Towards Physical Hybrid Systems[⋆]

Katherine Cordwell[1] and André Platzer[1,2]

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, USA
{kcordwel,aplatzer}@cs.cmu.edu
[2] Fakultät für Informatik, Technische Universität München

Abstract. Some hybrid systems models are unsafe for mathematically correct but physically unrealistic reasons. For example, mathematical models can classify a system as being unsafe on a set that is too small to have physical importance. In particular, differences in measure zero sets in models of cyber-physical systems (CPS) have significant mathematical impact on the mathematical safety of these models even though differences on measure zero sets have no tangible physical effect in a real system. We develop the concept of "physical hybrid systems" (PHS) to help reunite mathematical models with physical reality. We modify a hybrid systems logic (differential temporal dynamic logic) by adding a first-class operator to elide distinctions on measure zero sets of time within CPS models. This approach facilitates modeling since it admits the verification of a wider class of models, including some physically realistic models that would otherwise be classified as mathematically unsafe. We also develop a proof calculus to help with the verification of PHS.

Keywords: hybrid systems · almost everywhere · differential temporal dynamic logic · proof calculus

1 Introduction

Hybrid systems [24,1], which have interacting discrete and continuous dynamics, provide all the necessary mathematical precision to describe and verify the behavior of safety-critical *cyber-physical systems* (CPS), such as self-driving cars, surgical robots, and drones. Ironically, however, hybrid systems provide so much mathematical precision that they can distinguish models that exhibit no physically measurable difference. More specifically, since mathematical models are minutely precise, models can classify systems as being unsafe on minutely small

[⋆] This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1252522. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. This research was also sponsored by the AFOSR under grant number FA9550-16-1-0288 and by the Alexander von Humboldt Foundation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

P. Fontaine (Ed.): CADE 2019, LNAI 11716, pp. 216–232, 2019.
DOI: 10.1007/978-3-030-29436-6_13

sets—even when these sets have no physical significance. For example, a mathematical model could classify a system as being mathematically unsafe at a single instant in time—but why should the safety of a model give more weight to such glitches than even the very notion of solutions of differential equations, which is unaffected [30] by changes on sets of measure zero in time? Practically speaking, a physical system is only unsafe at a single instant of time if it is also already unsafe at a significantly larger set of times. In the worst case, such degenerate counterexamples could detract attention from real unsafeties in a model.

That is why this paper calls for a shift in perspective toward *physical hybrid systems* (PHS) that are more attuned to the limitations and necessities of physics than pure mathematical models. PHS are hybrid systems that behave safely "almost everywhere" (in a measure theoretic sense) and thus, physically speaking, are safe systems. While different flavors of attaining PHS are possible and should be pursued, we propose arguably the tamest one, which merely disregards differences in safety on sets of time of measure zero. As our ultimate hope is that models of PHS can be (correctly) formally verified without introducing any burden on the user, we introduce the ability to rigorously ignore sets of time of measure zero into logic. A major difficulty is that there is a delicate trade-off between the physical practicality of a definition (what real-world behavior it captures) and the logical practicality of a definition (what logical reasoning principles it supports). Our notion of safety almost everywhere in time not only enjoys a direct link with well-established mathematical principles of differential equations, but also satisfies key logical properties, such as compositionality.

We modify *differential temporal dynamic logic* (dTL) [15,25] to capture the notion of safety *time almost everywhere* (tae) along the execution trace of a hybrid system. dTL extends the hybrid systems logic *differential dynamic logic* (dL) with the ability to analyze system behavior over time. We call our new logic *physical differential temporal dynamic logic* (PdTL) to reflect its purpose. While PdTL is closely related to dTL in style and development, the formalization of safety tae is entirely new, and thus requires new reasoning. Guiding the development of PdTL is the following motivating example: Consider a train and a safety condition $v{<}100$ on the velocity of the train. Physically speaking, it is fine to allow $v{=}100$ for a split-second, because this has no measurable impact. PdTL is designed to classify the situation where the train continuously accelerates until $v{=}100$ and immediately brakes whenever it reaches $v{=}100$ as tae safe.

2 Related Work

Since all systems inherently suffer from imprecision, several approaches develop *robust hybrid systems*, which are stable up to small perturbations—for example, in the contexts of decidability [2,10,11], runtime monitoring [8,9], and controls [18,19,22]. If systems are robust, they provide a fair amount of automation [11,17]. Both our approach and robustness hinge on building in an awareness of physics to hybrid systems verification. However, robustness is fundamentally different from our deductive verification approach. The analysis of robust systems often relies on a reachability analysis (see, e.g., [21]) which loses much of the logical precision present in deductive verification, e.g., decidability of differential equation invariants [27]. Further, by building on dL, which is a general purpose hybrid systems logic, we are able to handle a wide class of models, whereas tools like dReach [17] tend to be slightly more limited in scope. In particular, our deductive approach for PHS admits an induction principle—making it possible to verify safety properties of controllers that run in loops for any amount of time—whereas robustness approaches are presently limited to bounded model checking. We advocate for robustness in that models can, and should be, written with an awareness of imprecision. However, we recognize that modeling is difficult, and even well-intentioned models can suffer from nonphysical glitches—hence PHS.

Non-classical solutions of ODEs [4,6], especially Filippov and Carathéodory solutions, align well with the PHS intuition as they often inherently ignore sets of measure zero. *Filippov solutions* consider vector fields equivalent up to differences on sets of measure zero. *Carathéodory solutions* satisfy a differential equation everywhere except on a set of measure zero. Hybrid systems models do not usually make use of non-classical solutions, since admitting them would require a relaxed notion of safety. Non-classical solutions are sometimes used in the context of controls: Goebel et al. [13] generalized the notion of a solution to a hybrid system by using non-classical solutions of ODEs by Filippov and Krasovskii, with a view towards obtaining robustness properties, and a later work [12] allows solutions that fit a system of ODEs almost everywhere in a time interval. The temporal approach of PdTL naturally admits Carathéodory solutions.

Eliding sets of measure zero can be computationally significant—notably, in quantifier elimination, which arises in the last step of hybrid systems proofs. Despite having no physical meaning, measure zero sets have a significant impact on the efficiency of real arithmetic, and thus on the overall hybrid systems proofs. The enabling factor behind efficient arithmetic [20] is to ignore sets of measure zero and thus remove the need to compute with irrational algebraic numbers. A potential computational benefit is an encouraging motivation for PHS.

3 Syntax of PdTL

In pursuit of enabling the statement of physical safety properties of hybrid systems, we develop PdTL, which builds on concepts from dTL [15,25] to introduce an “almost everywhere in time” operator, $\Box_{\text{tae}}$, which makes it possible to disregard minor glitches violating safety conditions on sets of time of measure zero. When possible, we keep our notation consistent with that of dTL [25], so that the syntax of PdTL is very similar to the syntax of dTL—the key difference being that we eschew dTL trace formulas $\Box\phi$ and $\Diamond\phi$ in favor of $\Box_{\text{tae}}\phi$.

The new PdTL formula $[\alpha]\Box_{\text{tae}}\phi$ expresses that along each run of the hybrid system α, the formula ϕ is true at almost every time (“tae” stands for “time almost everywhere”). This formula remains true even in cases where ϕ is false at only a measure zero set of points in time along α. Because a hybrid system may exhibit different behaviors, the particular measure zero set of points in time at which ϕ is false is allowed to depend on the particular run of α.

Fix a set Σ containing real-valued variables, function symbols, and predicate symbols. In particular, Σ contains the symbols needed for first-order logic of real arithmetic (FOL). We use Σ_{var} to denote the set of real-valued variables in Σ, and let $\text{Trm}(\Sigma)$ denote the set of (polynomial) terms over Σ (as in FOL).

We now define the syntax of *hybrid programs* (which model hybrid systems) and formulas capable of expressing physical properties of hybrid programs. Hybrid programs [25,26] are allowed to assign values to variables (with the := operator), test the truth of formulas (with the ? operator), evolve along systems of differential equations, and branch nondeterministically (with the $\cup$ operator). Hybrid programs are also sequentially composable with the ; operator, and can be run in loops with the * operator.

Definition 1. *Hybrid programs are given by a grammar, where α and β are hybrid programs, $e \in Trm(\Sigma)$, x is a variable, and P and R are FOL formulas:*

$$\alpha, \beta \ ::= \ x := e \mid ?P \mid x' = f(x) \& R \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$$

As in CTL* [7] and dTL, we split *PdTL formulas* into state formulas that are true or false in a *state* (i.e., at a snapshot in time) and trace formulas that are true or false along a fixed *trace* that keeps track of the behavior of a system over time.

Definition 2. *The state formulas are given by the following grammar, where $p \in \Sigma$ is a predicate symbol of arity $n{\geq}0$, $e_1, \ldots, e_n \in Trm(\Sigma)$, ϕ and ψ are state formulas, α is a hybrid program, κ is a trace formula, and x is a variable:*

$$\phi, \psi \ ::= \ p(e_1, \ldots, e_n) \mid \neg\phi \mid \phi \wedge \psi \mid \forall x\, \phi \mid [\alpha]\kappa \mid \langle\alpha\rangle\kappa$$

Trace formulas are given by the following grammar, where ϕ is a state formula:

$$\kappa \ ::= \ \phi \mid \Box_{tae}\phi$$

We will also allow the use of the standard logical operators $\vee$, $\rightarrow$, and $\leftrightarrow$, which are defined in terms of $\neg$ and $\wedge$ as usual in classical logic.

Our motivating example can be modeled in PdTL as follows:

$$a{=}0 \wedge v{=}0 \rightarrow [\big(((?(v{<}100); a := 1) \cup (?(v{=}100); a := -1));$$
$$\{x'{=}v, v'{=}a \ \& \ 0{\leq}v{\leq}100\}\big)^*]\Box_{\text{tae}}\, v{<}100$$

This claims that if the initial velocity and acceleration are both 0, then along any run of the system, $v{<}100$ holds at almost all times. The train accelerates if $v{<}100$ and brakes if $v{=}100$; it moves according to the system of ODEs $x'{=}v, v'{=}a$. The evolution domain constraint $v{\leq}100$ indicates an event-triggered controller [26].

A natural question is why we choose to build in reasoning about $\Box_{\text{tae}}$ by developing PdTL instead of having the user edit the model by, for example, making the postcondition $v{\leq}100$ instead of $v{<}100$. Indeed, in this particular case that would make the program safe at every moment in time. However, in other examples, editing the postcondition in a similar way may be unwise. For example, although $[x := 0; y := 0; \{x' = 0, y' = 1\}]\Box_{\text{tae}}(y{>}0 \rightarrow x{>}1 \vee x{<}1)$ is valid, if we relax the inequalities, $[x := 0; y := 0; \{x' = 0, y' = 1\}]\Box_{\text{tae}}(y{\geq}0 \rightarrow x{\geq}1 \vee x{\leq}1)$ is not. As reasoning about hybrid systems is so subtle, the most user-friendly approach to eliding sets of measure zero is to specifically introduce the rigorous ability to ignore sets of measure zero into the logic.

4 Semantics of PdTL

We now report a trace semantics for hybrid programs, based on which we give meaning to the informal concept of formulas being true almost everywhere in time, first along an individual trace and then along all traces of a hybrid program.

4.1 Semantics of State Formulas

State formulas are evaluated at states, which capture the behavior of the hybrid program at an instant in time. Each state contains the values of all relevant variables at a given instant. We formalize this in the following definition.

Definition 3. *A state is a map $\omega : \Sigma_{var} \rightarrow \mathbb{R}$. We distinguish a separate state Λ to indicate the failure of a system run. The set of all states is $Sta(\Sigma_{var})$.*

We now give the semantics of state formulas. The $val(\omega, \phi)$ operator determines whether state formula ϕ is true or false in state ω. The valuations of the state formulas $[\alpha]\kappa$ and $\langle\alpha\rangle\kappa$ depend on the semantics of traces σ (especially the notion of first σ), which is explained in Def. 5, and on the semantics of the trace formula κ (i.e., $val(\omega, \kappa)$) which is given later, in Def. 12, and may be undefined.

Definition 4 ([25]). *The valuation of state formulas with respect to state ω is defined inductively:*

1. *$val(\omega, p(\theta_1, \ldots, \theta_n))$ is $p^\ell(val(\omega, \theta_1), \ldots, val(\omega, \theta_n))$ where p^ℓ is the relation associated with p under the semantics of real arithmetic*
2. *$val(\omega, \neg\phi)$ is true iff $val(\omega, \phi)$ is false*
3. *$val(\omega, \phi \wedge \psi)$ is true iff $val(\omega, \phi)$ is true and $val(\omega, \psi)$ is true*
4. *$val(\omega, \forall x\, \phi)$ is true iff $val(\omega_x^d, \phi)$ is true for all $d \in \mathbb{R}$, where ω_x^d is the state that is identical to ω, except x has the value d.*
5. *$val(\omega, [\alpha]\kappa)$ is true iff for every trace σ of α that starts in first $\sigma = \omega$, if $val(\sigma, \kappa)$ is defined, then $val(\sigma, \kappa)$ is true*
6. *$val(\omega, \langle\alpha\rangle\kappa)$ is true iff there is some trace σ of α where first $\sigma = \omega$ and $val(\sigma, \kappa)$ is true*

We write $\omega \models \phi$ when $val(\omega, \phi)$ is true. We write $\omega \not\models \phi$ when $val(\omega, \phi)$ is false.

4.2 Traces of Hybrid Programs

Trace formulas are evaluated with respect to an execution trace of a hybrid program. Intuitively, a trace of a hybrid program views its behavior over time as a sequence of functions, where each function corresponds to a particular discrete or continuous portion of the dynamics. Most hybrid systems are associated with multiple traces (to reflect the variety of behaviors that a given program can exhibit). Each function within a trace maps from a time interval to states of the hybrid program. Continuous portions of traces are functions from an uncountable time interval, and thus are associated to uncountably many states, whereas discrete portions involve just a single state. Significantly, traces are allowed to end in an abort state Λ, which indicates an unsuccessful run of a program. Aborts are incurred when tests fail and when evolution domain constraints are not initially satisfied. No program can run past Λ. We review the formal definition below.

Definition 5 ([25]). *A trace σ of a hybrid program α is a sequence of functions $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_n)$ where $\sigma_i : [0, r_i] \to Sta(\Sigma_{var})$. We will denote the length of the interval associated to σ_i by $|\sigma_i|$ (so that if σ_i maps from $[0, r_i]$ to states of α, $|\sigma_i| = r_i$). A position of σ is a tuple (i, ζ) where $i \in \mathbb{N}$, $\zeta \in [0, r_i]$. Each position (i, ζ) is associated with the corresponding state $\sigma_i(\zeta)$. A trace $(\sigma_0, \sigma_1, \ldots, \sigma_n)$ is said to terminate if it does not end in the abort state, i.e. if $\sigma_n(|\sigma_n|) \neq \Lambda$, and write last $\sigma \equiv \sigma_n(|\sigma_n|)$ in that case. We write first $\sigma \equiv \sigma_0(0)$ for the first state.*

We now modify the trace semantics [25] using Carathéodory solutions for ODEs [30] using notation as in [26, Definition 2.6].

Definition 6. *The state ν is* reachable in the extended sense *from initial state ω by $x_1' = \theta_1, \ldots, x_n' = \theta_n$ & R iff there is a function $\varphi : [0, r] \to Sta(\Sigma_{var})$ s.t.:*

1. *Initial and final states match: $\varphi(0) = \omega, \varphi(r) = \nu$.*
2. *φ is absolutely continuous.*
3. *φ respects the differential equations almost everywhere: For each variable x_i, $\varphi(z)(x_i)$ is continuous in z on $[0, r]$ and if $r>0$, $\varphi(z)(x_i)$ has a time-derivative of value $\varphi(z)(\theta_i)$ at all $z \in [0, r] \setminus \mathcal{U}$, for some set $\mathcal{U} \subset [0, r]$ that has Lebesgue measure zero.*
4. *The value of other variables $y \notin \{x_1, \ldots, x_n\}$ remains constant throughout the continuous evolution, that is $\varphi(z)(y) = \omega(y)$ for all times $z \in [0, r]$;*
5. *φ respects the evolution domain at all times: $\varphi(z) \models R$ for all $z \in [0, r]$.*

If such a φ exists, we say that $\varphi \models x_1' = \theta_1 \wedge \cdots \wedge x_n' = \theta_n$ & R almost everywhere.

This change highlights how the PHS intuition aligns with the intuition behind Carathéodory solutions. However, we have left the syntax of hybrid programs unchanged, and any system of differential equations in this syntax has a unique classical solution by Picard-Lindelöf [26]. We believe that in order to determine a suitable generalization of the syntax of hybrid programs to allow systems of ODEs with Carathéodory solutions, one should first develop strategies for reasoning about ODEs in PdTL that are beyond the scope of this work (for example, a notion of differential invariants—see [26,27]).

Note that in condition 5 of Def. 6, the solution is required to stay within the evolution domain constraint at *all* times. This is because evolution domain constraints, when used correctly, are nonnegotiable—for example, a correct use of evolution domain constraints is to reflect some underlying property of physics, like that the speed of a decelerating system is always nonnegative.

We define the trace semantics [25], with the above change for ODEs. Like evolution domain constraints, we treat tests as nonnegotiable, so as not to interfere with a user's ability to write precise models. We do not intend to secretly change the meaning of models, but rather to identify physically correct models.

Definition 7. *The trace semantics, $\tau(\alpha)$, of a hybrid program α is the set of all its possible hybrid traces and is defined inductively as follows (where $e \in Trm(\Sigma)$), $x' = f(x)$ is a vectorial ODE, R and P are FOL formulas, β is a hybrid program, and where for a state ω, $\hat{\omega}$ is the function from $[0,0] \to Sta(\Sigma_{var})$ with $\hat{\omega}(0) = \omega$):*

1. $\tau(x := e) = \{(\hat{\omega}, \hat{\nu}) : \nu = \omega_x^{val(\omega,e)}$ *for* $\omega \in Sta(\Sigma_{var})\}$
2. $\tau(x' = f(x)\ \&\ R) = \{(\varphi) : \varphi \models x' = f(x)\ \&\ R$ *almost everywhere* $\} \cup \{(\hat{\omega}, \hat{\Lambda}) : \omega \not\models R\}$
3. $\tau(\alpha \cup \beta) = \tau(\alpha) \cup \tau(\beta)$
4. $\tau(?P) = \{(\hat{\omega}) : val(\omega, P) = true\} \cup \{(\hat{\omega}, \hat{\Lambda}) : val(\omega, P) = false\}$
5. $\tau(\alpha; \beta) = \{\sigma \circ \zeta : \sigma \in \tau(\alpha), \zeta \in \tau(\beta)$ *when* $\sigma \circ \zeta$ *is defined*$\}$*; the composition of* $\sigma = (\sigma_0, \sigma_1, \sigma_2, \dots)$ *and* $\zeta = (\zeta_0, \zeta_1, \zeta_2, \dots)$ *is*

$$\sigma \circ \zeta := \begin{cases} (\sigma_0, \dots, \sigma_n, \zeta_0, \zeta_1, \dots) & \textit{if } \sigma \textit{ terminates at } \sigma_n \textit{ and last } \sigma\textit{=first } \zeta \\ \sigma & \textit{if } \sigma \textit{ does not terminate} \\ \textit{not defined} & \textit{otherwise} \end{cases}$$

6. $\tau(\alpha^*) = \bigcup_{n \in \mathbb{N}} \tau(\alpha^n)$*, where* $\alpha^{n+1} := (\alpha^n; \alpha)$ *for* $n \geq 1$*, and* $\alpha^0 := ?(true)$

As an important remark, notice that if we have a trace $\sigma = (\sigma_0, \dots, \sigma_n)$ and $|\sigma_i| > 0$, then $\sigma_i \in \tau(x' = f(x)\ \&\ R)$ for some (vectorial) ODE $x' = f(x)$ and some evolution domain constraint R. In other words, only continuous portions of a trace have nonzero duration, and continuous portions are only introduced when our system is evolving subject to a system of differential equations.

We are almost ready to give the semantics of trace formulas, but first we need to take a slight detour to discuss what formulas make "physical sense".

4.3 Physical Formulas

One feature of our motivating example is that $v<100$ is not a physically meaningful postcondition: If v is allowed to get arbitrarily close to 100, $v=100$ should also be allowed, since there is no physically measurable difference between $v<100$ and $v \leq 100$. The postcondition $v \leq 100$ is, mathematically speaking, less restrictive than $v<100$, but also practically speaking, the same as $v<100$. Motivated by this intuition, we define "physical formulas".

Physical Formulas and $\overline{\phi}$ Geometrically, the set of states in which a state formula ϕ in n variables is true is a subset, $[\![\phi]\!] = \{(x_1, \dots, x_n) \in \mathbb{R}^n \mid \phi(x_1, \dots, x_n)\}$, of $\mathbb{R}^n$. We use this correspondence to define the physical version of ϕ.

Definition 8. *A state formula ϕ is called* physical *iff $[\![\phi]\!]$ is topologically closed. If ϕ is a formula in n variables $x_1, \dots, x_n$, then the physical version of ϕ is the closure, denoted by $\overline{\phi}$, and is, indeed, definable [3] by:*

$$\forall \epsilon{>}0\, \exists y_1, \dots, y_n\ \left(\phi(y_1, \dots, y_n) \wedge (x_1 - y_1)^2 + \dots + (x_n - y_n)^2 {<} \epsilon^2\right)$$

This satisfies $[\![\overline{\phi}]\!] = \overline{[\![\phi]\!]}$, where $\overline{[\![\phi]\!]}$ is the topological closure of $[\![\phi]\!]$. Quantifier elimination can compute a quantifier-free equivalent of $\overline{\phi}$ that is often preferable.

Associating a state formula to subset of $\mathbb{R}^n$ is useful for identifying which points are "almost included", which will be crucial knowledge for our temporal approach. In the train example, $v{=}100$ is "almost included" in the postcondition $v{<}100$. These points that are "almost included" in formula ϕ are exactly the limit points of the set associated to ϕ, so $\overline{[\![\phi]\!]}$ adds in all of these limit points. We will make use of the following properties of the physical version of ϕ. Both are proved in a companion report [5].

Proposition 9. *For any state formula ϕ, $\phi \to \overline{\phi}$ is valid (i.e., true in all states).*

Proposition 10. *The following proof rule is sound for state formulas ϕ, ψ (i.e., the validity of all premises implies the validity of the conclusion):*

$$\frac{\phi \to \psi}{\overline{\phi} \to \overline{\psi}}\ \mathit{TopCl}$$

4.4 Semantics of Trace Formulas

Intuitively, we want to say that, for a trace σ, $\sigma \models \Box_{\text{t.a.e}}\phi$ when there is only a "small" set of positions (i, ζ) where $\sigma_i(\zeta) \not\models \phi$ and where the discrete portions of σ satisfy a reasonable constraint. To formalize the notion of a "small" set of positions, we map positions of σ to $\mathbb{R}$, since $\mathbb{R}$ admits the Lebesgue measure.

Definition 11. *Given a trace $\sigma = (\sigma_0, \dots, \sigma_n)$ of a hybrid program α with $|\sigma_i| = r_i$, map each position (i, ζ) of σ to $\zeta + i + \sum_{k=0}^{i-1} |\sigma_k|$, so that the positions $(0, 0), \dots, (0, r_0)$ cover the interval $[0, r_0]$, the positions $(1, 0), \dots, (1, r_1)$ cover the interval $[r_0 + 1, r_0 + r_1 + 1]$, and so on. In this way we have an injection, which we call f, from positions of a trace σ to (a subset of) $\mathbb{R}$.*

Fig. 1 illustrates the mapping f, which is obtained by first concatenating the positions between each discrete step (i.e. the positions for each continuous function σ_i) and then projecting these concatenations onto a single time axis so that the images of the states for σ_i and the states for σ_j are disjoint when $i \neq j$. To ensure disjointness, our mapping places an open interval of unit length between the images of the states of σ_i and the states of σ_{i+1} (for all i). The unit length was chosen arbitrarily—any nonzero length would work—but it is important that the positions (i, r_i) and $(i+1, 0)$ have different projections, since discrete changes can cause their states $\sigma_i(r_i)$ and $\sigma_{i+1}(0)$ to be different.

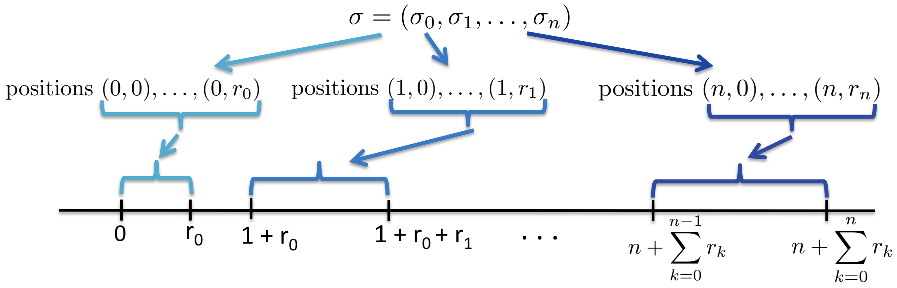
$\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_n)$
positions $(0,0), \ldots, (0, r_0)$
positions $(1,0), \ldots, (1, r_1)$
positions $(n,0), \ldots, (n, r_n)$
0
r_0
$1 + r_0$
$1 + r_0 + r_1$
. . .
$n + \sum_{k=0}^{n-1} r_k$
$n + \sum_{k=0}^{n} r_k$

Fig. 1. Injectively mapping positions of a trace to times in $\mathbb{R}$

Definition 12. *The valuation of a trace formula κ with respect to trace σ is defined as:*

1. *$val(\sigma, \phi) = val(last\ \sigma, \phi)$ if σ terminates. If σ does not terminate, then $val(\sigma, \phi)$ is undefined. We write $\sigma \models \phi$ when $val(\sigma, \phi)$ is true. We write $\sigma \not\models \phi$ when $val(\sigma, \phi)$ is false.*
2. *Let $\mathcal{U}$ be the set of positions (i, ζ) where corresponding states $\sigma_i(\zeta)$ satisfy $\sigma_i(\zeta) \not\models \phi$ and $\sigma_i(\zeta) \neq \Lambda$. We say that $val(\sigma, \square_{tae}\phi)$ is true iff the following two conditions are satisfied:*
   (a) *(Discrete condition) For all i, if $|\sigma_i| = 0$ and $\sigma_i(0) \neq \Lambda$, then $\sigma_i(0) \models \overline{\phi}$.*
   (b) *(Continuous condition) $f(\mathcal{U}) \subseteq \mathbb{R}$ has measure zero with respect to the Lebesgue measure, where f is the mapping defined in Def. 11 for σ; i.e., for all $\epsilon>0$ there exist intervals $I_p = [a_p, b_p]$ so that $f(\mathcal{U}) \subseteq \bigcup_{p=0}^{\infty} I_p$ and $\sum_{p=0}^{\infty} |b_p - a_p| < \epsilon$ (see [16, Section 1-1013], or [28]).*

   We write $\sigma \models \square_{tae}\phi$ when $val(\sigma, \square_{tae}\phi)$ is true. We write $\sigma \not\models \square_{tae}\phi$ when $val(\sigma, \square_{tae}\phi)$ is false.

A short primer on the measure theory we need is in a companion report [5].

With this mapping, the condition that $f(\mathcal{U})$ has measure zero with respect to the Lebesgue measure enforces the t.a.e. constraint for the continuous portions of our program. The image of the states for σ_i is an interval of length r_i. In order for the t.a.e constraint to be satisfied, σ_i cannot go wrong except at a "small set" of states, where we use our mapping to formalize the notion of a "small set" in terms of measure zero.

Next, the condition that $\sigma_i(0) \models \overline{\phi}$ whenever $|\sigma_i| = 0$ constrains the discrete portions of a program. This constraint will be important for induction; it also ensures that discrete programs behave reasonably within our logic. For example, let α be the fully discrete program $x := 5; (x := x + 1)^*$, and take a trace σ of α. The states of σ map to the points $5, 6, 7, \ldots, n$, and *without the discrete condition*, we would be able to show that $[x := 5; (x := x+1)^*]\square_{\text{tae}} x<5$ is valid, even though x is never less than 5 along the trace.

To understand why the discrete condition specifies $\sigma_i(0) \models \overline{\phi}$ when $|\sigma_i| = 0$ instead of $\sigma_i(0) \models \phi$ when $|\sigma_i| = 0$, recall the motivating train control example.

We want to allow the velocity of the train to evolve from a safe state where $v{<}100$ to an unsafe state where $v = 100$, as long as the train then immediately brakes (sets its acceleration to a negative value). If we specified that $\sigma_i(0) \models \phi$, the train would *not* be allowed to accelerate from $v{<}100$ to $v = 100$ and then brake, because at the discrete braking point, the train would be in a state that is unsafe mathematically (though still safe physically).

Given a hybrid program α, postcondition ϕ, and a state ω, we are interested in determining whether $\omega \models [\alpha]\Box_{\text{tae}}\phi$ holds, because when such a formula is true for a given hybrid program, that indicates that no matter how that particular hybrid program runs, it will be "safe almost everywhere". Following Def. 4, this is true iff for each trace $\sigma \in \tau(\alpha)$ with first $\sigma = \omega$, $\sigma \models \Box_{\text{tae}}\phi$.

5 Discussion

Now that we have developed the semantics, we step back to consider how PdTL makes progress towards PHS, and why PdTL is a good way to introduce PHS.

Impact on Modeling. PdTL allows the verification of several classes of physically realistic models that are mathematically not quite safe. For example, in Section 7 we will explain how PdTL allows the verification of the train control model. This simple train example is representative of a greater class of examples—tiny glitches are common in event-triggered controllers, since the event that is being detected is often an almost unsafe event that requires the controller to immediately change behavior.

Other examples do not involve time-triggered controllers, but rather suffer from tiny glitches at handover points between the discrete and continuous dynamics within a hybrid program. Consider the safety postcondition $x^2 + y^2{<}1$ and the hybrid program $x := 0; y := 1; \{x' = -x, y' = -y\}$. The only glitch is that the program starts ever so slightly outside the safe set. Since it immediately moves into the safe set, all runs of this hybrid program are safe tae. PdTL is designed to classify $[x := 0; y := 1; \{x' = -x, y' = -y\}]\Box_{\text{tae}}x^2 + y^2{<}1$ as valid.

In another class of examples, our approach handles tiny glitches within the continuous portion of the program. This can easily happen if the postcondition is missing some small regions. For example, consider two robots that are moving, one in front of the other. Since we do not want the robots to collide, it is unsafe for the second robot to accelerate while the first robot is braking. Say we model this with safety postcondition $\neg(a_1{\leq}0 \wedge a_2{\geq}0)$. This is a small modeling mistake, because we should allow the point where $a_1{=}0$ and $a_2{=}0$. Now, if our controller is $a_1 := -1; a_2 := -1; \{a_1' = 1, a_2' = 1\}$, any run of this hybrid program is tae safe, but not safe at all points in time (as some runs will contain the origin). Notably, the very similar controller $a_1 := -1; a_2 := -1; \{a_1' = 1, a_2' = 2\}$ is *not* tae safe. PdTL is designed to distinguish between these two controllers.

Why tae? The tae safety notion along the trace of a hybrid system is a natural approach with strong mathematical underpinnings (e.g., from the invariance of

Lebesgue integrals up to sets of measure zero, and from Carathéodory solutions), and with physical motivation from examples like those just discussed. Introducing tae is a good way to begin PHS, because it may be the closest possible PHS construct to the canonical notion of "safety everywhere". However, this closeness to safety everywhere does make tae more restrictive than some other possible PHS notions—for example, a notion of safety "space almost everywhere", or sae.

Consider a self-driving car moving in $\mathbb{R}^3$. The final states of a hybrid program α modeling the car correspond to positions in $\mathbb{R}^3$, so here we may wish to consider safety almost everywhere with respect to the Lebesgue measure *on the set of all possible final states of* α as follows: Given a hybrid program α and precondition ψ, let $\mathcal{F} = \{\text{last } \sigma \text{ s.t. } \sigma \in \tau(\alpha), \text{first } \sigma \models \psi\}$. Say that $\psi \vdash [\alpha]\Box_{\mu\mathrm{sae}}\phi$ if $\mu(\{\omega \in \mathcal{F} \mid \omega \not\models \phi\}) = 0$, where μ is the Lebesgue measure on $\mathbb{R}^3$. This modality has an intuitive geometric interpretation and is more permissive than tae.

However, $\Box_{\mu\mathrm{sae}}$ applies only when there is a natural measure μ on the set of all possible final states of α. Furthermore, $\Box_{\mu\mathrm{sae}}$ is not compositional. Given $P \vdash [\alpha]\Box_{\mu\mathrm{sae}}P$ and $P \vdash [\beta]\Box_{\mu\mathrm{sae}}P$, in order to conclude that $P \vdash [\alpha;\beta]\Box_{\mu\mathrm{sae}}P$, one needs to know that β is sae safe when starting in $[\![P]\!] \cup Q$, for any measure zero set Q reachable from α. This is not always true—for example, let P be $x^2+y^2<1$, α be $\{x' = 1, y' = 1 \;\&\; x^2 + y^2 \leq 1\}$ and β be $?(x^2 + y^2 = 1); \{x' = 1, y' = 1\}$. Further, given α, it is unclear how to syntactically classify such sets Q—as sae is more relaxed than tae, it also seems to be less well-behaved. Thus, although $\Box_{\mu\mathrm{sae}}$ has some advantages, it is not clear how to constrain it to achieve desirable logical properties like compositionality. Although we hope that future work will develop a notion of safety sae, the challenges therein are no small matter. In contrast, tae satisfies many nice logical properties, which we now turn our attention to.

6 Proof Calculus and Properties of PdTL

Before developing the proof calculus for PdTL, we discuss some key properties. First, PdTL is a *conservative* extension of dL, i.e. all valid formulas of dL are still valid in PdTL. The proof of this, discussed in a companion report [5], is essentially the same as the proof that dTL is a conservative extension of dL [25, Proposition 4.1]. This conservativity property is useful, since if we are able to reduce temporal PdTL formulas to dL formulas, we can use the extensive machinery built for dL to close proofs. Indeed, our proof calculus is designed to reduce temporal PdTL formulas into nontemporal formulas to rely on dL's capabilities for the latter.

Key dL axioms are proved sound for PdTL in a companion report [5], which is useful as sometimes a dL axiom is needed to reduce the goal in a proof, as we will see when we analyze the train example in Section 7.

Next, we state three properties of temporal PdTL formulas which underlie some of the soundness proofs for rules in the proof calculus. These properties hold by construction. All proofs are in the companion report [5].

Lemma 13. *If* $\sigma \models \Box_{tae}\phi$ *for a terminating* σ*, then last* $\sigma \models \overline{\phi}$*.*

Corollary 14. *The formula* $[\alpha]\Box_{tae}\phi \rightarrow [\alpha]\overline{\phi}$ *is valid.*

Lemma 15. *If* ξ *and* η *are traces of hybrid programs where* ξ *terminates and last* $\xi =$ *first* η*, then* $\xi \circ \eta \models \Box_{tae}\phi$ *iff both* $\xi \models \Box_{tae}\phi$ *and* $\eta \models \Box_{tae}\phi$*.*

6.1 Proof Calculus

The proof calculus for PdTL is shown in Fig. 2. Intuitively, all of the axioms are designed to successively decompose complicated formulas into structurally simpler formulas while successively reducing trace formulas into state formulas. The test axiom, assignment axiom, solution axiom, and solution with evolution domain constraint axiom ($[?]_{\text{tae}}$, $[:=]_{\text{tae}}$, $[']_{\text{tae}}$, and $['\&]_{\text{tae}}$) remove instances of $\Box_{\text{tae}}$. The nondeterministic choice axiom $[\cup]_{\text{tae}}$ reduces a choice between two hybrid programs to two separate programs. The induction axiom I_{tae} reduces a loop property involving a trace formula to a loop property involving a state formula; I_{tae} also allows us to derive two very useful proof rules.

The Gödel generalization rule (G_{tae}) proves that if formula ϕ is valid, then it is also true, tae, along the trace of any hybrid program. The modal modus ponens rule (K_{tae}) allows us to derive a monotonicity property. Our approach occasionally introduces extra premises; for example, the modal modus ponens rule (K_{tae}), has an extra goal $\overline{\phi} \rightarrow \overline{\psi}$ due to the discrete condition of Def. 12. Many of these extra premises will be easy to prove—if our models make use of physical formulas, which are closed, then these extra cases will prove immediately.

$$[?]_{\text{tae}} \quad [?P]\Box_{\text{tae}}\phi \leftrightarrow \overline{\phi}$$

$$[\cup]_{\text{tae}} \quad [\alpha \cup \beta]\Box_{\text{tae}}\phi \leftrightarrow [\alpha]\Box_{\text{tae}}\phi \wedge [\beta]\Box_{\text{tae}}\phi$$

$$[:=]_{\text{tae}} \quad [x := e]\Box_{\text{tae}}\phi \leftrightarrow \overline{\phi} \wedge [x := e]\overline{\phi}$$

$$[;]_{\text{tae}} \quad [\alpha;\beta]\Box_{\text{tae}}\phi \leftrightarrow ([\alpha]\Box_{\text{tae}}\phi \wedge [\alpha][\beta]\Box_{\text{tae}}\phi)$$

$$\text{I}_{\text{tae}} \quad [\alpha^*]\Box_{\text{tae}}\phi \leftrightarrow \big(\overline{\phi} \wedge [\alpha^*](\overline{\phi} \rightarrow [\alpha]\Box_{\text{tae}}\phi)\big)$$

$$[']_{\text{tae}} \quad [x' = f(x)]\Box_{\text{tae}}P \leftrightarrow \overline{P} \wedge \forall t{\geq}0 Q$$

$$['\&]_{\text{tae}} \quad [x' = f(x)\&R]\Box_{\text{tae}}P \leftrightarrow \overline{P} \wedge \forall t{>}0 \, ((\forall 0{\leq}s{\leq}t \; [x := y(s)]R) \rightarrow Q)$$

$$\frac{\phi}{[\alpha]\Box_{\text{tae}}\phi}\ \text{G}_{\text{tae}}$$

$$\frac{\overline{\phi} \rightarrow \overline{\psi} \qquad [\alpha]\Box_{\text{tae}}(\phi \rightarrow \psi)}{[\alpha]\Box_{\text{tae}}\phi \rightarrow [\alpha]\Box_{\text{tae}}\psi}\ K_{\text{tae}}$$

$$\frac{\phi \rightarrow \psi}{\overline{\phi} \rightarrow \overline{\psi}}\ \text{TopCl}$$

$$\text{CGG} \quad [\alpha]\Box_{\text{tae}}\phi \rightarrow [\alpha]\overline{\phi}$$

Fig. 2. Proof calculus for PdTL[3]

[3] Here, α and β are hybrid programs, ϕ and ψ are state formulas, P is a FOL formula, $y(t)$ is the unique global polynomial solution to the differential equation $x' = f(x)$, and the formula Q in $[']_{\text{tae}}$ and $['\&]_{\text{tae}}$ is the FOL formula constructed by Proposition 16 for $P(y(t))$. Although the "for almost all" quantifier is in general *not* definable in FOL [23], Proposition 16 justifies that "for almost all $t{\geq}0[x := y(t)]P$" is logically equivalent to "$\forall t{\geq}0 \; Q$".

Soundness proofs are in the report [5]. We discuss a few key high-level ideas.

Soundness of $[;]_{tae}$ *and* I_{tae} Sequential composition and induction are subtly challenging for PHS—since we are allowed to leave the safe set, handover points between hybrid programs are no longer guaranteed to be safe points. However, sequential composition and induction are crucial for the practicality of verification, which is predicated on having a good way of breaking down complicated formulas into simpler components. The soundness proof of $[;]_{\mathrm{tae}}$ exploits Lemma 15. The soundness proof of $\mathrm{I}_{\mathrm{tae}}$ is based on Lemma 13, which in turn relies on the discrete condition of Def. 12.

Differential Equations Reasoning about differential equations is one of the most challenging aspects of hybrid systems. In this work, we focus on relatively simple reasoning principles for differential equations, as justifying even simple principles is made much more challenging by introducing the notion of "safety almost everywhere". We leave the development of more complicated reasoning (for example, a notion of differential invariants for $\Box_{\mathrm{tae}}$) to future work.

For "sufficiently tame" systems of ODEs $x' = f(x)$, we might hope to replace $[x' = f(x)]\Box_{\mathrm{tae}}P$ with an equivalent expression without the $\Box_{\mathrm{tae}}$ modality. The cleanest case is when $x' = f(x)$ has a unique global polynomial solution, $y(t)$. Although we think of y as being a polynomial in t, y can involve any of the other parameters, call them $x_1, \ldots, x_n$, in f, from its dependency on initial values. We require that y is also polynomial in $x_1, \ldots, x_n$. This is the case handled by axiom $[']_{\mathrm{tae}}$: $[x' = f(x)]\Box_{\mathrm{tae}}P \leftrightarrow \overline{P} \ \wedge \ \forall t{\geq}0\, Q$, where Q is a FOL formula constructed so that "$\forall t{\geq}0\, Q$" expresses "for almost all $t{\geq}0\, [x := y(t)]P$". Axiom $['\&]_{\mathrm{tae}}$ generalizes $[']_{\mathrm{tae}}$ to ODEs with evolution domain constraints.

In particular, we get Q by applying Proposition 16 to $P(y(t))$, which is the formula obtained using the assignment axiom $[:=]$ of dL on $[x := y(t)]P$ (it is a FOL formula because $y(t)$ is polynomial and polynomials are closed under composition). Given any FOL formula P, Proposition 16 constructs a FOL formula Q so that "for almost all $t{\geq}0\, P$" is semantically equivalent to $\forall t{\geq}0\, Q$ (i.e., "for almost all $t{\geq}0\, P$" is true in a state ω iff $\forall t{\geq}0\, Q$ is true in ω).

Proposition 16. *Let P be a FOL formula. Using quantifier elimination [29], put it into one of the following normal forms: $e{=}0$, $e{\geq}0$, $e{<}0$, $P_1 \wedge P_2$, and $P_1 \vee P_2$, where e is a polynomial and P_1, P_2 are FOL formulas. Construct the FOL formula $Q = g(P)$ by structural induction on P as follows: $g(e{=}0)$ is $e{=}0$, $g(e{\geq}0)$ is $e{\geq}0$, $g(e{<}0)$ is $e{\leq}0 \wedge ((a_n{=}0 \wedge \cdots \wedge a_1{=}0) \rightarrow e{<}0)$, $g(P_1 \wedge P_2)$ is $g(P_1) \wedge g(P_2)$, and $g(P_1 \vee P_2)$ is $g(P_1) \vee g(P_2)$.*

Then, for any state ω, the following hold:

1. *Locally false: If $\omega_t^k \not\models Q$ for some $k{\geq}0$, then there is a nonempty interval $[k, \ell)$ so that for all $q \in [k, \ell)$, $\omega_t^q \not\models P$. Further, if $k{>}0$, then there is an interval (ℓ_1, ℓ_2) with $\ell_1{<}k{<}\ell_2$ so that for all $q \in (\ell_1, \ell_2)$, $\omega_t^q \not\models P$.*
2. *Finite difference: There are only finitely many values $k{\geq}0$ where $\omega_t^k \models Q \wedge \neg P$.*

The proof is by induction on the structure of P; details are in the report [5].

6.2 Derived Rules

We highlight some of the most useful derived rules for PdTL formulas in Fig. 3. Monotonicity properties are fundamental in logic. Our rule M_{tae} intuitively says that if $\phi \rightarrow \psi$ is valid, then if ϕ is true almost everywhere along every trace of a hybrid program, then ψ is also true almost everywhere along every trace of that hybrid program. The rule Ind_{tae} reduces proving a safety property of hybrid program α^* to proving a safety property of program α. When its premise proves, it effectively removes the need to reason about loops. The rule loop_{tae} provides us with a loop invariant rule. The rule Comp_{tae} reduces a property of $\alpha;\beta$ to individual properties of α and β. The derivations are given in the report [5].

$$\dfrac{\psi \rightarrow \phi}{[\alpha]\Box_{\text{tae}}\psi \rightarrow [\alpha]\Box_{\text{tae}}\phi}\ \mathrm{M}_{\text{tae}} \qquad \dfrac{\overline{\phi} \vdash [\alpha]\Box_{\text{tae}}\phi}{\overline{\phi} \vdash [\alpha^*]\Box_{\text{tae}}\phi}\ \mathrm{Ind}_{\text{tae}}$$

$$\dfrac{\Gamma \vdash \overline{\psi}, \Delta \qquad \overline{\psi} \vdash [\alpha]\Box_{\text{tae}}\psi \qquad \psi \vdash \phi}{\Gamma \vdash [\alpha^*]\Box_{\text{tae}}\phi, \Delta}\ \mathrm{loop}_{\text{tae}} \qquad \dfrac{\psi \rightarrow [\alpha]\Box_{\text{tae}}\phi \qquad \overline{\phi} \rightarrow [\beta]\Box_{\text{tae}}\phi}{\psi \rightarrow [\alpha;\beta]\Box_{\text{tae}}\phi}\ \mathrm{Comp}_{\text{tae}}$$

Fig. 3. Derived rules for PdTL

7 Proof of Motivating Example

We now apply our proof calculus to the model of the train example (Section 3). Full details are in the report [5]. Using structural rule $\rightarrow\!R$ and our induction proof rule loop_{tae} with invariant $v{<}100$, the proof reduces to showing $a{=}0 \wedge v{=}0 \vdash v{\leq}100$ (which holds by real arithmetic), $v{<}100 \vdash v{<}100$ (identically true), and

$$\begin{aligned} v{\leq}100 \vdash [\big((?(v{<}100); a := 1) \cup (?(v = 100); a := -1)\big);\\ \{x' = v, v' = a \;\&\; 0{\leq}v{\leq}100\}]\Box_{tae} v{<}100. \end{aligned}$$

Axiom $[;]_{\text{tae}}$ splits this into goals (1) and (2):

$$v{\leq}100 \vdash [(?(v{<}100); a := 1) \cup (?(v = 100); a := -1)]\Box_{tae} v{<}100 \tag{1}$$

$$\begin{aligned} v{\leq}100 \vdash [(?(v{<}100); a := 1) \cup (?(v = 100); a := -1)]\\ [\{x' = v, v' = a\&\; 0{\leq}v{\leq}100\}]\Box_{tae} v{<}100. \end{aligned} \tag{2}$$

(1) is straighforward. (2) is more complicated because it involves ODEs reasoning. The dL axioms $[\cup]$ and $\wedge R$ split the proof of (2) into (3) and (4):

$$v{\leq}100 \vdash [?(v{<}100); a := 1][\{x' = v, v' = a \;\&\; 0{\leq}v{\leq}100\}]\Box_{tae} v{<}100 \tag{3}$$

$$v{\leq}100 \vdash [?(v = 100); a := -1][\{x' = v, v' = a \;\&\; 0{\leq}v{\leq}100\}]\Box_{tae} v{<}100 \tag{4}$$

(3) and (4) require similar reasoning, so we focus on (3). The dL axioms [;], [:=], and [?] reduce (3) to

$$v\leq 100, v<100 \vdash [\{x' = v, v' = 1 \;\&\; 0\leq v\leq 100\}]\Box_{tae} v<100 \tag{5}$$

To prove (5), we need to use axiom $[' \&]_{\mathrm{tae}}$, which says:

$$[x' = f(x) \& R]\Box_{\mathrm{tae}} P \leftrightarrow \overline{P} \land \forall t{>}0\,((\forall 0{\leq}s{\leq}t\; [x := y(s)]R) \rightarrow Q)$$

For clarity, we use v_0 for the value of v in the initial state before it starts evolving along the ODEs, and similarly we use x_0 for the value of x in the initial state. Following Proposition 16, Q is $(1 = 0 \rightarrow t + v_0{<}100) \land t + v_0{\leq}100$. Since applying $[' \&]_{tae}$ reduces our goal to a dL formula, and since PdTL is a conservative extension of dL, we can use the contextual equivalence rules of dL to replace Q with the logically equivalent formula $t + v_0{\leq}100$, obtaining:

$$\begin{aligned} &v_0{\leq}100, v_0{<}100 \vdash v_0{\leq}100 \land \forall t{>}0 \\ &\big((\forall 0{\leq}s{\leq}t\; [x := .5s^2 + v_0 s + x_0][v := s + v_0]0{\leq}v{\leq}100\big) \rightarrow t + v_0{\leq}100\big) \end{aligned} \tag{6}$$

After using the dL axiom [:=], the proof closes by real arithmetic.

8 Conclusions and Future Work

We introduce PHS to help narrow the gap between mathematical models and physical reality. To enable logic to begin to distinguish between true unsafeties of systems and physically unrealistic unsafeties, we develop the notion of safety tae along the execution trace of a system. Our new logic, PdTL, contains the logical operator $\Box_{\mathrm{tae}}$, which elides sets of time that have measure zero.

A cornerstone of our approach is its logical practicality—in order to support verification, we develop a proof calculus for PdTL. We demonstrate the capability of the proof calculus by applying it to a motivating example. We think it is an interesting and challenging problem for future work to develop new ways of thinking about PHS, such as the notion of space almost everywhere discussed in Section 5, while maintaining this logical practicality.

Future work could continue to develop PdTL. It would be especially interesting to develop further differential equations reasoning, including an appropriate generalization of the syntax of hybrid programs to admit Carathéodory solutions.

9 Acknowledgements

We very much appreciate Yong Kiam Tan and Brandon Bohrer for many useful discussions and for feedback on the paper. Thank you also to the anonymous CADE'19 reviewers for their thorough feedback.

References

1. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman et al. [14], pp. 209–229
2. Asarin, E., Bouajjani, A.: Perturbed turing machines and hybrid systems. In: 16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings. pp. 269–278. IEEE Computer Society (2001). doi:10.1109/LICS.2001.932503
3. Bochnak, J., Coste, M., Roy, M.F.: Real Algebraic Geometry. Springer (1998). doi:10.1007/978-3-662-03718-8
4. Ceragioli, F.: Some remarks on stabilization by means of discontinuous feedbacks. Systems & Control Letters **45**(4), 271–281 (2002). doi:10.1016/S0167-6911(01)00185-2
5. Cordwell, K., Platzer, A.: Towards physical hybrid systems. CoRR **abs/1905.09520** (2019), http://arxiv.org/abs/1905.09520
6. Cortes, J.: Discontinuous dynamical systems. IEEE Control Systems **28**(3), 36–73 (2008). doi:10.1109/MCS.2008.919306
7. Dam, M.: CTL* and ECTL* as fragments of the modal μ-calculus. Theoretical Computer Science **126**(1), 77–96 (1994). doi:10.1016/0304-3975(94)90269-0
8. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. LNCS, vol. 8044, pp. 264–279. Springer (2013). doi:10.1007/978-3-642-39799-8_19
9. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theoretical Computer Science **410**(42), 4262–4291 (2009). doi:10.1016/j.tcs.2009.06.021
10. Fränzle, M.: Analysis of hybrid systems: An ounce of realism can save an infinity of states. In: Flum, J., Rodríguez-Artalejo, M. (eds.) Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings. LNCS, vol. 1683, pp. 126–140. Springer (1999). doi:10.1007/3-540-48168-0_10
11. Gao, S., Avigad, J., Clarke, E.M.: Delta-decidability over the reals. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012. pp. 305–314. IEEE Computer Society (2012). doi:10.1109/LICS.2012.41
12. Goebel, R., Sanfelice, R.G., Teel, A.R.: Hybrid dynamical systems. IEEE Control Systems Magazine **29**(2), 28–93 (April 2009). doi:10.1109/MCS.2008.931718
13. Goebel, R., Hespanha, J., Teel, A.R., Cai, C., Sanfelice, R.: Hybrid systems: Generalized solutions and robust stability. IFAC Proceedings Volumes **37**(13), 1 – 12 (2004). doi:10.1016/S1474-6670(17)31194-1, 6th IFAC Symposium on Nonlinear Control Systems 2004 (NOLCOS 2004), Stuttgart, Germany, 1-3 September, 2004
14. Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.): Hybrid Systems, vol. 736. Springer, Berlin (1993)
15. Jeannin, J., Platzer, A.: dTL^2: Differential temporal dynamic logic with nested temporalities for hybrid systems. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings. LNCS, vol. 8562, pp. 292–306. Springer (2014). doi:10.1007/978-3-319-08587-6_22

16. Jeffreys, H., Swirles, B.: Methods of Mathematical Physics. Cambridge University Press, Cambridge, 3rd edn. (1999). doi:10.1017/CBO9781139168489
17. Kong, S., Gao, S., Chen, W., Clarke, E.M.: dReach: δ-reachability analysis for hybrid systems. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings. LNCS, vol. 9035, pp. 200–205. Springer (2015). doi:10.1007/978-3-662-46681-0_15
18. Manthanwar, A., Sakizlis, V., Dua, V., Pistikopoulos, E.: Robust model-based predictive controller for hybrid system via parametric programming. In: Puigjaner, L., Espuña, A. (eds.) European Symposium on Computer-Aided Process Engineering-15, 38th European Symposium of the Working Party on Computer Aided Process Engineering, Computer Aided Chemical Engineering, vol. 20, pp. 1249 – 1254. Elsevier (2005). doi:10.1016/S1570-7946(05)80050-1
19. Mayhew, C.G., Sanfelice, R.G., Teel, A.R.: Robust source-seeking hybrid controllers for autonomous vehicles. In: 2007 American Control Conference. pp. 1185–1190 (July 2007). doi:10.1109/ACC.2007.4283016
20. McCallum, S.: Solving polynomial strict inequalities using cylindrical algebraic decomposition. The Computer Journal **36**(5), 432–438 (1993). doi:10.1093/comjnl/36.5.432
21. Moggi, E., Farjudian, A., Duracz, A., Taha, W.: Safe & robust reachability analysis of hybrid systems. Theoretical Computer Science **747**, 75–99 (2018). doi:10.1016/j.tcs.2018.06.020
22. Moor, T., Davoren, J.M.: Robust controller synthesis for hybrid systems using modal logic. In: Benedetto, M.D.D., Sangiovanni-Vincentelli, A.L. (eds.) Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings. LNCS, vol. 2034, pp. 433–446. Springer (2001). doi:10.1007/3-540-45351-2_35
23. Morgenstern, C.F.: The measure quantifier. J. Symb. Log. **44**(1), 103–108 (1979). doi:10.2307/2273708
24. Nerode, A., Kohn, W.: Models for hybrid systems: Automata, topologies, controllability, observability. In: Grossman et al. [14], pp. 317–356
25. Platzer, A.: Logical Analysis of Hybrid Systems - Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14509-4
26. Platzer, A.: Logical Foundations of Cyber-Physical Systems. Springer, Cham (2018). doi:10.1007/978-3-319-63588-0
27. 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). doi:10.1145/3209108.3209147
28. Royden, H.L., Fitzpatrick, P.M.: Real Analysis (Classic Version). Pearson, 4th edn. (2018)
29. Tarski, A.: A decision method for elementary algebra and geometry. In: Caviness, B.F., Johnson, J.R. (eds.) Quantifier Elimination and Cylindrical Algebraic Decomposition. pp. 24–84. Springer, Vienna (1998). doi:10.1007/978-3-7091-9459-1_3
30. Walter, W.: Ordinary Differential Equations, Graduate Texts in Mathematics, vol. 182. Springer, New York (1998). doi:10.1007/978-1-4612-0601-9