $\bar{\sigma} \in [\![\alpha]\!]_{\mathbb{T}}$. If $\bar{\sigma}$ is a finite trace, then

$$(\Box\Diamond\phi_{tr})(\bar{\sigma}) \iff \exists\bar{\sigma}' \in suffix_{\mathbb{T}}(\bar{\sigma}) \text{ valid such that } \phi_{tr}(\bar{\sigma}')$$

This can be made true by choosing $\bar{\sigma}' = final(\bar{\sigma})$, i.e., the suffix containing only the final environment of $\bar{\sigma}'$. Since $[\alpha]_{st}\phi$, by definition $\phi_{tr}(trace')$, and the property is shown.

($\leftarrow$). Suppose $[\alpha]_{tr}\Box\Diamond\phi_{tr}$. Then for all valid and finite traces $\bar{\sigma}$ in $[\![\alpha]\!]_{\mathbb{T}}$, $\Box\Diamond\phi_{tr}$. It follows that the suffix defined as the last environment of $\bar{\sigma}$ also satisfies the formula $\Box\Diamond\phi_{tr}$, i.e., $(\Box\Diamond\phi_{tr})(final(\bar{\sigma}))$. By definition of $\Box$ and $\Diamond$, it follows that $(\phi)(final(\bar{\sigma}))$ and, by definition of $[\cdot]_{st}$, it can be concluded that $[\alpha]_{st}\phi$. □

4 Embedding the dTL^2 Calculus in PVS

This section presents the proof calculus of dTL^2 as formalized in Plaidypvs. Reasoning about statements involving temporal properties on hybrid programs relies on the sequent calculus of dTL^2 that is built upon the sequent calculus of dL already implemented in Plaidypvs. In this section, the dTL^2-Sequent is defined and the temporal rules of dTL^2 are presented as formally proven lemmas in PVS. Due to the changes in the formula satisfaction of Definition 3.8, which discard error traces, certain proof rules differ from the ones defined in [21].

In the dTL^2 calculus, temporal operators are allowed in the proof calculus with the restriction that they can be nested at most twice, i.e., the only combinations allowed are $\Box\Diamond$ and $\Diamond\Box$. While this can look like a stringent limitation, all possible nesting of temporal operators can be expressed as the nesting of at most two temporal operators. The combination of the temporal operators with the dTL^2 run quantifications allows for reasoning on the reachable states of different computational paths. This corresponds to a non-trivial fragment of CTL*. Nevertheless, notice that the nesting of Boolean operators inside temporal formulas to express (for instance) liveness properties of the form $\Box(\phi \rightarrow \Diamond\psi)$ is not yet supported in dTL^2.

4.1 The dTL^2-Sequent

The dTL^2-sequent in PVS is denoted $\Gamma \vdash \Delta$, where Γ and Δ are lists of Boolean hybrid program expressions and $\vdash$ is a predicate that returns a value in $\mathbb{B}$ (PVS's Boolean type)

associated with the statement

$$\forall e \in \mathcal{E}.\ \bigwedge_i \Gamma_i(e) \Longrightarrow \bigvee_j \Delta_j(e),$$

where $\Longrightarrow$ is the implication in PVS. The expressions Γ, Δ in this context are referred to as the dTL^2-antecedent and dTL^2-consequent, respectively. The dTL^2-sequent can be understood simply as the statement "the conjunction of the antecedent formulas implies the disjunction of the consequent formula".

When proving a statement about hybrid programs with temporal operators in Plaidypvs, the method of proof relies on the use of the defined logical rules of dTL^2 to manipulate the sequent so that the conjunction of the resulting statement after the application of the rules implies the original. In order to have a formally verified statement at each proof step, the rules of dTL^2 are written as formally proven lemmas in PVS.

A dTL^2 rule is applied to rewrite the statement of the sequent so that the sequent has no temporal operators, at which time the task of proving can now be done using the logic of dL with the rules specified in Plaidypvs.

4.2 Trace Formulas Normalization

The proof rules of dTL^2 are defined over normalized trace formulas. Normalization becomes necessary to reason on both a temporal property and the final state of a trace, if it exists. More details on the necessity of normalization can be found in [21].

The satisfaction of a normalized trace formula ξ is defined by cases depending if the trace is terminating or infinite.

Definition 4.1 *Given $\bar{\sigma} \in \mathbb{T}$ and $\phi, \psi \in \mathcal{B}$, normalized formulas are defined as follows.*

$$\bar{\sigma} \models \phi \sqcap \Box\psi \iff \begin{cases} \phi(\mathit{final}(\bar{\sigma})) \text{ and } \bar{\sigma} \models \Box\psi & \text{if } \bar{\sigma} \text{ is finite} \\ \bar{\sigma} \models \Box\psi & \text{otherwise} \end{cases}$$

$$\bar{\sigma} \models \phi \sqcup \Diamond\psi \iff \begin{cases} \phi(\mathit{final}(\bar{\sigma})) \text{ or } \bar{\sigma} \models \Diamond\psi & \text{if } \bar{\sigma} \text{ is finite} \\ \bar{\sigma} \models \Diamond\psi & \text{otherwise} \end{cases}$$

$$\bar{\sigma} \models (\phi \blacktriangleleft \pi) \iff \begin{cases} \phi(\mathit{final}(\bar{\sigma})) & \text{if } \bar{\sigma} \text{ is finite} \\ \bar{\sigma} \models \pi & \text{otherwise} \end{cases}$$

where π is either $\Box\Diamond\psi$ or $\Diamond\Box\psi$.

The first normalized trace formula is the normalized globally formula which extends the globally temporal operator $\Box\psi$ by also checking if ϕ is satisfied on the last state when the trace is finite. The second normalized trace formula is the normalized eventually formula which is equivalent to $\Diamond\psi$ for infinite traces and in the finite case either ϕ is satisfied on the last state or $\Diamond\psi$ holds. The last normalized formula handles the nested temporal operators. In the infinite trace case the normalized formula reduces to the nested temporal operator while the finite case is satisfied by checking the last state of the trace on ϕ.

$$\Box\phi \rightsquigarrow \top \sqcap \Box\phi \qquad \Box\Diamond\phi \rightsquigarrow \phi \blacktriangleleft \Box\Diamond\phi$$
$$\Diamond\phi \rightsquigarrow \bot \sqcup \Diamond\phi \qquad \Diamond\Box\phi \rightsquigarrow \phi \blacktriangleleft \Diamond\Box\phi$$

Figure 1. Rewrites for the normalization of trace formulas.

The proof calculus of dTL^2 includes rules to normalize any trace formula. These rewriting rules are given in Figure 1. With these rules, one more layer is added to the proof method of dTL^2. In fact, the replacement of any non-normalized formulas with an equivalent normalized formula is needed before applying any other dTL^2 rule.

4.3 dTL^2 Rules

A dTL^2 rule has the form

$$\frac{P}{Q}$$

meaning that P implies Q. The following double-bar notation

$$\frac{P}{\overline{\overline{Q}}}$$

is introduced to denote that P and Q are equivalent.

The dTL^2 rules are shown in Figure 2 and the associated dual rules are in Figure 3. Both sets of rules are formalized and proved correct in PVS and have been included in Plaidypvs.

As mentioned before, the modification of the satisfaction relation in Definition 3.8 that considers just valid traces causes some rules defined in [21] to no longer hold. For instance, consider the rule ($[?]\sqcap$) in [21] given by

$$\frac{(\neg\chi \vee \psi) \wedge \phi}{\overline{\overline{[?\,\chi]_{tr}(\psi \sqcap \Box\phi)}}} \tag{4.1}$$

This statement is true for both directions of the implication if error traces are considered. However, without error traces the backward implication is not true anymore. This is because there is no guarantee that the initial state satisfies the Boolean expression in the test.

Nevertheless, it is useful to have these equivalences as rewriting rules, especially in PVS where this property will allow for a direct simplification of the sequent. To this aim, in addition to the original rules in [21] that still hold removing error traces, a set of modified equivalence rules have been added to the dTL^2 PVS formalization. For instance, the rewriting rule in (4.1) has been replaced with the following one in Plaidypvs.

$$\frac{\neg\chi \vee (\psi \wedge \phi)}{\overline{\overline{[?\chi]_{tr}(\psi \sqcap \Box\phi)}}} \tag{4.2}$$

In addition, the direction of (4.1), which still holds, has also been included in Plaidypvs.

Figure 2 groups the dTL^2 proof rules by the type of hybrid program they consider. The rules with $[;]$ are the sequential

$$\frac{[\alpha]_{tr}(([\beta]_{tr}(\phi \sqcap \Box\psi)) \sqcap \Box\psi)}{[\alpha;\beta]_{tr}(\phi \sqcap \Box\psi)}\ ([;]\sqcap) \qquad \frac{[\alpha;\beta]_{tr}(\phi \sqcap \Box\psi)}{[\alpha]_{tr}(\phi \sqcap \Box\psi) \vee \langle\alpha\rangle_{st}([\beta]_{tr}(\phi \sqcap \Box\psi))}\ ([;] \sqcap \text{con})$$

$$\frac{[\alpha]_{tr}(([\beta]_{tr}(\phi \sqcup \Diamond\psi)) \sqcup \Diamond\psi)}{[\alpha;\beta]_{tr}(\phi \sqcup \Diamond\psi)}\ ([;]\sqcup) \qquad \frac{[\alpha]_{tr}(([\beta]_{tr}(\phi \blacktriangleleft \Box\Diamond\pi)) \blacktriangleleft \Box\Diamond\pi)}{[\alpha;\beta]_{tr}(\phi \blacktriangleleft \Box\Diamond\pi)}\ ([;] \blacktriangleleft \Box)$$

$$\frac{[\alpha]_{tr}(([\beta]_{tr}(\phi \blacktriangleleft \Diamond\Box\pi)) \blacktriangleleft \Diamond\Box\pi)}{[\alpha;\beta]_{tr}(\phi \blacktriangleleft \Diamond\Box\pi)}\ ([;] \blacktriangleleft \Diamond) \qquad \frac{[\alpha]_{tr}\xi \wedge [\beta]_{tr}\xi}{[\alpha \cup \beta]_{tr}\xi}\ ([\cup]\xi)$$

$$\frac{\neg\chi \vee (\phi \wedge \psi)}{[?\chi]_{tr}(\phi \sqcap \Box\psi)}\ ([?]\sqcap) \qquad \frac{\neg\chi \vee (\phi \vee \psi)}{[?\chi]_{tr}(\phi \sqcup \Diamond\psi)}\ ([?]\sqcup) \qquad \frac{\neg\chi \vee \phi}{[?\chi]_{tr}(\phi \blacktriangleleft \Box\Diamond\psi)}\ ([?] \blacktriangleleft \Box) \qquad \frac{\neg\chi \vee \phi}{[?\chi]_{tr}(\phi \blacktriangleleft \Diamond\Box\psi)}\ ([?] \blacktriangleleft \Diamond)$$

$$\frac{\phi \wedge [x := \theta]_{st}(\psi \wedge \phi)}{[x := \theta]_{tr}(\psi \sqcap \Box\phi)}\ ([:=]\sqcap) \qquad \frac{\phi \vee [x := \theta]_{st}(\psi \vee \phi)}{[x := \theta]_{tr}(\psi \sqcup \Diamond\phi)}\ ([:=]\sqcup) \qquad \frac{[x := \theta]_{st}\phi}{[x := \theta]_{tr}(\phi \blacktriangleleft \pi)}\ ([:=] \blacktriangleleft)$$

$$\frac{\neg\chi \vee (\phi \wedge [\bar{x} = \bar{\theta}\&\chi]_{st}(\psi \wedge \phi))}{[\bar{x} = \bar{\theta}\&\chi]_{tr}(\psi \sqcap \Box\phi)}\ ([']\sqcap) \qquad \frac{(\neg\chi \vee \phi \vee \psi) \wedge [\bar{x} = \bar{\theta}\&\chi \wedge \neg\phi]_{st}\psi \wedge (\neg\chi \vee \langle\bar{x} = \bar{\theta}\rangle_{st}(\neg\chi \vee \phi))}{[\bar{x} = \bar{\theta}\&\chi]_{tr}(\psi \sqcup \Diamond\phi)}\ ([']\sqcup)$$

$$\frac{(\neg\chi \vee \psi) \wedge [\bar{x} = \bar{\theta}\&\chi]_{st}\psi \wedge (\langle\bar{x} = \bar{\theta}\rangle_{st}\neg\chi \vee [\bar{x} = \bar{\theta}]_{st}\langle\bar{x} = \bar{\theta}\rangle_{st}\phi)}{[\bar{x} = \bar{\theta}\&\chi]_{tr}(\psi \blacktriangleleft \Box\Diamond\phi)}\ (['] \blacktriangleleft \Box)$$

$$\frac{(\neg\chi \vee \psi) \wedge [\bar{x} = \bar{\theta}\&\chi]_{st}\psi \wedge (\langle\bar{x} = \bar{\theta}\rangle_{st}\neg\chi \vee \langle\bar{x} = \bar{\theta}\rangle_{st}[\bar{x} = \bar{\theta}]_{st}\phi)}{[\bar{x} = \bar{\theta}\&\chi]_{tr}(\psi \blacktriangleleft \Diamond\Box\phi)}\ (['] \blacktriangleleft \Diamond)$$

$$\frac{(\phi \wedge \psi) \wedge [\alpha^*]_{st}[\alpha]_{tr}(\phi \sqcap \Box\psi)}{[\alpha^*]_{tr}(\phi \sqcap \Box\psi)}\ ([*]\sqcap) \qquad \frac{\phi \vee \psi \wedge [\alpha;\alpha^*]_{tr}(\phi \sqcup \Diamond\psi)}{[\alpha^*]_{tr}(\phi \sqcup \Diamond\psi)}\ ([*]\sqcup) \qquad \frac{\phi \Rightarrow [\alpha]_{tr}(\phi \sqcup \Diamond\psi)}{[\alpha^*]_{tr}(\phi \sqcup \Diamond\psi)}\ (\text{ind}\sqcup)$$

$$\frac{\phi \wedge [\alpha^*]_{st}[\alpha]_{tr}(\phi \blacktriangleleft \pi)}{[\alpha^*]_{tr}(\phi \blacktriangleleft \pi)}\ ([*] \blacktriangleleft) \qquad \frac{\forall r > 0.\,(\varphi(r) \Rightarrow \langle\alpha\rangle_{tr}(\varphi(r-1) \sqcap \Box\psi))}{(\exists r\, \varphi(r)) \wedge \psi \Rightarrow \langle\alpha^*\rangle_{tr}((\exists r \leq 0.\,\varphi(r)) \sqcap \Box\psi)}\ (\text{con}\sqcap)$$

Figure 2. Rules of the dTL^2 proof calculus.

composition rules. The sequential composition is the case that required the most care in the development of the dTL^2 formalization. Notice that these rules keep a temporal operator in the statement, even with replacement. This may seem like a problem if the goal is to eventually rely on the proof rules of dL, but notice that unless α or β are defined by a sequential hybrid program, in one or two more replacement rules the temporal statement can be transformed into a non-temporal statement. This exact situation can be seen in Example 5.4.

The rule [$\cup$] for the non-deterministic choice extends the corresponding rule of dL. The assignment rules, [:=], behave as expected since the traces that satisfy the semantics of an assign statement always end in a finite and discrete trace. Therefore, the normalized formula is always interpreted in the finite case.

For the test rules [?], the only valid traces in the semantics are the ones where the test passes, meaning the initial environment belongs to the Boolean expression in the test. This forces any valid traces to be finite, in addition, they will have a length equal to one and the element in the trace list will be of type $\mathcal{E}$. Thus, the normalization formulas will all be in the first case of their definition. For example, the temporal statement in rule ([?]$\sqcap$) is equivalent to the statement $\chi \Rightarrow \phi \wedge \psi$. In fact, since the trace is of length one, the globally and eventually operators are equivalent in these rules.

Traces that satisfy the differential hybrid program can be either finite or infinite. An added level of complexity comes

$$\frac{\langle\alpha\rangle_{tr}((\langle\beta\rangle_{tr}(\phi \sqcap \Box\psi)) \sqcap \Box\psi)}{\langle\alpha;\beta\rangle_{tr}(\phi \sqcap \Box\psi)}\ (\langle;\rangle\sqcap)$$

$$\frac{\langle\alpha\rangle_{tr}(\langle\beta\rangle_{tr}(\phi \sqcup \Diamond\psi)) \vee \langle\alpha;\bot\rangle_{tr}(\phi \sqcup \Diamond\psi)}{\langle\alpha;\beta\rangle_{tr}(\phi \sqcup \Diamond\psi)}\ (\langle;\rangle\sqcup) \qquad \frac{\langle\alpha;\beta\rangle_{tr}(\phi \sqcup \Diamond\psi)}{\langle\alpha\rangle_{st}(\langle\beta\rangle_{tr}(\phi \sqcup \Diamond\psi))}\ (\langle;\rangle \sqcup \text{con})$$

$$\frac{\langle\alpha\rangle_{tr}((\langle\beta\rangle_{tr}(\phi \blacktriangleleft \Box\Diamond\pi)) \blacktriangleleft \Box\Diamond\pi)}{\langle\alpha;\beta\rangle_{tr}(\phi \blacktriangleleft \Box\Diamond\pi)}\ (\langle;\rangle \blacktriangleleft\Box)$$

$$\frac{\langle\alpha\rangle_{tr}((\langle\beta\rangle_{tr}(\phi \blacktriangleleft \Diamond\Box\pi)) \blacktriangleleft \Diamond\Box\pi)}{\langle\alpha;\beta\rangle_{tr}(\phi \blacktriangleleft \Diamond\Box\pi)}\ (\langle;\rangle \blacktriangleleft\Diamond) \qquad \frac{\langle\alpha\rangle_{tr}\xi \vee \langle\beta\rangle_{tr}\xi}{\langle\alpha \cup \beta\rangle_{tr}\xi}\ (\langle\cup\rangle\xi)$$

$$\frac{\chi \wedge (\phi \wedge \psi)}{\langle?\chi\rangle_{tr}(\phi \sqcap \Box\psi)}\ (\langle?\rangle\sqcap) \qquad \frac{\chi \wedge (\phi \vee \psi)}{\langle?\chi\rangle_{tr}(\phi \sqcup \Diamond\psi)}\ (\langle?\rangle\sqcup) \qquad \frac{\chi \wedge \phi}{\langle?\chi\rangle_{tr}(\phi \blacktriangleleft \Box\Diamond\psi)}\ (\langle?\rangle \blacktriangleleft\Box) \qquad \frac{\chi \wedge \phi}{\langle?\chi\rangle_{tr}(\phi \blacktriangleleft \Diamond\Box\psi)}\ (\langle?\rangle \blacktriangleleft\Diamond)$$

$$\frac{\phi \wedge \langle x := \theta\rangle_{st}(\psi \wedge \phi)}{\langle x := \theta\rangle_{tr}(\psi \sqcap \Box\phi)}\ (\langle:=\rangle\sqcap) \qquad \frac{\phi \vee \langle x := \theta\rangle_{st}(\psi \vee \phi)}{\langle x := \theta\rangle_{tr}(\psi \sqcup \Diamond\phi)}\ (\langle:=\rangle\sqcup) \qquad \frac{\langle x := \theta\rangle_{st}\phi}{\langle x := \theta\rangle_{tr}(\phi \blacktriangleleft \pi)}\ (\langle:=\rangle \blacktriangleleft)$$

$$\frac{(\chi \wedge \phi \wedge \psi) \vee \langle\bar{x} = \bar{\theta}\&\chi \wedge \phi\rangle_{st}\psi \vee [\bar{x} = \bar{\theta}]_{st}(\chi \wedge \phi)}{\langle\bar{x} = \bar{\theta}\&\chi\rangle_{tr}(\psi \sqcap \Box\phi)}\ (\langle'\rangle\sqcap) \qquad \frac{(\chi \wedge (\phi \vee \psi)) \vee \langle\bar{x} = \bar{\theta}\&\chi\rangle_{st}(\phi \vee \psi)}{\langle\bar{x} = \bar{\theta}\&\chi\rangle_{tr}(\psi \sqcup \Diamond\phi)}\ (\langle'\rangle\sqcup)$$

$$\frac{(\chi \wedge \psi) \vee \langle\bar{x} = \bar{\theta}\&\chi\rangle_{st}\psi \vee ([\bar{x} = \bar{\theta}]_{st}\chi \wedge \langle\bar{x} = \bar{\theta}\rangle_{st}[\bar{x} = \bar{\theta}]_{st}\phi)}{\langle\bar{x} = \bar{\theta}\&\chi\rangle_{tr}(\psi \blacktriangleleft \Box\Diamond\phi)}\ (\langle'\rangle \blacktriangleleft\Box)$$

$$\frac{(\chi \wedge \psi) \vee \langle\bar{x} = \bar{\theta}\&\chi\rangle_{st}\psi \vee ([\bar{x} = \bar{\theta}]_{st}\chi \wedge \langle\bar{x} = \bar{\theta}\rangle_{st}[\bar{x} = \bar{\theta}]_{st}\phi)}{\langle\bar{x} = \bar{\theta}\&\chi\rangle_{tr}(\psi \blacktriangleleft \Diamond\Box\phi)}\ (\langle'\rangle \blacktriangleleft\Diamond)$$

$$\frac{\phi \vee \langle\alpha^*\rangle_{st}\langle\alpha\rangle_{tr}(\phi \sqcup \Diamond\psi)}{\langle\alpha^*\rangle_{tr}(\phi \sqcup \Diamond\psi)}\ (\langle*\rangle\sqcup) \qquad \frac{\psi \wedge (\phi \vee \langle\alpha;\alpha^*\rangle_{tr}(\phi \sqcap \Box\psi))}{\langle\alpha^*\rangle_{tr}(\phi \sqcap \Box\psi)}\ (\langle*\rangle\sqcap) \qquad \frac{\phi \vee \langle\alpha^*\rangle_{st}\langle\alpha\rangle_{tr}(\phi \blacktriangleleft \pi)}{\langle\alpha^*\rangle_{tr}(\phi \blacktriangleleft \pi)}\ (\langle*\rangle \blacktriangleleft)$$

Figure 3. Dual rules of the dTL^2 proof calculus.

from the fact that a differential can end at any time in the execution, which leads to interesting behaviors when reasoning about the temporal properties of a system. All the differential rules $[']$ transform a temporal formula into a temporal free formula that can then be handled by the proof calculus of dL. In the $([']\ \blacktriangleleft\Diamond)$ and $([']\ \blacktriangleleft\Box)$ rules there is a nesting of allruns and someruns operators that handles the case of infinite traces. The proof of these rules requires leveraging the uniqueness of a solution to a differential equation. For example, in the proof of $([']\ \blacktriangleleft\Box)$ one branch assumes that for a differential system $[\bar{x} = \bar{\theta}]\langle\bar{x}' = \bar{\theta}\rangle\psi$ and a given initial state e_i a function $\bar{f}$ exists such that $sol(\mathbb{R}_{\geq 0}, \bar{x}, \bar{\theta}, e_i, \bar{f})$. Then, the nesting of the modal operators someruns and allruns requires proving that for a state occurring at time r, where $e_o = env(\bar{x}, \bar{f}, e_i, r)$, there exists a unique solution $\bar{g}$ on $\mathbb{R}_{\geq 0}$ with e_o as the initial state such that $\forall t \in \mathbb{R}_{\geq 0}.\ \bar{g}(t) = \bar{f}(t+r)$.

The last set of rules in Figure 2 are the repetition rules $[*]$. Since only finite, but unbounded, repetitions are allowed, these are proved through various induction arguments. The rules $([*]\sqcap)$ and $([*]\ \blacktriangleleft)$ transform a temporal statement about a repetition into a temporal statement about the hybrid program in the repetition, which can then be handled by the other rules.

The rule $(\text{ind}\sqcup)$ is the induction rule and extends its dL analog. Rule $(\text{con}\sqcap)$ is a convergence rule for the decreasing invariant φ that extends the corresponding rule in dL. Rules $(\text{ind}\sqcup)$ and $(\text{con}\sqcap)$ do not have a dual counterpart.

The dual rules in Figure 3 are organized similarly to Figure 2 and behave analogously to their counterpart. Similarly

to the rules in Figure 2, the dual rules in Figure 3 are modified from the rules for dTL^2 found in [21] due to the elimination of error traces.

5 Proving Temporal Properties in Plaidypvs

With the work presented in this paper, Plaidypvs now allows for reasoning about hybrid programs using dTL^2 within PVS. Examples that showcase the types of properties that are now possible to prove formally using this temporal extension can be found in the PVS library . The proof structure of these examples generally follows the following procedure. First, given a statement written with the temporal operators $\Diamond$ and $\Box$ the replacement formulas from Figure 1 are used. Then the rules from Figure 2 and Figure 3 are used until either the statement can be proven through algebraic manipulations or by calling rules from the core of dL.

The following examples are proven in PVS. These examples show both the functionality of Plaidypvs and give details of the proof method.

Example 5.1 Consider the hybrid program discussed in Example 2.1 given by

$$x' := 6; x := \lfloor x \rfloor \tag{5.1}$$

This hybrid program has the property that every output of the variable x is an integer, written in the logic of dL as the following sequent.

$$\vdash [x' := 6; x := \lfloor x \rfloor](x \in \mathbb{Z}). \tag{5.2}$$

As noted in Example 2.1, it is possible that there is a run with intermediate states where x takes non-integer values. This statement written in dTL^2 is

$$\vdash \langle x' = 6; x := \lfloor x \rfloor \rangle_{tr} \Diamond (x \notin \mathbb{Z}). \tag{5.3}$$

The specification of this hybrid program and these properties in Plaidypvs is shown in Figure 4. The PVS function `State_Trace` lifts a state formula ϕ to a trace one ϕ_{tr}. The lemma `eventually_not_int` formalize the property in (5.3), while the lemma `end_state_int` formalize the property in (5.2).

```
floor_re(re:RealExpr)(env:Environment): real = floor(re(env))
integer_pred_re?(re:RealExpr)(env:Environment): bool = integer_pred(re(env))

prove | discharge-tccs | status-proofchain | show-prooflite
eventually_not_int: LEMMA
SOMERUNS_tr(SEQ(DIFF((:(x,cnst(6)):)),ASSIGN((:(x,floor_re(val(x))):))),
            DLEVENTUALLY(State_Trace(NOT integer_pred_re?(val(x)))))

prove | discharge-tccs | status-proofchain | show-prooflite
end_state_int: LEMMA
ALLRUNS(SEQ(DIFF((:(x,cnst(6)):)),ASSIGN((:(x,floor_re(val(x))):))),
                                    integer_pred_re?(val(x)))
```

Figure 4. Specification of the input/output property (5.2) and temporal property (5.3) for the hybrid program (5.1).

These properties specified in Plaidypvs are proved using the interactive theorem prover environment of PVS. For the non-temporal rule, the application of dL rules is done using proof strategies specified in Plaidypvs. Figure 5 illustrates part of the proof tree consisting of dL commands. The commands (dl-assert) and (dl-grind) simplify the statement using rewrites. The (dl-solve) command solves linear differential equations. For the proof of the temporal property, the application of the dTL^2 rules transforms the statement to a non-temporal statement. This can be seen in Figure 6. At this point, the application of dL rules can be done to complete the proof. Notice that the problem reduces to proving

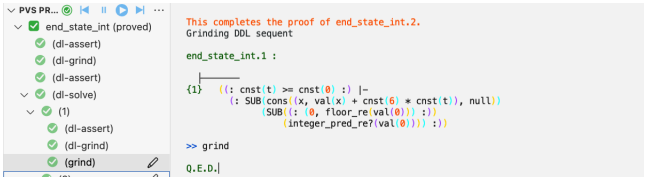

Figure 5. Prover session in Plaidypvs for the non-temporal sequent (5.2).

Figure 6. Prover session in Plaidypvs for the temporal sequent (5.3).

the statement that there is some run of the differential equation that reaches a non-integer state. This can be shown by applying the dl-solve prover command. ■

Example 5.2 The dynamics of a fixed-wing aircraft turning at a constant velocity, as depicted in Figure 7, can be modeled as the following hybrid program

$$\alpha_{fw} \triangleq (x' := -y, y' := x),$$

where x and y denote the 2D position of the aircraft turning [51]. The following temporal property

$$P \vdash [\alpha_{fw}; ?\bot]_{tr} \Box \Diamond P, \tag{5.4}$$

where $P \triangleq (x = x_0 \wedge y = y_0)$, states that with the turning dynamics, the aircraft returns to its original position an arbitrary number of times. Note that the semantics of the hybrid program $\alpha_{fw}; ?\bot$ exclusively contains the infinite traces of α_{fw}. In fact, any finite trace will end in an environment that does not satisfy $\bot$ and the corresponding trace will end with an error, according to Definition 3.4.

The specification of the hybrid program and property in Plaidypvs is given in Figure 8.

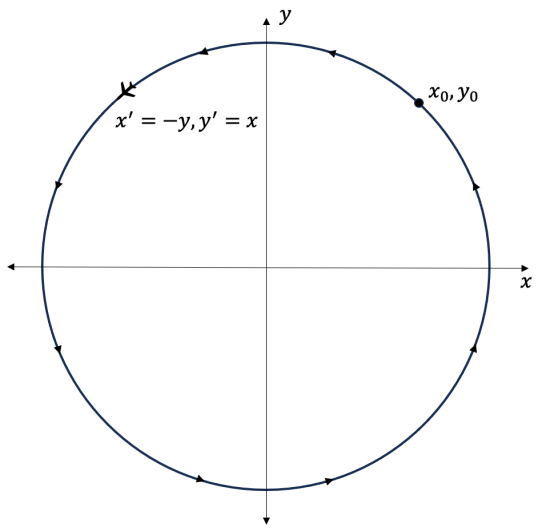

Figure 7. Visualization of fixed-wing aircraft dynamics as described in Example 5.2. The aircraft turning at a constant rate repeatedly crosses the point (x_0, y_0).

```
prove | discharge-tccs | status-proofchain | show-prooflite
return_turn: LEMMA
FORALL(x0,y0:real):
val(x)=cnst(x0) AND val(y)=cnst(y0) IMPLIES
 ALLRUNS_tr(SEQ(DIFF(turn_odes),TEST(DLFALSE)),
  DLGLOBALLY(DLEVENTUALLY(State_Trace(DLAND(val(x)=cnst(x0),
                                            val(y)=cnst(y0)))))))
```

Figure 8. Specification of temporal property given in (5.4) on the dynamics of a turning fixed-wing aircraft.

The property (5.4) can be proven in Plaidypvs using the rules of dL and dTL^2. After the rewrites in Figure 1 are applied, the sequent is

$$P \vdash [\alpha_{fw}; ?\,\bot]_{tr} P \blacktriangleleft \Box\Diamond P. \tag{5.5}$$

Applying the $([;] \blacktriangleleft \Box)$ then the $([']\blacktriangleleft\Box)$ rule generates three subgoals

$$P \vdash \neg\top \vee ([?\,\bot]_{tr} P \blacktriangleleft \Box\Diamond P) \tag{5.6}$$
$$P \vdash [\alpha_{fw}]_{st} ([?\,\bot]_{tr} P \blacktriangleleft \Box\Diamond P) \tag{5.7}$$
$$P \vdash \langle\alpha_{fw}\rangle_{st} \neg\top \vee [\alpha_{fw}]_{st} \langle\alpha_{fw}\rangle_{st} P \tag{5.8}$$

Figure 9 depicts the interactive theorem prover environment with the subgoals (5.6), (5.7), and (5.8). The proof of subgoal (5.6) uses the $([?]\blacktriangleleft\Box)$ rule, as well as the differential weakening rule (dl-weak) and propositional simplification rule (assert) from Plaidypvs and is given by

$$\dfrac{\dfrac{\dfrac{\top}{\vdash \neg\bot \vee P}\,\text{(dl-assert)}}{\vdash ([?\,\bot]_{tr} P \blacktriangleleft \Box\Diamond P)}\,([?]\blacktriangleleft\Box)}{P \vdash [\alpha]_{st} ([?\,\bot]_{tr} P \blacktriangleleft \Box\Diamond P)}\,\text{(dl-weak)}$$

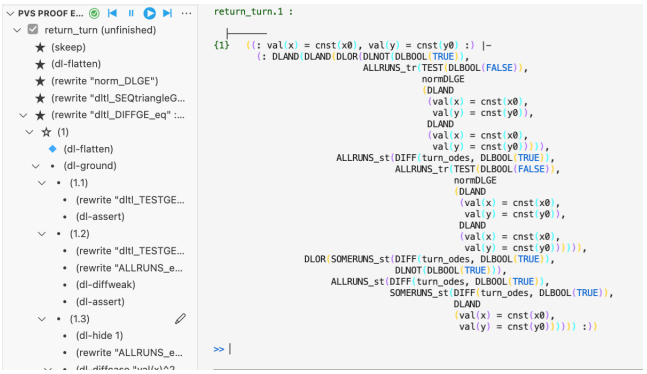

Figure 9. The interactive theorem prover environment of PVS with the three subgoals (5.6), (5.7), and (5.8).

The proof of subgoal (5.7) also uses the $([?]\blacktriangleleft\Box)$ rule and the (dl-assert) rule from Plaidypvs.

$$\dfrac{\dfrac{\dfrac{\top}{P \vdash \neg\bot \vee P}\,\text{(dl-assert)}}{P \vdash ([?\,\bot]_{tr} P \blacktriangleleft \Box\Diamond P)}\,([?]\blacktriangleleft\Box)}{P \vdash \neg\top \vee ([?\,\bot]_{tr} P \blacktriangleleft \Box\Diamond P)}\,\text{(dl-assert)}$$

The proof of the final subgoal (5.8) is shown in Figure 10. The (dl-hide) rule hides part of a disjunction, the (dl-cut) rule is the differential cut rule that splits the sequent in two cases: one shows that the dynamics never leaves a region, and the other restricts the continuous dynamics to that region. The (dl-solve) replaces a continuous evolution with the solutions of its differential equation. The (dl-inv) rule is the differential invariant rule that checks if a property is an invariant through rules of differentiation and substitutions. See [51] for more details about these rules.

■

Example 5.3 The example depicted in Figure 11 shows that an aircraft beginning in an unsafe region, under some return-to-safe dynamics, eventually returns and stays in a safe region. Let $S \in \mathbb{R}^2$ be the safe region, $s \in \mathbb{R}^2$, and $C_{s,r}$ be the circle of radius $r > 0$ centered at s such that $C_{s,r} \subseteq S$. Define the return-to-safe dynamics for some $k > 0$

$$rts \triangleq (?(x, y \notin C_{s,r}); (x' = -k(x - s_x), y' = -k(y - s_y))) \\ \cup\ (?(x, y \in C_{s,r}); (x' = 0, y' = 0))$$

which corresponds to the aircraft flying towards the point s, until it is within r to it, at which point the aircraft stops moving. For a starting point $p_0 \notin C_{s,r}$, it can be shown these dynamics will eventually always be in S.

$$[rts^*; ?\bot]_{tr} \Diamond\Box(x, y \in S).$$

Normalizing this expression becomes:

$$[rts^*; ?\bot]_{tr} x, y \in S \blacktriangleleft \Diamond\Box x, y \in S.$$

Applying the $([;]\blacktriangleleft\Diamond)$, $([*]\blacktriangleleft)$ rule generates two subgoals

$$\vdash [?\,\bot]_{tr}\,(x, y \in S \blacktriangleleft \Diamond\Box x, y \in S)\,, \tag{5.9}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{x(0)^2+y(0)^2=x_0^2+y_0^2 \vdash \exists t \geq 0.\, x(0)\cos(t)-y(0)\sin(t)=x_0 \wedge x(0)\sin(t)+y(0)\cos(t)=y_0}\,(\text{dl-solve})}{x^2+y^2=x_0^2+y_0^2 \vdash \langle\alpha\rangle_{st}P}\,(\text{dl-weak})}{P \vdash [x'=y,y'=-x\,\&\,x^2+y^2=x_0^2+y_0^2]_{st}\langle\alpha\rangle_{st}P} \quad \dfrac{\dfrac{\top}{P \vdash x^2+y^2=x_0^2+y_0^2}\,(\text{dl-assert}) \quad \dfrac{\top}{\vdash 2x(-y)+2yx=0}\,(\text{dl-assert})}{P \vdash [\alpha]_{st}Q}\,(\text{dl-inv})}{P \vdash [\alpha]_{st}\langle\alpha\rangle_{st}P}\,(\text{dl-cut})}{P \vdash \langle\alpha\rangle_{st}\neg\top \vee [\alpha]_{st}\langle\alpha\rangle_{st}P}\,(\text{dl-hide})$$

Figure 10. Proof of subgoal (5.8) in Example 5.2

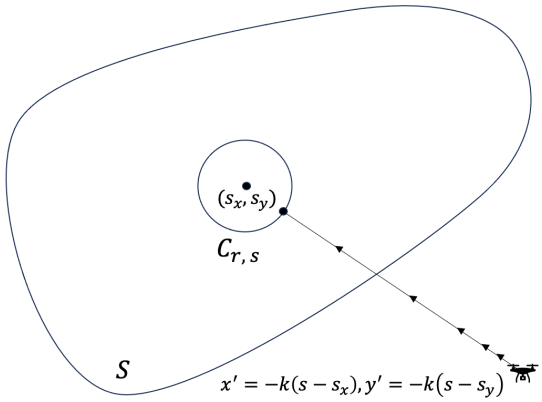

Figure 11. Visualization of Example 5.3, which shows an aircraft returning to a safe region S. This shows one run where the aircraft flies to the boundary of the circle of radius r around some point $s \in S$.

$$\vdash [rts^*]_{st}[rts]_{tr}\,([?\,\bot]_{tr}\,(x,y \in S \blacktriangleleft \Diamond\Box x, y \in S) \blacktriangleleft \Diamond\Box x, y \in S) \tag{5.10}$$

The proof of subgoal (5.10) uses the $([?] \blacktriangleleft \Diamond)$ rule and propositional simplification rule (assert) from Plaidypvs and is given by

$$\dfrac{\dfrac{\top}{\vdash \neg\bot \vee (x,y \in S \blacktriangleleft \Diamond\Box x, y \in S)}\,(\text{dl-assert})}{\vdash [?\,\bot]_{tr}\,(x,y \in S \blacktriangleleft \Diamond\Box x, y \in S)}\,([?] \blacktriangleleft \Diamond)$$

Applying the dL differential weakening (dl-weak) rule to subgoal (5.10) gives the simplified sequent

$$\vdash [rts]_{tr}\,([?\,\bot]_{tr}\,(x,y \in S \blacktriangleleft \Diamond\Box x, y \in S) \blacktriangleleft \Diamond\Box x, y \in S)\,. \tag{5.11}$$

Applying the $([\cup]\xi)$ to subgoal (5.11) gives the following two subgoals

$$\vdash [?(x,y \in C_{s,r}); (x'=0, y'=0))]_{tr}\,([?\,\bot]_{tr}\,(x,y \in S \blacktriangleleft \Diamond\Box x, y \in S) \blacktriangleleft \Diamond\Box x, y \in S)\,, \tag{5.12}$$

and

$$\vdash [?(x,y \notin C_{s,r}); (x'=-k(x-s_x), y'=-k(y-s_y))]_{tr}\,([?\,\bot]_{tr}\,(x,y \in S \blacktriangleleft \Diamond\Box x, y \in S) \blacktriangleleft \Diamond\Box x, y \in S) \tag{5.13}$$

Both subgoals (5.11) and (5.12), can be simplified to dL expressions using the rules of dTL^2, and have been proven in Plaidypvs. The details of the proof of subgoal (5.13) are shown in Figure 12. ■

Example 5.4 Plaidypvs supports reasoning on subtypes of hybrid programs. This feature allows a user to define features of the hybrid program without specifying the exact instance to prove general properties about a family of programs. For example, consider the collection of differential equations with unique solutions of the form

$$F \triangleq \{x' = f_x, y' = f_y \mid \exists \epsilon > 0\ \forall t \in \mathbb{R}_{\geq 0}.f_y(x(t),y(t)) > \epsilon\}$$

that models a collection of aircraft dynamics as depicted in Figure 13. Take an element $f \in F$, then it can be shown that

$$[f;?(y \geq 0)]_{tr}\Diamond\Box y \geq 0 \tag{5.14}$$

meaning that a differential equation defined where the derivative of y is increasing eventually reaches the safe region defined by the set $y \geq 0$ and will stay in that set for the remainder of the run.

The PVS specification of the problem is shown in Figure 14. Figure 15 gives an example of calling a rule in the interactive theorem prover environment for the proof of the specification of (5.14).

Note that this statement is trivial for finite traces. This can be seen by considering the sequential operator with a test at the end. In one case the differential equation reaches the set in the test, thus the test is redundant. In the other case, the dynamics never allow a solution to reach the region in the test, so no traces belong to the semantics, making the statement vacuously true.

This property becomes very interesting when considering infinite traces. In fact, it can be shown that for time going to infinity, the solution of the differential equation will stay in the set. This case is non-trivial to prove as it relies on showing that the differential equation satisfies

$$\langle f\rangle_{st}[f]_{st}y \geq 0 \tag{5.15}$$

$$\dfrac{\dfrac{\dfrac{\top}{\vdash \neg\top \vee \neg\bot \vee x, y \in S}\,(\text{dl-assert})}{\vdash \neg\top \vee Q}\,([?] \blacktriangleleft \Diamond) \qquad \dfrac{\dfrac{\top}{\vdash \neg\bot \vee x, y \in S}\,(\text{dl-assert})}{\vdash Q}\,(\text{dl-weak})}{\vdash \left([(x' = -k(x - s_x), y' = -k(y - s_y))]_{tr} Q\right.} \qquad \dfrac{\vdots}{\vdash \langle (x' = -k(x - s_x), y' = -k(y - s_y))\rangle_{st} \neg\top \vee \langle (x' = -k(x - s_x), y' = -k(y - s_y))\rangle_{st} [(x' = -k(x - s_x), y' = -k(y - s_y))]_{st} x, y \in S}\,([']\blacktriangleleft\Diamond)$$

$$\dfrac{\dfrac{\vdash \left([(x' = -k(x - s_x), y' = -k(y - s_y))]_{tr} ([?\,\bot]_{tr} (x, y \in S \blacktriangleleft \Diamond\Box x, y \in S) \blacktriangleleft \Diamond\Box x, y \in S)\right)}{\vdash \neg\left(?(x, y \notin C_{s,r})\right]) \vee \left([(x' = -k(x - s_x), y' = -k(y - s_y))]_{tr} (Q \blacktriangleleft \Diamond\Box x, y \in S)\right)}\,(\text{dl-hide})}{\vdash [?(x, y \notin C_{s,r})]_{tr}}\,([?] \blacktriangleleft \Diamond)$$

$$\dfrac{\left([(x' = -k(x - s_x), y' = -k(y - s_y))]_{tr} ([?\,\bot]_{tr} (x, y \in S \blacktriangleleft \Diamond\Box x, y \in S) \blacktriangleleft \Diamond\Box x, y \in S)\right) \blacktriangleleft \Diamond\Box x, y \in S}{\vdash [(?(x, y \notin C_{s,r}); (x' = -k(x - s_x), y' = -k(y - s_y)))]_{tr} ([?\,\bot]_{tr} (x, y \in S \blacktriangleleft \Diamond\Box x, y \in S) \blacktriangleleft \Diamond\Box x, y \in S)}\,([;] \blacktriangleleft \Diamond)$$

Figure 12. Proof of subgoal (5.13) in Example 5.3 where $Q = [?\,\bot]_{tr}\,(x, y \in S \blacktriangleleft \Diamond\Box x, y \in S)$.

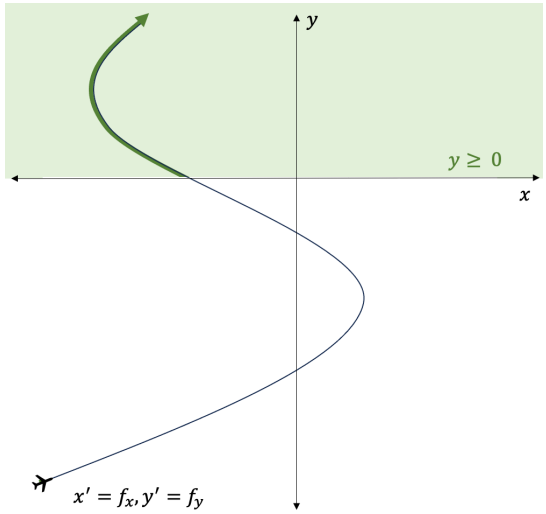

Figure 13. Visualization of aircraft dynamics described in Example 5.4. The velocities are given by the general type that ensures the velocity in the y direction is positive, i.e. $f_y \geq \epsilon > 0$. With velocities of this type, the aircraft eventually gets to the safe region $y \geq 0$.

```
%%Dynamics defined by
f:{ff:(ODEs_s?(hp(0))) | length(ff)=2 AND nth(ff,1)`1 = y AND
        EXISTS(eps:posreal):
          FORALL(env:Environment): (nth(ff,1)`2)(env) > eps}

prove | discharge-tccs | status-proofchain | show-prooflite
Always_eventually_safe: LEMMA
ALLRUNS_tr(SEQ(DIFF(f),TEST((Y>=0))),
          DLEVENTUALLY(DLGLOBALLY(State_Trace((Y>=0)))))
```

Figure 14. Specification of the set of differential equations F and the reachability property defined in (5.14).

which amounts to showing that the solution to the differential equation satisfies the property that it will reach the set

```
Always_eventually_safe :

  ├──────
{1}   (: :) |-
      (: ALLRUNS_tr(SEQ(DIFF(f, DLBOOL(TRUE)), TEST((Y >= cnst(0)))),
                    DLEVENTUALLY(DLGLOBALLY(State_Trace
                                            ((Y >= cnst(0)))))) :)

>> (rewrite "norm_DLEG")

Found matching substitution:
P: BoolExpr gets (Y >= cnst(0)),
Rewriting using norm_DLEG, matching in *

Always_eventually_safe :

  ├──────
{1}   (: :) |-
      (: ALLRUNS_tr(SEQ(DIFF(f, DLBOOL(TRUE)), TEST((Y >= cnst(0)))),
                    normDLEG((Y >= cnst(0)), (Y >= cnst(0)))) :)

>> |
```

Figure 15. The interactive theorem prover environment with one of the dTL^2 normalization rewriting rules called as a lemma in the proof for (5.14).

$y \geq 0$ and remain there for all time after. Figure 16 shows this property in the interactive theorem prover.

```
Always_eventually_safe.1 :

  ├──────
{1}   (: :) |-
      (: DLAND(ALLRUNS_st(DIFF(f, DLBOOL(TRUE)), DLBOOL(TRUE)),
               DLOR(SOMERUNS_st(DIFF(f, DLBOOL(TRUE)), DLBOOL(FALSE)),
                    SOMERUNS_st(DIFF(f, DLBOOL(TRUE)),
                                ALLRUNS_st(DIFF(f, DLBOOL(TRUE)),
                                           (Y >= cnst(0)))))) :)

>> |
```

Figure 16. The interactive theorem prover where the only non-trivial statement is to show Equation (5.15).

This type of stability property is useful in the field of safety-critical air traffic systems, where it is important to verify that a vehicle has the ability to reach and stay within some geofenced area. ■

6 Related Work

Extensive work has been done in the field of formal verification of hybrid systems and different languages and techniques have been proposed to specify and analyze these systems. Among these techniques, dL([37, 40, 46, 48]) has been introduced as a rigorous formalism to reason about the combination of discrete and continuous dynamics. This formalism has been successfully applied to the verification of different applications such as robotics [7, 24], aircraft collision avoidance [8, 20], reinforcement learning [16], and railways [22, 23].

In [6], the authors formally verified the soundness of dL in Coq and Isabelle. The work in [6] focuses on a full formal verification of soundness of dL, with the goal of a formally verified prover kernel for KeYmaera X. This work resulted in proof checkers in Coq and Isabelle for dL proofs. In contrast to [6], the goal of Plaidypvs is a verified embedding of dL and dTL^2 in PVS that allows for the specification and verification of hybrid programs within the PVS theorem prover.

Over the years, several extensions of dL have been proposed. For instance, differential game logic has been introduced in [44, 47] to model adversarial cyber-physical systems. There are also extensions for distributed hybrid systems (quantified differential dynamic logic, [42]), stochastic hybrid systems (stochastic differential dynamic logic, [43]), and differential algebraic programs (differential-algebraic dynamic logic, [41]),

A temporal extension of dL, called differential temporal dynamic logic, was defined in [38]. This logic enhanced dL with the temporal operators globally and eventually. In [21], the differential temporal dynamic logic has been further extended with nested temporal operators. Recently, a signal temporal logic extension to dL has been defined in [2]. In this logic, the scope of temporal operators can be bound to a certain time interval. To the best of the authors' knowledge, these temporal extensions of dL have not been formally formalized in a theorem prover.

There has been significant work done in the PVS theorem prover [1, 52], Event-B [13], and Isabelle/HOL [15, 17–19, 50, 53, 55, 56] on the verification of hybrid systems outside the dL framework. In addition, PVS has been successfully applied to several applications including formal verification of aircraft avoidance systems [31], path planning algorithms [4, 9], unmanned aircraft systems [29], position reporting algorithms of aircraft [14, 27], sensor uncertainty mitigation [35], floating-point round-off error analysis [26, 54], genetic algorithms [49], nonlinear control systems [5], and structured natural language requirements [10]. PVS has decades of developments that give it many unique features and automation capabilities. In addition to advanced real number reasoning [11, 25, 30, 32–34], previous work has connected PVS to the automated theorem prover for real-valued functions MetiTarski [3, 12]. The development of Plaidypvs adds to this extensive collection of PVS theories providing a new framework for the formal verification of hybrid programs within PVS.

7 Conclusion

This paper presents a PVS formalization of dTL^2, a temporal extension of dL. This formalization has been implemented as part of the Plaidypvs tool and is available in the NASA PVS library. The addition of temporal operators to dL enhances the logic with the possibility of reasoning about intermediate states of different computational paths of a hybrid program. The combination of dL run quantifiers and temporal operators enables an interesting fragment of the differential counterpart of CTL*. To the best of the authors' knowledge, this is the first formalization of a temporal extension of dL and the first functional implementation of dTL^2.

The goal of Plaidypvs and its temporal extension is to add hybrid systems reasoning to the ecosystem of PVS. Since Plaidypvs is embedded in PVS, it can leverage the numerous theories already formally verified and available. In fact, the embedding enables user-defined functions and meta-reasoning about hybrid programs. These are unique features of Plaidypvs with respect to existing implementations of dL.

In the future, the authors plan to implement proof strategies to automatically verify dTL^2 formulas within PVS. Different extensions of dTL^2 can be explored. For instance, the addition of the Until and Release operators, or the support for liveness properties of the form $\Box(\phi \rightarrow \Diamond\psi)$. Another interesting direction is the extension of the language to include discrete infinite traces and operators for parallel and synchronous compositions of hybrid programs. These features would enable the modeling of a wider range of reactive hybrid systems. Another topic of future interest includes extending the semantics of Plaidypvs to allow for reasoning on differential systems that do not have unique solutions.

References

[1] Erika Ábrahám, Ulrich Hannemann, and Martin Steffen. 2001. Verification of hybrid systems: Formalization and proof rules in PVS. In *Proceedings Seventh IEEE International Conference on Engineering of Complex Computer Systems*. IEEE, 48–57. https://doi.org/10.1109/ICECCS.2001.930163

[2] Hammad Ahmad and Jean-Baptiste Jeannin. 2021. A program logic to verify signal temporal logic specifications of hybrid systems. In *Proceedings 24th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '21)*. ACM, 10:1–10:11. https://doi.org/10.1145/3447928.3456648

[3] Behzad Akbarpour and Lawrence Charles Paulson. 2010. MetiTarski: An automatic theorem prover for real-valued special functions. *Journal of Automated Reasoning* 44, 3 (2010), 175–205. https://doi.org/10.1007/s10817-009-9149-2

[4] Swee Balachandran, Anthony Narkawicz, César A. Muñoz, and María Consiglio. 2017. A path planning algorithm to enable well-clear low altitude UAS operation beyond visual line of sight. In *Twelfth USA/Europe Air Traffic Management Research and Development Seminar (ATM 2017)*.

[5] Cinzia Bernardeschi and Andrea Domenici. 2016. Verifying safety properties of a nonlinear control by interactive theorem proving with

the Prototype Verification System. *Inform. Process. Lett.* 116, 6 (2016), 409–415. https://doi.org/10.1016/j.ipl.2016.02.001

[6] Brandon Bohrer, Vincent Rahli, Ivana Vukotic, Marcus Völp, and André Platzer. 2017. Formally verified differential dynamic logic. In *Proceedings 6th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2017)*. 208–221. https://doi.org/10.1145/3018610.3018616

[7] Brandon Bohrer, Yong Kiam Tan, Stefan Mitsch, Andrew Sogokon, and André Platzer. 2019. A Formal Safety Net for Waypoint Following in Ground Robots. *IEEE Robotics and Automation Letters* 4, 3 (2019), 2910–2917. https://doi.org/10.1109/LRA.2019.2923099

[8] Rachel Cleaveland, Stefan Mitsch, and André Platzer. 2023. Formally Verified Next-Generation Airborne Collision Avoidance Games in ACAS X. *ACM Trans. Embed. Comput. Syst.* 22, 1 (2023), 1–30. https://doi.org/10.1145/3544970

[9] Brendon K. Colbert, J Tanner Slagel, Luis G. Crespo, Swee Balachandran, and César A. Muñoz. 2020. PolySafe: A Formally Verified Algorithm for Conflict Detection on a Polynomial Airspace. *IFAC-PapersOnLine* 53, 2 (2020), 15615–15620.

[10] Esther Conrad, Laura Titolo, Dimitra Giannakopoulou, Thomas Pressburger, and Aaron Dutle. 2022. A compositional proof framework for FRETish requirements. In *Proceedings 11th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2022)*. 68–81. https://doi.org/10.1145/3497775.3503685

[11] Marc Daumas, David Lester, and César A. Muñoz. 2008. Verified real number calculations: A library for interval arithmetic. *IEEE Trans. Comput.* 58, 2 (2008), 226–237. https://doi.org/10.1109/tc.2008.213

[12] William Denman and César Munoz. 2014. Automated Real Proving in PVS via MetiTarski. In *Proceedings 19th International Symposium on Formal Methods (FM 2014) (Lecture Notes in Computer Science, Vol. 8442)*. Springer, 194–199. https://doi.org/10.1007/978-3-319-06410-9_14

[13] Guillaume Dupont. 2021. *Correct-by-construction design of hybrid systems based on refinement and proof*. Ph. D. Dissertation. https://oatao.univ-toulouse.fr/28190/1/Dupont_Guillaume.pdf

[14] Aaron Dutle, Mariano M. Moscato, Laura Titolo, and César A. Muñoz. 2017. A Formal Analysis of the Compact Position Reporting Algorithm. *9th Working Conference on Verified Software: Theories, Tools, and Experiments, VSTTE 2017, Revised Selected Papers* 10712 (2017), 19–34.

[15] Simon Foster, Jonathan Julián Huerta y Munive, Mario Gleirscher, and Georg Struth. 2021. Hybrid systems verification with Isabelle/HOL: Simpler syntax, better models, faster proofs. In *Proceedings Formal Methods: 24th International Symposium (FM 2021) (Lecture Notes in Computer Science, Vol. 13047)*. Springer, 367–386. https://doi.org/10.1007/978-3-030-90870-6_20

[16] Nathan Fulton and André Platzer. 2018. Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. In *Proceedings Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*. AAAI Press, 6485–6492. https://doi.org/10.1609/aaai.v32i1.12107

[17] Jonathan Julian Huerta y Munive. 2020. *Algebraic verification of hybrid systems in Isabelle/HOL*. Ph. D. Dissertation. University of Sheffield. https://etheses.whiterose.ac.uk/28886/

[18] Jonathan Julián Huerta y Munive and Georg Struth. 2018. Verifying hybrid systems with modal Kleene algebra. In *Proceedings 17th International Conference on Relational and Algebraic Methods in Computer Science (RAMiCS 2018)*. Springer, 225–243. https://doi.org/10.1007/978-3-030-02149-8_14

[19] Jonathan Julián Huerta y Munive and Georg Struth. 2022. Predicate Transformer Semantics for Hybrid Systems. *Journal of Automated Reasoning* 66, 1 (2022), 93–139. https://doi.org/10.1007/s10817-021-09607-x

[20] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Aurora C. Schmidt, Ryan W. Gardner, Stefan Mitsch, and André Platzer. 2017. A Formally Verified Hybrid System for Safe Advisories in the Next-generation Airborne Collision Avoidance System. *International Journal on Software Tools for Technology Transfer* 19, 6 (2017), 717–741. https://doi.org/10.1007/s10009-016-0434-1

[21] Jean-Baptiste Jeannin and André Platzer. 2014. dTL2: Differential Temporal Dynamic Logic with Nested Temporalities for Hybrid Systems. In *Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR 2014) (Lecture Notes in Computer Science, Vol. 8562)*. Springer, 292–306. https://doi.org/10.1007/978-3-319-08587-6_22

[22] Aditi Kabra, Stefan Mitsch, and André Platzer. 2022. Verified Train Controllers for the Federal Railroad Administration Train Kinematics Model: Balancing Competing Brake and Track Forces. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4409–4420. https://doi.org/10.1109/TCAD.2022.3197690

[23] Stefan Mitsch, Marco Gario, Christof J. Budnik, Michael Golm, and André Platzer. 2017. Formal Verification of Train Control with Air Pressure Brakes. In *Proceedings 2nd International Conference Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification (RSSRail 2017) (Lecture Notes in Computer Science, Vol. 10598)*. Springer, 173–191. https://doi.org/10.1007/978-3-319-68499-4_12

[24] Stefan Mitsch, Khalil Ghorbal, David Vogelbacher, and André Platzer. 2017. Formal Verification of Obstacle Avoidance and Navigation of Ground Robots. *International Journal of Robotics Research* 36, 12 (2017), 1312–1340. https://doi.org/10.1177/0278364917733549

[25] Mariano M. Moscato, César A. Muñoz, and Andrew P. Smith. 2015. Affine Arithmetic and Applications to Real-Number Proving. In *Proceedings 6th International on Conference Interactive Theorem Proving (ITP 2015) (Lecture Notes in Computer Science, Vol. 9236)*. Springer, 294–309. https://doi.org/10.1007/978-3-319-22102-1_20

[26] Mariano M. Moscato, Laura Titolo, Aaron Dutle, and César A. Muñoz. 2017. Automatic Estimation of Verified Floating-Point Round-Off Errors via Static Analysis. In *Proceedings of the 36th International Conference on Computer Safety, Reliability, and Security, SAFECOMP 2017*. Springer.

[27] Mariano M. Moscato, Laura Titolo, Marco A. Feliú, and César A. Muñoz. 2019. Provably Correct Floating-Point Implementation of a Point-in-Polygon Algorithm. In *Proceedings of the 23nd International Symposium on Formal Methods (FM 2019) (Lecture Notes in Computer Science, Vol. 11800)*. Springer, 21–37. https://doi.org/10.1007/978-3-030-30942-8_3

[28] Andreas Müller, Stefan Mitsch, Werner Retschitzegger, Wieland Schwinger, and André Platzer. 2017. Change and Delay Contracts for Hybrid System Component Verification. In *Proceedings of the 20th International Conference on Fundamental Approaches to Software Engineering (FASE 2017) (Lecture Notes in Computer Science, Vol. 10202)*. Springer, 134–151. https://doi.org/10.1007/978-3-662-54494-5_8

[29] César A. Muñoz, Aaron Dutle, Anthony Narkawicz, and Jason Upchurch. 2016. Unmanned aircraft systems in the national airspace system: a formal methods perspective. *ACM SIGLOG News* 3, 3 (2016), 67–76.

[30] César A. Muñoz and Anthony Narkawicz. 2013. Formalization of Bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning* 51, 2 (2013), 151–196. https://doi.org/10.1007/s10817-012-9256-3

[31] César A. Muñoz, Anthony Narkawicz, George Hagen, Jason Upchurch, Aaron Dutle, María Consiglio, and James Chamberlain. 2015. DAIDALUS: detect and avoid alerting logic for unmanned systems. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*. IEEE, 5A1–1.

[32] Anthony Narkawicz and César A. Muñoz. 2013. A formally verified generic branching algorithm for global optimization. In *Working Conference on Verified Software: Theories, Tools, and Experiments*. Springer, 326–343. https://doi.org/10.1007/978-3-642-54108-7_17

[33] Anthony Narkawicz and César A. Muñoz. 2014. A Formally Verified Generic Branching Algorithm for Global Optimization. In *Proceedings*

5th International Conference on Verified Software: Theories, Tools, and Experiments (VSTTE 2013) (Lecture Notes in Computer Science, Vol. 8164). Springer, 326–343. https://doi.org/10.1007/978-3-642-54108-7_17

[34] Anthony Narkawicz, César A. Muñoz, and Aaron Dutle. 2015. Formally-verified decision procedures for univariate polynomial computation based on Sturm's and Tarski's theorems. *Journal of Automated Reasoning* 54, 4 (2015), 285–326. https://doi.org/10.1007/s10817-015-9320-x

[35] Anthony Narkawicz, César A. Muñoz, and Aaron Dutle. 2018. Sensor uncertainty mitigation and dynamic well clear volumes in DAIDALUS. In *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*. IEEE, 1–8.

[36] Sam Owre, John M. Rushby, and Natarajan Shankar. 1992. PVS: A Prototype Verification System. In *Proceedings of the 11th International Conference on Automated Deduction (CADE 1992)*. Springer, 748–752.

[37] André Platzer. 2007. Differential Dynamic Logic for Verifying Parametric Hybrid Systems. In *Proceedings 16th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2007) (Lecture Notes in Computer Science, Vol. 4548)*. Springer, 216–232. https://doi.org/10.1007/978-3-540-73099-6_17

[38] André Platzer. 2007. A Temporal Dynamic Logic for Verifying Hybrid System Invariants. In *Proceedings 5th International Symposium on Logical Foundations of Computer Science (LFCS'07) (Lecture Notes in Computer Science, Vol. 4514)*. Springer, 457–471. https://doi.org/10.1007/978-3-540-72734-7_32

[39] André Platzer. 2008. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning* 41, 2 (2008), 143–189. https://doi.org/10.1007/s10817-008-9103-8

[40] André Platzer. 2008. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning* 41, 2 (2008), 143–189. https://doi.org/10.1007/s10817-008-9103-8

[41] André Platzer. 2010. Differential-algebraic Dynamic Logic for Differential-algebraic Programs. *J. Log. Comput.* 20, 1 (2010), 309–352. https://doi.org/10.1093/logcom/exn070 Advance Access published on November 18, 2008.

[42] André Platzer. 2010. Quantified Differential Dynamic Logic for Distributed Hybrid Systems. In *Proceedings 24th International Workshop on Computer Science Logic (CSL 2010) (Lecture Notes in Computer Science, Vol. 6247)*, Anuj Dawar and Helmut Veith (Eds.). Springer, 469–483. https://doi.org/10.1007/978-3-642-15205-4_36

[43] André Platzer. 2011. Stochastic Differential Dynamic Logic for Stochastic Hybrid Programs. In *Proceedings International Conference on Automated Deduction (CADE-23) (Lecture Notes in Computer Science, Vol. 6803)*. Springer, 446–460. https://doi.org/10.1007/978-3-642-22438-6_34

[44] André Platzer. 2015. Differential Game Logic. *ACM Trans. Comput. Log.* 17, 1 (2015), 1:1–1:51. https://doi.org/10.1145/2817824

[45] André Platzer. 2017. A complete uniform substitution calculus for differential dynamic logic. *Journal of Automated Reasoning* 59, 2 (2017), 219–265. https://doi.org/10.1007/s10817-016-9385-1

[46] André Platzer. 2017. A complete uniform substitution calculus for differential dynamic logic. *Journal of Automated Reasoning* 59, 2 (2017), 219–265. https://doi.org/10.1007/s10817-016-9385-1

[47] André Platzer. 2017. Differential Hybrid Games. *ACM Trans. Comput. Log.* 18, 3 (2017), 19:1–19:44. https://doi.org/10.1145/3091123

[48] André Platzer. 2018. *Logical Foundations of Cyber-Physical Systems*. Springer. https://doi.org/10.1007/978-3-319-63588-0

[49] Muhammad Saqib Nawaz, Muhammad Ikram Ullah Lali, and Muhammad A. Pasha. 2013. Formal verification of crossover operator in genetic algorithms using prototype verification system (PVS). In *2013 IEEE 9th International Conference on Emerging Technologies (ICET)*. IEEE, 1–6.

[50] Huanhuan Sheng, Alexander Bentkamp, and Bohua Zhan. 2023. HHLPy: Practical Verification of Hybrid Systems Using Hoare Logic. In *Proceedings 25th International Symposium on Formal Methods (FM 2023) (Lecture Notes in Computer Science, Vol. 14000)*. Springer, 160–178. https://doi.org/10.1007/978-3-031-27481-7_11

[51] J. Tanner Slagel, Mariano M. Moscato, Lauren White, César Muñoz, Swee Balachandran, and Aaron Dutle. 2023. Embedding Differential Dynamic Logic in PVS. In *Proceedings of the 18th International Conference on Logical and Semantic Frameworks, with Applications (LSFA 2023)*.

[52] J. Tanner Slagel, Lauren White, and Aaron Dutle. 2021. Formal verification of semi-algebraic sets and real analytic functions. In *Proceedings 10th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2021)*. 278–290. https://doi.org/10.1145/3437992.3439933

[53] Georg Struth. 2021. Hybrid Systems Verification with Isabelle/HOL: Simpler Syntax, Better Models, Faster Proofs. In *Formal Methods: 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings*, Vol. 13047. Springer Nature, 367. https://doi.org/10.1007/978-3-030-90870-6_20

[54] Laura Titolo, Marco A. Feliú, Mariano M. Moscato, and César A. Muñoz. 2018. An Abstract Interpretation Framework for the Round-Off Error Analysis of Floating-Point Programs. In *Proceedings of the 19th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Springer, 516–537.

[55] Shuling Wang, Naijun Zhan, and Liang Zou. 2015. An Improved HHL Prover: An Interactive Theorem Prover for Hybrid Systems. In *Proceedings 17th International Conference on Formal Engineering Methods (ICFEM 2015) (Lecture Notes in Computer Science, Vol. 9407)*. Springer, 382–399. https://doi.org/10.1007/978-3-319-25423-4_25

[56] Liang Zou, Naijun Zhan, Shuling Wang, and Martin Fränzle. 2015. Formal verification of Simulink/Stateflow diagrams. In *Proceedings 13th International Symposium on Automated Technology for Verification and Analysis (ATVA 2015) (Lecture Notes in Computer Science, Vol. 9364)*. Springer, 464–481. https://doi.org/10.1007/978-3-319-24953-7_33

Received 2023-09-19; accepted 2023-11-25