15-424: Foundations of Cyber-Physical Systems

Lecture Notes on Differential & Temporal Logics

André Platzer

Carnegie Mellon University
Lecture 16

1 Introduction

This course is devoted to the study of the Foundations of Cyber-Physical Systems [Pla12c, Pla12b]. Lecture 3 on Choice & Control explained hybrid programs, a program notation for hybrid systems [Pla08, Pla10, Pla12c, Pla12a]. Lecture 4 on Safety & Contracts defined differential dynamic logic [Pla08, Pla10, Pla12c, Pla12a] as a specification and verification logic for hybrid programs. Lecture 5 on Dynamical Systems & Dynamic Axioms and subsequent lectures studied proof principles for differential dynamic logic with which we can prove correctness properties of hybrid systems. In your labs, you have demonstrated aptly how you can model, specify, and verify quite sophisticated and challenging robots.

Yet, there was one rather puzzling phenomenon that we noticed in Lecture 4 only then did not have a chance to consider any further. For a hybrid program α and differential dynamic logic formula ϕ, the modal formula

$$[\alpha]\phi$$

expresses that all *final* states reached by all runs of α satisfy the logical formula ϕ. The modal formula $[\alpha]\phi$ is, consequently, false exactly in those states from which α can reach a final state that violates the safety condition ϕ. Yet, what about states from which the final state reached by running α is safe but some intermediate state along the execution of α was not safe?

Shouldn't systems that violate safety conditino ϕ at an intermediate state be considered unsafe as well?

The short answer is: that depends.

Does it even make a difference whether we study intermediate states as well or only worry about final states?

The short answer is again: that depends.

What exactly it depends on and how to systematically approach the general case of safety *throughout* the system execution is what today's lecture studies. The key to the answer will be understanding the temporal behavior of hybrid programs. The hybrid trace semantics of hybrid programs will also give us a deeper understanding of the hybrid aspect of time in hybrid systems.

This lecture is based on [Pla10, Chapter 4], which is a significant extension of [Pla07], and incorporates some aspects of follow-up work [JP14] to which we refer for a more general account to temporal aspects in hybrid systems verification. The thoughts on time in this lecture are related to an upcoming article [Pla14].

The most important learning goals of this lecture are:

Modeling and Control: We find identify one additional dynamical aspect, the aspect of temporal dynamics, i.e. how state changes over time throughout a system execution. It is important to learn to judge under which circumstance temporal dynamics is important for understanding a CPS and when in can be neglected without loss. Part of today's lecture is also about understanding time, never a bad goal to have.

Computational Thinking: This lecture addresses subtle aspects with identifying specifications and critical properties of CPS. We will also see how to express temporal variations of postconditions for CPS models. This lecture introduces *differential temporal dynamic logic* dTL [Pla10] extending the differential dynamic logic that is used as the specification and verification language for CPS in the other parts of this course by temporal aspects. Secondary goals in this lecture are practicing the first half of the logical trinity consisting of the relationship of syntax and semantics.

CPS Skills: We add a new dimension into our understanding of the semantics of a CPS model: the temporal dimension corresponding to how exactly a system changes state as a function of time. Such temporal changes have been implicit in the semantics of hybrid programs so far, because that was based on reachability relations. Today's lecture will make the temporal change explicit as a function of time. This helps understanding nuances in the semantics of hybrid systems either based on state reachability or on temporal traces, which further helps sharpen our intuition for the operational effects of CPS as dynamic functions over time.

2 Temporalizing Hybrid Systems

In order to be able to distinguish whether a CPS is safe at the end of its run or safe always throughout its run, differential dynamic logic $\mathsf{d}\mathcal{L}$ will be extended with additional temporal modalities. The resulting logic extends $\mathsf{d}\mathcal{L}$ and is called *differential temporal dynamic logic* (dTL) [Pla10, Chapter 4]. The modal formula

$$[\alpha]\phi$$

of dℒ [Pla08, Pla12c] expresses that all *final* states reached by all runs of α satisfy the logical formula ϕ. The same dℒ formula $[\alpha]\phi$ is allowed in the logic dTL and has the same semantics [Pla10, Chapter 4]. The new temporal modal dTL formula

$$[\alpha]\Box\phi$$

instead, expresses that all states reached *all along* all traces of α satisfy ϕ. Those two modalities can be be used to distinguish systems that are always throughout from those that are only safe in final states. For example, if the dTL formula

$$[\alpha]\phi \wedge \neg[\alpha]\Box\phi$$

is true in an initial state ν, then the system α will be safe (in the sense of satisfying ϕ) in all final states reached after running α from ν, but is not safe always throughout all traces of all runs of α from ν. Can that happen?

You should try to answer this question before it is discussed in a later part of these lecture notes.

3 Syntax of Differential Temporal Dynamic Logic

The *differential temporal dynamic logic* dTL extends differential dynamic logic [Pla08, Pla10, Pla12c] with temporal modalities for verifying temporal specifications of hybrid systems. Hence, dTL has two kinds of modalities:

Modal operators. Modalities of dynamic logic express statements about all possible behaviour ($[\alpha]\pi$) of a system α, or about the existence of a trace ($\langle\alpha\rangle\pi$), satisfying condition π. Unlike in standard dynamic logic, α is a model of a hybrid system. The logic dTL uses hybrid programs to describe α as in previous lectures. Yet, unlike in standard dynamic logic [HKT00] or dℒ, π is a *trace formula* in dTL, and π can refer to all states that occur *during* a trace using temporal operators.

Temporal operators. For dTL, the temporal trace formula $\Box\phi$ expresses that the formula ϕ holds all along a trace selected by $[\alpha]$ or $\langle\alpha\rangle$. For instance, the state formula $\langle\alpha\rangle\Box\phi$ says that the state formula ϕ holds at every state along at least one trace of α. Dually, the trace formula $\Diamond\phi$ expresses that ϕ holds at some point during such a trace. It can occur in a state formula $\langle\alpha\rangle\Diamond\phi$ to express that there is such a state in some trace of α, or as $[\alpha]\Diamond\phi$ to say that along each trace there is a state satisfying ϕ. The primary focus of attention in today's lecture is on homogeneous combinations of path and trace quantifiers like $[\alpha]\Box\phi$ or $\langle\alpha\rangle\Diamond\phi$.

The formulas of dTL are defined similarly to differential dynamic logic. However, the modalities $[\alpha]$ and $\langle\alpha\rangle$ accept trace formulas that refer to the temporal behavior of *all* states along a trace. Inspired by CTL and CTL* [EC82, EH86], dTL distinguishes between state formulas, which are true or false in states, and trace formulas, which are true or false for system traces. The sets Fml of state formulas and Fml_T of trace formulas with variables in Σ are simultaneously inductively defined in Def. 1.

Definition 1 (dTL formula)**.** The *(state) formulas of differential temporal dynamic logic* (dTL) are defined by the grammar (where ϕ, ψ are dTL state formulas, π is a dTL trace formula, θ_1, θ_2 (polynomial) terms, x a variable, α a HP):

$$\phi, \psi \ ::= \ \theta_1 = \theta_2 \mid \theta_1 \geq \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x\, \phi \mid \exists x\, \phi \mid [\alpha]\pi \mid \langle\alpha\rangle\pi$$

The *trace formulas of* dTL are defined by the grammar (where ϕ is a dTL state formula):

$$\pi ::= \phi \mid \Box\phi \mid \Diamond\phi$$

Operators $>, \leq, <, \leftrightarrow$ *can be defined as usual, e.g.,* $\phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$.

Formulas without $\Box$ and $\Diamond$ are *nontemporal formulas* and have the same semantics as the corresponding dℒ formulas. Unlike in CTL, dTL state formulas are true on a trace if they hold for the *last* state of a trace, not for the first. Thus, when ϕ is a state formula, dTL formula $[\alpha]\phi$ expresses that ϕ is true at the end of each trace of α, which is the same as the dℒ formula $[\alpha]\phi$. In contrast, $[\alpha]\Box\phi$ expresses that ϕ is true *all along* all states of every trace of α. This combination gives a smooth embedding of nontemporal dℒ into dTL and makes it possible to define a compositional calculus. Like CTL, dTL allows nesting with a branching time semantics [EC82], e.g., $[\alpha]\Box(x \geq 2 \rightarrow \langle\beta\rangle\Diamond x \leq 0)$.

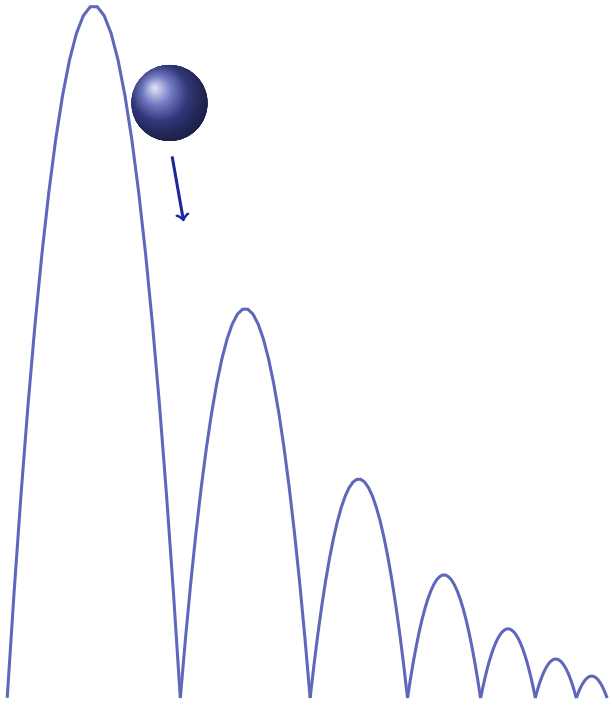

Figure 1: Sample trajectory of a bouncing ball (plotted as position over real time)

4 Hybrid Time

The semantics of differential temporal dynamic logic refers to the temporal behavior of hybrid programs along a trace over time. Our first goal will therefore be to find the right notion of time for the job.

Let us consider the familiar bouncing ball; see Fig. 1. The bouncing ball is flying through the air toward the ground, bounces back up when it hits the ground, and will again fly up. Then, as gravity wins over, it will fly down again for a second bounce, and so forth, leading to a lot of interesting physics including questions of how the kinetic energy transforms into potential energy as the ball deforms by an elastic collision on the ground and then reverses the deformation to gain kinetic energy [Cro00].

Alternatively, we decided in Lecture 4 on Safety & Contracts to put our multi-dynamical systems glasses on [Pla12c] and realized that the bouncing ball dynamics consists of two phases that, individually, are easy to describe and interact to form a hybrid system. There is the flying part, where the ball does not do anything but move according to gravity.[1] And then there is the bouncing part, where the ball bounces back from the ground. While there is more physics involved in the bouncing, a simple description is that the bounce on the ground will make the ball invert its velocity vector (from down to up) and slow down a little (since the friction loses energy). Both aspects separately, the flying and the bouncing, are easy to understand. They interact as a hybrid system, where the ball flies continuously through the air until it hits the ground where it bounces back up by a discrete jump of its velocity from negative to positive. These thoughts led us to a hybrid program model for the bouncing ball along with its specification in differential dynamic logic from Lecture 4 on Safety & Contracts:

$$0 \leq x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 > c \geq 0 \rightarrow \\ \left[\left(x' = v, v' = -g \,\&\, x \geq 0;\ \texttt{if}(x = 0)\, v := -cv\right)^*\right](0 \leq x \wedge x \leq H) \quad (1)$$

A typical run of the bouncing ball program in (1) will follow an alternating succession of a continuous trajectory for the differential equation $x' = v, v' = -g$ within the evolution domain $x \geq 0$ for a certain nonzero duration and an instantaneous discrete jump following the discrete change $\texttt{if}(x = 0)\, v := -cv$ at a discrete instant of time. This succession gives rise to a subtlety. If we ask what value the velocity and height of the bouncing ball have at a certain point in time t_1, chances are that that is not going to have an unambiguous answer. Whenever the ball is on the ground ($x = 0$) bouncing back up, there are two velocities we could be referring to. The negative velocity where the ball was still flying downwards and the subsequent positive velocity after the bounce which reverted direction by $v := -cv$. So if we are trying to define a trace as a function $\sigma : \mathbb{R} \to \mathcal{S}$ from real time $\mathbb{R}$ to the state space $\mathcal{S}$, we will utterly fail producing a function, because the velocity values at a time t_1 with $\sigma(t)(x) = 0$ are not unique. What could we do about that?

Before you read on, see if you can find the answer for yourself.

[1] Taking the usual models of air resistance into account turned out to be easy as well as we saw in Lecture 11 on Differential Equations & Proofs.

The way out of this dilemma is to blow up time. Ever since Lecture 12 on Ghosts & Differential Ghosts, we are used to seeing extra dimensions everywhere. Time is one of those cases where a spooky extra dimension can help clarify what is going on. Let's add a second dimension to time so that we can distinguish between the first and the second time that the ball was at the ordinary real time t_1. The first such time (maybe denoted $t_1.1$) will have negative velocity, the second one (denoted $t_1.2$) positive velocity. Think of this as buying a pair of chronophotographic hyper-time spectacles and looking at the same world as before with a more fine-grained notion of time to discover that there is succession in things that looked indistinguishable before.[2]

It turns out, however, that rather than suffixing real points in time t_1 with a natural number j to form $t_1.j$, it is more convenient to turn it around and consider time T as a cartesian product $\mathbb{N} \times \mathbb{R}$ such that a point in time (j, t) consists of a natural number $j \in \mathbb{N}$ counting how many discrete steps have happened so far and a real number $t \in \mathbb{R}, t \geq 0$ indicating the real amount of time it took to get there.

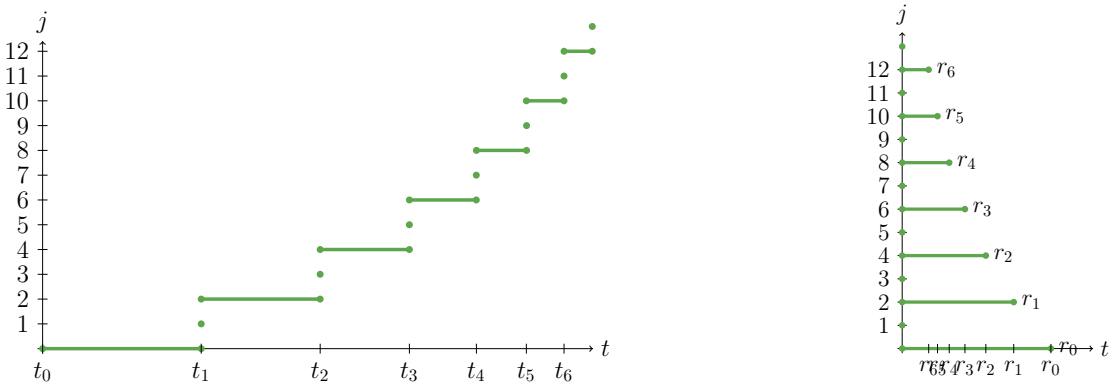

Figure 2: Two possible hybrid time domains for the sample trajectory of a bouncing ball with discrete time step j and continuous time t

This succession of continuous and discrete transitions in Fig. 1 gives rise to the hybrid time domain T shown in Fig. 2(left). Here, the intervals are either compact intervals $[t_i, t_{i+1}]$ of positive duration $t_{i+1} - t_i > 0$ during which the ball is flying through the air continuously according to the differential equation, or they are point intervals $[t_i, t_i]$ and a discrete transition happens at that single point in time that changes the sign and magnitude of the ball's velocity by a bounce described in the assignment. For example, $[t_1, t_2]$ is the time interval during which the ball is flying after its first bounce. And the point interval $[t_2, t_2]$ represents the point in time during which the subsequent discrete transition of bouncing happened, while $[t_2, t_3]$ would be the flying phase after the second bounce.

While this choice of a hybrid time domain gives a nice visual representation of the overall progress of time, the lower bound of the intervals is not particularly informa-

[2]This is one of the many amazing cases where we follow Wheeler's expression of Henri Poincaré's thoughts: "Time is defined so that motion looks simple" [MTW73, p. 23].

tive, because it coincides with the upper bound of the previous interval of time. It gets notationally easier if all lower bounds of all intervals are normalized to 0 and only the duration $r_i = t_{i+1} - t_i$ is retained as the upper bound; see Fig. 2(right). Both hybrid time domains in Fig. 2 are ultimately equivalent but the one on the right is easier to work with. Fig. 3 shows the particular sample trajectory of the bouncing ball from Fig. 1 plotted on its corresponding hybrid time domain T from Fig. 2(right). That illustration separates out the various discrete and continuous pieces of the trajectory of the bouncing ball into separate fragments of the two-dimensional hybrid time.

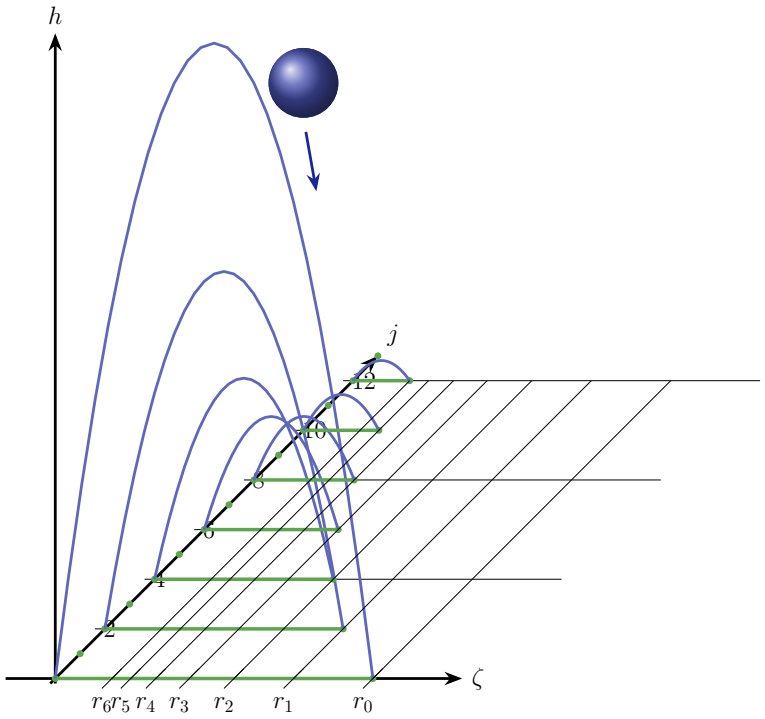

Figure 3: Sample trajectory of a bouncing ball plotted as position h over its hybrid time domain with discrete time step j and continuous time ζ

This particular illustration nicely highlights the hybrid nature of the bouncing ball dynamics. The downside, however, is that the hybrid domain T shown in Fig. 2 is specific to the particular bouncing ball trajectory from Fig. 1 and Fig. 3 and does not fit to any other bouncing ball trajectories.

Before we proceed, we illustrate two more phenomena that are worth noticing: subdivision and super-dense computations. While Fig. 2 shows one hybrid time domain for the sample trajectory in Fig. 1, there are infinitely many other hybrid time domains that fit to the original sample trajectory shown in Fig. 1 and just subdivide one of the intervals of a flying phase into two subintervals during which the ball just keeps on flying the way it did before. The first flying phase, for example, could just as well be subdivided into the continuous phase where the ball is flying up according to the dif-

ferential equation followed by a continuous phase where the ball is flying down, still according to the same differential equation. This happens whenever the continuous evolution stops before the ball was on the ground, in which case the hybrid program from (1) will loop around without actually changing any variables. That would yield a different hybrid time domain with multiple intervals of positive duration in immediate succession but still essentially the same behavior of the hybrid system in the end. So subdivision of time domains does not yield characteristically different behavior. Likewise, there can be hybrid systems that have multiple discrete steps (corresponding to point intervals in the hybrid time domain) in immediate succession before a continuous transition happens again. For example, a car could, successively, switch gears and disable the adaptive cruise control system and engage a warning light to alert the driver before it ceases control again to the continuous driving behavior. Hence, while strict alternation of discrete and continuous transitions may be the canonical example to have in mind, it is most definitely not the only relevant scenario.

5 Trace Semantics of Hybrid Programs

In differential dynamic dℒ [Pla08, Pla12c] from Lecture 4, modalities only refer to the final states of system runs and the semantics is a reachability relation on states: State ω is reachable from state ν using system α if there is a run of α which terminates in ω when started in ν. For dTL, however, formulas can refer to intermediate states of runs as well. To capture this, we change the semantics of a hybrid system α to be the set of its possible *traces*, i.e., successions of states that occur during the evolution of α. The relation between the initial and the final state alone is not sufficient.

States define the values of system variables during a hybrid evolution. A *state* is a map $\nu : \Sigma \to \mathbb{R}$. In addition, we distinguish a separate state Λ to denote the failure of a system run when it is *aborted* due to a test $?H$ that yields false. In particular, Λ can only occur at the end of an aborted system run and marks that no further extension of that trace is possible because of a failed test. The set of all states is denoted by $\mathcal{S}$.

Hybrid systems evolve along piecewise continuous traces in multi-dimensional space as time passes. Continuous phases are governed by differential equations, whereas discontinuities are caused by discrete jumps in state space. Unlike in discrete cases [Pra79, BS01], traces are not just sequences of states, since hybrid systems pass through uncountably many states even in bounded time. Beyond that, continuous changes are more involved than in pure real time [ACD90, HNSY92], because all variables can evolve along differential equations with different slopes. Generalizing the real-time traces of [HNSY92], the following definition captures hybrid behaviour by splitting the uncountable succession of states into periods σ_i that are regulated by the same control law. For discrete jumps, some of those periods are point flows of duration 0.

The (trace) semantics of hybrid programs is compositional, that is, the semantics of a complex program is defined as a simple function of the trace semantics of its parts. What a hybrid trace captures is the full temporal evolution of a hybrid system. Hybrid systems can behave in different ways, so their trace semantics will be a set of hybrid

traces, each of which describes one particular temporal evolution over time. Time, however, is hybridized to a pair (i,ζ) of a discrete time index $i \in \mathbb{N}$ and a real time point $\zeta \in \mathbb{R}$. A single time component $\zeta \in \mathbb{R}$ itself would an inadequate model of time for hybrid systems, because hybrid systems can make progress by a discrete transition without continuous time passing. That happens whenever discrete controls take action. Continuous time only passes during continuous evolutions along differential equations. Discrete actions only make discrete time index i pass.

Definition 2 (Hybrid trace)**.** A *trace* is a (nonempty) finite sequence

$$\sigma = (\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_n)$$

of functions $\sigma_i : [0, r_i] \to \mathcal{S}$ with their respective durations $r_i \in \mathbb{R}$ (for $i \in \mathbb{N}$). A *position* of σ is a pair (i,ζ) with $i \in \mathbb{N}, i \leq n$ and ζ in the interval $[0, r_i]$; the state of σ at (i,ζ) is $\sigma_i(\zeta)$. Positions of σ are ordered lexicographically by $(i,\zeta) \prec (j,\xi)$ iff either $i < j$, or $i = j$ and $\zeta < \xi$. Further, for a state $\nu \in \mathcal{S}$, $\hat{\nu} : [0,0] \to \mathcal{S}; 0 \mapsto \nu$ is the *point flow* at ν with duration 0, which is only defined at the time 0 as $\hat{\nu}(0) = \nu$.

A trace *terminates* if it is a finite sequence $(\sigma_0, \sigma_1, \ldots, \sigma_n)$ with $\sigma_n(r_n) \neq \Lambda$. In that case, the last state $\sigma_n(r_n)$ is denoted by $\operatorname{last}\sigma$, otherwise $\operatorname{last}\sigma$ is undefined. The first state $\sigma_0(0)$ is denoted by $\operatorname{first}\sigma$. A trace is an *error trace* if it is a finite sequence $(\sigma_0, \sigma_1, \ldots, \sigma_n)$ with $\sigma_n(r_n) = \Lambda$. Error traces of hybrid programs cannot be continued any further, so if a trace has the error state Λ anywhere, then only as the last state $\sigma_n(r_n)$.

Unlike in [ACD90, HNSY92], the definition of traces also admits finite traces of bounded duration, which is necessary for compositionality of traces in $\alpha;\beta$. The semantics of hybrid programs α as the set $\tau(\alpha)$ of its possible traces depends on valuations $[\![\cdot]\!]_\nu$ of formulas and terms at intermediate states ν. The valuation of terms and interpretations of function and predicate symbols are as for real arithmetic (Lecture 4). The valuation of formulas will be defined in Def. 6. Again, we use ν_x^d to denote the *modification* that agrees with state ν on all variables except for the symbol x, which is changed to $d \in \mathbb{R}$.

Definition 3 (Trace semantics of hybrid programs)**.** The *trace semantics, $\tau(\alpha)$, of a hybrid program α*, is the set of all its possible hybrid traces and is defined inductively as follows:

1. $\tau(x := \theta) = \{(\hat{\nu}, \hat{\omega}) \ : \ \omega = \nu \text{ except that } [\![x]\!]_\omega = [\![\theta]\!]_\nu \text{ for } \nu \in \mathcal{S}\}$
2. $\tau(x' = \theta \,\&\, H) = \{(\varphi) \ : \ \varphi(t) \models x' = \theta \text{ and } \varphi(t) \models H \text{ for all } 0 \leq t \leq r \text{ for a solution } \varphi : [0, r] \to \mathcal{S} \text{ of any duration } r\} \cup \{(\hat{\nu}, \hat{\Lambda}) \ : \ \nu \not\models H\}$;
   i.e., with $\varphi(t)(x') \overset{\text{def}}{=} \frac{\mathsf{d}\varphi(\zeta)(x)}{\mathsf{d}\zeta}(t)$, φ solves the differential equation and satisfies H at all times, see Lecture 2. If even the initial state ν does not satisfy H, there can be no evolution except from the current state ν to an error state Λ.
3. $\tau(?H) = \{(\hat{\nu}) \ : \ \nu \models H\} \cup \{(\hat{\nu}, \hat{\Lambda}) \ : \ \nu \not\models H\}$
4. $\tau(\alpha \cup \beta) = \tau(\alpha) \cup \tau(\beta)$
5. $\tau(\alpha; \beta) = \{\sigma \circ \varsigma \ : \ \sigma \in \tau(\alpha)\,, \varsigma \in \tau(\beta) \text{ when } \sigma \circ \varsigma \text{ is defined}\}$;
   the composition of $\sigma = (\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_n)$ and $\varsigma = (\varsigma_0, \varsigma_1, \varsigma_2, \ldots, \varsigma_m)$ is
   $$\sigma \circ \varsigma := \begin{cases} (\sigma_0, \ldots, \sigma_n, \varsigma_0, \varsigma_1, \ldots, \varsigma_m) & \text{if } \sigma \text{ terminates and } \operatorname{last}\sigma = \operatorname{first}\varsigma \\ \sigma & \text{if } \sigma \text{ does not terminate} \\ \text{not defined} & \text{otherwise} \end{cases}$$
6. $\tau(\alpha^*) = \bigcup_{n \in \mathbb{N}} \tau(\alpha^n)$, where $\alpha^{n+1} \overset{\text{def}}{\equiv} (\alpha^n; \alpha)$ for $n \geq 1$, as well as $\alpha^1 \overset{\text{def}}{\equiv} \alpha$ and $\alpha^0 \overset{\text{def}}{\equiv} (?true)$.

Time passes differently during discrete and continuous change. During continuous evolutions, the discrete step index i of positions (i, ζ) remains constant, whereas the continuous duration ζ remains 0 during discrete point flows. This permits multiple discrete state changes to happen at the same (super-dense) continuous time, unlike in other approaches [ACD90].

Example 4. For comparing the transition semantics of hybrid programs for $\mathsf{d}\mathcal{L}$ from Lecture 3 and the trace semantics of hybrid programs for dTL from Def. 3, consider the following simple hybrid program α:

$$a := -2a;\ a := a^2.$$

The transition semantics is just the relation between initial and final states:

$$\rho(\alpha) \equiv \{(\nu, \omega) \ : \ \omega \text{ is like } \nu \text{ except that } \omega(a) = 4\nu(a)^2\}.$$

In particular, the $\mathsf{d}\mathcal{L}$ formula $[\alpha]a \geq 0$ is valid, because all final states have a square as

the value of a. In contrast, the trace semantics of α retains all intermediate states:

$$\tau(\alpha) \equiv \{(\hat{\nu}, \hat{s}, \hat{\omega}) \ : \ s \text{ is like } \nu \text{ except } s(a) = -2\nu(a) \\ \text{and } \omega \text{ is like } s \text{ except } \omega(a) = s(a)^2 = 4\nu(a)^2\}.$$

During these traces, $a \geq 0$ does not hold at all states. If the trace starts with a positive value ($\nu \models a > 0$), then it will become negative at the point flow s (where $s \models a < 0$), yet recover to a positive value ($\omega \models a > 0$) at the end.

Example 5. The previous example only had discrete jumps, and, thus, the traces only involved point flows. Now consider the hybrid program β from the train context:

$$a := -b;\ z' = v, v' = a;\ ?v \geq 0;\ a := A;\ z' = v, v' = a.$$

The transition semantics of this program only considers successful runs to completion. In particular, if $A > 0$, the velocity v will always be nonnegative at the end (otherwise the test $?v \geq 0$ in the middle fails and the program aborts), because the last differential equation will accelerate and increase the velocity again. Thus, the position z at the end of the program run will never be smaller than at the beginning.

If, instead, we consider the trace semantics of β, all intermediate states are in the set of traces:

$$\begin{aligned}\tau(\beta) \ \equiv \ &\{(\hat{\mu_0}, \hat{\mu_1}, \varphi_1, \hat{\mu_2}, \hat{\mu_3}, \varphi_2) \ : \ \mu_1 = \mu_0[a \mapsto -\mu_0(b)] \text{ and} \\ &\quad \varphi_1 \text{ is a state flow of some duration } r_1 \geq 0 \text{ with } \varphi_1 \models z' = v \wedge v' = a \\ &\quad \text{starting in } \varphi_1(0) = \mu_1 \text{ and ending in a state with } \varphi_1(r_1)(v) \geq 0 \\ &\quad \text{and } \mu_2 = \varphi_1(r_1), \mu_3 = \varphi_1(r_1)[a \mapsto \varphi_1(r_1)(A)] \text{ and} \\ &\quad \varphi_2 \text{ is a state flow of some duration } r_2 \geq 0 \text{ with } \varphi_2 \models z' = v \wedge v' = a \\ &\quad \text{starting in } \varphi_2(0) = \mu_3 \text{ and ending in state } \varphi_2(r_2)\} \\ \cup \ &\{(\hat{\mu_0}, \hat{\mu_1}, \varphi_1, \hat{\mu_2}, \hat{\Lambda}) \ : \ \mu_1 = \mu_0[a \mapsto -\mu_0(b)] \text{ and} \\ &\quad \varphi_1 \text{ is a state flow of some duration } r \geq 0 \text{ with } \varphi_1 \models z' = v \wedge v' = a \\ &\quad \text{starting in } \varphi_1(0) = \mu_1 \text{ and ending in a state with } \varphi_1(r)(v) < 0 \\ &\quad \text{further } \mu_2 = \varphi_1(r)\}.\end{aligned}$$

The first set is the set of traces where the test $?v \geq 0$ in the middle succeeds and the system continues. The second set (after the union) is the set of traces that are aborted with $\hat{\Lambda}$ during their execution, because the middle test fails. Note that the traces in the first set have two continuous flows φ_1, φ_2 and four point flows $\hat{\mu_0}, \hat{\mu_1}, \hat{\mu_2}, \hat{\mu_3}$ in each trace. The traces in the second set have only one continuous flow φ_1 and three point flows $\hat{\mu_0}, \hat{\mu_1}, \hat{\mu_2}$, because the subsequent aborting point flow $\hat{\Lambda}$ does not terminate and aborts all further execution. In the trace semantics, $v < 0$ is possible in the middle of some traces, which is a fact that the transition semantics does not notice. Combining traces for $\alpha \cup \beta$, that is, for

$$(a := -2a;\ a := a^2) \cup (a := -b;\ z' = v, v' = a;\ ?v \geq 0;\ a := A;\ z' = v, v' = a)$$

is just the union $\tau(\alpha) \cup \tau(\beta)$ of the traces $\tau(\alpha)$ and $\tau(\beta)$ from Examples 4 and 5. Note that $a \leq 0$ will hold at least once during every trace of $\alpha \cup \beta$, either in the beginning, or after setting $a := -2a$ or $a := -b$, respectively, when we assume $b > 0$.

6 Semantics of State and Trace Formulas

In the semantics of dTL formulas, the dynamic modalities determine the set of traces according to the trace semantics of hybrid programs, and, independently, the temporal modalities determine at which points in time the respective postcondition needs to hold. The semantics of formulas is compositional and denotational, that is, the semantics of a complex formula is defined as a simple function of the semantics of its subformulas.

Definition 6 (dTL semantics)**.** The *satisfaction relation* $\nu \models \phi$ for a dTL (state) formula ϕ in state ν is defined inductively:

- $\nu \models (\theta_1 = \theta_2)$ iff $[\![\theta_1]\!]_\nu = [\![\theta_2]\!]_\nu$.
- $\nu \models (\theta_1 \geq \theta_2)$ iff $[\![\theta_1]\!]_\nu \geq [\![\theta_2]\!]_\nu$.
- $\nu \models \neg\phi$ iff $\nu \not\models \phi$, i.e. if it is not the case that $\nu \models \phi$.
- $\nu \models \phi \wedge \psi$ iff $\nu \models \phi$ and $\nu \models \psi$.
- $\nu \models \phi \vee \psi$ iff $\nu \models \phi$ or $\nu \models \psi$.
- $\nu \models \phi \rightarrow \psi$ iff $\nu \not\models \phi$ or $\nu \models \psi$.
- $\nu \models \phi \leftrightarrow \psi$ iff ($\nu \models \phi$ and $\nu \models \psi$) or ($\nu \not\models \phi$ and $\nu \not\models \psi$).
- $\nu \models \forall x\, \phi$ iff $\nu_x^d \models \phi$ for all $d \in \mathbb{R}$.
- $\nu \models \exists x\, \phi$ iff $\nu_x^d \models \phi$ for some $d \in \mathbb{R}$.
- $\nu \models [\alpha]\pi$ iff for each trace $\sigma \in \tau(\alpha)$ that starts in $\text{first}\,\sigma = \nu$, if $[\![\pi]\!]_\sigma$ is defined, then $[\![\pi]\!]_\sigma = \mathit{true}$.
- $\nu \models \langle\alpha\rangle\pi$ iff there is a trace $\sigma \in \tau(\alpha)$ starting in $\text{first}\,\sigma = \nu$ such that $[\![\pi]\!]_\sigma$ is defined and $[\![\pi]\!]_\sigma = \mathit{true}$.

For trace formulas, the *valuation* $[\![\cdot]\!]_\sigma$ with respect to trace σ is defined inductively as:

1. If ϕ is a state formula, then $[\![\phi]\!]_\sigma = [\![\phi]\!]_{\text{last}\,\sigma}$ if σ terminates, whereas $[\![\phi]\!]_\sigma$ is *not defined* if σ does not terminate.
2. $[\![\Box\phi]\!]_\sigma = \mathit{true}$ iff $\sigma_i(\zeta) \models \phi$ holds for all positions (i, ζ) of σ with $\sigma_i(\zeta) \neq \Lambda$.
3. $[\![\Diamond\phi]\!]_\sigma = \mathit{true}$ iff $\sigma_i(\zeta) \models \phi$ holds for some position (i, ζ) of σ with $\sigma_i(\zeta) \neq \Lambda$.

As usual, a (state) formula is *valid* if it is true in all states. If $\nu \models \phi$, then we say that dTL state formula ϕ is true at ν or that ν is a model of ϕ. A (state) formula ϕ is *valid*, written $\vDash \phi$, iff $\nu \models \phi$ for all states ν. A formula ϕ is a *consequence* of a set of formulas Γ, written $\Gamma \vDash \phi$, iff, for each ν: ($\nu \models \psi$ for all $\psi \in \Gamma$) implies that $\nu \models \phi$. Likewise, for trace formula π and trace σ we write $\sigma \models \pi$ iff $[\![\pi]\!]_\sigma = \mathit{true}$ and $\sigma \not\models \pi$ iff $[\![\pi]\!]_\sigma = \mathit{false}$. In particular, we only write $\sigma \models \pi$ or $\sigma \not\models \pi$ if $[\![\pi]\!]_\sigma$ is defined, which it is not the case if π is a state formula and σ does not terminate. The points where a dTL property ϕ has to hold for the various combinations of temporal and dynamic modalities are illustrated in Fig. 4.

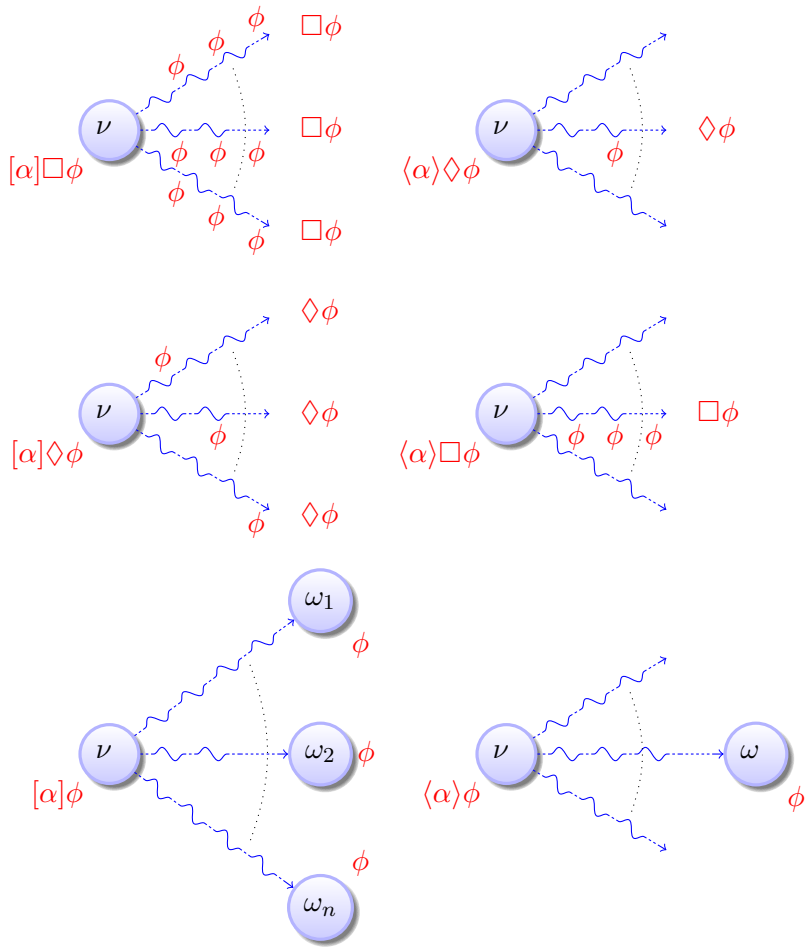

Figure 4: Trace semantics of dTL formulas

7 Conservative Temporal Extension

The following result shows that the extension by temporal operators that dTL provides does not change the meaning of nontemporal dℒ formulas. The trace semantics given in Def. 6 is equivalent to the final state reachability relation semantics given in Lecture 4 for the sublogic dℒ of dTL.

Proposition 7 (Conservative temporal extension [Pla10, Proposition 4.1])**.** *The logic* dTL *is a* conservative extension *of nontemporal* dℒ*, i.e., the set of valid* dℒ *formulas is the same with respect to transition reachability semantics of* dℒ *(Lecture 4) as with respect to the trace semantics of* dTL *(Def. 6).*

The proof is by induction using that the reachability relation fits to the trace semantics. That is, the reachability relation semantics of hybrid programs agrees with the first and last states of the traces in the trace semantics.

Lemma 8 (Trace relation [Pla10, Lemma 4.1])**.** *For hybrid programs* α*:*

$$\rho(\alpha) = \{(\text{first}\,\sigma, \text{last}\,\sigma) : \sigma \in \tau(\alpha) \textit{ terminates}\}.$$

In particular, the trace semantics from today's lecture fits seamlessly to the original reachability semantics that was the basis for the previous lectures. The trace semantics exactly satisfies the objective of characterizing the same reachability relation between initial and final states, while, in addition, keeping a trace of all intermediate states around. For nontemporal dTL formulas and for dℒ formulas, this full trace with intermediate states is not needed, because the reachability relation between initial and final states is sufficient to define the meaning For temporal dTL formulas, instead, the trace is crucial to give a meaning to $\Box$ and $\Diamond$.

8 Summary

This lecture introduced a temporal extension of the logic dℒ and a trace semantics of hybrid programs. This extends the syntax and semantics to the presence of temporal modalities. The next lecture investigates how to prove temporal properties of hybrid systems. Part of the value of today's lecture was to learn about how to state temporal properties of hybrid systems in differential temporal dynamic logic. An indirect aspect is, however, that it gave us a deeper understanding of the temporal behavior of hybrid systems even in cases where we continue to operate in differential dynamic logic.

Exercises

Exercise 1. Can you give a formula of the following form that is valid?

$$[\alpha]\Box\phi \wedge \neg[\alpha]\phi$$

Exercise 2. Plot the counterpart of the sample trajectory from Fig. 3 for the alternative hybrid time domain in Fig. 2(left).

Exercise 3. In which case does the temporal $[\alpha]\Box\phi$ differ from the nontemporal $[\alpha]\phi$.

Exercise 4. Def. 3 defined the trace semantics of tests as

$$\tau(?H) = \{(\hat{\nu}) \,:\, \nu \models H\} \cup \{(\hat{\nu}, \hat{\Lambda}) \,:\, \nu \not\models H\}$$

What would change if this definition would be modified to include an extra state $\hat{\nu}$ in the case of successful tests to record the fact that a test has happened:

$$\tau(?H) = \{(\hat{\nu}, \hat{\nu}) \,:\, \nu \models H\} \cup \{(\hat{\nu}, \hat{\Lambda}) \,:\, \nu \not\models H\}$$

Is there a dTL formula that is true in one semantics but not in the other? Would the semantics be the same when, instead, modifying the semantics of tests to elide the initial state:

$$\tau(?H) = \{(\hat{\nu}) \,:\, \nu \models H\} \cup \{(\hat{\Lambda}) \,:\, \nu \not\models H\}$$

Is there a dTL formula that is true in one semantics but not in the other?

Exercise 5. No traces ever start in error states. Is the semantics of sequential composition the same when dropping the requirement of termination and using the following definition instead:

$$\sigma \circ \varsigma := \begin{cases} (\sigma_0, \ldots, \sigma_n, \varsigma_0, \varsigma_1, \ldots, \varsigma_m) & \text{if } \operatorname{last}\sigma = \operatorname{first}\varsigma \\ \sigma & \text{if } \operatorname{last}\sigma = \Lambda \\ \text{not defined} & \text{otherwise} \end{cases}$$

This exercise assumes that $\operatorname{last}\sigma$ is defined as $\sigma_n(r_n)$ whether it terminates without error or with error.

Exercise 6. Is the formula (1) equivalent to the following dTL formula?

$$0 \le x \wedge x = H \wedge v = 0 \wedge g > 0 \wedge 1 > c \ge 0 \rightarrow \\ \big[\big(x' = v, v' = -g \,\&\, x \ge 0;\ \texttt{if}(x = 0)\, v := -cv\big)^*\big]\Box(0 \le x \wedge x \le H)$$

What if the differential equation is replaced by

$$x' = v, v' = -g; ?x \ge 0$$

Are the corresponding temporal and nontemporal formulas equivalent in that case?

References

[ACD90] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *LICS*, pages 414–425. IEEE Computer Society, 1990.

[BS01] Bernhard Beckert and Steffen Schlager. A sequent calculus for first-order dynamic logic with trace modalities. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *IJCAR*, volume 2083 of *LNCS*, pages 626–641. Springer, 2001.

[Cro00] Rod Cross. The coefficient of restitution for collisions of happy balls, unhappy balls, and tennis balls. *Am. J. Phys.*, 68(11):1025–1031, 2000. doi:10.1119/1.1285945.

[EC82] E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.

[EH86] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.

[HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic logic*. MIT Press, 2000.

[HNSY92] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. In *LICS*, pages 394–406. IEEE Computer Society, 1992.

[JP14] Jean-Baptiste Jeannin and André Platzer. dTL^2: Differential temporal dynamic logic with nested temporalities for hybrid systems. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *IJCAR*, volume 8562 of *LNCS*, pages 292–306. Springer, 2014. doi:10.1007/978-3-319-08587-6_22.

[LIC12] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.

[MTW73] Charles W. Misner, Kip S. Thorne, and John Archibald Wheeler. *Gravitation*. W. H. Freeman, New York, 1973.

[Pla07] André Platzer. A temporal dynamic logic for verifying hybrid system invariants. In Sergei N. Artëmov and Anil Nerode, editors, *LFCS*, volume 4514 of *LNCS*, pages 457–471. Springer, 2007. doi:10.1007/978-3-540-72734-7_32.

[Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):143–189, 2008. doi:10.1007/s10817-008-9103-8.

[Pla10] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. doi:10.1007/978-3-642-14509-4.

[Pla12a] André Platzer. The complete proof theory of hybrid systems. In LICS [LIC12], pages 541–550. doi:10.1109/LICS.2012.64.

[Pla12b] André Platzer. Dynamic logics of dynamical systems. *CoRR*, abs/1205.4788, 2012. arXiv:1205.4788.

[Pla12c] André Platzer. Logics of dynamical systems. In LICS [LIC12], pages 13–24. doi:10.1109/LICS.2012.13.

[Pla14] André Platzer. Analog and hybrid computation: Dynamical systems and programming languages. *Bulletin of the EATCS*, 114, 2014. URL: http://bulletin.eatcs.org/index.php/beatcs/article/viewFile/292/274.

[Pra79] Vaughan R. Pratt. Process logic. In *POPL*, pages 93–100, 1979.