dTL^2: Differential Temporal Dynamic Logic with Nested Temporalities for Hybrid Systems

Jean-Baptiste Jeannin and André Platzer

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

Abstract. The differential temporal dynamic logic dTL^2 is a logic to specify temporal properties of hybrid systems. It combines differential dynamic logic with temporal logic to reason about the intermediate states reached by a hybrid system. The logic dTL^2 supports some linear time temporal properties of LTL. It extends differential temporal dynamic logic dTL with nested temporalities. We provide a semantics and a proof system for the logic dTL^2, and show its usefulness for nontrivial temporal properties of hybrid systems. We take particular care to handle the case of alternating universal dynamic and existential temporal modalities and its dual, solving an open problem formulated in previous work.

1 Introduction

A major task of computer science is to program objects of our physical world: cars, trains, airplanes, robots, etc. — often grouped under the denomination of cyber-physical systems (CPS). A CPS is governed by its programmable controllers, but also by the laws of physics. To fully verify it, one thus needs to model the controllers and their software as well as the relevant laws of physics in the same system. Such a system then becomes hybrid: the controllers are *discrete* while the laws of physics are *continuous*.

In recent years, a number of systems have been explored to reason about such *hybrid systems*. In particular, this paper is based on *differential dynamic logic* [14], [16, chapter 4], a logic based on dynamic logic [15,17], [16, chapter 2] and including programs enabling discrete assignments and discrete control structures, but also execution of differential equations. Differential dynamic logic comes with a semantics as well as a proof system, which is sound and relatively complete.

Based on dynamic logic, differential dynamic logic only reasons about the end state of a system. However, to ensure that a system always stays within some structural limits, or always accomplishes a certain task, one needs to reason about its intermediate states as well. CPSs that are safe when their systems terminate but have been unsafe in the middle of the program run are still not safe to use. The idea is to use both dynamic logic — to quantify over possible executions — and temporal logic — to quantify over the states in the trace of each execution. This is not a new idea, but previous work [1,14] focuses only on the non-alternating cases: "some property is always verified during all executions" and "something happens during some execution."

In this paper, we are developing a *differential temporal dynamic logic* dTL^2 inspired from LTL, and we are focusing on correctly handling the more complex alternating cases: "something happens during all executions" and "there is an execution where some property is always verified," as well as nested temporal modalities. In particular, a property checking that a task is always accomplished can now be checked. This logic is an important stepping stone towards full dTL*, the differential analog of CTL*.

As a simple example, let us look at a satellite with position x trying to leave the solar system, avoiding planets. To simplify, let us consider only two planets with radiuses r_1 and r_2, at (evolving) positions p_1 and p_2. The satellite can be controlled either by a pilot who can set its steering ω to left or right then let x evolve according to differential equation $\mathsf{flight}(\omega)$, or by an autopilot following a PID controller with target direction set to d. During each evolution, the positions of the planets continue to evolve, following differential equation $\mathsf{planets}(p_1, p_2)$. The program of the satellite and its safety property ϕ — expressing that there exists a steering avoiding all planets — can be expressed as:

$$\begin{aligned}\mathsf{satellite} ::=& (((\omega := \mathsf{left} \cup \omega := \mathsf{right}); x' = \mathsf{flight}(\omega), (p_1', p_2') = \mathsf{planets}(p_1, p_2))\\ &\cup (d := *; x' = \mathsf{PID}(d), (p_1', p_2') = \mathsf{planets}(p_1, p_2)))^*;\\ &\mathsf{control} := \mathsf{lost}; d := *; x' = \mathsf{PID}(d), (p_1', p_2') = \mathsf{planets}(p_1, p_2)\\ \phi ::=& \langle\mathsf{satellite}\rangle\Box(\mathsf{dist}(x, p_1) > r_1 \land \mathsf{dist}(x, p_2) > r_2 \land \mathsf{control} \neq \mathsf{lost})\end{aligned}$$

This example shows several features of hybrid programs and the logic dTL^2. Under the pilot's command, the variable ω can be *assigned* to either left or right, following a *nondeterministic choice* $\cup$. Then x, p_1 and p_2 follow a *differential equation* modeling the continuous evolution of the system, including movement of the planets. Under the autopilot's command, d is *nondeterministically assigned* $(d := *)$. There is a nondeterministic choice between the two commands, followed by a star * representing *repetition*. In case of mechanical or communication failure, control could be lost, which we represent by a variable assignment, and the system continues to evolve. The formula ϕ says that there exists a possible evolution ($\langle\mathsf{satellite}\rangle$) such that throughout this evolution ($\Box$), the satellite does not hit any planet; namely, the evolution avoiding planets where control is never lost. The formula ϕ is expressible in dTL^2, and shows how dTL^2 handles alternating and nested program ($\langle\mathsf{satellite}\rangle$) and temporal modalities ($\Box$ and $\Diamond$). The focus of this paper is to create a semantics and a proof calculus for dTL^2.

There are three main contributions to this paper. First, we show how to correctly handle the alternating cases of a universal dynamic modality followed by an existential temporal modality, and its dual an existential dynamic modality followed by a universal temporal modality. This solves an open problem identified in 2001 [1] and identified as a problem for hybrid systems in 2007 [14], [16, chapter 4]. Second, we offer a treatment where programs are not duplicated by proof rules, solving another open problem formulated in [14], [16, chapter 4]. This is significant for proving hybrid systems in practice, because previous approaches led to a duplication of proof effort, once for intermediate and once for final states. Third and finally, we extend the logic to nested temporal quantifiers, show that all formulas of interest are equivalent to formulas containing at most two quantifiers — thus the name dTL^2 — by identifying the resemblance to modal system S4.2, and develop a logic and proof calculus for the new temporal formulas.

The paper is organized as follows. After presenting the syntax and semantics of Differential Temporal Dynamic Logic dTL^2 in Section 2, we show how to normalize trace formulas and how to axiomatize dTL^2 in Section 3. We study alternative proof systems in Section 4 and related work in Section 5, before concluding in Section 6.

2 Differential Temporal Dynamic Logic dTL^2

This section defines the syntax and semantics of hybrid programs and trace formulas formally. The development mostly follows and extends previous work on differential temporal dynamic logic [14], [16, chapter 4]; we explicitly point out differences and extensions from the previous work.

2.1 Hybrid Programs

We use *hybrid programs* (HP) [15,17], [16, chapter 2] α, β to model hybrid systems. Syntactically, hybrid programs can be *atomic* hybrid programs or *compound* hybrid programs. Atomic hybrid programs can be discrete jump *assignments* ($x := \theta$), *tests* ($?\chi$) and *differential equations* evolving within an evolution domain constraint χ — meaning that the system can evolve following a solution of the differential equation as long as χ remains true ($x' = \theta \ \&\ \chi$). Terms θ are polynomials with rational coefficients, and conditions χ are first-order formulas of real arithmetic.[1] Compound hybrid programs are *nondeterministic choice* ($\alpha \cup \beta$), *sequential composition* ($\alpha; \beta$) and *nondeterministic finite repetition* (α^*):

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

The *trace semantics* of hybrid programs assigns to each program α a set of *traces* $\tau(\alpha)$. The set of *states* Sta is the set of (total) functions from variables to the reals $\mathbb{R}$. In addition, we consider a separate state Λ (not in Sta) denoting a failure of the system. For $v \in \mathsf{Sta}$ or $v = \Lambda$, we denote by $\hat{v}$ the function $\sigma : \{0\} \to \{v\}, 0 \mapsto v$, defined only on the singleton interval $[0, 0]$. A *trace* is a (nonempty) finite sequence $\sigma = (\sigma_0, \sigma_1, ..., \sigma_n)$ of functions σ_i. For $0 \leq i < n$, the piece σ_i is a function $\sigma_i : [0, r_i] \to \mathsf{Sta}$, where $r_i \geq 0$ is the duration of this step. For $i = n$, the piece σ_n is either a function:

- $\sigma_n : [0, r_i] \to \mathsf{Sta}$; we then say that σ is a *terminating* trace; or
- $\sigma_n : [0, +\infty) \to \mathsf{Sta}$; we then say that σ is an *infinite* trace; or
- $\sigma_n : \{0\} \to \{\Lambda\}, 0 \mapsto \Lambda$, for $n \geq 1$;[2] we then say that σ is an *error* trace.

We often collectively refer to infinite and error traces as *nonterminating*; thus when we refer to terminating traces, we only refer to those traces that terminate but not with an error state Λ. We write Tra for the set of all traces. A *position* of σ is a pair (i, ζ) with $0 \leq i \leq n$ and ζ in the domain of definition of σ_i; the state of σ at (i, ζ) is $\sigma_i(\zeta)$. For any trace σ, we denote by $\mathsf{first}\ \sigma$ the state $\sigma_0(0)$; we informally say that "σ starts with v" to say that $v = \mathsf{first}\ \sigma$. If $\sigma = (\sigma_0, \ldots, \sigma_n)$ terminates (and only in that case), we also denote by $\mathsf{last}\ \sigma$ the state $\sigma_n(r_n)$; when σ does not terminate, $\mathsf{last}\ \sigma$ is undefined. We denote by $\mathsf{val}(v, \theta)$ the value of term θ in state v, and by $v[x \mapsto r]$ the valuation assigning variable x to $r \in \mathbb{R}$ and matching with v on all other variables. We also write $v \vDash \chi$ if state v satisfies condition χ, and $v \nvDash \chi$ otherwise.

[1] using first-order formulas or real arithmetic results in a poor-test version of the logic. Our results generalize to a rich-test version, where a condition χ is instead defined as any formula ϕ of dTL^2 (see Section 2.2).

[2] We impose $n \geq 1$ so that $(\hat{\Lambda})$ is not considered a trace

Given two traces $\sigma = (\sigma_0, \dots, \sigma_n)$ and $\rho = (\rho_0, \dots, \rho_m)$, we say that ρ is a *prefix* of σ if it describes the trace σ truncated at some position. Formally, ρ is a prefix of σ if and only if $\rho = \sigma$ — a condition ensuring that nonterminating traces are also suffixes of themselves — or there exists a position (i, ζ) of σ such that:

– traces $(\sigma_0, \dots, \sigma_{i-1})$ and $(\rho_0, \dots, \rho_{m-1})$ are identical.[3] In particular this imposes that $i = m$; and
– the domain of definition of ρ_m is exactly $[0, \zeta]$ and is included in the domain of definition of σ_m, and for all $d \in [0, \zeta]$, $\sigma_m(d) = \rho_m(d)$.

Symmetrically, we say that ρ is a *suffix* of σ if it starts at some position of σ then follows σ. Formally, ρ is a suffix of σ if and only if there exists a position (i, ζ) of σ such that:

– if σ_i has domain of definition $[0, r_i]$, then the domain of definition of ρ_0 is exactly $[0, r_i - \zeta]$ and for all $d \in [\zeta, r_i]$, $\sigma_i(d) = \rho_0(d - \zeta)$; and in the case where σ_i has domain of definition $[0, +\infty)$, the domain of definition of ρ_0 is also $[0, +\infty)$ and for all $d \in [\zeta, +\infty)$, $\sigma_i(d) = \rho_0(d - \zeta)$; and
– $(\sigma_{i+1}, \dots, \sigma_n)$ and $(\rho_1, \dots, \rho_m)$ are identical, which imposes that $n - i = m$.

Definition 1 (Trace Semantics of Hybrid Programs). *The* trace semantics $\tau(\alpha) \subseteq 2^{\mathsf{Tra}}$ *of a hybrid program α is then defined inductively as follows:*

– $\tau(x := \theta) = \{(\hat{v}, \hat{w}) \mid w = v[x \mapsto \mathsf{val}(v, \theta)]\}$*;*
– $\tau(x' = \theta \;\&\; \chi) = \{(\sigma) \;:\;$ *σ is a state flow of order 1 [15] defined on* $[0, r]$ *or* $[0, +\infty)$ *solution of $x' = \theta$, and for all t in its domain of definition, $\sigma(t) \vDash \chi\}$* $\cup \{(\hat{v}, \hat{\Lambda}) \;:\; v \nvDash \chi\}$*;*[4]
– $\tau(?\chi) = \{(\hat{v}) \;:\; v \vDash \chi\} \cup \{(\hat{v}, \hat{\Lambda}) \;:\; v \nvDash \chi\}$*;*
– $\tau(\alpha \cup \beta) = \tau(\alpha) \cup \tau(\beta)$*;*
– $\tau(\alpha; \beta) = \{\sigma \circ \rho \;:\; \sigma \in \tau(\alpha), \rho \in \tau(\beta)$ *when $\sigma \circ \rho$ is defined$\}$;*
where the composition $\sigma \circ \rho$ of $\sigma = (\sigma_0, \dots, \sigma_n)$ and $\rho = (\rho_0, \dots, \rho_m)$ is
  • $\sigma \circ \rho = (\sigma_0, \dots, \sigma_n, \rho_0, \dots, \rho_m)$ *if σ terminates and* $\mathsf{last}\ \sigma = \mathsf{first}\ \rho$ *(since σ terminates,* $\mathsf{last}\ \sigma$ *is well-defined);*
  • *σ if σ does not terminate;*
  • *undefined otherwise;*
– $\tau(\alpha^*) = \bigcup_{n \in \mathbb{N}} \tau(\alpha^n)$*, where α^0 is defined as* $?\mathsf{true}$*, α^1 is defined as α and α^{n+1} is defined as $\alpha^n; \alpha$ for $n \geq 1$.*

An important property of this trace semantics is that for all programs α and states v, there exists a trace σ of α starting with v (even if it might be an error trace). This property will be key to proving the soundness of assignment rules.

Aside from the correction on $\tau(x' = \theta \;\&\; \chi)$, this definition is slightly different from [14], [16, chapter 4] in two ways: these previous papers also consider infinite sequences $\sigma = (\sigma_0, \sigma_1, \dots)$, but infinite sequences are not part of the semantics of any program; and these papers do not consider infinite traces in the semantics. Still, we can prove that the interpretation of trace formulas (Section 2.2) is the same on the subset of trace formulas they consider.

[3] if $i = m = 0$, $(\sigma_0, \dots, \sigma_{i-1})$ and $(\rho_0, \dots, \rho_{m-1})$ are empty and thus not formally traces, but we still consider the condition fulfilled.

[4] this case is corrected from [14], [16, chapter 4], which wrongly forget the error traces of ordinary differential equations — when χ is initially false.

2.2 State and Trace Formulas

To reason about hybrid programs, we use *state formulas* and *trace formulas*. State formulas express properties about states, while trace formulas express properties about traces; their definitions are mutually inductive. A state formula ϕ, ψ can be a *comparison of terms* ($\theta_1 \geq \theta_2$); a *negation* of a state formula ($\neg\phi$); a *conjunction* ($\phi \wedge \psi$) or a *disjunction* ($\phi \vee \psi$) of state formulas; a *universally quantified* ($\forall x\ \phi$) or *existentially quantified* ($\exists x\ \phi$) state formula — quantification of a variable x is over the set of reals $\mathbb{R}$. Finally, a state formula can also be a *program necessity* ($[\alpha]\pi$) — expressing that all traces of hybrid program α starting at the current state satisfy trace formula π — or its dual, a *program possibility* ($\langle\alpha\rangle\pi$) — expressing that there is a trace of α starting at the current state satisfying trace formula π.

A trace formula π can be a *state formula* (ϕ); a *negation* of a trace formula ($\neg\pi$); a *temporal necessity* of a trace formula ($\Box\pi$) — expressing that every suffix of the current trace satisfies π — or its dual, a *temporal possibility* of a trace formula ($\Diamond\pi$) — expressing that there is a suffix of the current trace satisfying π. The syntax of state and trace formulas is thus given by:

$$\begin{aligned}\phi, \psi &::= \theta_1 \geq \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \forall x\ \phi \mid \exists x\ \phi \mid [\alpha]\pi \mid \langle\alpha\rangle\pi \\ \pi &::= \phi \mid \neg\pi \mid \Box\pi \mid \Diamond\pi\end{aligned}$$

Additionally, as in classical logic, the implication $\phi \rightarrow \psi$ is defined as $\neg\phi \vee \psi$. When a trace formula also happens to be a state formula ϕ, the formula $\neg\phi$ means the same whether it is seen as a state or trace formula; in the rest of the paper we collude the two. We are now ready to define satisfaction of state and trace formulas.

Definition 2 (Satisfaction of dTL^2 Formulas). *For state formulas, we write $v \vDash \phi$ to say that state $v \in$ Sta satisfies state formula ϕ. Satisfaction of state formulas with respect to a state v is defined inductively as follows:*

- $v \vDash \theta_1 \geq \theta_2$ *if and only if* $\mathsf{val}(v, \theta_1) \geq \mathsf{val}(v, \theta_2)$
- $v \vDash \neg\phi$ *if and only if* $v \vDash \phi$ *does not hold.*
- $v \vDash \phi \wedge \psi$ *if and only if* $v \vDash \phi$ *and* $v \vDash \psi$.
- $v \vDash \phi \vee \psi$ *if and only if* $v \vDash \phi$ *or* $v \vDash \psi$.
- $v \vDash \forall x\, \phi$ *if and only if* $v[x \mapsto d] \vDash \phi$ *holds for all* $d \in \mathbb{R}$.
- $v \vDash \exists x\, \phi$ *if and only if* $v[x \mapsto d] \vDash \phi$ *holds for some* $d \in \mathbb{R}$.
- *for ϕ a state formula,* $v \vDash [\alpha]\phi$ *if and only if for each trace* $\sigma \in \tau(\alpha)$ *that starts in* first $\sigma = v$, *if* σ *terminates, then* last $\sigma \vDash \phi$.
- *for ϕ a state formula,* $v \vDash \langle\alpha\rangle\phi$ *if and only if there is a trace* $\sigma \in \tau(\alpha)$ *starting in* first $\sigma = v$ *such that* σ *terminates and* last $\sigma \vDash \phi$.
- *If π is not a state formula,* $v \vDash [\alpha]\pi$ *if and only* $\sigma \vDash \pi$ *for each trace* $\sigma \in \tau(\alpha)$ *that starts in* first $\sigma = v$.
- *If π is not a state formula,* $v \vDash \langle\alpha\rangle\pi$ *if and only* $\sigma \vDash \pi$ *for some trace* $\sigma \in \tau(\alpha)$ *that starts in* first $\sigma = v$.

For trace formulas, we write $\sigma \vDash \pi$ to say that trace $\sigma \in$ Tra satisfies trace formula π. Satisfaction of trace formulas with respect to a trace σ is defined inductively as follows:

- $\sigma \vDash \phi$ *if and only if* $\mathsf{first}\ \sigma \vDash \phi$.
- $\sigma \vDash \neg\pi$ *if and only if* $\sigma \vDash \pi$ *does not hold.*
- $\sigma \vDash \Box\pi$ *if and only if* $\rho \vDash \pi$ *holds for all suffixes* ρ *of* σ *that are different from* $(\hat{\Lambda})$.
- $\sigma \vDash \Diamond\pi$ *if and only if* $\rho \vDash \pi$ *holds for some suffix* ρ *of* σ *that is different from* $(\hat{\Lambda})$.

This definition follows the intuition given when presenting the syntax of state and trace formulas, except for one point. Note that in the definitions of $\sigma \vDash \Box\pi$ and $\sigma \vDash \Diamond\pi$, the suffix ρ of σ does not have to be proper, and we can have $\rho = \sigma$. When seen as a trace formula, a state formula ϕ can express a property on a trace σ. We then say that σ satisfies ϕ if and only if the first state of σ satisfies ϕ (condition $\mathsf{first}\ \sigma \vDash \phi$ in the definition of $\sigma \vDash \phi$). However, there is an exception to this definition: when ϕ appears directly after a program necessity (as in $[\alpha]\phi$) or a program possibility (as in $\langle\alpha\rangle\phi$), ϕ only refers to *terminating* traces, and we say that σ satisfies ϕ if and only if the *last* state of σ satisfies ϕ (condition $\sigma \vDash \mathsf{last}\ \phi$ in the definitions of $\sigma \vDash \langle\alpha\rangle\phi$ and $\sigma \vDash [\alpha]\phi$). This discontinuity in the definition of the satisfaction of ϕ enables following both the usual semantics of dynamic logic and of temporal logic, and was also adopted in previous work [7,14], [16, chapter 4]. It is also useful for proof rules as temporal properties often reduce to what happens after a program.

The syntax of dTL^2 formulas extends the syntax of trace formulas given in [14], [16, chapter 4] by allowing nesting of temporal modalities, and otherwise agrees with it. The satisfaction of dTL^2 formulas given in Def. 2, although presented in a slightly different way, agrees with the definitions given in [14], [16, chapter 4] on trace formulas without nested temporal modalities.

3 Proof Calculus

3.1 Equivalence of Trace Formulas

Trace formulas follow the axioms of modal system S4.2 [9], therefore there are only four proper affirmative modalities $\Box\phi$, $\Diamond\phi$, $\Box\Diamond\phi$ or $\Diamond\Box\phi$. Intuitively, because formulas $\neg\Box\pi$ and $\Diamond\neg\pi$ are equivalent — in the sense that they are satisfied by the same traces — formulas can always be expressed in a way where only state formulas have negations. Similarly, formulas $\Box\pi$ and $\Box\Box\pi$ are equivalent, therefore a trace formula containing exclusively $\Box$ temporalities followed by a state formula ϕ is equivalent to $\Box\phi$. Moreover, a formula containing both $\Box$ and $\Diamond$ temporalities, finishing by a $\Diamond$ temporality followed by a state formula ϕ is equivalent to $\Box\Diamond\phi$. Similar properties are true for their duals. This is formalized by the following lemma, proved in [10].

Lemma 1 (Equivalence of Trace Formulas). *For any trace formula* π_1*, there exists a trace formula* π_2 *of the form* ϕ*,* $\Box\phi$*,* $\Diamond\phi$*,* $\Box\Diamond\phi$ *or* $\Diamond\Box\phi$ *such that* $\sigma \vDash \pi_1$ *if and only if* $\sigma \vDash \pi_2$*. Such a* π_2 *can be computed from* π_1 *in linear time in the number of temporal modalities and negations in* π_1*.*

Remark 1. Lemma 1 tells us that the only interesting trace formulas of our system are those of the form ϕ, $\Box\phi$, $\Diamond\phi$, $\Box\Diamond\phi$ and $\Diamond\Box\phi$. For any trace σ, the intuitive meaning of $\sigma \vDash \pi$ for π of the form ϕ, $\Box\phi$ or $\Diamond\phi$ is clear: we have $\sigma \vDash \phi$ if and only if σ

starts in a state satisfying ϕ; we have $\sigma \vDash \Box\phi$ if and only if all non-error states of the trace σ satisfy ϕ; and we have $\sigma \vDash \Diamond\phi$ if and only if there is a non-error state of trace σ satisfying ϕ. When π is of the form $\Box\Diamond\phi$ and $\Diamond\Box\phi$, we get a better intuition by distinguishing cases:

- if σ is a terminating trace, $\sigma \vDash \Diamond\Box\phi$ if and only if $\mathsf{last}\ \sigma \vDash \phi$, and $\sigma \vDash \Box\Diamond\phi$ if and only if $\mathsf{last}\ \sigma \vDash \phi$ as well;
- if σ is an error trace, σ can be written $(\sigma_0, \ldots, \sigma_{n-1}, \hat{\Lambda})$. Let $\rho = (\sigma_0, \ldots, \sigma_{n-1})$, then ρ is a terminating trace and a prefix of σ. Moreover, both $\sigma \vDash \Diamond\Box\phi$ and $\sigma \vDash \Box\Diamond\phi$ are equivalent to $\mathsf{last}\ \rho \vDash \phi$;
- if σ is an infinite trace, $\sigma \vDash \Diamond\Box\phi$ holds if and only if ϕ holds on all states of σ after some position, and $\sigma \vDash \Box\Diamond\phi$ holds if and only if any state of σ has a later state satisfying ϕ (if we did not have continuous dynamics, this would be the same as ϕ being true infinitely often along σ; but here it is not sufficient).

3.2 Normalization of Trace Formulas

The primary goal of this paper is to establish a proof system for differential temporal dynamic logic dTL^2. As for d$\mathcal{L}$ and dTL, rules typically decompose programs syntactically. Let us look at the state formula $\langle\alpha;\beta\rangle\Box\phi$, and to simplify, let us only consider terminating traces for now. Intuitively, this formula says that there exists a trace in $\tau(\alpha;\beta)$ throughout which ϕ holds. Considering only terminating traces, this is true as long as there exists a trace σ of α throughout which ϕ is true, and a trace ρ of β starting at $\mathsf{last}\ \sigma$ throughout which ϕ is also true. It is thus tempting to write the following rule:

$$\frac{\langle\alpha\rangle\Box\phi \wedge \langle\alpha\rangle\langle\beta\rangle\Box\phi}{\langle\alpha;\beta\rangle\Box\phi}\ \text{(unsound)}$$

This rule is unsound because α is possibly nondeterministic. Its premise says that there is a trace σ of α throughout which ϕ is true, and a trace σ' of α followed by a trace ρ of β throughout which ϕ is true. But σ and σ' do not have to be the same trace; the trick is that ϕ is not necessarily true throughout σ'. To fix this rule, we need to express that traces σ and σ' are the same, thus writing a premise resembling:

$$\langle\alpha\rangle(\Box\phi \wedge \langle\beta\rangle\Box\phi) \tag{1}$$

Unfortunately, this is not directly expressible with dTL^2, without using the program $\alpha;\beta$ again: the missing piece is the expressibility of a conjunction on traces that simultaneously talks about temporal properties like $\Box\phi$ and properties true at the end of the trace. To achieve this expressibility, we extend the logic with *normalized trace formulas* to make conjunction of temporal formulas expressible as needed in (1).

A normalized trace formula ξ can be of different forms: for terminating traces, the formula $\phi\sqcap\Box\psi$ captures the conjunction of ending in a state satisfying ϕ, and satisfying $\Box\psi$; and the formula $\phi \sqcup \Diamond\psi$ captures the disjunction of ending in a state satisfying ϕ, or satisfying $\Diamond\psi$. For nonterminating traces, $\phi \sqcap \Box\psi$ is the same as $\Box\psi$, and $\phi \sqcup \Diamond\psi$ is the same as $\Diamond\psi$, because there is no terminal state in which it makes sense to evaluate ϕ. Additionally, the formula $\phi\blacktriangleleft\Box\Diamond\psi$ captures ending in a state satisfying ϕ if terminating, and satisfying $\Box\Diamond\psi$ otherwise; and similarly, the formula $\phi \blacktriangleleft \Diamond\Box\psi$ captures ending in a state satisfying ϕ if terminating, and satisfying $\Diamond\Box\psi$ otherwise.

Formulas $\phi \blacktriangleleft \Diamond\Box\psi$ and $\phi \blacktriangleleft \Box\Diamond\psi$ play the same role for formulas $\Diamond\Box\psi$ and $\Box\Diamond\psi$ as formulas $\phi \sqcap \Box\psi$ and $\phi \sqcup \Diamond\psi$ play for formulas $\Box\psi$ and $\Diamond\psi$: they allow us to define premises of modular inference rules for sequential composition as in (1). Like standard trace formulas, normalized trace formulas can appear after a program necessity $[\alpha]$ or a program possibility $\langle\alpha\rangle$. We therefore extend state formulas to accept normalized trace formulas, and define normalized trace formulas as:

$$\begin{aligned} \phi,\psi &::= \dots \mid [\alpha]\xi \mid \langle\alpha\rangle\xi \\ \xi &::= \phi \sqcap \Box\psi \mid \phi \sqcup \Diamond\psi \mid \phi \blacktriangleleft \Box\Diamond\psi \mid \phi \blacktriangleleft \Diamond\Box\psi \end{aligned}$$

Sometimes we will also use the notation $\phi \blacktriangleleft \pi$, with the understanding that in such cases π can only be of the form $\Box\Diamond\psi$ or $\Diamond\Box\psi$.

Coming back to our example, a sound rule for $\langle\alpha;\beta\rangle\Box\phi$ can be expressed as:

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

In the form of its dual $[;]\Diamond$, this rule will be discussed later and proved sound in [10]. Observe how $\langle;\rangle\Box$ does not even duplicate α and β.

Extending Def. 2, the satisfaction of trace formulas $[\alpha]\xi$ and $\langle\alpha\rangle\xi$ is defined in the same way as trace formulas $[\alpha]\pi$ and $\langle\alpha\rangle\pi$ (if π is not a state formula):

- $v \vDash [\alpha]\xi$ if and only $\sigma \vDash \xi$ for each trace $\sigma \in \tau(\alpha)$ that starts in first $\sigma = v$.
- $v \vDash \langle\alpha\rangle\xi$ if and only $\sigma \vDash \xi$ for some trace $\sigma \in \tau(\alpha)$ that starts in first $\sigma = v$.

Satisfaction of normalized trace formulas carefully distinguishes between terminating and nonterminating traces, and is defined as follows.

Definition 3 (Semantics of Normalized dTL^2 Trace Formulas). *For normalized trace formulas, we write $\sigma \vDash \xi$ to say that trace σ satisfies normalized state formula ξ. Satisfaction of normalized trace formulas with respect to a trace σ is defined inductively:*

$$\begin{aligned}
\sigma \vDash \phi \sqcup \Diamond\psi \ \textit{if and only if} &\begin{cases} \mathsf{last}\ \sigma \vDash \phi \ \textit{or}\ \sigma \vDash \Diamond\psi & \textit{if } \sigma \textit{ terminates} \\ \sigma \vDash \Diamond\psi & \textit{otherwise} \end{cases} \\
\sigma \vDash \phi \sqcap \Box\psi \ \textit{if and only if} &\begin{cases} \mathsf{last}\ \sigma \vDash \phi \ \textit{and}\ \sigma \vDash \Box\psi & \textit{if } \sigma \textit{ terminates} \\ \sigma \vDash \Box\psi & \textit{otherwise} \end{cases} \\
\sigma \vDash \phi \blacktriangleleft \pi \quad \textit{if and only if} &\begin{cases} \mathsf{last}\ \sigma \vDash \phi & \textit{if } \sigma \textit{ terminates} \\ \sigma \vDash \pi & \textit{otherwise} \end{cases}
\end{aligned}$$

Not only can normalized trace formulas help express rules like $\langle;\rangle\Box$, they can also, along with state formulas, express all possible trace formulas. In Lemma 1, we have shown how to express any trace formula in the form ϕ, $\Box\phi$, $\Diamond\phi$, $\Box\Diamond\phi$ or $\Diamond\Box\phi$. Building on this result, we now show how to *normalize* every trace formula into a state formula or a normalized trace formula. To this effect, we define a relation $\leadsto$ between the set of state formulas and trace formulas, and the set of state formulas and normalized trace formulas. This simplifies the axiomatization of dTL^2 by allowing us to only consider cases containing normalized trace formulas.

$$\Box\phi \rightsquigarrow \mathsf{true} \sqcap \Box\phi \;(\rightsquigarrow\sqcap) \qquad \Diamond\phi \rightsquigarrow \mathsf{false} \sqcup \Diamond\phi \;(\rightsquigarrow\sqcup)$$

$$\Box\Diamond\phi \rightsquigarrow \phi \blacktriangleleft \Box\Diamond\phi \;(\rightsquigarrow\blacktriangleleft\Box) \qquad \Diamond\Box\phi \rightsquigarrow \phi \blacktriangleleft \Diamond\Box\phi \;(\rightsquigarrow\blacktriangleleft\Diamond)$$

$$\phi \rightsquigarrow \phi \;(\rightsquigarrow\phi) \qquad \frac{\pi_1 \sim \pi_2 \qquad \pi_2 \rightsquigarrow \xi}{\pi_1 \rightsquigarrow \xi}\;(\sim\rightsquigarrow)$$

Fig. 1. Normalization rules for trace formulas

The normalization is sound, meaning that two related formulas are satisfied by the same trace. Additionally, every trace formula is related to either a state formula or a normalized trace formula, which can be found in linear time.

Lemma 2 (Soundness of Normalization). *If $\pi \rightsquigarrow \xi$ then for all traces σ, $\sigma \vDash \pi$ if and only if $\sigma \vDash \xi$.*

Proof. Soundness of $\rightsquigarrow \phi$ is trivial. Soundness of proof rules $\rightsquigarrow \sqcap$, $\rightsquigarrow \sqcup$, $\rightsquigarrow \blacktriangleleft\Box$ and $\rightsquigarrow \blacktriangleleft\Diamond$ is true by Def. 3, keeping in mind the intuition given in Remark 1. Soundness of proof rule $\sim\rightsquigarrow$ is by induction and using Lemma 1. □

Lemma 3 (Existence of a Normalized Form). *For any trace formula π there exists a state formula ϕ such that $\pi \rightsquigarrow \phi$, or a normalized trace formula ξ such that $\pi \rightsquigarrow \xi$. Such a ϕ or ξ can be computed from π in linear time.*

Proof. This lemma is a direct consequence of Lemma 1, using the identities of Fig. 1. Unless π is itself a state formula ϕ, it is related to a normalized trace formula ξ. □

Lemma 3 concludes our study of normalized forms. Since every trace formula is related (and thus semantically equivalent by Lemma 2) to a state formula or a normalized trace formula, we can limit our axiomatization to the study of state formulas and normalized trace formulas. Formulas of the form $[\alpha]\phi$ or $\langle\alpha\rangle\phi$ involving state formulas have already been axiomatized in d$\mathcal{L}$ [15,17], [16, chapter 2]. The rest of this paper focuses on axiomatizing formulas of the form $[\alpha]\xi$ or $\langle\alpha\rangle\xi$ involving normalized trace formulas. In [10], we come back to trace formulas to study a direct treatment of proof rules for state formulas of the form $[\alpha]\pi$ and $\langle\alpha\rangle\pi$ in order to make the system more efficient.

3.3 Proof Calculus for dTL^2

In this section we present a proof calculus for dTL^2 for verifying temporal properties of hybrid programs specified in the differential temporal dynamic logic dTL^2. The basic idea of the proof calculus is symbolic decomposition. The calculus progressively transforms formulas to simpler formulas, often by inductively decomposing programs that are in program modalities. In particular, the temporal rules progressively transform temporal formulas to temporal-free formulas, in order to leverage the nontemporal rules of d$\mathcal{L}$. The proof system inherits its nontemporal rules from the d$\mathcal{L}$ proof system [15,17], [16, chapter 2], and adds its own temporal rules. As is the case for d$\mathcal{L}$, the basis of our proof system is real arithmetic, and we integrate it as in d$\mathcal{L}$ [15,17], [16, chapter 2]. We first present how to use the rules, then a brief overview on the inherited nontemporal rules from d$\mathcal{L}$, and finally a detailed account of the new temporal rules of dTL^2, summarized in Fig. 2.

Usage of the Rules. Rules are to be used in the same way as in the $\mathsf{d}\mathcal{L}$ calculus. We do, however, use a new double bar notation by writing some rules in the form

$$\frac{\phi}{\overline{\psi}}$$

This notation denotes equivalence of the premise and its conclusion. This means that there exists a dual rule, hence the two following rules are true

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

For space reasons we do not list dual rules explicitly but give them in [10].

Inherited Nontemporal Rules. On top of the temporal rules presented in Fig. 2, the proof calculus of dTL^2 also inherits the rules of the proof calculus of $\mathsf{d}\mathcal{L}$. Since the semantics of dTL^2 conservatively extends the semantics of dTL, which itself conservatively extends the semantics of $\mathsf{d}\mathcal{L}$ [14], [16, chapter 4], it is sound to inherit the $\mathsf{d}\mathcal{L}$ calculus. While we inherit the nontemporal rules of $\mathsf{d}\mathcal{L}$, we do not inherit — but rather reformulate with normalized trace formulas — the temporal rules of dTL [14], [16, chapter 4], thus enabling more efficient proofs by exploiting normalized trace formulas.

Temporal Rules. The temporal rules of the proof calculus of dTL^2 are presented in Fig. 2, in which they are grouped by program construct. Rules $[\,]\rightsquigarrow$ and $\langle\,\rangle\rightsquigarrow$ lift trace formula normalization to program modalities. Rule $[\cup]\xi$ for nondeterministic choice easily extends corresponding rule $[\cup]$ of $\mathsf{d}\mathcal{L}$, and assignment rules behave as expected, largely because assignments always terminate.

The sequential composition rules exhibit how nicely the normalized formula interact with sequential composition; remember that sequential composition is one of the main technical difficulties of a calculus handling alternating program and temporal modalities. Normalized trace formulas were designed for these rules, and particular care was taken in considering nonterminating traces. Rule $[;]\sqcap$ expresses that all traces of the composition of two programs α and β satisfies $\phi \sqcap \Box\psi$ if and only if all traces of α satisfy $\Box\psi$, and for terminating traces of α, if all following traces of β satisfy $\phi \sqcap \Box\psi$. In particular, this rule improves on the corresponding rule $[;]\Box$ of dTL by *not* duplicating program modality $[\beta]$, thus eliminating proofs that are exponential in the number of sequential compositions. Rule $[;]\sqcup$ is the main rule for alternating program and temporal modalities in the context of sequential composition. It expresses that all traces of the composition of two programs α and β satisfies $\phi \sqcap \Diamond\psi$ if and only if all traces of α either satisfy $\Diamond\psi$, or are terminating and followed only by traces of β satisfying $\phi \sqcup \Diamond\psi$. Finally, rule $[;]\blacktriangleleft$ similarly handles sequential compositions followed by a $\blacktriangleleft$ operator.

For the test rules, let us remember that a test trace terminates only if the test passes, and is otherwise an error trace. Any trace of test $?\chi$ satisfies $\phi \sqcap \Box\psi$ if and only if its initial state satisfies $\phi \wedge \psi$ when it terminates, or satisfies just ψ when it doesn't terminate; this can be summarized as $(\neg\chi \vee \phi) \wedge \psi$ as in rule $[?]\sqcap$. Rule $[?]\sqcup$ is similar. Any trace of test $?\chi$ satisfies $\phi \blacktriangleleft \Diamond\Box\psi$ if and only if it terminates and its initial state satisfied ϕ, or it doesn't terminate and its initial state satisfied ψ; this can be summarized as $(\chi \wedge \phi) \vee (\neg\chi \wedge \psi)$ as in rule $[?]\blacktriangleleft\Diamond$. Rule $[?]\blacktriangleleft\Box$ is similar.

Normalization of Trace Formulas

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

Sequential Composition

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

Nondeterministic Choice

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

Test

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

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

Assignment

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

Ordinary Differential Equation

$$\frac{\psi \wedge [x' = \theta \,\&\, \chi](\phi \wedge \psi)}{[x' = \theta \,\&\, \chi](\phi \sqcap \Box\psi)}\,([']\sqcap)$$

$$\frac{(\chi \vee \psi) \wedge [x' = \theta \,\&\, (\chi \wedge \neg\psi)]\phi \wedge \langle x' = \theta\rangle(\neg\chi \vee \psi)}{[x' = \theta \,\&\, \chi](\phi \sqcup \Diamond\psi)}\,([']\sqcup)$$

$$\frac{(\chi \vee \psi) \wedge [x' = \theta \,\&\, \chi]\phi \wedge (\langle x' = \theta\rangle(\neg\chi) \vee \langle x' = \theta\rangle[x' = \theta]\psi)}{[x' = \theta \,\&\, \chi](\phi \blacktriangleleft \Diamond\Box\psi)}\,([']\blacktriangleleft\Diamond)$$

$$\frac{(\chi \vee \psi) \wedge [x' = \theta \,\&\, \chi]\phi \wedge (\langle x' = \theta\rangle(\neg\chi) \vee [x' = \theta]\langle x' = \theta\rangle\psi)}{[x' = \theta \,\&\, \chi](\phi \blacktriangleleft \Box\Diamond\psi)}\,([']\blacktriangleleft\Box)$$

Repetition

$$\frac{\phi \wedge [\alpha^*][\alpha](\phi \sqcap \Box\psi)}{[\alpha^*](\phi \sqcap \Box\psi)}\,([^*]\sqcap) \qquad \frac{\psi \vee (\phi \wedge [\alpha;\alpha^*](\phi \sqcup \Diamond\psi))}{[\alpha^*](\phi \sqcup \Diamond\psi)}\,([^{*n}]\sqcup)$$

$$\frac{\forall^\alpha(\phi \rightarrow [\alpha](\phi \sqcup \Diamond\psi))}{\phi \rightarrow [\alpha^*](\phi \sqcup \Diamond\psi)}\,(\mathsf{ind}\sqcup) \qquad \frac{\phi \wedge [\alpha^*][\alpha](\phi \blacktriangleleft \pi)}{[\alpha^*](\phi \blacktriangleleft \pi)}\,([^*]\blacktriangleleft)$$

$$\frac{\forall^\alpha \forall r > 0\ (\varphi(r) \rightarrow \langle\alpha\rangle(\varphi(r-1) \sqcap \Box\psi))}{(\exists r\ \varphi(r)) \wedge \psi \rightarrow \langle\alpha^*\rangle((\exists r \le 0\ \varphi(r)) \sqcap \Box\psi)}\,(\mathsf{con}\sqcap)$$

Fig. 2. Rule schemata of the proof calculus for dTL^2

Ordinary differential equations have terminating traces, but also infinite and error traces. Additionally, the execution can exit a differential equation at any moment, even if the evolution constraint domain it still verified; thus formulas like $[x' = \theta \,\&\, \chi]\phi$ and $[x' = \theta \,\&\, \chi]\Box\phi$ are equivalent in a state satisfying χ. Rules for ordinary differential equations transform formulas into temporal-free formulas, on which the $\mathsf{d}\mathcal{L}$ proof calculus and in particular differential invariants can be used. In rule $[']\sqcap$, the first conjunct ψ is necessary to handle error traces, when χ is initially false. In rule $[']\sqcup$, the first conjunct $\chi \vee \psi$ expresses that the differential equation can evolve or has satisfied $\Diamond\psi$ initially. The second conjunct handles traces that never satisfy ψ and thus have to satisfy ϕ, and the third conjunct makes sure there is either no infinite trace ($\langle x' = \theta\rangle\neg\chi$), or that such an infinite trace satisfies $\Diamond\psi$ (condition $\langle x' = \theta\rangle\psi$, equivalent to $\langle x' = \theta\rangle\Diamond\psi$). The first conjunct of rule $[']\blacktriangleleft\Diamond$ again handles error traces as in rule $[']\sqcup$. The second conjunct ensures all terminating traces finish in a state satisfying ϕ, and its third conjunct handles infinite traces by making sure they don't exist ($\langle x' = \theta\rangle\neg\chi$) or that they satisfy $\Diamond\Box\psi$ (condition $\langle x' = \theta\rangle[x' = \theta]\psi$). Rule $[']\blacktriangleleft\Box$ is similar.

In some way, repetition rules are easier because as long as a repetition only repeats a terminating trace, it is itself terminating. Rules $[^*]\sqcap$ and $[^*]\blacktriangleleft$ are particularly satisfying because their premise no longer contains a temporal property of a loop, but only a non-temporal postcondition of a loop, which is thus provable by ordinary, non-temporal induction. Only the postcondition still has a temporal property but no more loops. That is, these rules reduce temporal properties of loops to nontemporal properties of loops, or more complicated temporal properties on a program without the loop. In rule $[^*]\sqcap$, the first disjunct expresses that $\Diamond\psi$ holds without repeating if ψ holds initially. The first conjunct ϕ of the second disjunct is necessary when α repeats zero times; while the second conjunct executes α any number of times n, then checks that the $(n+1)$-st execution of α also satisfies $\phi\sqcap\Box\psi$. The treatment of rule $[^*]\blacktriangleleft$ is similar. Rule $[^{*n}]$ is less satisfying because it leaves an α^* inside a program modality followed by a normalized trace formula. If ψ is true then the conclusion trivially holds; otherwise the rule relies on the fact that α^* is equivalent to $?\mathsf{true}\cup\alpha;\alpha^*$ and just unwinds the loop once. Program $\alpha;\alpha^*$ in the modality could as well be the equivalent $\alpha^*;\alpha$. The same thing is *not* true for rule $[^*]\sqcap$, where $[\alpha^*][\alpha](\phi\sqcap\Box\phi)$ ensures progress of the proof, while writing $[\alpha][\alpha^*](\phi\sqcap\Box\phi)$ would not. Rules $\mathsf{ind}\sqcup$ and $\mathsf{con}\sqcap$ extend induction (ind) and convergence (con) rules of $\mathsf{d}\mathcal{L}$ to normalized trace formulas. As in $\mathsf{d}\mathcal{L}$, they are not equivalences; and also as in $\mathsf{d}\mathcal{L}$, they use the notation $\forall^\alpha$, which quantifies over all variables possibly assigned by α in assignments or differential equations. Rule $\mathsf{ind}\sqcup$ shows that ϕ is inductive with exit clause $\Diamond\psi$, i.e., ϕ holds after all traces of α from any state where ϕ holds, except when exit condition ψ was true at some point during that trace. If ψ was true initially, rule $[^{*n}]$ applies instead. Rule $\mathsf{con}\sqcap$ proves that φ is a variant of some trace of α (i.e., its level r decreases) during which ψ always holds true. Then starting from some initial r (assumption of conclusion), an r for which $\varphi(r)$ holds will ultimately be ≤ 0 without having violated when repeating α^* often enough.

3.4 Meta-Results

Soundness. The following result shows that verification with the dTL^2 calculus always produces correct results about the temporal behavior of hybrid systems, i.e., the dTL^2 calculus presented in Fig. 2 is sound. Theorem 1 is proved in [10].

Theorem 1 (Soundness of dTL^2). *The dTL^2 calculus presented in Fig. 2 is sound, i.e., derivable state formulas are valid, i.e., valid in every state.*

Incompleteness of dTL^2. In [15,17], [16, chapter 2] it was shown that the discrete and continuous fragments of $\mathsf{d}\mathcal{L}$ are non-axiomatizable. An extension of $\mathsf{d}\mathcal{L}$, the logic dTL is also non-axiomatizable [14], [16, chapter 4]. Since dTL^2 is a conservative extension of both $\mathsf{d}\mathcal{L}$ and dTL, those results lift to dTL^2. Therefore the discrete and continuous fragments of dTL^2, even if only containing nontemporal formulas are non-axiomatizable. In particular dTL^2 is non-axiomatizable.

Relative Completeness for Star-Free Expressions. We now show how to lift the relative completeness result of $\mathsf{d}\mathcal{L}$ [15,17], [16, chapter 2] to dTL^2; this completeness result is relative to first order logic of differential equations (FOD), i.e., first-order real arithmetic augmented with formulas expressing properties of differential equations [15,17], [16, chapter 2].

Theorem 2 (Relative completeness for star-free expressions). *The dTL^2 calculus restricted to *-free programs is complete relative to FOD, i.e., every valid dTL^2 formula with only star-free programs can be derived from FOD tautologies.*

Theorem 2 is proved in [10]. We conjecture that the proof system of dTL^2 is also relatively complete relative to FOD for all expressions, including repetitions.

4 Alternative Proof Systems

Normalizing all temporal formulas before applying the rules of Fig. 2 can sometimes result in longer proofs than necessary. In [10] we study a proof system directly handling (non-normalized) trace formulas. This extended proof system alleviates the need for normalizing all trace formulas, and is thus more efficient.

Another alternative, that we also study in [10], is to suppress all the $[\,]\sqcap$ rules of Fig. 2 (rules $[;]\sqcap$, $[?]\sqcap$, $[:=]\sqcap$, $[']\sqcap$ and $[^*]\sqcap$) and replace them by rules directly handling formulas of the form $[\alpha]\Box\phi$, and the following rule:

$$\frac{[\alpha]\phi \wedge [\alpha]\Box\psi}{[\alpha](\phi \sqcap \Box\psi)}([\,]\sqcap)$$

This results in a simpler system, because some of the rules are less complicated. However the system is not as efficient, because it duplicates the symbolic execution of α.

5 Related Work

In this section we study work related specifically to temporal reasoning of hybrid systems. For a more general account of previous work on verification of hybrid systems we refer to [15,17], [16, chapter 2].

This paper is based on work by Platzer introducing a temporal dynamic logic for hybrid systems [14], extending previous work by Beckert and Schlager [1] to hybrid programs. Both papers present a relatively complete calculus; however Beckert and Schlager only consider discrete state spaces, and only study temporal formulas of the form $[\alpha]\Box\phi$ and its dual $\langle\alpha\rangle\Diamond\phi$, leaving out any mixed cases alternating program and temporal modalities $[\alpha]\Diamond\phi$ or $[\alpha]\Box\Diamond\phi$. Platzer proposes to handle mixed cases by non-local program transformation, but does not show how to handle them compositionally.

Process logic [7,12,13,20] initially used temporal logic [6,19] in the context of dynamic logic [8] to reason about temporal behavior of programs. It is well studied, but limited to discrete programs. It also only considers an abstract notion of atomic program, without explicitly considering assignments and tests.

Davoren and Nerode [5] study hybrid systems and their topological aspects in the context of the propositional modal μ-calculus. Davoren, Coulthard, Markey and Moor [4] also give a semantics in general flow systems for a generalization of CTL*. In both [5] and [4], the authors provide Hilbert-style calculi to prove formulas of their systems, but in a propositional — not first-order — system, without specific proof rules to handle ordinary differential equations. Zhou, Ravn and Hansen [21] present a duration calculus extended by mathematical expressions with derivatives of state variables. Their system requires external mathematical reasoning about derivatives and continuity.

Other authors have studied temporal properties of hybrid systems in the context of model checking. Mysore, Piazza and Mishra [11] study model checking of semi-algebraic hybrid systems for TCTL (Timed Computation Tree Logic) properties and prove undecidability. They do bounded model checking for differential equations with polynomial solutions only, while we handle more general polynomial differential equations and unbounded safety verification. Additionally TCTL does not allow nesting of temporal modalities as we do. Cimatti, Roveri and Tonetta [3] present HRELTL, a linear temporal logic with regular expressions for hybrid traces. Their work is inspired by requirements validation for the European Train Control System, and uses bounded model checking and satisfiability modulo theory. More recently, Bresolin [2] develops HyLTL, a temporal logic for model checking hybrid systems, and shows how to solve the model checking problem by translating formulas into equivalent hybrid automata.

6 Conclusion and Future Work

In this paper we have presented a proof calculus for dTL^2, extending dTL by allowing nesting of temporal modalities. We showed proof rules for handling compositionally alternating program and temporal modalities, solving an open problem formulated in 2001 [1] and identified as a problem for hybrid systems in 2007 [14], [16, chapter 4]. We also offered a treatment where programs are not duplicated by proof rules, solving another open problem formulated by [14], [16, chapter 4]. We showed that the system is relatively complete with respect to FOD for *-free hybrid programs. The treatment of infinite traces is crucial to make the logic interesting, as temporal properties on terminating and error traces simplify greatly (Remark 1).

Future work includes proving our conjecture that the system is relatively complete with respect to FOD for all expressions; extending the semantics and the proof system to allow repetition — and not just differential equations — to create infinite traces; and implementing our proof rules in a tool such as KeYmaera [18].

A number of extensions to dTL^2 should be explored, such as inclusion of the temporal Until operator, or nested conjunctions and disjunctions inside temporal formulas. Some of these extensions can be handled by program transformations [14], [16, chapter 4], but a compositional proof system such as the one presented here would be more interesting. The proof system of dTL^2 is an important step towards a more general system dTL*, extending dTL^2 with formulas of CTL*, and expressing formulas such as $[\alpha]\Box(\Diamond\phi \wedge \psi)$. We would like to develop a semantics and a proof system for dTL*.

Acknowledgements. We are grateful to Khalil Ghorbal, Dexter Kozen, Sarah Loos, Stefan Mitsch, Ed Morehouse, Jan-David Quesel, Marcus Völp, and the anonymous referees for helpful comments and discussions. This material is based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246, NSF EXPEDITION CNS-0926181 and under Grant No. CNS-0931985.

References

1. Beckert, B., Schlager, S.: A sequent calculus for first-order dynamic logic with trace modalities. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) IJCAR. LNCS, vol. 2083, pp. 626–641. Springer (2001)
2. Bresolin, D.: HyLTL: a temporal logic for model checking hybrid systems. In: Bortolussi, L., Bujorianu, M.L., Pola, G. (eds.) HAS. EPTCS, vol. 124, pp. 73–84 (2013)
3. Cimatti, A., Roveri, M., Tonetta, S.: Requirements validation for hybrid systems. In: Bouajjani, A., Maler, O. (eds.) CAV. LNCS, vol. 5643, pp. 188–203. Springer (2009)
4. Davoren, J.M., Coulthard, V., Markey, N., Moor, T.: Non-deterministic temporal logics for general flow systems. In: Alur, R., Pappas, G.J. (eds.) HSCC. LNCS, vol. 2993, pp. 280–295. Springer (2004)
5. Davoren, J.M., Nerode, A.: Logics for hybrid systems. In: Proc. IEEE. Springer (2000)
6. Emerson, E.A., Halpern, J.Y.: "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. J. ACM 33(1), 151–178 (1986)
7. Harel, D., Kozen, D., Parikh, R.: Process logic: Expressiveness, decidability, completeness. J. Comput. Syst. Sci. 25(2), 144–170 (1982)
8. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge, MA (2000)
9. Hughes, G., Cresswell, M.: A New Introduction to Modal Logic. Routledge (1996)
10. Jeannin, J.B., Platzer, A.: dTL^2: Differential temporal dynamic logic with nested temporalities for hybrid systems. Tech. Rep. CMU-CS-14-109, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 15213 (May 2014), `http://reports-archive.adm.cs.cmu.edu/anon/2013/abstracts/14-109.html`
11. Mysore, V., Piazza, C., Mishra, B.: Algorithmic algebraic model checking II: Decidability of semi-algebraic model checking and its applications to systems biology. In: Peled, D., Tsay, Y.K. (eds.) ATVA. LNCS, vol. 3707, pp. 217–233. Springer (2005)
12. Nishimura, H.: Descriptively complete process logic. Acta Inf. 14, 359–369 (1980)
13. Parikh, R.: A decidability result for a second order process logic. In: FOCS. pp. 177–183. IEEE Comp. Soc. (1978)
14. Platzer, A.: A temporal dynamic logic for verifying hybrid system invariants. In: Artëmov, S.N., Nerode, A. (eds.) LFCS. LNCS, vol. 4514, pp. 457–471. Springer (2007)
15. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. 41(2), 143–189 (2008)
16. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010)
17. Platzer, A.: Logics of dynamical systems. In: LICS. pp. 13–24. IEEE (2012)
18. Platzer, A., Quesel, J.D.: KeYmaera: A hybrid theorem prover for hybrid systems. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR. LNCS, vol. 5195, pp. 171–178. Springer (2008)
19. Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57. IEEE Comp. Soc. (1977)
20. Pratt, V.R.: Process logic. In: Aho, A.V., Zilles, S.N., Rosen, B.K. (eds.) POPL. pp. 93–100. ACM (1979)
21. Zhou, C., Ravn, A.P., Hansen, M.R.: An extended duration calculus for hybrid real-time systems. In: Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) Hybrid Systems. LNCS, vol. 736, pp. 36–59. Springer (1992)