A Program Logic to Verify Signal Temporal Logic Specifications of Hybrid Systems

Hammad Ahmad
hammada@umich.edu
University of Michigan, Ann Arbor
Ann Arbor, Michigan

Jean-Baptiste Jeannin
jeannin@umich.edu
University of Michigan, Ann Arbor
Ann Arbor, Michigan

ABSTRACT

Signal temporal logic (STL) was introduced for monitoring temporal properties of continuous-time signals for continuous and hybrid systems. Differential dynamic logic (d$\mathcal{L}$) was introduced to reason about the end states of a hybrid program. Over the past decade, STL and its variants have significantly gained in popularity in the industry for monitoring purposes, while d$\mathcal{L}$ has gained in popularity for verification of hybrid systems. In this paper, we bridge the gap between the two different logics by introducing signal temporal dynamic logic (STd$\mathcal{L}$) – a dynamic logic that reasons about a subset of STL specifications over executions of hybrid systems. Our work demonstrates that STL can be used for deductive verification of hybrid systems. STd$\mathcal{L}$ significantly augments the expressiveness of d$\mathcal{L}$ by allowing reasoning about temporal properties in given time intervals. We provide a semantics and a proof calculus for STd$\mathcal{L}$, along with a proof of soundness and relative completeness.

CCS CONCEPTS

• **Theory of computation → Modal and temporal logics**; **Programming logic**; • **Computer systems organization → Embedded and cyber-physical systems**.

KEYWORDS

hybrid systems, dynamic logic, signal temporal logic, formal verification

ACM Reference Format:
Hammad Ahmad and Jean-Baptiste Jeannin. 2021. A Program Logic to Verify Signal Temporal Logic Specifications of Hybrid Systems. In *24th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '21), May 19–21, 2021, Nashville, TN, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447928.3456648

1 INTRODUCTION

Recent technological advances have made our transportation, manufacturing and communication facilities safer, cheaper, and more reliable. However, they have also increased our reliance on computer systems modeling and controlling objects of our physical world. Prime examples of such objects include cars on our roads, robots in our manufacturing plants, and satellites orbiting our planet. Such systems, referred to as *cyber-physical systems* (CPSs) [27], often fall under the category of *hybrid systems*: their programmable controllers typically exhibit discrete behavior, while the laws of physics that the systems are restricted by are continuous in nature.

The prevalence of hybrid systems around us, coupled with our increased reliance on these systems, has necessitated further exploration of reasoning about such systems. This process involves reasoning about the *states* of the hybrid system. A state is considered *safe* if it does not violate any safety property of the system, and considered *live* if the system can make some useful progress from that state. Verifying a system guarantees safety and liveness in the system. Signal temporal logic (STL) [15, 16] was introduced to monitor properties over continuous-time signals of continuous and hybrid systems in given time intervals, and has since been used primarily for monitoring purposes. Dynamic logic [10] was introduced as a formal system for reasoning about programs. Differential dynamic logic (d$\mathcal{L}$) [20] was built on top of dynamic logic to reason about the end states of a hybrid program, to ensure that the end state is a safe state. However, a hybrid system that is in a safe state at the end of a program's execution may not have been in a safe state throughout the program's execution: it is possible for a safety property to be violated during the execution of a program and be held at the termination of the program. Therefore, it is vital to verify that hybrid systems are safe during execution in addition to being safe upon termination. Differential temporal dynamic logic (dTL) [21] and differential temporal dynamic logic with nested temporalities (dTL^2) [12, 13] use both dynamic logic – to reason about all possible executions of a program – and a fragment of linear temporal logic (LTL) – to reason about intermediate states of each execution – to tackle this challenge.

While dTL and dTL^2 are able to reason about intermediate states of a hybrid system during program execution, the logics are still unable to reason about intermediate states of a system in given time intervals. This is a major limitation of the logics, since such reasoning abilities can be crucial in ensuring safety of a hybrid system (e.g., ensuring that a self-driving car applies its brakes within x seconds of spotting a stop sign, as opposed to ensuring that the car applies its brakes *eventually* after spotting a stop sign). STL is able to prove properties about a system in given time intervals, but the logic reasons about only *one* execution of a system, not all possible executions. Therefore, using STL alone to reason about safety in hybrid systems is not sufficient.

In this paper, we present signal temporal dynamic logic (STd$\mathcal{L}$), a logic that integrates a fragment of STL with differential dynamic logic (d$\mathcal{L}$) to reason both about the intermediate states of a hybrid system *in given time intervals*, and about the final states of the system. This reasoning is enabled by our use of STL, which natively supports formulas of the form $\Box_{[a,b]}\phi$ (i.e., for all times between $t + a$ and $t + b$, where t is the current time, the property ϕ is true) and $\Diamond_{[a,b]}\phi$ (i.e., there exists a time between $t + a$ and $t + b$ such that the property ϕ is true), but has historically been used mainly for monitoring purposes. We show that STL can be used for full deductive reasoning of hybrid systems.

The main contributions of this work are as follows:

– We introduce STd$\mathcal{L}$ – a logic that reasons about STL formulas for the first time in the context of d$\mathcal{L}$, bringing together results from two different communities with little overlap into a common framework.
– We introduce a notion of timing hybrid programs to bridge the gap between d$\mathcal{L}$ and STL for verification purposes.
– We provide a semantics for STd$\mathcal{L}$ and sound proof calculus for the logic, along with a proof of soundness and relative completeness.

The rest of the paper is organized as follows. Section 2 motivates STd$\mathcal{L}$ by introducing a running example of a use-case from the industry highlighting the power of the logic. Section 3 introduces the syntax and semantics of STd$\mathcal{L}$. Section 4 motivates the concept of normalization of trace formulas in STd$\mathcal{L}$ and presents the proof system of STd$\mathcal{L}$. Section 5 discusses future directions for STd$\mathcal{L}$. Section 6 outlines some related work, and Section 7 parts with concluding thoughts.

2 MOTIVATION AND RUNNING EXAMPLE

Throughout this paper, we use a simplified example of a use-case for STd$\mathcal{L}$ inspired by industry. As we note in Section 1, a major limitation of the program logics preceding STd$\mathcal{L}$ is their inability to reason about temporal properties in specified time intervals. Such reasoning abilities can be crucial in verifying a hybrid system. While STL is able to handle formulas specifying properties in given time intervals, the logic is only able to prove properties about *one* execution of a hybrid system, and not all possible executions; we need to be able to reason about every execution of a hybrid system to be able to claim with certainty correctness of the system. As such, none of differential dynamic logic d$\mathcal{L}$, differential temporal dynamic logics dTL or dTL^2, or signal temporal logic STL – or other variants of these logics – alone is sufficient to reason about safety and liveness in hybrid systems.

To see why, let us examine a simplified version of traction assist from the automobile industry. Consider a car with some accelerator input and braking force cruising on the road. The car has a signal that streams a binary value corresponding to whether or not the car's sensors detect that the car is skidding or losing traction, and a boolean flag corresponding to whether or not the vehicle's traction control is engaged. For simplicity, assume the accelerator input can have a positive or negative value corresponding to acceleration and deceleration respectively, or a value of zero corresponding to no acceleration. Assume further that the braking force is a non-negative integer. Let the wheel rotation of the car's wheels evolve according to some differential equation. The car has a safety property requiring that in the event that the car is skidding, a vehicle traction assist program executes to help gain traction again and slow down the wheel spin to stop the skidding, following which the car can accelerate again. According to the safety property, after running the traction assist program, the car's traction control should turn on and within 1 to 5 seconds, the car's wheel rotation should fall to under some threshold value (to help regain control).

As we introduce key concepts in the following sections, we also present the differential equation, the hybrid program, and the safety property for the car in STd$\mathcal{L}$. We present a proof sketch of the safety property using the STd$\mathcal{L}$ calculus. We note that safety properties of this class (i.e., containing temporal references for specified time intervals) are expressible directly in STd$\mathcal{L}$ (but, to the best of our knowledge and in the context of a program logic, not in any other logic preceding STd$\mathcal{L}$), and remain crucial in verifying correctness of hybrid systems.

3 SIGNAL TEMPORAL DYNAMIC LOGIC

This section formally defines the syntax and semantics of hybrid programs and state and trace formulas in STd$\mathcal{L}$. We take special care to ensure that STd$\mathcal{L}$ is a conservative extension of d$\mathcal{L}$, i.e. the non-temporal aspects of the state semantics for STd$\mathcal{L}$ are equivalent to the non-temporal transition semantics of d$\mathcal{L}$ (Definition 5 in [20]).

3.1 Hybrid Programs

We use *hybrid programs* to model hybrid systems in our work. A hybrid program α, β could be a discrete assignment ($x := \theta$), a test ($?\chi$), an ordinary differential equation ($x' = \theta \,\&\, \chi$), a non-deterministic choice ($\alpha \cup \beta$), a sequential composition ($\alpha; \beta$), or a non-deterministic finite repetition (α^*). As in d$\mathcal{L}$, a term θ can be any polynomial with a rational coefficient, and a condition χ can be any first-order formula of real arithmetic.

The syntax of hybrid programs can be summarized as:

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

For the semantics of hybrid programs in STd$\mathcal{L}$, the set of *states* Sta is the set of functions from variables to $\mathbb{R}$. A special state $\Lambda \notin \mathrm{Sta}$ denotes a failure state for the hybrid system. The *trace semantics* of a hybrid program α assign a set of traces $[\![\alpha]\!]$ to the program. For $v \in \mathrm{Sta} \cup \{\Lambda\}$, we express the function $\sigma : [0,0] \to \{v\}, 0 \mapsto v$ using $\hat{v}$, and $\hat{v}$ is defined only on the singleton interval [0,0].[1] A trace, then, is a non-empty, finite sequence $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_n)$ of subtraces σ_i. For $0 \le i < n$, the piece σ_i is a function $\sigma_i : [r_{i-1}, r_i] \to \mathrm{Sta}$, with the convention $r_{-1} = 0$, where $r_i - r_{i-1}$ is the duration of this step and $r_i \ge r_{i-1}$. Where $i = n$, σ_n can be defined as:

– $\sigma_n : [r_{n-1}, r_n] \to \mathrm{Sta}$, in which case we refer to σ as a *terminating* trace;
– $\sigma_n : [r_{n-1}, +\infty) \to \mathrm{Sta}$, in which case we refer to σ as an *infinite* trace;
– $\sigma_n : [r_{n-1}, r_{n-1}] \to \{\Lambda\}$ with $\sigma(r_{n-1}) = \Lambda$, for $n \ge 1$, in which case we refer σ as an *error* trace. $n \ge 1$ ensures that $(\hat{\Lambda})$ is not considered as a trace.

For a trace $\sigma = (\sigma_0, \ldots, \sigma_n)$, we define a *position* of σ as a pair (i, t) such that $0 \le i < n$ and t is in the domain of definition of σ_i. We write $\sigma_i(t)$ to refer to the state of σ at (i, t), i.e. $\sigma(i, t) = \sigma_i(t)$, and

[1]We often informally refer to a trace defined on a singleton interval $\{i\}$, e.g. $(\hat{v})$, as a trace that executes in zero time.

define the domain of σ as:

$$\mathsf{dom}(\sigma) = \bigcup_{i=0}^{n} \left(\bigcup_{t \in \mathsf{dom}(\sigma_i)} (i, t) \right)$$

We can now define the lengths of traces of hybrid programs.

Definition 3.1 (Length of traces of hybrid programs). The length of a trace $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_n) \in [\![\alpha]\!]$, denoted by $|\sigma| \in \mathbb{R}_+ \cup \{+\infty\}$, is defined inductively as follows:

- $|\sigma| = r_n$ if $\sigma_n : [r_{n-1}, r_n] \longrightarrow \mathsf{Sta}$;
- $|\sigma| = +\infty$ if $\sigma_n : [r_{n-1}, +\infty) \longrightarrow \mathsf{Sta}$;
- $|\sigma| = r_{n-1}$ if $\sigma_n : [r_{n-1}, r_{n-1}] \longrightarrow \{\Lambda\}$.

The set of all *traces* of a hybrid program is referred to as Tra , and we collectively refer to infinite traces and error traces as non-terminating traces. For a trace σ, we refer to the state $\sigma_0(0)$ as first σ, and we often say that "σ starts with v" if first $\sigma = v$. Likewise, for a finite trace σ, if σ terminates in a non-error state, we refer to the state $\sigma_n(r_n)$ as last σ; otherwise, we refer to the state Λ as last σ. Note that for any trace σ, first σ is always well-defined, but last σ may not be (since infinite traces have no last state). The value of term θ in state v is denoted by $val(v, \theta)$, and the valuation assigning variable x to $r \in \mathbb{R}$ while matching with v on all other variables is denoted by $v[x \mapsto r]$. If a state v satisfies some condition χ, we write $v \vDash \chi$; if v does not satisfy condition χ, we write $v \nvDash \chi$. Finally, given a trace σ and an $x \in \mathbb{R}$, we use the notation $\mathsf{dom}(\sigma) \oplus x$ to denote the domain of σ shifted by a value of $+x$. For example, if $\mathsf{dom}(\sigma) = [a, b]$, then $\mathsf{dom}(\sigma) \oplus x = [a + x, b + x]$.

Definition 3.2 (Trace semantics of hybrid programs). The trace semantics $[\![\alpha]\!]$ of a hybrid program α is defined as follows:

- $[\![x := \theta]\!] = \{(\hat{v}, \hat{w}) \mid w = v[x \mapsto val(v, \theta)]\}$;
- $[\![x' = \theta \ \& \ \chi]\!] = \{(\sigma) : \sigma$ is a state flow of order 1 [20] defined on $[0, r]$ or $[0, +\infty)$ solution of $x' = \theta$, and for all t in its definition domain, $\sigma(t) \vDash \chi\} \cup \{(\hat{v}, \hat{\Lambda}) \mid v \nvDash \chi\}$;
- $[\![?\chi]\!] = \{(\hat{v}) \mid v \vDash \chi\} \cup \{(\hat{v}, \hat{\Lambda}) \mid v \nvDash \chi\}$;
- $[\![\alpha \cup \beta]\!] = [\![\alpha]\!] \cup [\![\beta]\!]$;
- $[\![\alpha; \beta]\!] = \{\sigma \circ \rho \mid \sigma \in [\![\alpha]\!], \rho \in [\![\beta]\!]$ when $\sigma \circ \rho$ is defined$\}$, where the composition $\sigma \circ \rho$ of $\sigma = (\sigma_0, \ldots, \sigma_n)$ and $\rho = (\rho_0, \ldots, \rho_m)$ is
  - $\sigma \circ \rho = (\sigma_0, \ldots, \sigma_n, \bar{\rho}_0, \ldots, \bar{\rho}_m)$ if σ terminates and last σ = first ρ, where $\bar{\rho} = (\bar{\rho}_0, \ldots, \bar{\rho}_m)$ is a trace with $\mathsf{dom}(\bar{\rho}) = \mathsf{dom}(\rho) \oplus |\sigma|$ and for each $i \in \{0 \ldots m\}$, for each $t \in \mathsf{dom}(\rho_i)$, $\bar{\rho}_i(t) = \rho_i(t - |\sigma|)$,[2]
  - σ if σ does not terminate,
  - undefined otherwise;
- $[\![\alpha^*]\!] = \bigcup_{n \in \mathbb{N}} [\![\alpha^n]\!]$, where α^0 is defined as ?true, α^1 is defined as α, and α^{n+1} is defined as $\alpha^n; \alpha$ for $n \geq 1$.

These semantics for hybrid programs are adopted from dTL^2 [12]. As in dTL^2, an important property of the trace semantics of hybrid programs is that for any hybrid program α and state v, there always exists a trace $\sigma \in [\![\alpha]\!]$ such that first $\sigma = v$ (even if σ is an error trace). A key difference between the semantics of dTL^2 and our work is that for a trace $\sigma = (\sigma_0, \ldots, \sigma_n)$, while the former define the domain of each σ_i from 0 to r_i, we define the domain of each σ_i from r_{i-1} to r_i, to enable easier reasoning about temporal formulas in given time intervals. As such, our semantics for the composition $\sigma \circ \rho$ between traces σ and ρ requires trace ρ to be shifted in time by a value of $+|\sigma|$.

[2]Informally, $\bar{\rho}$ is merely the trace ρ shifted to the right by a value of $+|\sigma|$.

3.1.1 Running Example: Traction Assist in Cars. Having introduced the semantics of hybrid programs in STd$\mathcal{L}$, we now formally specify a simplified version of the differential equation cruise that varies the car's wheel rotation ρ. For ω the acceleration of the car, φ the braking force applied to each of the car's wheels, and some positive constants k and j, we have

$$\mathsf{cruise}(\omega, \varphi) ::= \omega \times k - \varphi \times j$$

Note that in practice, each of the car's wheels could have a different wheel rotation and braking force. For the sake of simplicity, and to avoid presenting four separate proofs for this example, we assume that each wheel has the same rotation and braking force.

A very simple version of the hybrid program traction_assist can then take the form

$$\mathsf{traction\_assist} ::= (?(\mathsf{no\_traction}); \tag{1}$$
$$\mathsf{traction\_control} := 1; \tag{2}$$
$$\omega := -1;\ \varphi := 10;\ \rho' = \mathsf{cruise}(\omega, \varphi)); \tag{3}$$

where the signal no_traction is a binary value of true or false corresponding to whether or not the car's sensors detect that the car is losing traction and the Boolean flag traction_control keeps track of whether or not the vehicle's traction control is engaged.[3]

Several properties of hybrid programs are present in the program traction_assist. (1) denotes a test to check whether the car has lost traction; (2) represents an assignment statement setting traction_control to on; and (3) shows an evolving ordinary differential equation that changes the wheel rotation of the car. The sequential composition operator joins the individual statements together to form a single hybrid program.

3.2 State and Trace Formulas

State and trace formulas are used to reason about hybrid programs. A state formula is used to express properties about a state, whereas a trace formula is used to express properties about a trace. The syntax of state and trace formulas in STd$\mathcal{L}$ can then be summarized as:

$$\phi, \psi ::= \theta_1 \geq \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \forall x.\phi \mid [\alpha]\pi$$
$$\pi ::= \phi \mid \neg\pi \mid \Box_{[a,b]}\phi$$
$$a, b ::= \theta \mid \max(\theta_1, \theta_2) \mid \min(\theta_1, \theta_2) \mid a + b \mid a - b$$

A state formula ϕ or ψ could express a comparison of two terms ($\theta_1 \geq \theta_2$), a negation of a state formula ($\neg\phi$), a conjunction of two state formulas ($\phi \wedge \psi$), a universally quantified ($\forall x.\phi$) state formula over a variable $x \in \mathbb{R}$, or a program necessity ($[\alpha]\pi$) indicating that all traces of program α starting from the current state satisfy π. For a disjunction of two state formulas ($\phi \vee \psi$), we define as an abbreviation $\phi \vee \psi \equiv \neg(\phi \wedge \psi)$; for an existentially quantified ($\exists x.\phi$) over a variable $x \in \mathbb{R}$, we define $\exists x.\phi \equiv \neg\forall x.\neg\phi$; and for a program possibility ($\langle\alpha\rangle\pi$) over a trace formula π indicating that there exists a trace of program α starting from the current state that satisfies π, we define $\langle\alpha\rangle\pi \equiv \neg[\alpha]\neg\pi$.

[3]The variable φ is set to an arbitrary non-negative integer for the purposes of this example.

A trace formula π can express a state formula (ϕ), a negation of a trace formula ($\neg\pi$), or a temporal necessity ($\Box_{[a,b]}\phi$) indicating that given the current time t, every trace starting in the current state satisfies ϕ from time $t+a$ and $t+b$. A temporal possibility ($\Diamond_{[a,b]}\phi$) indicating that every trace starting in the current state satisfies ϕ at some point between time $t+a$ and time $t+b$ is defined as the abbreviation $\Diamond_{[a,b]}\phi \equiv \neg\Box_{[a,b]}\neg\phi$. For time intervals of the form $[a,b]$, a and b are terms in the hybrid program evaluated in the first state of a trace (which is always well-defined, see Definition 3.5), or the min or max of two terms in the hybrid program. We allow for a and b to be terms in the hybrid program, and not mere constants, since we need to allow for a program variable to appear as the lower or upper bound of an interval $[a,b]$ (see Section 3.5, where the timing variable q is introduced to appear inside the temporal intervals of an STd$\mathcal{L}$ formula for interval shifting).

3.3 Length of Traces and Trace Formulas

Previous works supporting temporal operators within the context of d$\mathcal{L}$ did not need to reason about the length of a trace or a trace formula, due to their use of linear temporal logic operators that do not support reasoning about formulas in time intervals. However, since STd$\mathcal{L}$ involves verifying a trace over specified time intervals, we need to incorporate reasoning about lengths of traces and trace formulas to determine the satisfaction of formulas over traces of hybrid programs. More specifically, we require that for a hybrid program α, a trace $\sigma \in [\![\alpha]\!]$ be sufficiently long to determine the satisfaction of the program necessities and possibilities. This requirement is inspired by that of STL with respect to signal lengths [15, 16], and is similarly justified for STd$\mathcal{L}$ since it is intuitively nonsensical to verify the satisfaction of a trace formula of length φ against a trace of length $\varphi_0 < \varphi$.

Definition 3.3 (Minimum length of trace formulas). The necessary length associated with trace formula π, written as $\|\pi\|$, to determine the satisfaction of a program necessity or possibility is defined inductively as follows:

$$\|\phi\| = 0$$
$$\|\neg\pi\| = \|\pi\|$$
$$\|\Box_{[a,b]}\phi\| = b$$

3.4 Satisfaction of State and Trace Formulas

The satisfaction of state and trace formulas in STd$\mathcal{L}$ is defined as follows:

Definition 3.4 (Satisfaction of state formulas). For a state formula ϕ and state $v \in \mathrm{Sta}$, we say $v \vDash \phi$ if v satisfies ϕ. Satisfaction of state formulas with respect to state v is then defined inductively as follows:

- $v \vDash \theta_1 \geq \theta_2$ if and only if $val(v,\theta_1) \geq val(v,\theta_2)$;
- $v \vDash \neg\phi$ if and only if $v \nvDash \phi$;
- $v \vDash \phi \wedge \psi$ if and only if $v \vDash \phi$ and $v \vDash \psi$;
- $v \vDash \forall x.\phi$ if and only if $v[x \mapsto d] \vDash \phi$ for all $d \in \mathbb{R}$;
- For ϕ a state formula, $v \vDash [\alpha]\phi$ if and only if for every trace $\sigma \in [\![\alpha]\!]$ such that first $\sigma = v$, if σ terminates, then last $\sigma \vDash \phi$;
- For π a trace formula, $v \vDash [\alpha]\pi$ if and only if for every trace $\sigma \in [\![\alpha]\!]$ such that first $\sigma = v$, if $|\sigma| \geq val(\text{first}\,\sigma, \|\pi\|)$, then we also have that $\sigma \vDash \pi$; notably, there is no requirement on traces σ that do not have the minimum length, i.e., for which $|\sigma| < val(\text{first}\,\sigma, \|\pi\|)$.

We discuss alternative choices for semantics of program necessities and possibilities with respect to the length of a trace formula π, and justify the intuition behind our choice, in an extended technical report [1].

Definition 3.5 (Satisfaction of trace formulas). For a trace formula π and trace $\sigma = (\sigma_0, \ldots, \sigma_n) \in \mathrm{Tra}$, we say $(\sigma,(i,t)) \vDash \pi$ if σ satisfies π starting from subtrace σ_i at time t. We use $\sigma \vDash \pi$ to say that $(\sigma,(0,0)) \vDash \pi$. Satisfaction of trace formulas with respect to a trace σ is then defined inductively as follows:

- For ϕ a state formula, $(\sigma,(i,t)) \vDash \phi$ if and only if $|\sigma| \geq t$ and $\sigma_i(t) \vDash \phi$;
- $(\sigma,(i,t)) \vDash \neg\pi$ if and only if $(\sigma,(i,t)) \nvDash \pi$;
- $(\sigma,(i,t)) \vDash \Box_{[a,b]}\phi$ if and only if for every $t' \in [t+val(\text{first}\,\sigma,a), t+val(\text{first}\,\sigma,b)]$ and for every i such that $(i,t') \in \mathrm{dom}(\sigma)$, it follows that $(\sigma,(i,t')) \vDash \phi$.

Given a trace σ and an interval $[a,b]$ such that $val(\text{first}\,\sigma,b) < val(\text{first}\,\sigma,a)$, we define the interval to be an empty set. As such, formulas such as $\Box_{[a,b]}\phi$ and $\Diamond_{[a,b]}\phi$ are defined to be trivially true and trivially false respectively over this empty interval. This choice deviates from the norm set by STL: formulas like $\Box_{[a,b]}\phi$ and $\Diamond_{[a,b]}\phi$ in STL require that for constants a and b, we have $a \geq 0$ and $b \geq a$. This requirement is more difficult to impose in STd$\mathcal{L}$, since time interval shifting due to sequential composition (see Section 3.5) could result in a formula where $val(\text{first}\,\sigma,b) < val(\text{first}\,\sigma,a)$, and we need the semantics of STd$\mathcal{L}$ to handle such cases appropriately. In the rest of the paper, given a trace σ and an interval $[a,b]$, we refer to $val(\text{first}\,\sigma,a)$ and $val(\text{first}\,\sigma,b)$ as simply a and b respectively for easier readability.

3.5 Timing Hybrid Programs

A major technical difficulty arising from our integration of STL with d$\mathcal{L}$ is the fact that we now need to reason about not only the time intervals where a certain temporal property holds, but also about how the length of a trace of a hybrid program affects the time intervals under consideration. This problem surfaces immediately for the sequential composition of two programs α and β, but is in fact a general challenge with the integration of continuous traces from d$\mathcal{L}$ and temporal operators from STL.

Let us consider a trace $\sigma = \sigma_\alpha \circ \sigma_\beta \in [\![\alpha;\beta]\!]$ such that $\sigma_\alpha \in [\![\alpha]\!]$ terminates at time c, following which $\sigma_\beta \in [\![\beta]\!]$ begins. For simplicity, let us also assume that $a < c < b$ in determining the satisfiability of $\Box_{[a,b]}\psi$ by σ. Note that $\sigma \vDash \Box_{[a,b]}\psi$ if and only if $\sigma_\alpha \vDash \Box_{[a,c]}\psi$ and $\sigma_\beta \vDash \Box_{[0,b-c]}\psi$. Intuitively, this means that α runs first *until time* c and σ_α satisfies ψ from time $t+a$ to time $t+c$ (where t is the current time), following which β runs and σ_β satisfies ψ from the time it starts to the time $t+(b-c)$ (due to a shifting of the time interval, since part of the interval $[a,b]$ was already satisfied by σ_α). A key property that this rule relies on is the termination of program α at time c. The value of c is not known by a programmer in advance (since a program can have non-deterministic properties), although a programmer could annotate the code to enforce the termination of a program at a certain time. A more elegant solution, however, is to *measure* the time it takes for a program α to run, and use the measured value for the time offset for any subsequent temporal operators that may need interval shifting.

Definition 3.6 (Timing of hybrid programs). Given hybrid programs α and β, and a variable q *fresh* in α and β, the timing of hybrid programs is defined inductively as follows:

- $time(x := \theta, q) \triangleq x := \theta$
- $time(x' = \theta \,\&\, \chi, q) \triangleq \{x' = \theta, q' = 1 \,\&\, \chi\}$
- $time(?\chi, q) \triangleq ?\chi$
- $time(\alpha \cup \beta, q) \triangleq time(\alpha, q) \cup time(\beta, q)$
- $time(\alpha; \beta, q) \triangleq time(\alpha, q); time(\beta, q)$
- $time(\alpha^*, q) \triangleq (time(\alpha, q))^*$

The time taken q by a hybrid program α is then given by the program:

$$timed(\alpha, q) \equiv q := 0;\ time(\alpha, q)$$

Recall that a trace σ is a function that maps a pair (i, t) to a state $v \in \mathrm{Sta}$, whereas a state v is a function from the set of variables Var to $\mathbb{R}$. For $\sigma_\alpha \in [\![\alpha]\!]$, we write $\sigma_\alpha|_S$ to refer to σ restricted to variables in the set $S \subseteq \mathrm{Var}$. Mathematically, $\sigma_\alpha|_S : (\mathbb{R}\times\mathbb{N}) \to S \to \mathbb{R}$, where $(i, t) \mapsto \sigma_\alpha(i, t)|_S$. We can then define an equality between timed and untimed hybrid programs as follows:

Lemma 3.7 (Equality of timed and untimed hybrid programs). *Given a hybrid program α, the following set equality always holds:*

$$\{\sigma_\alpha|_{\mathrm{Var}-\{q\}} : \sigma_\alpha \in [\![\alpha]\!]\} = \{\sigma_{timed(\alpha,q)}|_{\mathrm{Var}-\{q\}} : \sigma_{timed(\alpha,q)} \in [\![timed(\alpha, q)]\!]\}$$

Proof. The proof of Lemma 3.7 is true by Definition 3.6, keeping in mind the fact that q is fresh in $timed(\alpha, q)$. □

Intuitively, Lemma 3.7 expresses that for a trace $\sigma_\alpha \in [\![timed(\alpha, q)]\!]$, there always exists a corresponding trace $\sigma'_\alpha \in [\![\alpha]\!]$, and vice versa, such that σ_α and σ'_α are identical with respect to every variable except the fresh variable q introduced by $timed(\alpha, q)$. We rely on this lemma for the proof of soundness of the STd$\mathcal{L}$ calculus.

4 PROOF CALCULUS

In this section, we outline a proof calculus for STd$\mathcal{L}$, and present a proof of soundness for the rules in the schemata of the proof calculus.

4.1 Normalization of Trace Formulas

Sequential composition of two traces is a major challenge in a calculus handling alternating program and temporal modalities. To see why, let us consider a state formula $\langle\alpha; \beta\rangle\Box_{[a,b]}\phi$ limited to terminating traces only for simplicity. This formula states that there exists a trace $\sigma_\alpha \in [\![\alpha]\!]$ followed by the trace $\sigma_\beta \in [\![\beta]\!]$ such that sequential composition of the traces satisfies $\Box_{[a,b]}\phi$. Let us assume further for simplicity that all traces $\sigma_\alpha \in [\![\alpha]\!]$ terminate between time a and b. A first attempt at writing a rule for this state formula could take the form:

$$\frac{\langle\alpha\rangle\Box_{[a,b]}\phi \wedge \langle timed(\alpha, q)\rangle\langle\beta\rangle\Box_{[0,b-q]}\phi}{\langle\alpha; \beta\rangle\Box_{[a,b]}\phi}$$

Unfortunately, this rule is intuitive but not sound, since the choice of σ_α and σ_β could be non-deterministic. The premise says that there exists a trace $\sigma_\alpha \in [\![\alpha]\!]$ in which $\Box_{[a,b]}\phi$ is true, and a trace trace $\sigma'_\alpha \in [\![\alpha]\!]$ followed by σ_β in which $\Box_{[0,b-q]}\phi$ is true, but σ_α and σ'_α need not necessarily be the same trace. To capture the fact that σ_α and σ'_α are indeed the same traces, we need a premise resembling:

$$\langle timed(\alpha, q)\rangle(\Box_{[a,b]}\phi \wedge \langle\beta\rangle\Box_{[0,b-q]}\phi)$$

The rule is not in the syntax of STd$\mathcal{L}$, since it involves a conjunction between a state formula and a trace formula. We could choose to add this conjunction to the syntax of the logic, but we would still need to reason about the meaning of this conjunction if the trace σ_α is non-terminating.

To circumvent this problem cleanly, we need a conjunction operator that reasons about properties like ϕ that are true at the end of a trace and properties like $\Box_{[a,b]}\phi$ that are true during a trace. dTL^2 introduces a notion of normalized trace formulas to achieve the expressibility needed for sequential composition for LTL formulas within the context of hybrid systems by introducing a conjunction operator $\sqcap$ and a disjunction operator $\sqcup$ [12]. We extend STd$\mathcal{L}$ with a similar normalization of trace formulas to reason about time-bounded trace properties during the execution of a trace and state properties at the end of a trace. We augment the syntax of state formulas to accept normalized trace formulas, and define the syntax of a normalized trace formula ξ as:

$$\phi, \psi ::= \dots \mid [\alpha]\xi \mid \langle\alpha\rangle\xi$$
$$\xi ::= \phi \sqcap \Box_{[a,b]}\psi \mid \phi \sqcup \Diamond_{[a,b]}\psi$$

Definition 4.1 (Semantics of normalized trace formulas). For a normalized trace formula ξ and trace $\sigma = (\sigma_0, \dots, \sigma_n) \in \mathrm{Tra}$, we say $(\sigma, (i, t)) \vDash \xi$ if σ satisfies ξ starting from subtrace σ_i at time t. We say that $\sigma \vDash \xi$ if $(\sigma, (0, 0)) \vDash \xi$. Satisfaction of normalized trace formulas with respect to a trace σ is then defined inductively as follows:

- $\sigma \vDash \phi \sqcap \Box_{[a,b]}\psi$ if and only if
  - last $\sigma \vDash \phi$ and $\sigma \vDash \Box_{[a,b]}\psi$, if σ terminates,
  - $\sigma \vDash \Box_{[a,b]}\psi$ otherwise;
- $\sigma \vDash \phi \sqcup \Diamond_{[a,b]}\psi$ if and only if
  - last $\sigma \vDash \phi$ or $\sigma \vDash \Diamond_{[a,b]}\psi$, if σ terminates,
  - $\sigma \vDash \Diamond_{[a,b]}\psi$ otherwise.

Given a normalized state formula ξ, we use the notation ξ_{sta} to refer to the state formula in ξ, and we use the notation ξ_{tra} to refer to the trace formula in ξ. For example, $(\phi \sqcap \Box_{[a,b]}\psi)_{sta} = \phi$, and $(\phi \sqcap \Box_{[a,b]}\psi)_{tra} = \Box_{[a,b]}\psi$.

We define the minimum length of normalized trace formulas required to determine the satisfaction of program necessities and possibilities as follows:

Definition 4.2 (Minimum length of normalized trace formulas). The minimum length associated with a normalized trace formula ξ, denoted by $\|\xi\|$, to determine the satisfaction of a program necessity or possibility is defined as follows:

$$\|\phi \sqcap \Box_{[a,b]}\psi\| = b$$
$$\|\phi \sqcup \Diamond_{[a,b]}\psi\| = b$$

We build on Definition 3.4 for state formulas as follows:

$$\phi \rightsquigarrow \phi \qquad (\rightsquigarrow \phi)$$
$$\Box_{[a,b]}\phi \rightsquigarrow \text{true} \sqcap \Box_{[a,b]}\phi \qquad (\rightsquigarrow \Box_I)$$
$$\Diamond_{[a,b]}\phi \rightsquigarrow \text{false} \sqcup \Diamond_{[a,b]}\phi \qquad (\rightsquigarrow \Diamond_I)$$

Figure 1: Normalization of trace formulas in STd$\mathcal{L}$.

- $v \models [\alpha]\xi$ if and only if for each trace $\sigma \in [\![\alpha]\!]$ such that first $\sigma = v$ and if σ terminates then last $\sigma \models \xi_{sta}$, and if $|\sigma| \geq val(\text{first}\,\sigma, \|\xi\|)$, we also have that $\sigma \models \xi$;
- $v \models \langle\alpha\rangle\xi$ if and only if there exists trace $\sigma \in [\![\alpha]\!]$ such that first $\sigma = v$ and if σ terminates then last $\sigma \models \xi_{sta}$, and $|\sigma| \geq val(\text{first}\,\sigma, \|\xi\|)$ and $\sigma \models \xi$.

Given the semantics of normalized trace formulas in STd$\mathcal{L}$, we derive rules to transform any trace formula in STd$\mathcal{L}$ into a normalized trace formula. The rules for normalization are shown in Figure 1. The relation $\rightsquigarrow$ allows us to only consider normalized trace formulas for the rules of the proof calculus of STd$\mathcal{L}$, thereby simplifying the proof system greatly.

Lemma 4.3 (Soundness of normalized trace formulas). *If $\pi \rightsquigarrow \xi$, then for all traces σ, it follows that $\sigma \models \pi$ if and only if $\sigma \models \xi$.*

Proof. Soundness of rule $(\rightsquigarrow \phi)$ is trivial. Soundness of rules $(\rightsquigarrow \Box_I)$, $(\rightsquigarrow \Diamond_I)$ is true by the semantics in Definition 4.1. □

Lemma 4.4 (Existence of a normalized trace formula). *For any trace formula π, there exists a state formula ϕ such that $\pi \rightsquigarrow \phi$, or a normalized trace formula ξ such that $\pi \rightsquigarrow \xi$.*

Proof. This lemma is a consequent of the $\rightsquigarrow$ relation presented in Figure 1. □

Lemma 4.4 allows the proof system of STd$\mathcal{L}$ to just focus on axiomatizing only formulas that use normalized traces, and inherit non-temporal rules from d$\mathcal{L}$ [20, 22, 23]. This results in a cleaner, simpler proof calculus for STd$\mathcal{L}$.

4.1.1 Running Example: Traction Assist in Cars. Recall that our running example introduced a safety property, ϕ, that required a skidding car's traction assist to reduce the wheel rotation ρ of the car to some constant, ρ_0, within 1 to 5 seconds to help regain traction. This property can be expressed as a normalized STd$\mathcal{L}$ formula as follows:

$$\phi ::= [\text{traction\_assist}]\left(\ \neg\text{no\_traction} \sqcup \Diamond_{[1,5]}(\rho < \rho_0)\ \right)$$

For ease of understanding, the normalized disjunction $\neg$no_traction $\sqcup \Diamond_{[1,5]}(\rho < \rho_0)$ can be thought of as the implication no_traction $\Rightarrow \Diamond_{[1,5]}(\rho < \rho_0)$ (although this implication is not directly supported in the sytax of STd$\mathcal{L}$). We provide a proof sketch of this property in Section 4.2.3.

4.2 Proof Calculus

This section presents the proof calculus of STd$\mathcal{L}$. As in d$\mathcal{L}$, the rules in the proof calculus of STd$\mathcal{L}$ typically follow a symbolic decomposition pattern whereby hybrid programs may be decomposed syntactically as needed. The proof calculus transforms STL formulas into temporal-free formulas to leverage the non-temporal rules of d$\mathcal{L}$. As such, the proof system inherits its non-temporal rules from d$\mathcal{L}$ [20, 22, 23], and adds its own temporal rules to allow for expressing temporal formulas for given time intervals. All rules should be used in the same way as in the d$\mathcal{L}$ proof calculus.

Note that with the exceptions of rules (ind $\sqcup_t$) and (con $\sqcap_t$) (see Figure 2), all rules are actually equivalences between the premise and the conclusion. In other words, each rule has a dual such that the negation of both the premise and the conclusion is also true. Therefore, when we write rule $\frac{\rho}{\phi}$, the following two rules are both true:

$$\frac{\rho}{\phi} \qquad \frac{\neg\rho}{\neg\phi}$$

Such duals for the rules contain the proof rules for $\langle\alpha\rangle$ when the original rule contains the proof rules for $[\alpha]$, and vice versa (again, except for rules (ind $\sqcup_t$) and (con $\sqcap_t$)), but are omitted here for space reasons. We include the dual rules in an extended technical report [1] for reference.

4.2.1 Inheritance of Non-Temporal and Temporal Rules. In addition to the temporal rules introduced in Figure 2, STd$\mathcal{L}$ also uses the proof system of d$\mathcal{L}$. Indeed, the goal of the proof calculus introduced here is to leverage the non-temporal rules of d$\mathcal{L}$ to reason about temporal properties of formulas. Since we build STd$\mathcal{L}$ to conservatively extend d$\mathcal{L}$, it is sound to inherit the proof calculus of d$\mathcal{L}$.

4.2.2 Introduction of New Temporal Rules. This subsection introduces the temporal rules, grouped by program construct for hybrid programs, for the proof calculus of STd$\mathcal{L}$. A detailed rule schemata for the proof calculus is included in Figure 2.

Rules ([ ] $\rightsquigarrow$) and ($\langle\rangle \rightsquigarrow$) lift normalization of trace formulas to program necessities and possibilities respectively.

For assignment rule ([:=] $\sqcap_t$), the first disjunct expresses that for the time interval $[0, 0]$, ψ must hold initially, and after the execution of the program, must continue to hold in addition to ϕ, as summarized in clause $\psi \wedge [x := \theta](\phi \wedge \psi)$. The second disjunct expresses that for any interval $[a, b]$ where $a > 0$ and $b \geq a$, only ϕ needs to be true after execution of the assignment, since assignment occurs in zero time, and as such, the trace of $x := \theta$ would not be long enough to determine the satisfiability of $\Box_{[a,b]}\psi$ for $a > 0$. Similar reasoning is used for rule ([:=] $\sqcup_t$).

For the rules for test, as a reminder, a test trace only terminates if the test passes, and is a trace of the error state if the test fails. Rule ([?] $\sqcap_t$) encapsulates the fact that a trace of $?\chi$ satisfies $\phi \sqcap \Box_{[a,b]}\psi$ if and only if

- for $a = 0$ and $b = 0$, its initial state satisfies $\phi \wedge \psi$ if the test passes, or satisfies only ψ if the test fails;
- for $a > 0$ and $b \geq a$, its initial state satisfies just ϕ if the test passes.

Note that there is no satisfaction requirement on the trace of a failing test (i.e., $\neg\chi$ is true) when $a > 0$ and $b \geq a$, since the test also occurs in zero time, and as such, the trace of $?\chi$ would not be long enough to determine the satisfiability of $\Box_{[a,b]}\psi$ in this case. Similar reasoning is used for rule ([?] $\sqcup_t$).

Rules for ordinary differential equations (ODEs) look complex at first glance, but can be broken down in slightly simpler sub-rules. It is first important to remember that ODEs could have terminating

Normalization of Trace Formulas

$$\frac{\pi \rightsquigarrow \xi \;\; [\alpha]\xi}{[\alpha]\pi}\;([\,]\rightsquigarrow) \qquad \frac{\pi \rightsquigarrow \xi \;\; \langle\alpha\rangle\xi}{\langle\alpha\rangle\pi}\;(\langle\rangle\rightsquigarrow)$$

Assignment

$$\frac{\left(\begin{array}{c}(a = 0 \wedge b = 0) \wedge (\psi \wedge [x := \theta](\phi \wedge \psi)) \vee \\ ((a > 0 \wedge b \geq a) \wedge [x := \theta]\phi)\end{array}\right)}{[x := \theta](\phi \sqcap \Box_{[a,b]}\psi)}\;([:=]\sqcap_t)$$

$$\frac{\left(\begin{array}{c}(a = 0 \wedge b = 0) \wedge (\psi \vee [x := \theta](\phi \vee \psi)) \vee \\ ((a > 0 \wedge b \geq a) \wedge [x := \theta]\phi)\end{array}\right)}{[x := \theta](\phi \sqcup \Diamond_{[a,b]}\psi)}\;([:=]\sqcup_t)$$

Test

$$\frac{\left(\begin{array}{c}((a = 0 \wedge b = 0) \wedge ((\chi \wedge (\phi \wedge \psi)) \vee (\neg\chi \wedge \psi)) \vee \\ ((a > 0 \wedge b \geq a) \wedge (\neg\chi \vee (\chi \wedge \phi)))\end{array}\right)}{[?\chi](\phi \sqcap \Box_{[a,b]}\psi)}\;([?]\sqcap_t)$$

$$\frac{\left(\begin{array}{c}((a = 0 \wedge b = 0) \wedge ((\chi \wedge (\phi \vee \psi)) \vee (\neg\chi \wedge \psi)) \vee \\ ((a > 0 \wedge b \geq a) \wedge (\neg\chi \vee (\chi \wedge \phi)))\end{array}\right)}{[?\chi](\phi \sqcup \Diamond_{[a,b]}\psi)}\;([?]\sqcup_t)$$

Ordinary Differential Equation

$$\frac{\left(\begin{array}{c}(b < a \wedge [x' = \theta \;\&\; \chi]\phi) \vee \\ (\neg\chi \wedge (\neg(a = 0 \wedge b \geq a) \vee \psi)) \vee \\ ([t := 0; \{x' = \theta, t' = 1 \;\&\; (\chi \wedge t \leq a)\}; ?(t = a)] \\ [\{x' = \theta, t' = 1 \;\&\; (\chi \wedge t \leq b)\}]\psi \wedge [x' = \theta \;\&\; \chi]\phi)\end{array}\right)}{[x' = \theta \;\&\; \chi](\phi \sqcap \Box_{[a,b]}\psi)}\;([']\sqcap_t)$$

$$\frac{\left(\begin{array}{c}(b < a \wedge [x' = \theta \;\&\; \chi]\phi) \vee \\ ((\chi \vee [t := 0; \{x' = \theta, t' = 1 \;\&\; (\chi \wedge t \leq a)\}; ?(t = a)]\psi) \wedge \\ [x' = \theta \;\&\; (\chi \wedge \neg\psi)]\phi \wedge \\ [t := 0; \{x' = \theta, t' = 1 \;\&\; (\chi \wedge t \leq a)\}; ?(t = a)] \\ \langle\{x' = \theta, t' = 1 \;\&\; (t \leq b)\}\rangle(\neg\chi \vee \psi))\end{array}\right)}{[x' = \theta \;\&\; \chi](\phi \sqcup \Diamond_{[a,b]}\psi)}\;([']\sqcup_t)$$

Non-deterministic Choice

$$\frac{[\alpha]\xi \wedge [\beta]\xi}{[\alpha \cup \beta]\xi}\;([\cup]\;\xi)$$

Sequential Composition

$$\frac{[timed(\alpha, q)]([\beta](\phi \sqcap \Box_{[\max(0,a-q),b-q]}\psi) \sqcap \Box_{[a,\min(b,q)]}\psi)}{[\alpha;\beta](\phi \sqcap \Box_{[a,b]}\psi)}\;([;]\sqcap_t)$$

$$\frac{[timed(\alpha, q)]([\beta](\phi \sqcup \Diamond_{[\max(0,a-q),b-q]}\psi) \sqcup \Diamond_{[a,\min(b,q)]}\psi)}{[\alpha;\beta](\phi \sqcup \Diamond_{[a,b]}\psi)}\;([;]\sqcup_t)$$

Figure 2: Rule schemata of the proof calculus for STd$\mathcal{L}$.

Non-deterministic Finite Repetition

$$\frac{\left(\begin{array}{c}(\phi \wedge (\neg(a = 0 \wedge b = 0) \vee \psi)) \wedge \\ [timed(\alpha^*, q)][\alpha](\phi \sqcap \Box_{[max(0,a-q),b-q]}\psi)\end{array}\right)}{[\alpha^*](\phi \sqcap \Box_{[a,b]}\psi)}\;([^*]\sqcap_t)$$

$$\frac{(\psi \wedge (a = 0 \wedge b \geq a)) \vee (\phi \wedge [\alpha^*;\alpha](\phi \sqcup \Diamond_{[a,b]}\psi))}{[\alpha^*](\phi \sqcup \Diamond_{[a,b]}\psi)}\;([^{*n}]\sqcup_t)$$

$$\frac{\phi \implies [\alpha](\phi \sqcup \Diamond_{[a,b]}\psi)}{\forall^\alpha(\phi \implies [\alpha^*](\phi \sqcup \Diamond_{[a,b]}\psi))}\;(\mathrm{ind}\sqcup_t)$$

$$\frac{\forall^\alpha\forall r > 0\;(\varphi(r) \implies \langle\alpha\rangle(\varphi(r-1) \sqcap \Box_{[a,b]}\psi))}{(\exists r.\varphi(r)) \wedge \psi \implies \langle\alpha^*\rangle((\exists r.r \leq 0 \wedge \varphi(r)) \sqcap \Box_{[a,b]}\psi)}\;(\mathrm{con}\sqcap_t)$$

Figure 2 (continued): Rule schemata of the proof calculus for STd$\mathcal{L}$.

traces or error traces, and the rules for ODEs need to account of both possibilities. With that in mind, we conclude that an error trace of $x' = \theta \;\&\; \chi$ satisfies $\phi \sqcap \Box_{[a,b]}\psi$ if and only if $a = 0$ and $b \geq a$ implies ψ, as the second disjunct in rule ($[']\sqcap_t$). For non-error traces of $x' = \theta \;\&\; \chi$, we first transform the program into a program of the form $t := 0; \{x' = \theta, t' = 1 \;\&\; (\chi \wedge t \leq a)\}; ?(t = a)$ and $\{x' = \theta, t' = 1 \;\&\; (\chi \wedge t \leq b)\}$ to enforce that the differential equation first runs from time $t = 0$ to time $t = a$ without any satisfaction requirements on ψ, followed by running the equation from time $t = a$ to $t = b$, during which ψ must be true. In addition to this, ϕ must be true after running the program $x' = \theta \;\&\; \chi$, to deal with the case where the execution exits the differential equation before time a. This is summarized in the third disjunct of the rule ($[']\sqcap_t$). Note that the first disjunct of the rule deals with the case where $b < a$, so $\Box_{[a,b]}\psi$ is defined to be trivially true, and any trace of $x' = \theta \;\&\; \chi$ need only satisfy ϕ. Rule ($[']\sqcup_t$) expresses that a trace of $x' = \theta \;\&\; \chi$ satisfies $\phi \sqcup \Diamond_{[a,b]}\psi$ if and only if either $b < a$ and the trace satisfies ϕ upon termination, or

– the differential equation can evolve or has satisfied ψ at time $t = a$ (as in the first conjunct of the rule),
– if no trace of the differential equation can satisfy ψ, all traces must satisfy ϕ instead (as in the second conjunct of the rule),
– either there does not exist a non-terminating trace of the differential equation – transformed to a program as in rule ($[']\sqcap_t$) – or such a trace satisfies ψ between times a and b.

Rule ($[\cup]\;\xi$) for non-deterministic choice is lifted directly from the corresponding rule $[\cup]\Box$ in d$\mathcal{L}$.

The rules for sequential composition were one of the most challenging aspects of STd$\mathcal{L}$. Indeed, sequential composition is the sole reason why we use normalized trace formulas in STd$\mathcal{L}$ (see Section 4.1), and a primary reason why introduce the notion of recording the amount of time it takes for a hybrid program α to execute (see Section 3.5). As a reminder here, for a hybrid program α, executing $timed(\alpha, q)$ is equivalent to executing α while recording the amount of time the program takes to execute, following which the timed value is output as a fresh variable q. With that in mind, a trace of $\alpha;\beta$ satisfies $\phi \sqcap \Box_{[a,b]}\psi$ if and only if

– for $a \leq q \leq b$, all traces of α satisfy $\Box_{[a,q]}\psi$, and for traces of α that terminate at time q, all following traces of β satisfy $\phi \sqcap \Box_{[0,b-q]}\psi$,
– for $a \leq b \leq q$, all traces of α satisfy $\Box_{[a,b]}\psi$, and for traces of α that terminate at time q, all following traces of β satisfy ϕ,
– for $q \leq a \leq b$, for traces of α that terminate at time q, all following traces of β satisfy $\phi \sqcap \Box_{[a-q,b-q]}\psi$.

These properties for the cases of the relative ordering of a, b and q are captured succinctly in rule ($[;]\sqcap_t$) using `min` and `max`. Rule ($[;]\sqcup_t$) is similar.

$$\cfrac{\cfrac{\cfrac{\cfrac{\cdots}{[timed(\alpha_1, q_1)]([timed(\alpha_2, q_2)](\psi_3 \sqcup \Diamond_{[\max(0,1-q_1),\min(5-q_1,q_2)]}) \sqcup \Diamond_{[1,\min(5,q_1)]}\psi_2)}\;[']\sqcup_t}{[timed(\alpha_1, q_1)]([timed(\alpha_2, q_2)]([\alpha_3](\psi_1 \sqcup \Diamond_{[\max(0,\max(0,1-q_1)-q_2),5-q_1-q_2]}\psi_2) \sqcup \Diamond_{[\max(0,1-q_1),\min(5-q_1,q_2)]}) \sqcup \Diamond_{[1,\min(5,q_1)]}\psi_2)}\;[']\sqcup_t}{[timed(\alpha_1, q_1)]([\alpha_2;\alpha_3](\psi_1 \sqcup \Diamond_{[\max(0,1-q_1),5-q_1]}\psi_2) \sqcup \Diamond_{[1,\min(5,q_1)]}\psi_2)}\;[;]\sqcup_t}{[\alpha](\psi_1 \sqcup \Diamond_{[1,5]}\psi_2)}\;[;]\sqcup_t$$

Figure 3: Proof sketch of our running example. The state formula ψ_3 can be further proven using rules from d$\mathcal{L}$. We omit a full proof for space reasons.

For the rules for non-deterministic finite repetition, let us first remember that as long as a trace α is finite, its finite repetition α^* will also be finite. In general, the rules attempt to reduce temporal properties of loops into either non-temporal properties of loops, or slightly more complex temporal properties on a program but without any loops. The idea here is to make the rules provable by ordinary, non-temporal induction. The key intuition behind rule ($[^*] \sqcup_t$) comes from a very useful rule for repetition from d$\mathcal{L}$, which says that for a given trace formula π, the following is true:

$$\frac{[?true]\pi \wedge [\alpha^*;\alpha]\pi}{[\alpha^*]\pi}\;[;]$$

Rule ($[^*] \sqcap_t$) captures the fact that a trace of α^* satisfies $\phi \sqcap \Box_{[a,b]}\psi$ if and only if when α repeats zero times, ϕ is true, and if $a = 0$ and $b = 0$ then ψ is true as well, or α^* runs first followed by α, during which $\phi \sqcap \Box_{[a,b]}\psi$ with time interval shifting (similar to that for the sequential composition rules) holds. In rule ($[^*] \sqcup_t$), the first disjunct expresses that $\Diamond_{[a,b]}\psi$ holds without repeating α if $a = 0$ and $b \geq a$ and ψ is true initially; the first conjunct of the second disjunct deals with the case where α repeats zero times and ψ is false initially, while the second conjunct requires a sequential composition of $\alpha^*;\alpha$ to satisfy $\phi \sqcup \Diamond_{[a,b]}\psi$ according to the rule ([;]) from d$\mathcal{L}$ mentioned above. Note that for rule ($[^*] \sqcup_t$), the use of $\alpha^*;\alpha$ is equivalent to the use of $\alpha;\alpha^*$, and either variant of the sequential composition may be used. The rules (ind $\sqcup_t$) and (con $\sqcap_t$) extend the rules of induction (ind) and convergence (con) from d$\mathcal{L}$ to normalized trace formulas. Consistent with the rules from d$\mathcal{L}$, the rules (ind $\sqcup_t$) and (con $\sqcap_t$) are not equivalence relations (i.e., they do not have dual counterparts such that the negation of the premise and the conclusion is also a rule). The notation $\forall^\alpha$ from d$\mathcal{L}$ is a quantification over all variables that could be assigned by a hybrid program α in assignments or differential equations. Rule (ind $\sqcup$) expresses that ϕ is inductive with exit clause $\Diamond_{[a,b]}\psi$ (i.e., ϕ is true after all traces $\sigma \in [\![\alpha]\!]$ where first $\sigma \vDash \phi$, except when ψ was true at some point in the interval I during the execution of σ), while rule (con $\sqcap$) shows that φ is a variant of some trace $\sigma \in [\![\alpha]\!]$ (as in, its level r decreases) during which ψ is always true, and starting from an initial r, for an r for which $\varphi(r)$ holds, it will ultimately be the case that $r \leq 0$ without ψ being false if we repeat α^* often enough [13].

4.2.3 Running Example: Traction Assist in Cars. In this subsection, we present a proof sketch of the safety property for our example, highlighting how the property expressed in STd$\mathcal{L}$ is reduced to an equivalent d$\mathcal{L}$ formula to leverage the d$\mathcal{L}$ calculus. For ease of understanding of the proof sketch, and due to space concerns, we only consider the second half of the hybrid program traction_assist (referred to as α) – though the application of the STd$\mathcal{L}$ proof rules to the first half of the program is also fairly straightforward. We refer to the sequential composition components $\omega := -1$, $\varphi := 10$ and $\rho' = \text{cruise}(\omega)$) in α as α_1, α_2, and α_3 respectively. We then refer to the safety property ϕ as $[\alpha](\psi_1 \sqcup \Diamond_{[1,5]}\psi_2)$, where $\psi_1 \equiv \neg\text{no\_traction}$ and $\psi_2 \equiv (\rho < \rho_0)$.

For $\alpha \equiv (\alpha_1; (\alpha_2; \alpha_3))$, using the STd$\mathcal{L}$ proof calculus, we get a proof tree of the form presented in Figure 3, where ψ_3 is obtained by applying rule $[']\sqcup_t$ with $a = \max(0, \max(0, 1 - q_1) - q_2)$ and $b = 5 - q_1 - q_2$ as follows:

$$\psi_3 \equiv [\alpha_3]\left(\begin{array}{c} (b < a \wedge [\rho' = \omega \times k - \varphi \times j]\psi_1) \vee \\ (([t := 0; \{x' = \theta, t' = 1 \;\&\; (\chi \wedge t \leq a)\}; ?(t = a)]\psi_2) \wedge \\ [\rho' = \omega \times k - \varphi \times j \;\&\; (\neg\psi_2)]\psi_1 \wedge \\ [t := 0; \{\rho' = \omega \times k - \varphi \times j, t' = 1 \;\&\; (t \leq a)\}; ?(t = a)] \\ \langle\{\rho' = \omega \times k - \varphi \times j, t' = 1 \;\&\; (t \leq b)\}\rangle\psi_2) \end{array}\right)$$

The STd$\mathcal{L}$ state formula ψ_3 can be proven further using solely the non-temporal rules from d$\mathcal{L}$.

4.3 Soundness and Completeness of the STd$\mathcal{L}$ Proof Calculus

THEOREM 4.5. *The proof calculus for STd$\mathcal{L}$ is sound.*

Since STd$\mathcal{L}$ conservatively extends d$\mathcal{L}$, the soundness of the proof calculus of d$\mathcal{L}$ applies to STd$\mathcal{L}$ as well. We present the proof of soundness for the rules introduced by the STd$\mathcal{L}$ calculus.

PROOF. We prove the soundness of individual rules. By induction on the proof trees, soundness of the entire proof system is a corollary.

($[:=] \sqcap_t$): For any state v, there is a unique terminating trace $\sigma \in [\![x := \theta]\!]$ such that first $\sigma = v$. From the trace semantics of hybrid programs, we know that $\sigma = (\hat{v}, \hat{w})$ with $w = [x \mapsto val(v, \theta)]$. Therefore, $v \vDash [x := \theta](\phi \sqcap \Box_I\psi)$ if and only if

- for $I_s = 0$ and $I_e = 0$, $w \vDash \phi$, $v \vDash \psi$, and $w \vDash \psi$, which is true if and only if $v \vDash \psi \wedge [x := \theta](\phi \wedge \psi)$;
- for $I_s > 0$ and $I_e \geq I_s$, $w \vDash \phi$, which is true if and only if $v \vDash [x := \theta]\phi$.

In either case, it follows that $v \vDash [x := \theta](\phi \sqcap \Box_I\psi)$ if and only if $v \vDash ((I_s = 0 \wedge I_e = 0) \wedge (\psi \wedge [x := \theta](\phi \wedge \psi)) \vee ((I_s > 0 \wedge I_e \geq I_s) \wedge [x := \theta]\phi)$.

($[?] \sqcap_t$): ($\rightarrow$) Let $v \vDash ((a = 0 \wedge b = 0) \wedge ((\chi \wedge (\phi \vee \psi)) \vee (\neg\chi \wedge \psi)) \vee ((a > 0 \wedge b \geq a) \wedge (\neg\chi \vee (\chi \wedge \phi)))$, and let $\sigma \in [\![?\chi]\!]$ with first $\sigma = v$. If $v \vDash \neg\chi$, then $\sigma = (\hat{v}, \hat{\Lambda})$ (i.e., σ is the error trace). If $a = 0$ and $b = 0$, by our assumption, it follows that $v \vDash \psi$ (otherwise v does not satisfy anything). Since σ is a trace that occurs in zero time, it follows that $\sigma \vDash (\phi \sqcap \Box_{[a,b]}\psi)$. If, however, $v \vDash \chi$, then

$\sigma = (\hat{v})$, and by our assumption, if $a > 0$ and $b \geq a$, then $v \models \phi$ only (since the length of σ is not long enough to determine the satisfiability of $\Box_{[a,b]}\psi$). Therefore, $\sigma \models (\phi \sqcap \Box_{[a,b]}\psi)$ in this case as well.
($\leftarrow$) Conversely, assume that $v \models [?\chi](\phi \sqcap \Box_{[a,b]}\psi)$. Now, if $v \models \neg\chi$, then $\sigma = (\hat{v}, \hat{\Lambda})$ (which is a non-terminating state), and $v \models \psi$ only when $a = 0$ and $b = 0$. Otherwise, $v \models \chi$ and $\sigma = (\hat{v})$, and therefore $v \models (\phi \wedge \psi)$ when $a = 0$ and $b = 0$, or $v \models \phi$ when $a > 0$ and $b \geq a$. In either case, $v \models ((a = 0 \wedge b = 0) \wedge ((\chi \wedge (\phi \vee \psi)) \vee (\neg\chi \wedge \psi)) \vee ((a > 0 \wedge b \geq a) \wedge (\neg\chi \vee (\chi \wedge \phi)))$.

([′] $\sqcap_t$): ($\rightarrow$) Let $v \models (b < a \wedge [x' = \theta \,\&\, \chi]\phi) \vee (\neg\chi \wedge (\neg(a = 0 \wedge b \geq a) \vee \psi)) \vee ([t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)][\{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq b)\}]\psi \wedge [x' = \theta \,\&\, \chi]\phi)$, and let $\sigma \in [\![x' = \theta \,\&\, \chi]\!]$ such that first $\sigma = v$. If $b < a$, it is only required that $\sigma \models \phi$ (since $\Box_{[a,b]}\psi$ is trivially true in this case). If $v \models \neg\chi$, then σ is the non-terminating error trace $(\hat{v}, \hat{\Lambda})$ and $\sigma \models \Box_{[a,b]}\psi$ (since ψ is true when $a = 0$ and $b \leq a$). Therefore, $\sigma \models (\phi \sqcap \Box_{[a,b]}\psi)$. If $v \models \chi$, however, then $\sigma = \{f\}$ for a real function f defined on $D = [0, r]$ solution of $x' = \theta$, which satisfies χ on its domain of definition. Since $v \models [x' = \theta \,\&\, \chi]\phi$, for any σ that terminates, $\sigma \models \phi$. For a σ that does not terminate, $v \models [t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)][\{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq b)\}]\psi$, and therefore $\sigma \models \Box_{[a,b]}\psi$. In either case, $\sigma \models (\phi \sqcap \Box_{[a,b]}\psi)$.
($\leftarrow$) Conversely, assume $v \models [x' = \theta \,\&\, \chi](\phi \sqcap \Box_{[a,b]}\psi)$. By definition, there exists at least one trace $\sigma \in [\![x' = \theta \,\&\, \chi]\!]$ such that first $\sigma = v$ and $\sigma \models \Box_{[a,b]}\psi$. Now, if $v \models \neg\chi$, then $v \models \psi$ if $a = 0$ and $b \geq a$. Otherwise, for non-error traces of $x' = \theta \,\&\, \chi$,

- for a terminating trace $\sigma \in [\![x' = \theta \,\&\, \chi]\!]$, we have that $\sigma \models \phi \sqcap \Box_{[a,b]}\psi$, and in particular, we have that $\sigma \models \phi$
- for any trace $\sigma \in [\![x' = \theta \,\&\, \chi]\!]$ (terminating or otherwise), since $\sigma \models \phi \sqcap \Box_{[a,b]}\psi$, in particular we have that $\sigma \models \Box_{[a,b]}\psi$, and hence, $\sigma \models [t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)][\{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq b)\}]\psi$.

Therefore, $v \models (b < a \wedge [x' = \theta \,\&\, \chi]\phi) \vee (\neg\chi \wedge (\neg(a = 0 \wedge b \geq a) \vee \psi)) \vee ([t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)][\{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq b)\}]\psi \wedge [x' = \theta \,\&\, \chi]\phi)$.

([′] $\sqcup_t$): ($\rightarrow$) Assume $v \models (b < a \wedge [x' = \theta \,\&\, \chi]\phi) \vee ((\chi \vee [t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)]\psi) \wedge [x' = \theta \,\&\, (\chi \wedge \neg\psi)]\phi \wedge [t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)]\langle\{x' = \theta, t' = 1 \,\&\, (t \leq b)\}\rangle(\neg\chi \vee \psi))$ and let $\sigma \in [\![x' = \theta \,\&\, \chi]\!]$ such that first $\sigma = v$. If $b < a$, then $v \models [x' = \theta \,\&\, \chi]\phi$. If $v \models \neg\chi$, then σ is the non-terminating trace $(\hat{v}, \hat{\Lambda})$ such that $\sigma \models \Diamond_{[a,b]}\psi$. Therefore, $\sigma \models \phi \sqcup \Diamond_{[a,b]}\psi$. If $v \models \chi$, however, then $\sigma = \{f\}$ for a real function f defined on $D = [0, r]$ solution of $x' = \theta$, which satisfies χ on its domain of definition. If $\sigma \models \Diamond_{[a,b]}\psi$, then by definition, $\sigma \models \phi \sqcup \Diamond_{[a,b]}\psi$. Otherwise, if σ is terminating and no state of σ satisfies ψ, we have that $\sigma \in [\![x' = \theta \,\&\, (\chi \wedge \neg\psi]\!]$. From our assumption, we have that $\sigma \models \phi$, and as such, $\sigma \models \phi \sqcup \Diamond_{[a,b]}\psi$. Lastly, for the case case where $\sigma \not\models \Diamond_{[a,b]}\psi$, we cannot have a non-terminating σ. This is because such a σ would verify $\chi \wedge \neg\psi$ in all states, and could follow any trace $\sigma_\alpha \in [\![x' = \theta]\!]$, contradicting $v \models [t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)]\langle\{x' = \theta, t' = 1 \,\&\, (t \leq b)\}\rangle(\neg\chi \vee \psi)$ in the process.
($\leftarrow$) Conversely, let $v \models [x' = \theta \,\&\, \chi](\phi \sqcup \Diamond_{[a,b]}\psi)$, and let $\sigma \in [\![x' = \theta \,\&\, \chi]\!]$ such that first $\sigma = v$. First, if $b < a$, the $\Diamond_{[a,b]}\psi$ is vacuously false, and since $\sigma \models (\phi \sqcup \Diamond_{[a,b]}\psi)$, it must be the case that $\sigma \models \phi$. Otherwise, if $v \models \neg\chi$, the only trace of $[\![x' = \theta \,\&\, \chi]\!]$ such that first $\sigma = v$ is the trace $(\hat{v}, \hat{\Lambda})$. Since this trace satisfies $\Diamond_{[a,b]}\psi$, we have that $v \models [t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)]\psi$. Therefore, in all cases, we have $v \models \chi \vee [t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)]\psi$. To prove that $v \models [x' = \theta \,\&\, (\chi \wedge \neg\psi)]\phi$, we need only consider terminating tracing. Let σ be a terminating trace of $[\![x' = \theta \,\&\, (\chi \wedge \neg\psi)]\!]$. Then, in particular, $\sigma \in [\![x' = \theta \,\&\, \chi]\!]$, and as such, $\sigma \models \phi \sqcup \Diamond_{[a,b]}\psi$. Since σ also has $\neg\psi$ as domain constraint, it follows that $\sigma \not\models \Diamond_{[a,b]}\psi$, and as such, $\sigma \models \phi$. Finally, to prove the third conjunct of the rule, let us first consider the case where $v \models [t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)]\langle\{x' = \theta, t' = 1 \,\&\, (t \leq b)\}\rangle\neg\chi$. In this case, there is no non-terminating trace $\sigma \in [\![x' = \theta \,\&\, \chi]\!]$ such that first $\sigma = v$. For the case where $v \models [t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)]\langle\{x' = \theta, t' = 1 \,\&\, (t \leq b)\}\rangle\psi$, there exists a unique non-terminating trace $\sigma \in [\![x' = \theta \,\&\, \chi]\!]$ such that first $\sigma = v$. By our assumption, we have that $\sigma \models \Diamond_{[a,b]}\psi$. This means that ψ has to be true in some state that is reached by trace σ, and this notion is logically equivalent to $[t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)]\langle\{x' = \theta, t' = 1 \,\&\, (t \leq b)\}\rangle\psi$. From both of the cases mentioned above, we get $v \models [t := 0; \{x' = \theta, t' = 1 \,\&\, (\chi \wedge t \leq a)\}; ?(t = a)]\langle\{x' = \theta, t' = 1 \,\&\, (t \leq b)\}\rangle(\neg\chi \vee \psi)$.

The remainder of the proof of soundness for the STd$\mathcal{L}$ calculus is presented in an extended technical report [1] for space reasons. □

Theorem 4.6. *STd$\mathcal{L}$ is non-axiomatizable.*

Proof. Discrete and continuous fragments of d$\mathcal{L}$ were proved to not be axiomatizable in [20, 23]. Since STd$\mathcal{L}$ extends d$\mathcal{L}$, discrete and continuous fragments of STd$\mathcal{L}$ are also non-axiomatizable. Therefore, in general, STd$\mathcal{L}$ is non-axiomatizable. □

Even though STd$\mathcal{L}$ is non-axiomatizable in general, its proof system restricted programs without repetitions is complete relative to first-order logic of differential equations (i.e., first-order real arithmetic augmented with formulas expressing properties of differential equations) [20, 23], as was shown to be the case for d$\mathcal{L}$.

Theorem 4.7. *The proof calculus for STd$\mathcal{L}$ restricted to programs without non-deterministic finite repetitions is complete relative to first-order logic of differential equations.*

Proof. If we restrict STd$\mathcal{L}$ to programs without repetition, the proof calculus for STd$\mathcal{L}$ reduces temporal properties to non-temporal properties to leverage the calculus of d$\mathcal{L}$, which is proven to be complete relative to first-order logic of differential equations [20, 21, 23]. More specifically, any temporal rule in the STd$\mathcal{L}$ calculus transforms a normalized trace formula to a simpler normalized trace formula either without a temporal operator or with a temporal operator following a simpler, decomposed program. Every proof rule is an equivalence relation (i.e., the premise is equivalent to the conclusion), and Lemma 4.4 ensures that every trace formula in the syntax of STd$\mathcal{L}$ can be converted into a normalized trace formula able to be handled by the STd$\mathcal{L}$ calculus. Therefore, the relative completeness result of d$\mathcal{L}$ extends to STd$\mathcal{L}$ limited to programs without repetition. □

Indeed, we conjecture that the STd$\mathcal{L}$ proof calculus is complete relative to first-order logic of differential equations for all STd$\mathcal{L}$ programs. We leave a formal proof of full relative completeness as future work.

5 FUTURE WORK

We plan on working on the following improvements to STd$\mathcal{L}$ as future work:

- *Proving full relative completeness of STd$\mathcal{L}$:* While we prove that the calculus presented in STd$\mathcal{L}$ restricted to programs without non-deterministic repetition is complete relative to first-order logic of differential equations, we conjecture that the calculus is indeed complete relative to first-order logic of differential equations for all programs. We have yet to prove this conjecture formally.
- *Allowing for nested temporal operators in STd$\mathcal{L}$:* The fragment of STL currently supported by our work does not include properties with nested temporal operators, such as $\square_{[a,b]}\lozenge_{[c,d]}\phi$, to simplify the proof system. We do not consider this to be a significant drawback, since the fragment of STL considered is sufficient to cover a large amount of properties of interest expressed in previous case studies involving STL [3, 7, 14, 28, 29]. Nevertheless, we hope to remove this restriction in the future to further increase the expressive power of STd$\mathcal{L}$.
- *Implementing the rules for STd$\mathcal{L}$*: We hope to implement the rules for the STd$\mathcal{L}$ proof system into a theorem prover for hybrid systems such as KeYmaera X [8, 24].

6 RELATED WORK

In this section, we explore works related to reasoning about properties of hybrid systems and using STL for monitoring and verfication purposes.

STL [15, 16] was introduced for monitoring properties over continuous signals, and has since been studied widely, e.g., in Deshmukh et al. [5], Donzé and Maler [6], Maler et al. [17]. Most uses of STL have been mainly for monitoring purposes. However, there has been some work done on studying temporal properties of hybrid systems in the context of model checking. Mysore et al. [18] examine model checking of semi-algebraic hybrid systems for Timed Computation Tree Logic properties. Their work focuses on bounded model checking for differential equations with polynomial solutions only, while we allow for more general polynomial differential equations. Roehm et al. [30] define a new reachset temporal logic (RTL) and transform STL properties to RTL properties to perform model checking of continuous and hybrid systems. More recently, Bae and Lee [3] explore a bounded model checking of signal temporal logic properties using syntactic separation of STL. For both [3] and [18], the applications presented focus on bounded safety verification, while our work allows unbounded safety verification. Better still, our proof system enables proving strong liveness properties for hybrid systems, a trait not present in works like [3], [30], and [18].

Process logic [9, 19, 26] originally used Pnueli's temporal logic [25] in the context of Harel et al.'s dynamic logic [10] for temporal reasoning of hybrid systems. However, it is restricted to discrete programs and only considers an abstract notion of atomic programs, without supporting explicit assignments and tests. Platzer [20, 22, 23] introduce differential dynamic logic (d$\mathcal{L}$) to reason about the end states of a hybrid program, later followed by differential temporal dynamic logic (dTL) [21] to reason about intermediate states of hybrid programs throughout the execution of the program using some temporal operators of linear temporal logic. Jeannin and Platzer [12] present dTL^2, a logic that extends dTL and allows for alternating program and temporal modalities. While our work draws on the technical machinery from dTL^2, the logic has a significant drawback compared to STd$\mathcal{L}$ in that it does not support reasoning about properties in given time intervals. This nature of reasoning not only is often crucial to proving safety of hybrid systems, but also allows for expressing a significantly richer set of liveness properties.

Sogokon et al. present a proof method for proving eventuality properties [31] and persistence properties [32] in hybrid systems. Their methods focus on properties of the form $\lozenge_{[0,t]}\square_{[0,\infty)}P$, whereas our formalism is more general but does not support alternating temporal modalities – the properties that the two results focus on are complementary to each other. Note, however, that their formalism operates on the level of hybrid automata [2, 11], which unlike hybrid programs, do not enjoy the property of having a compositional semantics that can be used to verify systems by verifying properties of their parts in a theorem prover. Tan and Platzer [33] present an axiomatic approach for deductive verification of existence and liveness for ordinary differential equations with d$\mathcal{L}$, but their approach only focuses on liveness for differential equations, and not entire hybrid systems. They also only work on formulas of the form $\langle\alpha\rangle P$, which is a fairly limited form of liveness.

Davoren and Nerode [4] study hybrid systems in the context of the propositional μ-calculus. They provide a calculus to prove formulas in their systems, but with a propositional system (and not a first-order one). Furthermore, they do not provide specific rules in their proof system to handle ordinary differential equations.

7 CONCLUSION

In this work, we introduce signal temporal dynamic logic (STd$\mathcal{L}$), a logic that extends and combines differential dynamic logic (d$\mathcal{L}$) with a fragment of signal temporal logic (STL). STd$\mathcal{L}$ is a conservative extension of d$\mathcal{L}$ and allows reasoning not only about the final states of a hybrid system, but also the intermediate states of a hybrid system in given time intervals. While STL was originally intended to be a logic for monitoring systems, and has widely been used for exactly that purpose, we show that STL can very well be used for deductive verification of hybrid systems. STd$\mathcal{L}$ allows us to prove a greater set of both safety and liveness properties than was possible with logics preceding STd$\mathcal{L}$. We provide a semantics and a sound proof calculus for STd$\mathcal{L}$, along with proofs of soundness and relative completeness.

Acknowledgements

The authors would like to thank Nikos Aréchiga for insightful discussions. Toyota Research Institute ("TRI") provided funds to assist the authors with their research, but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

REFERENCES

[1] Hammad Ahmad and Jean-Baptiste Jeannin. 2021. *A Program Logic to Verify Signal Temporal Logic Specifications of Hybrid Systems: Extended Technical Report.* Technical Report CSE-TR-002-21. Computer Science and Engineering Department, University of Michigan, Ann Arbor, MI, 48109. https://www.eecs.umich.edu/techreports/cse/2021/CSE-TR-002-21.pdf

[2] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. 1992. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*. Springer, 209–229.

[3] Kyungmin Bae and Jia Lee. 2019. Bounded Model Checking of Signal Temporal Logic Properties Using Syntactic Separation. *Proc. ACM Program. Lang.* 3, POPL, Article 51 (Jan. 2019), 30 pages. https://doi.org/10.1145/3290364

[4] Jennifer Mary Davoren and Anil Nerode. 2000. Logics for hybrid systems. *Proc. IEEE* 88, 7 (2000), 985–1010.

[5] Jyotirmoy V Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A Seshia. 2017. Robust online monitoring of signal temporal logic. *Formal Methods in System Design* 51, 1 (2017), 5–30.

[6] Alexandre Donzé and Oded Maler. 2010. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 92–106.

[7] Alexandre Donzé, Oded Maler, Ezio Bartocci, Dejan Nickovic, Radu Grosu, and Scott Smolka. 2012. On temporal logic and signal processing. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 92–106.

[8] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. 2015. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In *International Conference on Automated Deduction*. Springer, 527–538.

[9] David Harel, Dexter Kozen, and Rohit Parikh. 1982. Process logic: Expressiveness, decidability, completeness. *Journal of computer and system sciences* 25, 2 (1982), 144–170.

[10] David Harel, Dexter Kozen, and Jerzy Tiuryn. 2001. Dynamic logic. In *Handbook of philosophical logic*. Springer, 99–217.

[11] Thomas A Henzinger. 2000. The theory of hybrid automata. In *Verification of digital and hybrid systems*. Springer, 265–292.

[12] Jean-Baptiste Jeannin and André Platzer. 2014. dTL^2: Differential temporal dynamic logic with nested temporalities for hybrid systems. In *International Joint Conference on Automated Reasoning*. Springer, 292–306.

[13] Jean-Baptiste Jeannin and André Platzer. 2014. *dTL^2: Differential Temporal Dynamic Logic with Nested Temporalities for Hybrid Systems.* Technical Report CMU-CS-14-109. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 15213. http://reports-archive.adm.cs.cmu.edu/anon/2013/abstracts/14-109.html

[14] Susmit Jha, Ashish Tiwari, Sanjit A Seshia, Tuhin Sahai, and Natarajan Shankar. 2019. TeLEx: learning signal temporal logic from positive examples using tightness metric. *Formal Methods in System Design* 54, 3 (2019), 364–387.

[15] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 152–166.

[16] Oded Maler and Dejan Ničković. 2013. Monitoring properties of analog and mixed-signal circuits. *International Journal on Software Tools for Technology Transfer* 15, 3 (2013), 247–268.

[17] Oded Maler, Dejan Nickovic, and Amir Pnueli. 2008. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of computer science*. Springer, 475–505.

[18] Venkatesh Mysore, Carla Piazza, and Bud Mishra. 2005. Algorithmic algebraic model checking II: Decidability of semi-algebraic model checking and its applications to systems biology. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 217–233.

[19] Hirokazu Nishimura. 1980. Descriptively complete process logic. *Acta Informatica* 14, 4 (1980), 359–369.

[20] André Platzer. 2008. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning* 41, 2 (2008), 143–189.

[21] André Platzer. 2010. *Differential Temporal Dynamic Logic dTL.* Springer Berlin Heidelberg, Berlin, Heidelberg, 203–230. https://doi.org/10.1007/978-3-642-14509-4_4

[22] André Platzer. 2010. *Logical analysis of hybrid systems: proving theorems for complex dynamics.* Springer Science & Business Media.

[23] André Platzer. 2012. Logics of dynamical systems. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*. IEEE Computer Society, 13–24.

[24] André Platzer and Jan-David Quesel. 2008. KeYmaera: A hybrid theorem prover for hybrid systems (system description). In *International Joint Conference on Automated Reasoning*. Springer, 171–178.

[25] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE, 46–57.

[26] V. R. Pratt. 1979. Process Logic: Preliminary Report. In *Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (San Antonio, Texas) *(POPL '79)*. Association for Computing Machinery, New York, NY, USA, 93–100. https://doi.org/10.1145/567752.567761

[27] Ragunathan Rajkumar, Insup Lee, Lui Sha, and John Stankovic. 2010. Cyber-physical systems: the next computing revolution. In *Design automation conference*. IEEE, 731–736.

[28] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. 2014. Model predictive control with signal temporal logic specifications. In *53rd IEEE Conference on Decision and Control*. IEEE, 81–87.

[29] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. 2015. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th international conference on hybrid systems: Computation and control*. 239–248.

[30] Hendrik Roehm, Jens Oehlerking, Thomas Heinz, and Matthias Althoff. 2016. STL Model Checking of Continuous and Hybrid Systems. In *Automated Technology for Verification and Analysis*, Cyrille Artho, Axel Legay, and Doron Peled (Eds.). Springer International Publishing, Cham, 412–427.

[31] Andrew Sogokon and Paul B Jackson. 2015. Direct formal verification of liveness properties in continuous and hybrid dynamical systems. In *International Symposium on Formal Methods*. Springer, 514–531.

[32] Andrew Sogokon, Paul B Jackson, and Taylor T Johnson. 2017. Verifying safety and persistence properties of hybrid systems using flowpipes and continuous invariants. In *NASA Formal Methods Symposium*. Springer, 194–211.

[33] Yong Kiam Tan and André Platzer. 2019. An Axiomatic Approach to Liveness for Differential Equations. In *FM (LNCS, Vol. 11800)*, Maurice ter Beek, Annabelle McIver, and José N. Oliviera (Eds.). Springer, 371–388. https://doi.org/10.1007/978-3-030-30942-8_23