The Logic in Computer Science Column

by

Yuri Gurevich

Microsoft Research
One Microsoft Way, Redmond WA 98052, USA
gurevich@microsoft.com

Analog and Hybrid Computation: Dynamical Systems and Programming Languages

André Platzer
Computer Science Department
Carnegie Mellon University
Pittsburgh, USA
aplatzer@cs.cmu.edu

Abstract

The purpose of this article is to serve as a light-weight introduction into the mysteries of analog and hybrid computing models from a dynamical systems and programming languages perspective. *Hybrid systems* are the dynamical systems that combine both models of computation, i.e., have interacting discrete and continuous dynamics. They have found widespread application as models for embedded computing in embedded systems as well as in cyber-physical systems. The primary role hybrid systems have played so far is to allow us to model how a (discrete) computer controller interacts with the (continuous) physical world and to analyze by means of formal proofs or reachability analyzes whether this interaction is safe or not. Without any doubt, such analyzes are of tremendous importance for our society, because they determine whether we can bet our lives on those systems.

But this article argues that hybrid systems also have computational consequences that make them an interesting subject to study from a computability theory perspective. Hybrid systems are described by hybrid programs or hybrid automata, both hybrid generalizations of corresponding discrete computational models. The phenomenon of discrete and continuous interplay, which hybrid systems provide, is fundamental and raises interesting computability questions. For example: what is computable using the analogue computation capabilities of continuous dynamical systems? How do the discrete computation capabilities of discrete dynamical systems relate to classical models of computation *à la* Church–Turing? What happens in hybrid computation, where discrete and continuous computation interact? Are the two facets of computation, discrete and continuous, of fundamentally different character

or are they two sides of the same computational coin? This article answers some of these questions using the rich theory that a logical characterization of hybrid systems in differential dynamic logic of hybrid programs provides. But the article is meant primarily as a manifesto for the significance and inherent beauty that these questions possess in the first place.

1 Introduction

Embedded computing may be the "third revolution in information technology after the birth of the computer itself and the introduction of the hyper-connected world of the Internet" [1]. This third revolution is connecting all computational power to the physical world and is raising the challenge of understanding how physics and computing interact. The interaction of physics and computation mixes analog and digital and is important not just in self-driving cars but also in aerospace applications, railway, robotics, and advanced medical devices. *Hybrid systems* [2–14] have been developed for the purpose of understanding such combinations of discrete and continuous dynamics. Hybrid systems play a major role in approaches for studying whether embedded computing systems and cyber-physical systems satisfy crucial safety properties [15–19]. Answering such correctness questions is, without any doubt, crucial to find out whether we can bet our lives on those systems, which is what we do every time we get on an airplane or recently-built car.

This article serves as a light-weight introduction into the mysteries of hybrid computation and hybrid systems from a dynamical systems and programming languages perspective. Its focus is on the impact that discrete and analog computation as well as discrete and continuous dynamical effects have on those systems. For example, while discrete and continuous systems first appear to be of fundamentally different character, which was the motivation for developing hybrid systems in the first place, they later turn out to be surprisingly intimately related [14]. The theory of hybrid systems builds a logical computational bridge between discrete and continuous systems (Fig. 1), bringing them into perfect proof-theoretical alignment [14]. This article is primarily meant as a manifesto for the significance and beauty of the intriguing questions related to a computational view on hybrid systems.

This article is based on previous work [12–14, 20] to which we refer for more details. The article serves as a gentle introduction with an explicit alignment with the theory of dynamical systems, based on prior work [20]. It also highlights the unnecessary complexities that the shortcomings of hybrid time domains cause and advocates for a simpler approach to hybrid systems that is based on programming languages and logic.

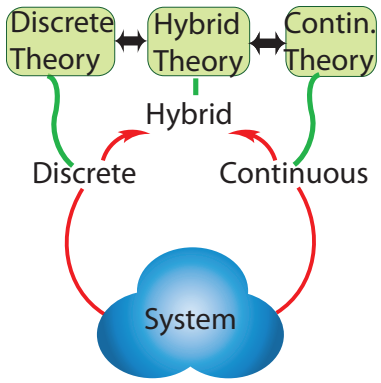

Figure 1: The proof theory of hybrid systems provides a complete proof-theoretical bridge aligning the theory of discrete systems and the theory of continuous systems

Structure of this Article. Section 2 starts with a light-weight introduction to general dynamical systems, discrete dynamical systems, continuous dynamical systems, and then illustrates important phenomena in hybrid systems. Section 3 discusses a programming language for hybrid systems, whose discrete and continuous fragments correspond to computational models for discrete dynamical systems and for continuous dynamical systems, respectively. Section 4 reviews a logical characterization of hybrid systems in differential dynamic logic [9]. Section 5 investigates the nature of hybridness by relating discrete and continuous dynamics by way of their common generalization as hybrid systems. Section 6 wraps up with concluding remarks and discusses interesting possibilities for future work.

2 Dynamical Systems

In this section, we survey the basic principles behind a number of important classes of dynamical systems. For a more comprehensive and more general overview and further extensions of dynamical systems, we refer to the prior work that this section is based on [20]. The theory of dynamical systems has been pioneered by Henri Poincaré [21].

2.1 General Dynamical Systems

A dynamical system [22, 23] is a mathematical model describing how a system changes its state over time. In a nutshell, a *dynamical system*[1] is a function $\varphi : T \times \mathcal{X} \to \mathcal{X}$ of time

[1] Formally, a *dynamical system* is an action of a monoid T on a state space $\mathcal{X}$. But this more general concept is not needed in this article.

T and state $\mathcal{X}$ whose value $\varphi_t(x) \in \mathcal{X}$ at time $t \in T$ denotes the state that the system has at time t when it originally started in the initial state $x \in \mathcal{X}$. The system starts at the initial state $\varphi_0(x) = x$ at time 0 and the evolution can proceed in stages, i.e., $\varphi_{t+s}(x) = \varphi_s(\varphi_t(x))$ for all $s, t \in T$ and all $x \in \mathcal{X}$; see Fig. 2. That is, if the dynamical system starts at x and evolves for time t to reach $\varphi_t(x)$ and, from that state, evolves again for time s to reach $\varphi_s(\varphi_t(x))$, then it reaches the same state $\varphi_{t+s}(x)$ by simply evolving for time $t + s$ starting from the initial state x right away.

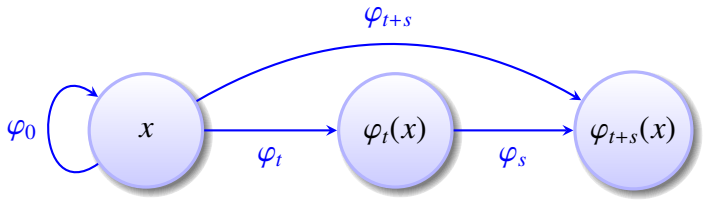

Figure 2: Dynamical systems can evolve in stages

Different choices of the time domain T and the state space $\mathcal{X}$ lead to different classes of dynamical systems. The time domain T is classically either discrete time ($T = \mathbb{N}$ or $T = \mathbb{Z}$), which proceeds in separate discrete steps, or continuous time ($T = \mathbb{R}$ or $T = [0, \infty)$), which has a dense continuous notion of progress of time. The state space $\mathcal{X}$ is typically a vector space such as $\mathbb{R}^d$ where $d \in \mathbb{N}$ is the dimension of the system.

2.2 Discrete Dynamical Systems

Discrete dynamical systems [23] have an integer notion of time (e.g., $T = \mathbb{N}$ or $T = \mathbb{Z}$) so that the state evolves in discrete time steps, one step at a time, as typically described by a difference equation or discrete state transition function. That is, one thing happens after the other in clearly discernible steps. Classical computer programs, for example, proceed in such discrete successive steps, with one computation step at a time.

Basic concept. A *discrete dynamical system*

$$\varphi_{n+1}(x) = f(\varphi_n(x)) \quad (n \in \mathbb{N}) \tag{1}$$

is fully described by its *generator* $f : \mathcal{X} \to \mathcal{X}$ or transition function, where $x \in \mathcal{X}$ is its initial state and $\varphi_n(x)$ the state at time $n \in \mathbb{N}$ after having started from initial state $x \in \mathcal{X}$. That is, the generator f specifies which next state $f(x)$ the discrete dynamical system

reaches after one step when it is currently in state x. The discrete dynamical system keeps on making steps according to the generator f. It will run as follows

$$x = \varphi_0(x) \overset{f}{\mapsto} \varphi_1(x) \overset{f}{\mapsto} \varphi_2(x) \overset{f}{\mapsto} \varphi_3(x) \overset{f}{\mapsto} \ldots$$

In other words, when f^n denotes the n-fold composition of f (so $f^{n+1}(x) = f(f^n(x))$ and $f^0(x) = x$), then the discrete dynamical system φ will run as

$$x \overset{f}{\mapsto} f(x) \overset{f}{\mapsto} f^2(x) \overset{f}{\mapsto} f^3(x) \overset{f}{\mapsto} \ldots$$

Example 2.1 (Mandelbrot set)**.** One simple example of a discrete dynamical system comes from the context of Mandelbrot fractals, where a simple discrete operation is repeated over and over again and its long-term behavior defines whether a point lies in that set or not. The Mandelbrot set is the set of all complex numbers $c \in \mathbb{C}$ for which $f^n(0)$ is bounded for all iterations n of the function $f(z) = z^2 + c$. Recall that $\mathbf{i}^2 = -1$, so $f(x + y\mathbf{i}) = (x + y\mathbf{i})^2 + c = (x^2 - y^2) + 2xy\mathbf{i} + c$ for a complex number $z = x + y\mathbf{i}$ with real part $x \in \mathbb{R}$ and imaginary part $y \in \mathbb{R}$. Hence, the generator corresponding to the complex number $c = a + b\mathbf{i}$ is the function

$$f(x + y\mathbf{i}) = (x^2 - y^2) + 2xy\mathbf{i} + c = (x^2 - y^2 + a) + (2xy + b)\mathbf{i}$$

When considering f as a real function of two real arguments x, y instead of one complex argument z, this yields:

$$f(x, y) = (x^2 - y^2 + a,\ 2xy + b)$$

The Mandelbrot set is the set of parameters $(a, b) \in \mathbb{R}^2$ for which the dynamical system

$$(0, 0) \overset{f}{\mapsto} f(0, 0) \overset{f}{\mapsto} f^2(0, 0) \overset{f}{\mapsto} f^3(0, 0) \overset{f}{\mapsto} \ldots$$

corresponding to the above generator f is bounded (it can be shown that the bound 2 is sufficient). The initial trajectory shown in Fig. 3(left) for the parameter $a = -0.6, b = -0.2$, for example, indicates that the state of the dynamical system stays bounded, which, indeed, it will remain forever in this case. The initial trajectory shown in Fig. 3(right) for the parameter $a = 0.41, b = 0.3$, however, will diverge, because it already leaves the Euclidean norm bound 2.

Note that the full behavior of a discrete dynamical system is determined entirely by its local generator f, which describes a step, plus the initial state, e.g., $(0, 0)$ in the case of the Mandelbrot system. It is still very complex to find out the global behavior of the dynamical system in the long run, but locally in one step, it is precisely captured by f.

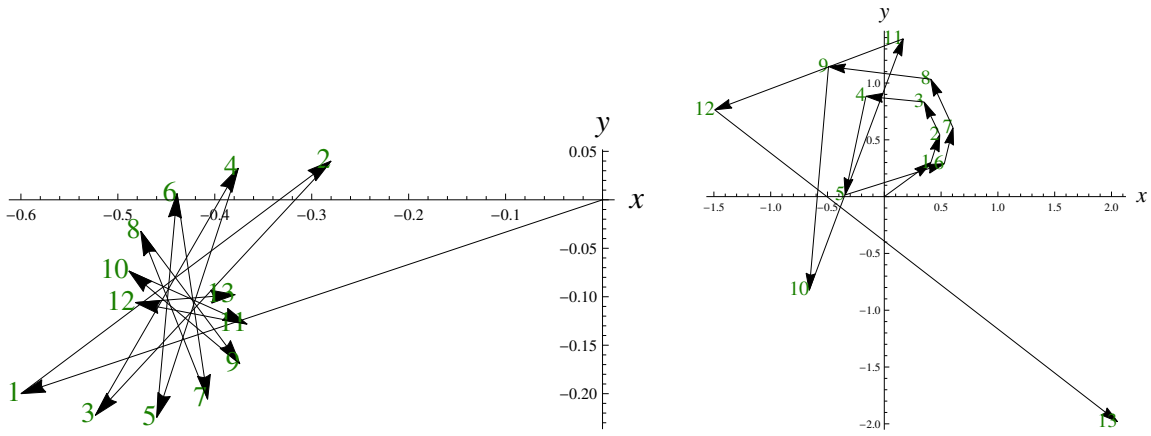

Figure 3: Trajectory of the Mandelbrot dynamical system for $a = -0.6, b = -0.2$ **(left)** and for $a = 0.41, b = 0.3$ **(right)** up to $n = 13$.

Difference equations. Another common way of describing the local generator of a discrete dynamical systems is by a difference equation. When defining $h(x) := f(x) - x$ the discrete dynamical system (1) can be described equivalently by the *difference equation*

$$\varphi_{n+1}(x) - \varphi_n(x) = h(\varphi_n(x)) \quad (n \in \mathbb{N}) \tag{2}$$

whenever the state space $\mathcal{X}$ is a vector space so that subtraction of states is defined. Both formulations, (1) and (2) are equivalent. The latter emphasizes the local change h of the state from one step to another as a function of the current state while the former emphasize the local state update f, instead. The vector from n to $n + 1$ shown in Fig. 3 directly illustrates the respective value of $h(\varphi_n(0))$, for example. Since there is a direct bijection between discrete dynamical systems in explicit form (1) and difference equations (2), both are often referred to informally as difference equations even if this is technically not quite correct.

Computational models. Computation processes can be described by discrete dynamical systems, for example. A computer system would start in an initial state $\varphi_0(x) = x$ at a time 0, perform a transition to a new state $\varphi_1(x) = f(x)$ at a time 1, then another transition to a state $\varphi_2(x) = f(f(x))$ at time 2, etc. until the computation terminates at a state $\varphi_n(x)$ at some time n. The scaling unit of these integer time steps is not relevant, but could be chosen, e.g., as the cycle time of a processor or discrete controller.

It is worth noting, however, that the dynamical systems induced by classical computer programs are both time- and space-discrete dynamical systems. That is, in addition to

having a discrete time domain $T = \mathbb{N}$, they also operate over a discrete state space $\mathcal{X}$ such as $\mathcal{X} = \mathbb{Z}^d$. In fact, when looking more closely, actual computers have finite memory so that $\mathcal{X}$ will even be a large but finite state space such as $\mathcal{X} = \{0, 1\}^d$. Program models and automata models have been used to describe discrete dynamical systems and have been used very successfully in verification [24–26].

In fact, the local generator f (respectively h when in difference equation form) needs to be sufficiently computational in order to have a chance of being used for any analytic purposes. Local generators often come from the transition function of a classical discrete computer program or the transition function of an automaton. But they can also be described using programs or machine models in more general models of computation such as the Blum-Shub-Smale model often called "real Turing machines" even if it is a random access machine [27]. In that case, the state space is some finite-dimensional real vector space $\mathcal{X} = \mathbb{R}^d$, because real Turing machines compute with real-valued data, but the time domain is still discrete $T = \mathbb{N}$. The computation of the generator for the Mandelbrot dynamical system can be described by a such real Turing machine [27]. It is, however, undecidable whether a point $a + b\mathbf{i}$ is in the Mandelbrot set, which corresponds to whether the Mandelbrot system for a, b always stays bounded, even in Blum-Shub-Smale's strong model of real computation [27]. Like everywhere else in computer science, it is, thus, imperative to distinguish between sets and their computational representation.

Other successful models of real computation are type II computable functions from the framework of computable analysis [28, 29], which, in a nutshell, study functions that can be computed up to arbitrary precision. Unlike non-quality, equality of real numbers, for example, is not type II computable, because, when two real numbers are different, we will ultimately find out by comparing their digits. But if they are the same, we will have to keep on comparing their digits for we will never be sure whether the next digit exhibits a difference or not.

Nondeterministic discrete dynamical systems. Discrete dynamical systems are described by transition functions, which makes them deterministic, i.e., for any initial state x and any time $n \in \mathbb{N}$ the discrete dynamical system will be in exactly one state $\varphi_n(x)$. This is at odds with understanding nondeterministic discrete systems, in which an initial state can have multiple successor states, because dynamical systems are supposed to be (deterministic) functions satisfying the staging property depicted in Fig. 2. For the staging property, $\varphi_n(x)$ has to have a unique value determined only by n and x and the dynamical system at hand, otherwise $\varphi_s(\varphi_t(x))$ does not have to agree with $\varphi_{t+s}(x)$ if $\varphi_t(x)$ were allowed to take on different values nondeterministically.

With a slight change in perspective, however, dynamical systems are equally useful for understanding nondeterministic discrete systems by going set-valued. The behavior

of systems with a discrete state transition relation $R \subseteq \mathcal{X} \times \mathcal{X}$ between previous states and successor states is nondeterministic, but can still be captured as a discrete dynamical system using the powerset $2^{\mathcal{X}}$ as the state space instead of $\mathcal{X}$:

$$\varphi_{n+1}(X) = f(\varphi_n(X)) = \{y \,:\, x \in \varphi_n(X) \text{ and } (x,y) \in R\} \quad (n \in \mathbb{N})$$

when starting from a set $X \subseteq \mathcal{X}$ of initial states. This principle is reminiscent of the powerset construction that converts nondeterministic finite automata into deterministic finite automata by considering a transition function on sets of states instead of a transition relation on individual states [30].

Limits of discrete dynamical systems. However useful discrete dynamical systems are, they cannot describe continuous processes, except as approximations at discrete points in time, e.g., with a uniform discretization grid $\frac{1}{n}$ at the discrete points in time $\frac{0}{n}, \frac{1}{n}, \frac{2}{n}, \ldots, \frac{n}{n}$. Discrete-time approximations give limited information about the behavior in between the $\frac{i}{n}$, which causes fundamental differences [31] but also surprising similarities [14].

2.3 Continuous Dynamical Systems

Continuous dynamical systems have a real continuous notion of time (e.g. $T = \mathbb{R}_{\geq 0}$ or $T = \mathbb{R}$) so that the state evolves continuously along a function of real time, typically described by a differential equation. The state of the system $\varphi_t(x)$ then is a function of continuous time t. In particular, unlike discrete dynamical systems, continuous dynamical systems have no notion of "next state" or "next time", because the time domain is (topologically) dense with a dense ordering relation $<$.

Basic concept. The *continuous dynamical system*

$$\begin{aligned} \frac{\mathsf{d}\varphi_t(x)}{\mathsf{d}t} &= f(\varphi_t(x)) \quad (t \in \mathbb{R}) \\ \varphi_0(x) &= x \end{aligned}$$

is fully described by its *generator* $f : \mathcal{X} \to \mathcal{X}$, where $x \in \mathcal{X}$ is the initial state at time 0. Depending on the duration of the solution of the above differential equation $\frac{\mathsf{d}\varphi_t(x)}{\mathsf{d}t} = f(\varphi_t(x))$, the continuous system may only be defined on some open subinterval of $\mathbb{R}$ rather than globally on $\mathbb{R}$. The time-derivative $\frac{\mathsf{d}}{\mathsf{d}t}$ is only well-defined under additional assumptions, e.g., that $\mathcal{X}$ is a differentiable manifold [22, 32] or simply some d-dimensional Euclidean space $\mathbb{R}^d$, which is what this article assumes. Many physical processes are continuous dynamical systems described by differential equations.

Example 2.2 (Motion with constant velocity along a straight line)**.** The movement of the longitudinal position of a car of velocity v down a straight road from initial position p_0 can be described by the differential equation $p'(t) = v$ with initial value $p(0) = p_0$. The state of the dynamical system at time t then is the solution $\varphi_t(p_0) = p_0 + tv$, which is defined at all times $t \in \mathbb{R}$.

Example 2.3 (Accelerated motion along a straight line)**.** The evolution of the state of a car accelerating with acceleration a on a straight line from initial position p_0 and initial velocity v_0 can be described by the differential equation system $p'(t) = v(t), v'(t) = a$ with initial value $p(0) = p_0$, $v(0) = v_0$. The state of that dynamical system at time t is then the vectorial solution

$$\varphi_t((p_0, v_0)) = \left(p_0 + tv_0 + \frac{a}{2}t^2,\ v_0 + at\right) \tag{3}$$

The notation p' for $\frac{\mathsf{d}p(t)}{\mathsf{d}t}$ is a common simplification, as is the implicit use of v instead of $v(t)$. Thus, the differential equation system for the accelerated car would often be written:

$$\begin{aligned} p' &= v \\ v' &= a \end{aligned} \tag{4}$$

Example 2.4 (Time square oscillator)**.** A simple example of a continuous dynamical system is described by the following differential equation

$$\begin{aligned} x' &= t^2 y \\ y' &= -t^2 x \\ t' &= 1 \end{aligned} \tag{5}$$

The initial trajectory shown in Fig. 4(left) for the initial value $x = 0, y = 1, t = 0$ illustrates that the dynamical system stays bounded but oscillates increasingly fast. In this case, the solution is

$$\begin{aligned} x(\tau) &= \sin\left(\frac{\tau^3}{3}\right) \\ y(\tau) &= \cos\left(\frac{\tau^3}{3}\right) \\ t(\tau) &= \tau \end{aligned} \tag{6}$$

Example 2.5 (Damped oscillator)**.** Another example of a continuous dynamical system is described by the following differential equation

$$\begin{aligned} x' &= y \\ y' &= -4x - 0.8y \end{aligned} \tag{7}$$

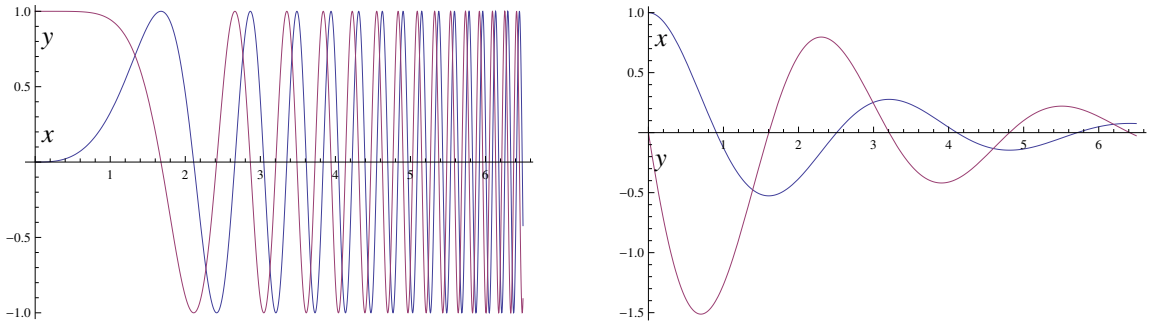

Figure 4: Trajectory of the time square oscillator for initial state $x = 0, y = 1, t = 0$ **(left)** and of the damped oscillator for initial state $x = 1, y = 0$ **(right)** up to time 6.5

The initial trajectory shown in Fig. 4(right) for the initial value $x = 1, y = 0$ illustrates that the dynamical system decays over time. In this case, the explicit global solution representing the dynamical system is more difficult.

More details and many more examples of continuous dynamical systems can be found in the literature [22, 32].

Computational models. Continuous processes can be described by the differential equations generating continuous dynamical systems. Just like discrete dynamical systems, which need to have suitable computational descriptions (e.g. by programs) in order to have a chance of being used for analytic purposes, continuous dynamical systems also need sufficiently computational descriptions.

One way of describing a continuous dynamical system in a computational model is to give a computational description of the system $\varphi_t(x)$ as a function of initial state x and time t. The motion with constant velocity from Example 2.2, for instance, can be described by a linear solution $\varphi_t(p_0) = p_0 + tv$. The accelerated motion from Example 2.3 can be described by the polynomial solution (3). Both symbolic expressions (linear and polynomial terms) are easily represented as arithmetic terms on a computer and their values can be computed easily, e.g., for every rational[2] $p_0, t \in \mathbb{Q}$.

That principle does not extend to Example 2.4, because its solution (6) is not polynomial. Even at rational $t \in \mathbb{Q}$, the value of the solution can only be approximated, because of the infinite power series $\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$ and likewise for cos. In computational

[2]Computations on bigger fields are possible, for example, for real algebraic $p_0, t \in \bar{\mathbb{Q}}$ using real algebraic number computations. Approximate computations are still possible for computable real numbers in the extended sense of increasingly fine approximations of type II computability in computable analysis [28, 29]. Polynomial computations for reals $p_0, t \in \mathbb{R}$ are allowed in Blum-Shub-Smale's computational model [27].

models for the reals that tolerate approximate answers, sin and cos are still computable [29], just only approximately so in the sense of type II computable analysis. For "most" dynamical systems, the situation is even more dire, because there is not even a closed-form symbolic solution of their differential equation at all. At least in those cases, the differential equation itself is a better computational representation of the continuous dynamical system. In fact, we argue that the differential equation is *always* a better computational representation, because the beautiful local perspective of differential equations is ruined when working with its complicated global solutions.

Under certain assumption, there are ways of computing approximate solutions of initial value problems of differential equations by numerical integration [14, 33, 34]. This depends crucially on additional assumptions on the system [31], such as known Lipschitz bounds or indirectly via known moduli of continuity in the case of type II computable functions [28, 29]. Otherwise, all relevant problems are undecidable even in strong models of computation even when tolerating arbitrarily large error bounds in the decision [31].

Type II computable functions in the sense of computable analysis [28, 29] have been identified [35] with a generalized understanding of Shannon's General Purpose Analog Computer (GPAC) [36] and with initial value problems of polynomial differential equations [35]. GPACs were originally meant as the mathematical model for the differential analyzer computer [37]. See Graça and Costa [38] for relations of GPACs to Moore's real recursive functions [39]. See Bournez et al. [35] for relations identifying GPACs, polynomial differential equations, and computable analysis when allowing for convergence [35] when, instead, considering a notion of computability for the GPACs that is based on convergence to the output in the limit with computable error bounds as considered in modern computability over the reals. Adding infinite convergent computations to the Blum-Shub-Smale model [27] has been considered in analytic machines [40]. Generalizations of finite automata from discrete time to continuous time have been considered as well [41] based on work by Trakhtenbrot [2001].

Limits of continuous dynamical systems. Continuous dynamical systems are continuous, so they have a hard time representing sudden discrete transitions. Discrete transitions lead to discontinuities, which lead to interesting but very complicated generalized notions of weak solutions, including Carathéodory solutions [34], Filippov solutions, Krasovskij solutions, and Hermes solutions; see Hájek for an overview [43].

Nondeterministic continuous dynamical systems. Nondeterminism is not a phenomenon that can only happen in discrete dynamical systems, but also in continuous dynamical systems; see [44] for an interesting perspective relating nondeterminism in continuous systems to the physical Church-Turing thesis. The most frequent source of nondeterminism

when working with continuous dynamical system comes from nondeterminism in the initial state while the rest of the continuous dynamical system stays deterministic. How long a continuous system is being followed is another important source of nondeterminism in a context where differential equations are embedded within hybrid systems.

Another source of nondeterminism directly in the continuous dynamical system itself comes from differential inequalities [34] or more general differential-algebraic constraints that also support nondeterministic disturbances [10]. In both cases, $p' \leq v$ would, for example, be a differential inequality describing that position p evolves with at most velocity v, possibly less. Likewise, the differential inequality $1 \leq p' \leq v$ describes a continuous dynamical system whose position changes with at most velocity v but at least velocity 1. It can have different velocities at different times, but is still restricted to be continuous, often even continuously differentiable (unlike in Carathéodory solutions [34] and Filippov solutions [45]). As in discrete dynamical systems, the fact that there is no unique velocity still results in a set-valued dynamical system φ to represent the nondeterminism as a function.

2.4 Hybrid Systems

Both discrete and continuous dynamical systems are useful and have their respective advantages depending on the situation that they model. Of course, there is no reason to believe that a given scenario only involves features that discrete dynamical systems are good at, or only features where continuous dynamical systems shine. More often than not, both features interact, and neither discrete nor continuous systems alone are a good fit for an application. In that case, hybrid dynamical systems are helpful, because they allow both discrete and continuous dynamics at once. Control decisions in systems are often of a more discrete nature, because they can be triggered suddenly, possibly by computerized controllers in response to certain events in the environment, while physical motion is a continuous phenomenon. But there are many other sources of hybridness as well, including fast physical processes that can suitably be abstracted by discrete dynamical systems.

Hybrid dynamical systems alias *hybrid systems* [2–14] are dynamical systems that combine discrete dynamical systems and continuous dynamical systems. Discrete and continuous dynamical systems are not just combined side by side to form hybrid systems, but they can interact in interesting ways. Part of the system can be described by discrete dynamics (e.g., decisions of a discrete-time controller), other parts are described by continuous dynamics (e.g., continuous movement of a physical process), and both kinds of dynamics interact freely in a hybrid system (e.g., when the discrete controller changes control variables of the continuous side by appropriate actuators such as when changing the acceleration input for the continuous dynamics, or when the continuous dynamics determines the values of sensor readings such as position or velocity for the discrete decisions).

Embedded systems and cyber-physical systems are often modeled as hybrid systems, because they involve both discrete control and physical effects.

A typical example of a hybrid system is a car that drives on a road according to a differential equation for the physical motion. This car is subject to discrete control decisions, where discrete controllers change the acceleration and braking of the wheels, e.g., when the adaptive cruise control or the electronic stability program takes effect. Figure 5 shows an example [46] how the acceleration of a car changes instantaneously by discrete control decisions (top), and how the velocity and position evolve continuously over time (middle and bottom) in response to the control input of acceleration. The situation in Fig. 5 illustrates bad control choices, where the follower car brakes too late (at time t_2) and then crashes into the leader car at time t_3. In particular, the follower car made a bad decision to keep on accelerating at some point before time t_2, when it should have activated the brakes instead, because, at time t_2, no control choice (within the physical acceleration limits $-b$ to A of the car) could still prevent the crash. This is one illustration of the phenomenon that bad control choices in the past cause unsafety in the future and that we need to verify our control choices now by considering their possible dynamical effects in the future.

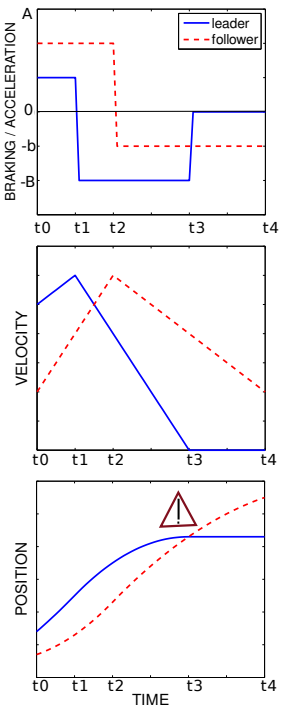

Figure 5: Example trajectory of a car control system where the follower collides with the leader car

When using hybrid systems instead of discrete dynamical systems, neither is there a need to use unnatural discretizations for continuous phenomena, because full continuous dynamics is allowed in hybrid systems. Nor is there a need to represent the system dynamics with the interesting but complicated discontinuous Carathéodory [34], Filippov, Krasovskij, or Hermes solutions [43] to understand jumps in continuous processes coming from sudden changes such as by decisions to activate the brakes. Discrete jumps are allowed directly as separate elements in hybrid systems. So, separately, both effects are easy to understand. The position changes continuously with the velocity, which changes continuously with the acceleration. And the acceleration is being decided by a computer controller. Each partial behavior alone is easy to understand and they just interact with one another to form a hybrid system. The overall system behavior can still be as complex as the original application demands. But the individual parts of the hybrid system have a simpler behavior that can be understood and analyzed by easier means.

Multi-dynamical systems. This phenomenon illustrates the keystone observation behind our philosophy of *multi-dynamical systems* [13, 20], i.e., the principle to understand complex systems as a combination of multiple elementary dynamical systems. The whole point of multi-dynamical systems is that the pieces are easier than the full system. That explains why multi-dynamical systems help tame the complexity of cyber-physical systems, because they understand systems in terms of their elementary parts, which are, by definition, easier than the full system. This compositional understanding of multi-dynamical systems carries over to their compositional analysis techniques [13, 20]. These techniques are based on proof steps that successively reduce a system to its parts and conclude correctness of the full system from correctness of its parts by compositional proof rules.

Basic concept. When formulating hybrid dynamical systems as a general dynamical system, we run into an immediate difficulty. What is the time domain T supposed to be for a hybrid system? It cannot be discrete $\mathbb{N}$, because hybrid systems can evolve continuously while their differential equations take effect. It cannot be continuous $\mathbb{R}$, either, though, because that does not fit to the discrete model of computation, one step at a time, that its discrete parts perform. In particular, a hybrid system might very well make a couple of discrete computation steps before proceeding with its continuous evolution again. Hence, the time domain is some combination of discrete time $\mathbb{N}$ and continuous time $\mathbb{R}$. There are different possibilities for the time domain but they follow the same essential idea [12, 47]. Hybrid time domains [47] are some subset $T \subset \mathbb{R} \times \mathbb{N}$, where the real component $t \in \mathbb{R}$ of a hybrid time point $(t, j) \in T$ measures the progress in real time and the natural number component $j \in \mathbb{N}$ measures the progress in time steps. Hybrid time domains are such that for each $j \in \mathbb{N}$ the set of all $t \in \mathbb{R}$ for which $(t, j) \in T$ is some interval in the reals. While there are a number of minor variations, such as whether the real intervals start at 0 or are consecutive intervals, the only important feature of hybrid time domains is that a hybrid time domain identifies a sequence of intervals. The complication is that the time domain T *depends on the particular execution of the hybrid system* and that executions of hybrid systems are highly nondeterministic. Since useful intuitions of more general interest arise from the study of the impact of time in hybrid systems, we illustrate the basic concept of a hybrid system by an instructive example.

Example 2.6 (Bouncing ball)**.** Let us consider a bouncing ball; see Fig. 6. 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 [48].

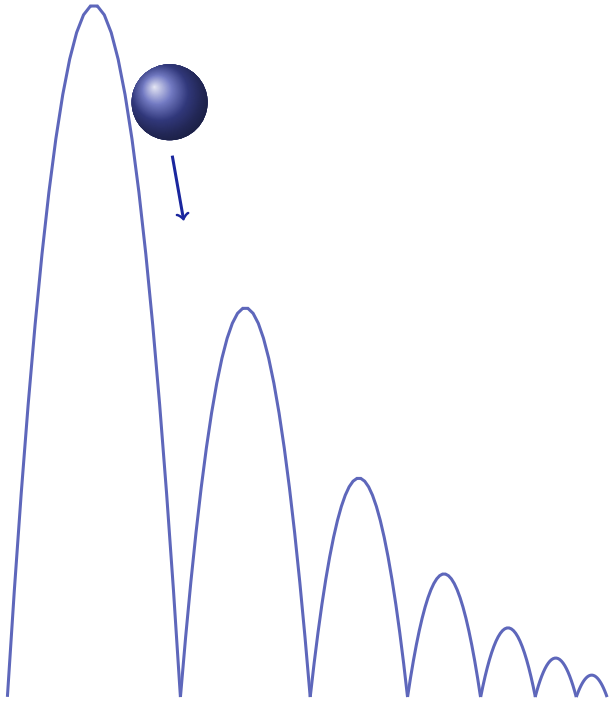

Figure 6: Sample trajectory of a bouncing ball (plotted as position over real time)

Alternatively, we can put our multi-dynamical systems glasses on and realize 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.[3] 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.

The continuous flying part of a bouncing ball is easy to describe by a differential equation, since the ball at height h with vertical velocity v is falling subject to gravity $g > 0$:

$$h' = v, v' = -g \tag{8}$$

The discrete bouncing part instantaneously negates the velocity of the ball around with a

[3]Taking the usual models of air resistance into account is not difficult either, but we refrain from doing so here for simplicity.

certain damping coefficient $0 \leq c < 1$:

$$v := -cv$$

This discrete change that updates the value of v to that of $-cv$ only happens when the ball just fell on the ground, which we posit is at height 0:

$$\mathsf{if}(h = 0)\, v := -cv \tag{9}$$

We postpone the question how to best represent how exactly the continuous flying dynamics (8) and the discrete bouncing dynamics (9) interact to form a hybrid system until we discuss the modeling language for hybrid systems in Section 3.

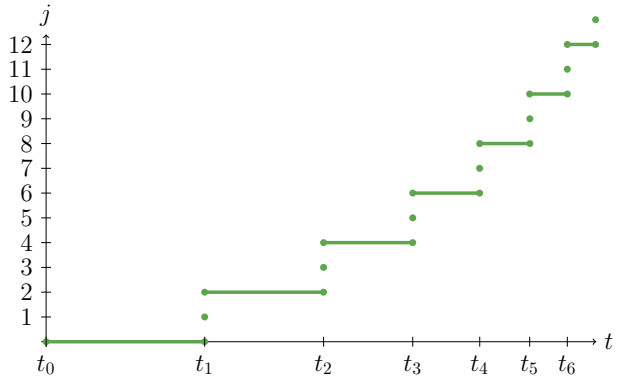

Figure 7: Hybrid time domain for the sample trajectory of a bouncing ball with discrete time step j and continuous time t

What we already observe about the bouncing ball is that its trajectory follows an alternating succession of a continuous trajectory following (8) for a certain nonzero duration and an instantaneous discrete jump following (9) at a discrete instant of time. This succession of continuous and discrete transitions in Fig. 6 gives rise to the hybrid time domain T shown in Fig. 7. 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 (8), 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 (9). 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 discrete transition of bouncing happened. Fig. 8 shows the particular sample trajectory of the bouncing ball from Fig. 6 plotted on its corresponding hybrid time domain T from Fig. 7. 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.

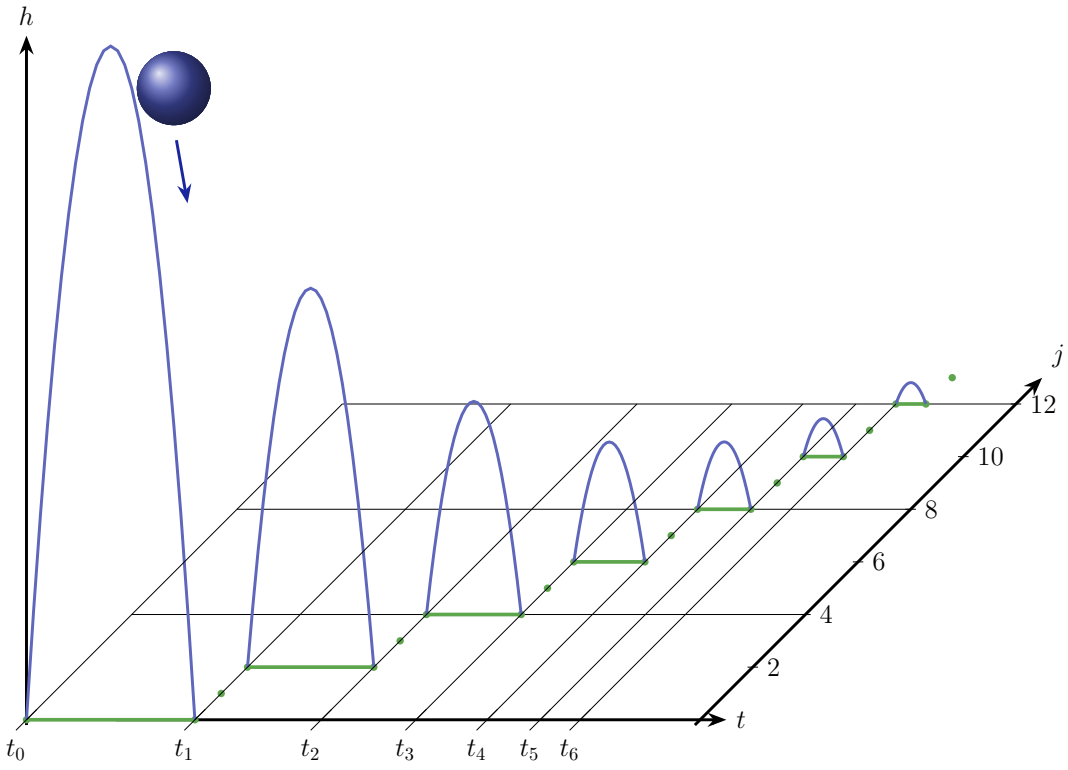

Figure 8: Sample trajectory of a bouncing ball plotted as position h over its hybrid time domain with discrete time step j and continuous time t

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. 7 is specific to the particular bouncing ball trajectory from Fig. 6 and Fig. 8 and does not fit to any other bouncing ball trajectories. This is in sharp contrast to the principles of general dynamical systems $\varphi : T \times X \to X$, in which the time domain T and state space X for φ are supposed to be a single set for all trajectories, and not depend on the particular sample trajectory considered so far. This is one of the reasons why we do not adopt the approach of working with hybrid times [47], but instead, leave time implicit in our models. If time is ever needed in a system, it can simply be added as a dedicated clock variable c with differential equation $c' = 1$ to the model.

Hybrid systems are just highly nondeterministic, even in their notion of time, which is a scenario to which programming language and formal language models are better adapted than the general dynamical systems model. Even the interaction of discrete and continuous dynamics is often characterized by nondeterminism, since there is not always just one point in time where control can pass from discrete to continuous or back. Nondeterminism, of course, breaks the staging property illustrated in Fig. 2, and requires a set-valued treatment to recover if only a fixed time domain T could be found. Having said that, it is perfectly possible to fit hybrid dynamical systems into the model of general dynamical systems. All it takes is a more sophisticated notion of time that remembers all previous actions (similar to the actions in the operational semantics of hybrid games [49]) and allows permanent forking of the subsequent execution to different futures. But these technical complications are unnecessary when working in a clean programming language (Section 3).

Before we give up on hybrid time domains, however, we illustrate two more phenomena that are worth noticing: subdivision and super-dense computations. While Fig. 7 shows one hybrid time domain for the sample trajectory in Fig. 6, there are infinitely many other hybrid time domains that fit to the original sample trajectory shown in Fig. 6 and just subdivide one of the intervals of a flying phase into two subintervals during which the ball just keeps on flying according to (8) 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 (8) followed by a continuous phase where the ball is flying down, still according to (8). 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.

Computational models. Like in the case of all other dynamical systems, hybrid systems need to be represented in suitable computational models to have a chance to be amenable to any form of computational analysis. There is a range of models for hybrid systems [50], including hybrid automata [51, 52] and its variations [53], process-algebraic models [54, 55], Petri nets [56], and programs [9–12]. All hybrid systems provide some form of discrete transitions and (various classes of) differential equations, but differ in terms of how those pieces are put together to form the hybrid systems. The representational differences may have important impact on the ease of analysis but are not fundamental, because translations between the models are possible at least in some cases [12, 55, 56].

Numerical approximation problem. What is important to realize for hybrid systems is the permanent presence of the numerical approximation problem, which, in terms of its ubiquity, is a numerical analogue of the halting problem. Verification of hybrid systems is a very challenging problem. The verification problem is the problem to decide whether a given hybrid system satisfies a given correctness property. Unfortunately, this problem is undecidable even for very simple hybrid systems [5, 57]. Even for absurdly limited models of hybrid systems, the verification problem is neither semidecidable nor co-semidecidable numerically, even for a bounded number of transitions and when tolerating arbitrarily large error bounds in the decision [31]. Minimal black box models of hybrid systems that only support numerical evaluation of the system and its derivatives at points are insufficient, because they lead to numerical undecidability even when tolerating arbitrarily large error bounds. That is why some form of additional input or symbolic representations are required in order to guarantee that analysis results can be correct.

The basic intuition behind the numerical undecidability result is shown in Fig. 9. Suppose an algorithm could decide safety of a system numerically by evaluating the value of the system flow φ at points. If the algorithm is a decision algorithm, it would have to terminate in finite time, hence, after evaluating a finite number of points, say x_1, x_2, x_3 in Fig. 9. But from the information that the algorithm has gathered at a finite number of points, it cannot distinguish the good behavior φ (solid flow safely outside B) from the bad behavior g (dashed flow reaching bad region B). The same undecidability result still holds even when restricting the flow φ to very special classes of functions and

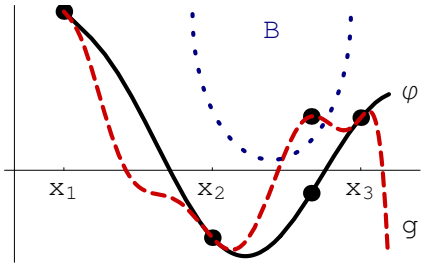

Figure 9: Safe and unsafe indistinguishable by $\varphi^{(j)}(x_i)$ (for $j \leq 2$)

when assuming that its derivatives $\varphi^{(j)}(x_i)$ could be evaluated and even when tolerating arbitrarily large error bounds in the decision. There is a series of extra assumptions and bounds that make the problem (approximately) decidable again by imposing extra constraints on the system. Yet, by the general undecidability result, these extra bounds (and several other bounds that have been proposed in related work) cannot be computed numerically. Because of this strong numerical undecidability result, it is surprisingly difficult but not impossible to get hybrid systems verification techniques sound using symbolic representations and/or assuming knowledge of the behavior of the system on intervals [12, 16, 58].

Limits of hybrid dynamical systems. Not all systems are hybrid systems. Some have more general effects that pure hybrid systems cannot represent properly. Yet, there are many interesting extensions of hybrid systems.

Distributed hybrid systems [59–66] are dynamical systems that combine distributed systems [26, 67, 68] with hybrid systems, and can, thus, model systems of systems aspects in hybrid systems (with their discrete and continuous dynamics). Distributed systems are systems consisting of multiple computers that interact through a communication network. They feature both (discrete) local computation and remote communication. Distributed hybrid systems, instead, consist of multiple hybrid systems that interact through a communication network, but may also interact through physical interactions. Distributed hybrid systems include multi-agent hybrid systems and hybrid systems where the number of agents involved in the system evolves over time. Typical examples of distributed hybrid systems are fleets of unmanned aerial vehicles or a platoon of cars on a highway.

Stochastic hybrid systems [62, 69–76] are dynamical systems that combine the dynamics of stochastic processes [77–79] with hybrid systems. They either feature stochastic effects only during the discrete dynamics [69] or during the continuous dynamics [70] or both [72, 73, 76]. Stochastic hybrid systems play a role when systems have a large degree of random noise and good probabilistic models are available for their distributions.

Hybrid games [49, 80–87] extend hybrid systems with adversarial effects coming from multiple players with different goals in the hybrid system. Hybrid games are relevant when it is important to understand how different agents with different goals might interact.

3 Models of Computation: Hybrid Programs

Hybrid programs (HP) [9, 12, 14, 88] are a programming language for hybrid systems. HPs combine differential equations with conventional program constructs and discrete assignments. In order to highlight the design features of HPs, we first take a detour with a

hybrid version of the programming language C.

Hybrid C. One way to think of HPs is to understand them as regular imperative programs that can additionally use differential equations as program statements. That intuition goes a long way except that it misses out on the other important feature of hybrid systems: their ubiquitous nondeterminism. We will, nevertheless, start this exposition first with this more narrow perspective of adding differential equations into conventional discrete programs and see where that gets us. To make things concrete, we consider a programming language with a notation akin to C, although any other imperative programming language would work as well. Let us call this programming language *Hybrid C*, since it is essentially C with differential equations.

The first attempt of representing the bouncing ball Example 2.6 in Hybrid C could be:

```
while (*) {
  if (h == 0) {
    v := -c*v;
  }
  h'=v,v'=-g;
}
```

This Hybrid C program consists of a loop that will repeatedly check with an if statement whether the height h is zero and then reverse the velocity v by a discrete assignment. For emphasis we use the notation `:=` for assignments to make sure they are not confused with differential equations. The most obvious problem with this Hybrid C program is that it is not clear when the while loop should stop, because it unclear how long the ball will be bouncing. And even a system component stops moving, we might still want to consider that a valid behavior for some while, e.g., until all other system components stopped as well. The right way of understanding hybrid systems is usually that they repeat nondeterministically any number of times, which we indicate by `while(*)` in Hybrid C.

Now the next problem with the above Hybrid C program is that it is unclear how long the system will follow the differential equation statement `h'=v,v'=-g`. Indeed, how long exactly a system follows a continuous dynamics before a discrete step happens again is usually highly nondeterministic. Even for time-triggered architecture implementations that are trying to operate at certain fixed frequencies, such as 10Hz, practice still holds phenomena like jitter in store, which cause variations in the time of operation. Indeed, for the bouncing ball, 10Hz or any other fixed sampling period would be unsuitable, because the system execution will never hit the interesting condition `if (h == 0)` that way.[4] Conse-

[4]This problem is intimately related to the zero-crossing problem in numerical algorithms. Indeed, floating-point algorithms approximating the executions of the Hybrid C program, e.g., by an Euler inte-

quently, the natural mode for a differential equation is that it evolves for a nondeterministic amount of time, just like `while(*)`.

Yet, hold on, the above Hybrid C program would also get in trouble if the differential equation evolved for too long. In that case, the ball would fall through the ground to a negative height ($h < 0$) and will then keep on falling forever, because the condition `if (h == 0)` will never be able to fire and rescue the ball by changing the sign of its velocity again. That would be a sad loss of a perfectly reasonable bouncing ball. Consequently, differential equations need to be constrained to remain within certain regions called evolution domains. The relevant evolution domain for the bouncing ball is $h \geq 0$, because physics constrains the ball to remain above the ground. The notation we will adopt to indicate that a continuous system follows a differential equation such as $h' = v, v' = -g$ only within such an evolution domain is conjunctively (&) as follows:

$$h' = v, v' = -g \,\&\, h \geq 0$$

Basic concept. Hybrid systems frequently exhibit nondeterminism in its various forms, including in the discrete control structure and continuous dynamics. Nondeterminism should, thus, be a first-class citizen in hybrid systems programming languages. That is why the programming language of hybrid programs [9, 12, 14, 88] embraces nondeterminism. In fact, hybrid programs make nondeterminism the norm and allow deterministic constructs as abbreviations for certain patterns of nondeterministic program operators. All classical programming constructs are definable in terms of the operators that hybrid programs provide.

HPs form a Kleene algebra with tests [89], that is, they are formed like regular expressions [90] just with more difficult atomic programs instead of letters of a finite alphabet. Atomic HPs are instantaneous discrete jump *assignments* $x := \theta$, *tests* $?H$ of a first-order formula[5] H of real arithmetic, and *differential equation (systems)* $x' = \theta \,\&\, H$ for a continuous evolution restricted to the domain of evolution H, where x' denotes the time-derivative of x. Compound HPs are generated from atomic HPs by nondeterministic choice ($\cup$), sequential composition (;), and Kleene's nondeterministic repetition (*). As *terms*, we use polynomials with rational coefficients here, but divisions can be allowed as well when guarding against singularities of divisions by zero; see [9, 12] for details.

Definition 3.1 (Hybrid program)**.** HPs are defined by the following grammar (α, β are HPs, x a variable, θ a term possibly containing x, and H a formula of first-order logic of

gration for the differential equation, will almost never satisfy the test `if (h == 0)`. Real executions of the bouncing ball, though, have no trouble finding when the height is zero and reacting appropriately. This problem is looming in some form or another in almost all simulation tools.

[5] The test $?H$ means "if H then *skip* else *abort*".

real arithmetic):

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

The first three cases are called atomic HPs, the last three compound. The *test* action $?H$ is used to define conditions. Its effect is that of a *no-op* if the formula H is true in the current state; otherwise, like *abort*, it allows no transitions so that system cannot execute. That is, if the test succeeds because formula H holds in the current state, then the state does not change, but the system execution continues normally. If the test fails because formula H does not hold in the current state, then the system cannot execute and such runs with failed tests are discarded and not considered any further.

Nondeterministic choice $\alpha \cup \beta$, sequential composition $\alpha; \beta$, and nondeterministic repetition α^* of programs are as in regular expressions but generalized to a semantics in hybrid systems. *Nondeterministic choice* $\alpha \cup \beta$ expresses behavioral alternatives between the runs of α and β. That is, the HP $\alpha \cup \beta$ can choose nondeterministically to follow the runs of HP α, or, instead, to follow the runs of HP β. The *sequential composition* $\alpha; \beta$ models that the HP β starts running after HP α has finished (β never starts if α does not terminate). In $\alpha; \beta$, the runs of α take effect first, until α terminates (if it does), and then β continues. Observe that, like repetitions, continuous evolutions within α can take more or less time, which causes uncountable nondeterminism. This nondeterminism occurs in hybrid systems, because they can operate in so many different ways, which is as such reflected in HPs. *Nondeterministic repetition* α^* is used to express that the HP α repeats any number of times, including zero times. When following α^*, the runs of HP α can be repeated over and over again, any nondeterministic number of times (≥ 0).

Example 3.2 (Single car)**.** As an example, consider a simple car control scenario. We denote the position of a car by x, its velocity by v, and its acceleration by a. From Newton's laws of mechanics, we obtain a simple kinematic model for the longitudinal motion of the car on a straight road, which can be described by the differential equation $x' = v, v' = a$. That is, the time-derivative of position is velocity ($x' = v$) and, simultaneously, the derivative of velocity is acceleration ($v' = a$). We restrict the car to never drive backwards by specifying the evolution domain constraint $v \geq 0$ and obtain the continuous dynamical system $x' = v, v' = a \,\&\, v \geq 0$. In addition, suppose the car controller can decide to accelerate (represented by $a := A$) or brake ($a := -b$), where $A \geq 0$ is a symbolic parameter for the maximum acceleration and $b > 0$ a symbolic parameter describing the brakes. The HP $a := -b \cup a := A$ describes a controller that can choose nondeterministically to brake or accelerate. Accelerating will only sometimes be a safe control decision, so the discrete controller in the following HP requires a test $?H$ to be passed in the acceleration choice:

$$car_s \equiv ((a := -b \cup (?H; a := A)); \ x' = v, v' = a \,\&\, v \geq 0)^* \tag{10}$$

This HP, which we abbreviate by car_s, first allows a nondeterministic choice of braking or acceleration (if the test H succeeds), and then follows the differential equation for an arbitrary period of time (that does not cause v to enter $v < 0$). The HP repeats nondeterministically as indicated by the * repetition operator. Note that the nondeterministic choice ($\cup$) in (10) can nondeterministically select to proceed with $a := -b$ or with $?H; a := A$. Yet the second choice can only continue if, indeed, formula H is true about the current state (then both choices are possible). Otherwise only the braking choice will run successfully, because the other choice will fail test $?H$ so that that run will be discarded. With this principle, HPs elegantly separate the fundamental principles of (nondeterministic) choice from conditional execution (tests).

Which formula is suitable for H depends on the control objective or property we care about. A simple guess for H like $v < 8$ has the effect that the controller can only choose to accelerate at lower speeds. This condition alone is insufficient for most control purposes and will leave the car possibly unsafe.

Semantics. HPs have a compositional semantics. We define their semantics by a reachability relation and refer to previous work for their trace semantics [12, 91]. The transition semantics of HP α is a relation $\rho(\alpha)$ defining which final states are reachable from which initial states by running α to completion. That is, $(\nu, \omega) \in \rho(\alpha)$ specifies that final state ω is reachable from the initial state ν by executing HP α. A *state* ν is a mapping from variables to $\mathbb{R}$. The set of states is denoted $\mathcal{S}$. We denote the value of term θ in ν by $[\![\theta]\!]_\nu$. The state ν_x^d agrees with ν except for the interpretation of variable x, which is changed to $d \in \mathbb{R}$. We write $\nu \models \chi$ iff the first-order formula χ is true in state ν (as defined formally in Section 4).

Definition 3.3 (Transition semantics of HPs)**.** Each HP α is interpreted semantically as a binary reachability relation $\rho(\alpha) \subseteq \mathcal{S} \times \mathcal{S}$ over states, defined inductively by

- $\rho(x := \theta) = \{(\nu, \omega) \,:\, \omega = \nu$ except that $[\![x]\!]_\omega = [\![\theta]\!]_\nu\}$
  That is, final state ω differs from initial state ν only in its interpretation of the variable x, which ω changes to the value that the right-hand side θ has in the initial state ω.

- $\rho(?H) = \{(\nu, \nu) \,:\, \nu \models H\}$
  That is, the final state ν is the same as the initial state ν (no change) but there only is such a self-loop transition if test formula H holds in ν, otherwise no transition is possible at all and the system is stuck because of a failed test.

- $\rho(x' = \theta \,\&\, H) = \{(\varphi(0), \varphi(r)) \,:\, \varphi(t) \models x' = \theta$ and $\varphi(t) \models H$ for all $0 \leq t \leq r$ for a solution $\varphi : [0, r] \to \mathcal{S}$ of any duration $r\}$
  That is, the final state $\varphi(r)$ is connected to the initial state $\varphi(0)$ by a continuous function of some duration $r \geq 0$ that solves the differential equation and satisfies H at all

times, when interpreting $\varphi(t)(x') \overset{\text{def}}{=} \frac{\mathrm{d}\varphi(\zeta)(x)}{\mathrm{d}\zeta}(t)$ as the derivative of the value of x over time [9].

- $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
  That is, $\alpha \cup \beta$ can do any of the transitions that α can do as well as any of the transitions that β is capable of.

- $\rho(\alpha;\beta) = \rho(\beta) \circ \rho(\alpha) = \{(\nu,\omega) : (\nu,\mu) \in \rho(\alpha), (\mu,\omega) \in \rho(\beta)\}$
  That is, $\alpha;\beta$ can do any transitions that go through any intermediate state μ to which α can make a transition from the initial state ν and from which β can make a transition to the final state ω.

- $\rho(\alpha^*) = \bigcup_{n\in\mathbb{N}} \rho(\alpha^n)$ with $\alpha^{n+1} \equiv \alpha^n;\alpha$ and $\alpha^0 \equiv ?\mathit{true}$.
  That is, α^* can repeat α any number of times, i.e., for any $n \in \mathbb{N}$, α^* can act like the n-fold sequential composition α^n would.

We refer to a book [12] for a comprehensive background and for an elaboration how the case $r = 0$ (in which the only condition is $\varphi(0) \models H$) is captured by the above definition for differential equations. Time itself does not play a special role. Whenever a clock variable t is needed in a HP, it can be axiomatized by $t' = 1$. Finally observe how easily the relational semantics of HPs deals with the ubiquitous nondeterminism of hybrid systems. The same simplicity can be obtained also for a trace semantics of hybrid programs that retains the intermediate states during hybrid trajectories [12, 91].

Example 3.4. Continuing Example 3.2, Fig. 10a illustrates the structure of the transition system of (10) for the (unsafe) choice of $H \equiv (v < 8)$. Fig. 10b illustrates how one particular transition from initial state ν to one final state follows the marked transitions through two iterations of the loop, which justifies $(\nu,\omega) \in \rho(car_s)$.

Definable operators. HPs only provide the logically fundamental operators of hybrid systems. All classical WHILE programming constructs and all hybrid systems can be defined from those fundamental operators [12] including the ones we alluded to in the development of the Hybrid C language. We, e.g., write $x' = \theta$ for the unrestricted differential equation $x' = \theta \,\&\, \mathit{true}$. We allow differential equation systems and use vectorial notation. Vectorial assignments are definable from scalar assignments and ; using auxiliary variables.[6] Other program constructs can be defined easily [12]. For example, nondeter-

[6] A vectorial assignment $x_1 := \theta_1, \ldots, x_n := \theta_n$ is definable by $\grave{x}_1 := x_1; \ldots; \grave{x}_n := x_n; x_1 := \grave{\theta}_1; \ldots; x_n := \grave{\theta}_n$ where $\grave{\theta}_i$ is θ_i with x_j replaced by $\grave{x}_j$ for all j. Memorizing the old values of x_j in $\grave{x}_j$ before assigning to x_i is necessary for a simultaneous vectorial assignment if θ_i mentions another x_j, which would already be overwritten if $j < i$.

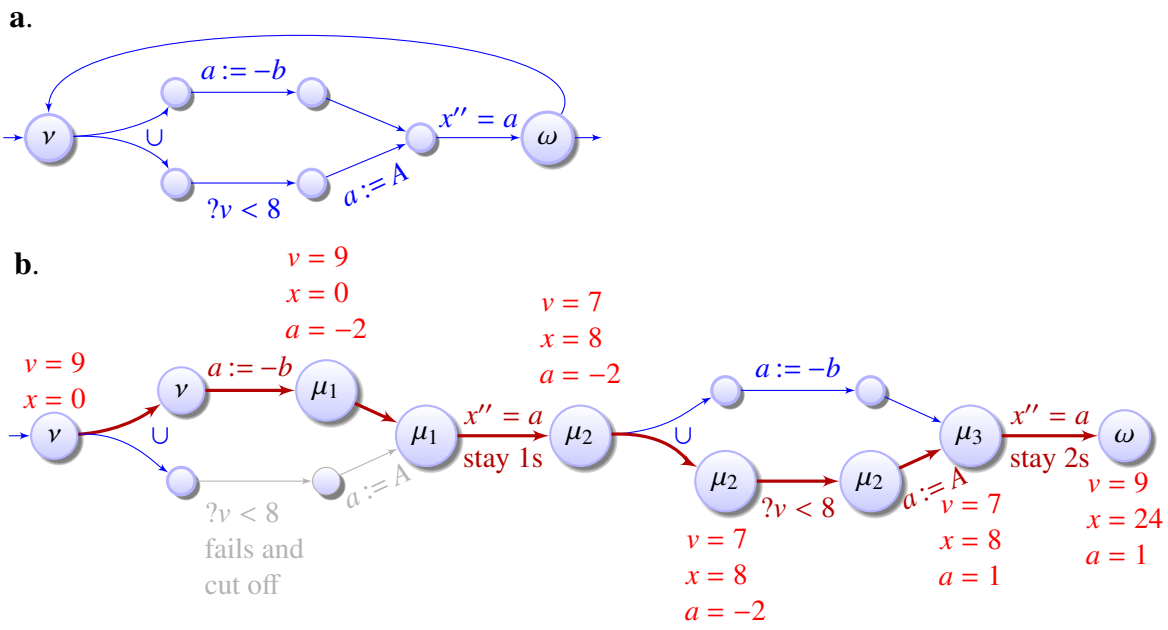

Figure 10: Transition structure and transition example in simple car

ministic assignments of any real value to x, if-then-else statements, and while loops can be defined by the following abbreviations, respectively:

$$\begin{aligned} x := * &\equiv x' = 1 \cup x' = -1 \\ \text{if } (H) \text{ then } \alpha \text{ else } \beta \text{ fi} &\equiv (?H;\alpha) \cup (?\neg H;\beta) \\ \text{if } (H) \text{ then } \alpha &\equiv (?H;\alpha) \cup ?\neg H \\ \mathsf{while}(H)\, \alpha &\equiv (?H;\alpha)^*; ?\neg H \end{aligned} \tag{11}$$

The reason why if (H) then α else β fi is the same as $(?H;\alpha) \cup (?\neg H;\beta)$, for example, is that, after the nondeterministic choice, exactly one of the two tests $?H$ and $?\neg H$ will succeed the other one will fail. Hence, even though the right-hand side of (11) starts out with a nondeterministic choice, only one choice will ever work out from any current state. That is, the only possible nondeterministic choices that are not aborted and discarded because of failing a subsequent test are those in which H holds and α executes or in which $\neg H$ holds and β executes. Since the if-then-else makes this determinism apparent that is implicit in the mutual exclusiveness of the test conditions, if-then-else is directly supported in the implementation of $\mathsf{d}\mathcal{L}$ in the theorem prover KeYmaera [17] even if it is not needed in theory.

Nondeterministic assignment $x := *$ assigns any real number to the variable x and is

frequently used in hybrid system models to represent that arbitrary control choices are possible. Often, those arbitrary control choices are subsequently restricted to a possible range using a test.

Example 3.5 (Bouncing ball)**.** Continuing Example 2.6, consider a hybrid program model of the bouncing ball:

$$
\begin{array}{l}
(\\
\quad \text{if } (h = 0) \text{ then}\\
\quad\quad c := *;\ ?(0 \leq c < 1);\\
\quad\quad v := -cv\\
\quad \text{fi};\\
\quad h' = v, v' = -g \,\&\, h \geq 0\\
)^*
\end{array}
$$

The if-then statement can be expanded using the definitions in (11), which leads to the hybrid program

$$
\begin{array}{l}
(\\
\quad (?(h = 0);\\
\quad\quad\quad c := *;\ ?(0 \leq c < 1);\\
\quad\quad\quad v := -cv\\
\quad\quad) \cup (?h \neq 0);\\
\quad h' = v, v' = -g \,\&\, h \geq 0\\
)^*
\end{array}
$$

Observe in both hybrid programs how the damping coefficient c is set to an arbitrary real number by way of $c := *$ and then subsequently restricted by the test $?(0 \leq c < 1)$ to lie within the interval $[0, 1)$. The overall effect of $c := *; ?(0 \leq c < 1)$ is to assign an arbitrary real number from $[0, 1)$ to c. This is a frequent modeling pattern to have a nondeterministic assignment followed by a test with the requisite range restrictions.

Hierarchies. Hybrid programs are designed as a minimal extension of conventional discrete programs. They characterize hybrid systems succinctly by adding continuous evolution along differential equations as the only additional primitive operation to a regular basis of conventional discrete programs. Their operations are interpreted over the domain of real numbers as required for hybrid systems. This gives rise to an elegant syntactic hierarchy [12] of discrete, continuous, and hybrid systems, for which the respective fragments of hybrid programs are a computational model, summarized in Table 1. The fragment consisting of just differential equations with evolution domain constraints corresponds to purely continuous dynamical systems [92]. The fragment of hybrid programs without differential equations corresponds to conventional discrete programs generalized over the reals or to discrete-time dynamical systems [93]. The fragment without discrete

assignments corresponds to switched continuous systems [6, 93]. Only the composition of mixed discrete assignments and continuous evolutions gives rise to truly hybrid behavior.

Table 1: Classification of hybrid programs and correspondence to dynamical systems

Hybrid program class	Dynamical systems class
differential equations	continuous dynamical systems
no assignments	switched continuous dynamical systems
no differential equations	discrete dynamical systems
no differential equations, over $\mathbb{N}$	discrete while programs
general hybrid programs	hybrid dynamical systems

4 Logical Characterizations of Hybrid Systems

Basic concept. Within a single specification and verification language, differential dynamic logic $\mathsf{d}\mathcal{L}$ [9, 12, 14, 88] combines operational system models with means to talk about the states that are reachable by system transitions. Differential dynamic logic $\mathsf{d}\mathcal{L}$ is a dynamic logic [94, 95] for hybrid systems. It combines first-order real arithmetic [96] with first-order modal logic [97, 98] and dynamic logic [94, 95] generalized to hybrid systems. (Nonlinear) real arithmetic is necessary for describing concepts like safe regions of the state space and real-valued quantifiers are for quantifying over the possible values of system parameters or states.

The logic $\mathsf{d}\mathcal{L}$ provides parametrized modal operators $[\alpha]$ and $\langle\alpha\rangle$ that refer to the states reachable by hybrid program α and can be placed in front of any formula. The modal operators $[\alpha]$ and $\langle\alpha\rangle$ refer to all (modal operator $[\alpha]$) or some (modal operator $\langle\alpha\rangle$) state reachable by following HP α. The formula $[\alpha]\phi$ expresses that all states reachable by hybrid program α satisfy formula ϕ. Likewise, $\langle\alpha\rangle\phi$ expresses that there is at least one state reachable by α for which ϕ holds. These modalities can be used to express necessary or possible properties of the transition behavior of α in a natural way. They can be nested or combined propositionally. The logic $\mathsf{d}\mathcal{L}$ supports quantifiers like $\exists p\,[\alpha]\langle\beta\rangle\phi$ which expresses that there is a choice of parameter p (expressed by $\exists p$) such that for all possible behaviors of hybrid program α (expressed by $[\alpha]$) there is a reaction of hybrid program β (i.e., $\langle\beta\rangle$) that ensures ϕ. The logic $\mathsf{d}\mathcal{L}$ is entirely flexible, so the parameter p that is quantified in these formulas may appear in the hybrid programs α,β as a system parameter as well as in the formula ϕ, where it would then be a parameter in the postcondition.

Definition 4.1 ($\mathsf{d}\mathcal{L}$ formula)**.** The *formulas of differential dynamic logic* ($\mathsf{d}\mathcal{L}$) are defined

by the grammar (where ϕ, ψ are dL formulas, θ_1, θ_2 terms, x a variable, α a HP):

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

Operators $>, \leq, <, \vee, \rightarrow, \leftrightarrow$ can be defined as usual in classical logic, e.g., $(\phi \rightarrow \psi) \equiv (\neg\phi \vee \psi)$. We use the notational convention that quantifiers and modal operators bind strong, i.e., their scope only extends to the formula immediately after. Thus, $[\alpha]\phi \wedge \psi \equiv ([\alpha]\phi) \wedge \psi$ and $\forall x\, \phi \wedge \psi \equiv (\forall x\, \phi) \wedge \psi$. In our notation, we also let $\neg$ bind stronger than $\wedge$, which binds stronger than $\vee$, which binds stronger than $\rightarrow, \leftrightarrow$. Thus, $\neg A \wedge B \vee C \rightarrow D \vee E \wedge F \equiv (((\neg A) \wedge B) \vee C) \rightarrow (D \vee (E \wedge F))$.

A dL formula is valid if it is true in all states (as will be defined in Def. 4.3 below). One common use case is the dL formula $A \rightarrow [\alpha]B$, which corresponds to a Hoare triple [99, 100], but for hybrid systems. It is valid if, for all states: if the dL formula A holds (in the initial state), then the dL formula B holds for all states reachable by following the HP α. That is, $A \rightarrow [\alpha]B$ is valid if B holds in all states reachable by HP α from initial states satisfying A.

Example 4.2 (Single car)**.** First, consider a very simple dL formula:

$$v \geq 0 \wedge A \geq 0 \rightarrow [a := A;\ x' = v, v' = a]v \geq 0$$

This dL formula expresses that, when, initially, the velocity v and maximal acceleration A are nonnegative, then all states reachable by the HP in the $[\cdot]$ modality have a nonnegative velocity ($v \geq 0$). The HP first performs a discrete assignment $a := A$ setting the acceleration a to maximal acceleration A, and then, after the sequential composition (;), follows the differential equation $x' = v, v' = a$ where the derivative of the position x is the velocity ($x' = v$) and the derivative of the velocity is the chosen acceleration a ($v' = a$). This dL formula is valid, because the velocity will never become negative when accelerating. It could, however, become negative when choosing a negative acceleration $a < 0$, which is what this simple dL formula does not allow.

Next, consider the following dL formula, where car_s denotes the HP from (10) in Example 3.2 that always allows braking but acceleration only when $\chi \equiv v \leq 20$ holds:

$$v \geq 0 \wedge A \geq 0 \wedge b > 0 \rightarrow [car_s]v \geq 0$$

This dL formula is trivially valid, simply because the postcondition $v \geq 0$ is implied by both the precondition and by the evolution domain constraint of (10). Because the invariant is (trivially) implied by the precondition, $v \geq 0$ also holds initially. It is also implied by the evolution domain constraint and the system has no runs that leave the evolution domain constraint. Note that this dL formula would not be valid, however, if we removed the evolution domain constraint, because the controller would then be allowed nondeterministically to choose a negative acceleration ($a := -b$) and stay in the continuous evolution arbitrarily long.

Semantics. The meaning of differential dynamic logic is a suitable combination of the semantics of first-order real arithmetic [96], first-order modal logic [97, 98], and dynamic logic [94, 95]. The semantics defines, which formula ϕ is true in which state ν. We write $\nu \models \phi$ if ϕ is true in state ν.

Definition 4.3 ($\mathsf{d}\mathcal{L}$ semantics)**.** The *satisfaction relation* $\nu \models \phi$ for $\mathsf{d}\mathcal{L}$ formula ϕ in state ν is defined inductively and as usual in first-order modal logic (of real arithmetic):

- $\nu \models (\theta_1 = \theta_2)$ iff $[\![\theta_1]\!]_\nu = [\![\theta_2]\!]_\nu$
  That is, an equation is true in a state ν iff the terms on both sides evaluate to the same number.

- $\nu \models (\theta_1 \geq \theta_2)$ iff $[\![\theta_1]\!]_\nu \geq [\![\theta_2]\!]_\nu$
  That is, a greater-or-equals inequality is true in a state ν iff the term on the left evaluate to a number that is greater or equal to the value of the right term.

- $\nu \models \neg\phi$ iff it is not the case that $\nu \models \phi$
  That is, a negated formula $\neg\phi$ is true in state ν iff the formula ϕ itself is not true in ν.

- $\nu \models \phi \wedge \psi$ iff $\nu \models \phi$ and $\nu \models \psi$
  That is, a conjunction is true in a state iff both conjuncts are true in said state.

- $\nu \models \forall x\, \phi$ iff $\nu_x^d \models \phi$ for all $d \in \mathbb{R}$
  That is, a universally quantified formula $\forall x\, \phi$ is true in a state iff its kernel ϕ is true in all variations of the state, no matter what real number d the quantified variable x evaluates to in the variation ν_x^d.

- $\nu \models \exists x\, \phi$ iff $\nu_x^d \models \phi$ for some $d \in \mathbb{R}$
  That is, an existentially quantified formula $\exists x\, \phi$ is true in a state iff its kernel ϕ is true in some variation of the state, for a suitable real number d that the quantified variable x evaluates to in the variation ν_x^d.

- $\nu \models [\alpha]\phi$ iff $\omega \models \phi$ for all ω with $(\nu, \omega) \in \rho(\alpha)$
  That is, a box modal formula $[\alpha]\phi$ is true in state ν iff postcondition ϕ is true in all states ω that are reachable by running α from ν.

- $\nu \models \langle\alpha\rangle\phi$ iff $\omega \models \phi$ for some ω with $(\nu, \omega) \in \rho(\alpha)$
  That is, a diamond modal formula $\langle\alpha\rangle\phi$ is true in state ν iff postcondition ϕ is true in at least one state ω that is reachable by running α from ν.

If $\nu \models \phi$, then we say that ϕ is true at ν. A $\mathsf{d}\mathcal{L}$ formula ϕ is *valid*, written $\vDash \phi$, iff $\nu \models \phi$ for all states ν.

Axiomatization. Differential dynamic logic dL is not just a specification language but also a verification language for hybrid systems. The logic dL comes with an axiomatization in proof calculi, including a Gentzen-type sequent calculus suitable for automation [9] as well as a Hilbert-type calculus characterizing the logical essentials [14]. Using this axiomatization, interesting properties of hybrid systems can be verified by a proof from the axioms. The Hilbert-type axiomatization of differential dynamic logic [13, 14] is shown in Fig. 11. Here, we highlight a few rules and refer to prior work [13, 14] for a detailed explanation of the axiomatization.

[:=] $[x := \theta]\phi(x) \leftrightarrow \phi(\theta)$

[?] $[?H]\phi \leftrightarrow (H \rightarrow \phi)$

[′] $[x' = \theta]\phi \leftrightarrow \forall t{\geq}0\, [x := y(t)]\phi$ $(y'(t) = \theta)$

[&] $[x' = \theta \,\&\, H]\phi \leftrightarrow \forall t_0{=}x_0\, [x' = \theta]([x' = -\theta](x_0 \geq t_0 \rightarrow H) \rightarrow \phi)$

[∪] $[\alpha \cup \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi$

[;] $[\alpha;\beta]\phi \leftrightarrow [\alpha][\beta]\phi$

[*] $[\alpha^*]\phi \leftrightarrow \phi \wedge [\alpha][\alpha^*]\phi$

K $[\alpha](\phi \rightarrow \psi) \rightarrow ([\alpha]\phi \rightarrow [\alpha]\psi)$

I $[\alpha^*](\phi \rightarrow [\alpha]\phi) \rightarrow (\phi \rightarrow [\alpha^*]\phi)$

C $[\alpha^*]\forall v{>}0\, (\varphi(v) \rightarrow \langle\alpha\rangle\varphi(v-1)) \rightarrow \forall v\, (\varphi(v) \rightarrow \langle\alpha^*\rangle\exists v{\leq}0\, \varphi(v))$ $(v \notin \alpha)$

B $\forall x\, [\alpha]\phi \rightarrow [\alpha]\forall x\, \phi$ $(x \notin \alpha)$

V $\phi \rightarrow [\alpha]\phi$ $(FV(\phi) \cap BV(\alpha) = \emptyset)$

G $\dfrac{\phi}{[\alpha]\phi}$

Figure 11: Differential dynamic logic axiomatization

We write $\vdash \phi$ iff dL formula ϕ can be *proved* with dL rules from dL axioms (including first-order rules and axioms); see Fig. 11. That is, a dL formula is inductively defined to be *provable* in the dL calculus if it is an instance of a dL axiom or if it is the conclusion (below the rule bar) of an instance of one of the dL proof rules Gödel generalization G,

modus ponens, ∀-generalization, whose premises (above the rule bar) are all provable. The dℒ axiomatization is sound and relatively complete [9, 14].

In axiom [′], $y(\cdot)$ is the (unique [34, Theorem 10.VI]) solution of the symbolic initial-value problem $y'(t) = \theta, y(0) = x$. Given such a solution $y(\cdot)$, continuous evolution along that differential equation can be replaced by a discrete assignment $x := y(t)$ with an additional quantifier for the evolution time t. It goes without saying that variables like t are fresh in Fig. 11. Notice that conventional initial-value problems are numerical with concrete numbers $x \in \mathbb{R}^d$ as initial values, not symbols x [34]. This would not be enough for our purpose, because we need to consider all states in which the system could start, which may be uncountably many. That is why axiom [′] solves one symbolic initial-value problem, because we could hardly solve uncountable many numerical initial-value problems. The side condition that $y(\cdot)$ is, indeed, a solution of the symbolic initial-value problem is decidable for simple solutions (such as polynomials). For more complicated differential equations, differential invariants and related techniques [10, 101, 102] are used to prove properties of differential equations by induction.

Sequential compositions are proven using nested modalities in axiom [;]. From right to left: If, after all α-runs, all β-runs lead to states satisfying ϕ (i.e., $[\alpha][\beta]\phi$ holds), then also all runs of the sequential composition $\alpha;\beta$ lead to states satisfying ϕ (i.e., $[\alpha;\beta]\phi$ holds). The converse implication uses the fact that if after all α-run all β-runs lead to ϕ (i.e., $[\alpha][\beta]\phi$), then all runs of $\alpha;\beta$ lead to ϕ (that is, $[\alpha;\beta]\phi$), because the runs of $\alpha;\beta$ are exactly those that first do any α-run, followed by any β-run. Again, it is crucial that dℒ is a full logic that considers reachability statements as modal operators, which can be nested, for then both sides in [;] are dℒ formulas again (unlike in Hoare logic [100], where intermediate assertions need to be guessed or computed as weakest preconditions for β and ϕ). Note that dℒ can directly express weakest preconditions, because the dℒ formula $[\beta]\phi$ or any formula equivalent to it already is the weakest precondition for β and ϕ. Strongest postconditions are expressible in dℒ as well.

Axiom I is an induction schema for repetitions. Axiom I says that, if, after any number of repetitions of α, invariant ϕ remains true after one (more) iteration of α (i.e., $[\alpha^*](\phi \to [\alpha]\phi)$), then ϕ holds after any number of repetitions of α (i.e., $[\alpha^*]\phi$) if ϕ holds initially. That is, if ϕ is true after running α whenever ϕ has been true before, then, if ϕ holds in the beginning, ϕ will continue to hold, no matter how often we repeat α in $[\alpha^*]\phi$.

The dℒ axiomatization in Fig. 11 uses a modular axiom [&] that reduces differential equations with evolution domain constraints to differential equations without them by checking the evolution domain constraint backwards along the reverse flow. It checks H backwards from the end of the evolution up to the initial time t_0, using that $x' = -\theta$ follows the same flow as $x' = \theta$, but backwards. See prior work for an elaboration and more details [13].

5 Hybrid Relations between Discrete and Continuous Dynamical Systems

Discrete dynamical systems and continuous dynamical systems start out on quite different premises, emphasizing step-wise discrete successions of change (Section 2.2) versus smooth or continuous forms of change (Section 2.3), respectively. That makes discrete and continuous dynamical systems and, thus, discrete and continuous computation, appear to be fundamentally and characteristically different. In fact, this difference was one important original motivation for inventing hybrid systems in the first place (Section 2.4) as a way of describing how two independent and different sources of dynamical behavior combine [2, 51, 103]; we refer to the literature for a review of the history of hybrid systems [104]. Of course, this also makes analysis questions of hybrid systems highly undecidable (not even semidecidable) and hybrid systems logics necessarily incomplete, because they combine two independent sources of incompleteness [9], the discrete and the continuous. Each of those sources of incompleteness follow by a simple corollary [9] to Gödel's incompleteness theorem [105].

Surprisingly, however, it turns out that discrete and continuous dynamics are not even quite so unrelated [9, 14]. For example, it has been shown that three-dimensional differential equations [106, 107] can simulate universal Turing machines on the relevant grid points. In an extended sense with approximation and robustness, so can polynomial differential equations [108]. The basic observations making these results happen is that Turing machines only take on values on a grid in time and space. That is, as discrete dynamical systems, they produce state change at a certain rate, say, 1 computation step per second, since $T = \mathbb{Z}$ or $T = \mathbb{N}$. They also only take on state values from a discrete set, say $T = \mathbb{Z}^d$. In a nutshell, continuous dynamical systems can be made to agree with the intended computations of a classical discrete Turing machine on a discrete grid, say $\mathbb{Z}^d$, that is chosen to correspond to the discrete states of the discrete dynamical system of a classical Turing machine. At the values off the grid, the continuous dynamical system can take on any value to continuously move from the previous state at time $n \in \mathbb{N}$ to the next state at time $n + 1$. Conversely, computability results for solutions of differential equations hold on open sets under existence and uniqueness assumptions and when rational interval approximations are given [109], which are necessary assumptions [31]. This result is based on enumerating all tubes around solutions and checking whether a tube covers the solution with the required accuracy.

What about general discrete dynamical systems, which, like Turing machines, have a discrete time domain $T = \mathbb{N}$, but, unlike classical Turing machines, can compute on a dense continuous state space $\mathcal{X} = \mathbb{R}^d$ rather than on a discrete $\mathcal{X} = \mathbb{Z}^d$ or even finite state space $\mathcal{X} = \{0, 1\}^d$ like Turing machines do? In that case, the relevant states are the dense

set $\mathbb{R}^d$, not just the grid $\mathbb{Z}^d$ with arbitrary values off grid, i.e., on $\mathbb{R}^d \setminus \mathbb{Z}^d$. Can discrete dynamical systems be simulated in some sense by continuous dynamical systems?

And what about hybrid systems? Hybrid systems mix discrete and continuous dynamics. Can their mixed discrete and continuous behavior be captured in some way using continuous dynamics alone? What if its behavior consists of some fixed finite number of discrete and continuous transitions? What if the hybrid system performs an arbitrary unknown number of repetitions of interactions of discrete and continuous dynamics like they usually do?

What about the other way around? Since at least some discrete dynamical systems like Turing machines can be emulated in continuous systems, can continuous systems also somehow be characterized in discrete systems?

Naïve ways of relating discrete and continuous dynamical systems are bound to fail. It is, for example, not generally the case that a property F transfers from a continuous system to its Euler discretization, nor vice versa. That is, neither the following equivalence nor the left-to-right implication nor the right-to-left implication generally holds:

$$[x' = \theta]F \overset{?}{\leftrightarrow} [(x := x + h\theta)^*]F \tag{12}$$

This formula would relate a property F of a continuous dynamical system $x' = \theta$ to property F of its Euler discretization $(x := x + h\theta)^*$ with discretization step size $h > 0$ *if only it were true*. Unfortunately, as such, the formula is not generally valid. Fig. 12 illustrates a counterexample to formula (12) from prior work [14], to which we refer for further details. The error of the Euler discretization grows quickly compared to the true solution in Fig. 12. For example, $F \equiv (x^2 + y^2 = 1)$ is an invariant of the true solution but not its approximation. On the bright side, the error can be smaller for *some* (not all) smaller discretization steps h and the error is quite reasonable for a certain period of time.

These aspects are one corner stone for a *complete logical alignment* of discrete and continuous dynamics using constructive proof-theoretical techniques [14]. The key to understanding how discrete and continuous dynamics relate is via their joint generalization as hybrid systems in their logical characterizations as fragments of differential dynamic logic [14]. Hybrid systems have been aligned with both continuous dynamical systems [9] and with discrete dynamical systems [14] by constructive completeness arguments showing that all valid properties of hybrid systems are provable in the $\mathsf{d}\mathcal{L}$ axiomatization from elementary properties of continuous systems to which they reduce constructively and likewise for discrete systems [9, 14]. Since every discrete system is a hybrid system and every continuous system also is a hybrid system, these two reductions mutually align discrete and continuous systems with one another [14, 110]. That is, discrete and continuous systems can be related to one another indirectly after embedding both into the joint generalization of hybrid systems and then analyzing how hybrid systems relate to their fragments;

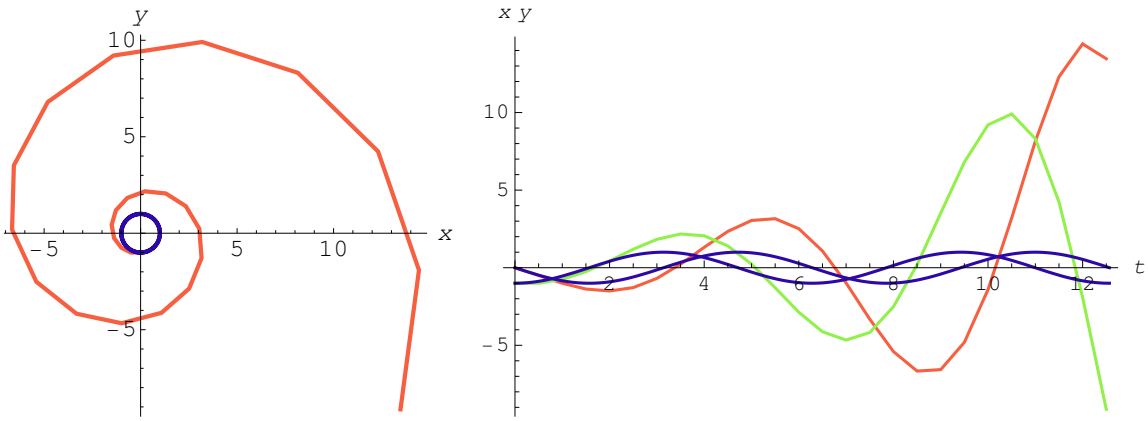

Figure 12: **(left)** Dark circle shows true solution, light line segments show Euler approximation for discretization step $h = \frac{1}{2}$ **(right)** Dark true bounded trigonometric solution and Euler approximation in lighter colors with increasing errors over time t

cf. Fig. 1.

From Hybrid to Continuous. Using the proof calculus of $\mathsf{d}\mathcal{L}$, the problem of proving properties of hybrid systems reduces completely to proving properties of elementary continuous systems [9].

Theorem 5.1 (Continuous relative completeness of $\mathsf{d}\mathcal{L}$ [9, 14])**.** *The $\mathsf{d}\mathcal{L}$ calculus is a* sound and complete axiomatization *of hybrid systems relative to differential equations, i.e., every valid $\mathsf{d}\mathcal{L}$ formula can be derived from elementary properties of differential equations.*

In particular, if we want to prove properties of hybrid systems, all we need to do is to prove properties of continuous systems, because the $\mathsf{d}\mathcal{L}$ calculus completely handles all other steps in the proofs that deal with discrete or hybrid systems. Of course, one has to be able to handle continuous systems in order to understand hybrid systems, because continuous systems are a special case of hybrid systems. But it turns out that this is actually all that one needs in order to verify hybrid systems, because the $\mathsf{d}\mathcal{L}$ proof calculus completely axiomatizes all the rest of hybrid systems.

Since the proof of Theorem 5.1 is constructive, there is even a complete constructive reduction of properties of hybrid systems to corresponding properties of continuous systems. The $\mathsf{d}\mathcal{L}$ calculus can prove hybrid systems properties exactly as good as properties of the corresponding continuous systems can be verified. One important step in the proof of Theorem 5.1 shows that all required invariants and variants for repetitions can be ex-

pressed in the logic dℒ. Furthermore, the dℒ calculus defines a decision procedure for dℒ sentences (i.e., closed formulas) relative to an oracle for differential equations [14].

This result implies that the continuous dynamics dominates the discrete dynamics since, once the continuous dynamics is handled, all discrete and hybrid dynamics can be handled as well. Therefore, verification of hybrid systems is not more complex than the verification of continuous systems. In particular, discrete systems verification is not more complex than the verification of continuous systems. This is reassuring, because we get the challenges of discrete dynamics solved for free (by the dℒ proof calculus) once we address continuous dynamics. In addition to its theoretical alignment of the landscape of complexity and reductions, this result emphasizes the importance of studying verification techniques for continuous systems, because the dℒ calculus makes those techniques hybrid.

From Hybrid to Discrete. In a certain sense, it may appear to be more complicated to handle continuous dynamics than discrete dynamics. If the continuous dynamics are not just subsuming discrete dynamics but if they were "inherently more", then one might wonder whether hybrid systems verification could be understood with a discrete dynamical system like a classical computer at all. Of course, such a naïve consideration would be quite insufficient, because, e.g., properties of objects in uncountable continuous spaces can very well follow from properties of finitary discrete objects. Finite dℒ proof objects, for example, already entail properties about uncountable continuous state spaces of systems.

Fortunately, all such worries about the insufficiency of discrete ways of understanding continuous phenomena can be settled once and for all by studying the proof-theoretical relationship between discrete and continuous dynamics. We have shown not only that the axiomatization of dℒ is complete relative to differential equations, but that it is also complete relative discrete systems [14].

Theorem 5.2 (Discrete relative completeness of dℒ [14])**.** *The* dℒ *calculus is a* sound and complete axiomatization *of hybrid systems relative to discrete systems, i.e., every valid* dℒ *formula can be derived from elementary properties of discrete systems.*

Thus, the dℒ calculus can also prove properties of hybrid systems exactly as good as properties of discrete systems can be proved. Again, the proof of Theorem 5.2 is constructive, entailing that there is a constructive way of reducing properties of hybrid systems to properties of discrete systems using the dℒ calculus. Furthermore, the dℒ calculus defines a decision procedure for dℒ sentences relative to an oracle for discrete systems [14]. Theorems 5.1 and 5.2 lead to a surprising result aligning discrete and continuous systems properties.

Theorem 5.3 (dℒ equi-expressibility [14])**.** *The logic* dℒ *is* expressible *in both its discrete and in its continuous fragment: for each* dℒ *formula* ϕ *there is a continuous formula* $\phi^\flat$ *that is equivalent, i.e.,* $\vDash \phi \leftrightarrow \phi^\flat$ *and a discrete formula* $\phi^\#$ *that is equivalent, i.e.,* $\vDash \phi \leftrightarrow \phi^\#$*. The converse holds trivially. Furthermore, the construction of* $\phi^\flat$ *and* $\phi^\#$ *is effective (and the equivalences are provable in the* dℒ *calculus).*

The proof of the surprising result Theorem 5.3 is constructive but rather nontrivial (some 20 pages). It uses a combination of Euler discretizations leading to "proof-uniform" approximations based on the existence of (not the values of) on-the-fly local Lipschitz bounds together with topological arguments on semi-algebraic base sets relating sets to quantified open neighborhoods and logical liftings using the Barcan axiom as well as real pairings by differential equations and relations between modalities and quantifiers. The usual challenges of evolution domain constraints are handled based on the "there and back again" axiom [&]. While several more efficient shortcuts exist, the overall proof is optimized for simplicity of the proof not for efficiency of the result, so it adds unnecessary complexity. But the proof also identifies cases, in which significantly more efficient reductions are possible, such as in the case of proving closed properties of open invariants. Whatever the added complexity may be, Theorem 5.3 does have interesting fundamental consequences.

Consequently, all hybrid questions (and, thus, also all discrete questions) can be formulated constructively equivalently as purely continuous questions and all hybrid questions (also all continuous questions) can be formulated constructively equivalently as purely discrete questions. There is a constructive and provable reduction from either side to the other.

As a corollary to Theorems 5.1 and 5.2, we can proof-theoretically and constructively equate

$$\text{hybrid} = \text{continuous} = \text{discrete}$$

by a *complete logical alignment* in the sense that proving properties of either of those classes of dynamical systems is the same as proving properties of any other of those classes, because all properties of one system can be provably reduced in a complete, constructive, and equivalent way to any of the other system classes. Even though each kind of dynamics comes from fundamentally different principles, they all meet in terms of their proof problems being interreducible, even constructively; recall Fig. 1. The proof problem of hybrid systems, the proof problem of continuous systems, and the proof problem of discrete systems are, thus, equivalent. Any proof technique for one of these classes of systems completely lifts to proof techniques for the other class of systems.

Since the proof problems interreduce constructively, every technique that is successful for one kind of dynamics lifts to the other kind of dynamics through the dℒ calculus

in a provably perfect way. Induction, for example, is the primary technique for proving properties of discrete systems. Hence, by Theorem 5.2, there is a corresponding induction technique for continuous systems and for hybrid systems. And, indeed, *differential invariants* [10, 101] are such an induction technique for differential equations that has been used very successfully for verifying hybrid systems with more advanced differential equations [12, 111–115]. In fact, differential invariants had already been introduced in 2008 [10] before Theorem 5.2 was proved [14], but Theorem 5.2 implies that a differential invariant induction technique has to exist. These results also show that there are sound ways of using discretization for differential equations [14] and that numerical integration schemes like, e.g., Euler's method or more elaborate methods can be used for hybrid systems verification, which is not at all clear a priori due to inherent numerical approximation errors, which may blur decisions either way [31].

Some ways of doing practical proof search and generation of invariants has been addressed in previous work [111, 112]. But many other proof search procedures could be useful to generate invariants more efficiently in practice. Such advances include, for example, techniques using the differential radical invariants extension of differential invariants [116] as well as combinations of differential invariants with Lie invariants [117] using differential cuts [10, 101]. Differential radical invariants provide a decision procedure for algebraic invariants of algebraic differential equations and a corresponding automatic invariant generation technique based on symbolic linear algebra [116]. Differential cuts, instead, generalize Gentzen's cut to differential equations but are fundamental, because they do not admit differential cut elimination [10, 101].

6 Conclusions and Future Work

This article gave a light-weight overview of analog and hybrid computing models from a dynamical systems perspective, with a tour of discrete dynamical systems, continuous dynamical systems, and their common generalization as hybrid (dynamical) systems, culminating in a logic and programming languages view of dynamical systems. The focus in this article was on an exposition of the basic principles and ideas. Deeper levels of sophistication are reserved for more in-depth expositions [12–14, 20]. The primary perspective here was on identifying and relating some surprising commonalities of discrete and continuous dynamics using the characterization of hybrid systems in differential dynamic logic [9, 12–14]. More consequences of the complete proof theoretical alignment are discussed in previous work [14]. We also remark that the approach shown in this paper generalizes to distributed hybrid systems [65], stochastic hybrid systems [76], and hybrid games [118].

The study of the relations of discrete and continuous systems is not only very exciting but also results in surprising relations [9, 13, 14, 20, 44, 106–108], bringing up many interesting questions for future work. We highlight that the complete alignments readily identify important cases for which the complexity is lower than what the constructive reductions use [14]. The reason is that the constructive proofs are optimized for simplicity not efficiency. This raises the question of the inherent complexity of the reductions. What is the lowest complexity achievable in which case?

Acknowledgments

We thank Gilles Dowek, Nachum Dershowitz, Olivier Bournez, Daniel Graça, and Yuri Gurevich for helpful feedback on this article.

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, by DARPA under agreement number FA8750-12-2-0291, by the Army Research Office under Award No. W911NF-09-1-0273, by University Transportation Center program grant funds from the U.S. Department of Transportation, and by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

References

[1] Laurent Doyen, Goran Frehse, George J. Pappas, and André Platzer. Verification of hybrid systems. In Edmund M. Clarke, Thomas A. Henzinger, and Helmut Veith, editors, *Handbook of Model Checking*, chapter 28. Springer, 2015.

[2] Anil Nerode and Wolf Kohn. Models for hybrid systems: Automata, topologies, controllability, observability. In Grossman et al. [119], pages 317–356. ISBN 3-540-57318-6.

[3] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138(1):3–34, 1995.

[4] Michael S. Branicky. General hybrid dynamical systems: Modeling, analysis, and control. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems*, volume 1066 of *LNCS*, pages 186–200. Springer, 1995. ISBN 3-540-61155-X.

[5] Thomas A. Henzinger. The theory of hybrid automata. In *LICS*, pages 278–292, Los Alamitos, 1996. IEEE Computer Society. doi:10.1109/LICS.1996.561342.

[6] Michael S. Branicky, Vivek S. Borkar, and Sanjoy K. Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE T. Automat. Contr.*, 43 (1):31–45, 1998.

[7] Jennifer M. Davoren and Anil Nerode. Logics for hybrid systems. *IEEE*, 88(7): 985–1010, July 2000.

[8] Rajeev Alur, Thomas Henzinger, Gerardo Lafferriere, and George J. Pappas. Discrete abstractions of hybrid systems. *Proc. IEEE*, 88(7):971–984, 2000.

[9] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41 (2):143–189, 2008. ISSN 0168-7433. doi:10.1007/s10817-008-9103-8.

[10] André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010. ISSN 0955-792X. doi:10.1093/logcom/exn070.

[11] André Platzer. *Differential Dynamic Logics: Automated Theorem Proving for Hybrid Systems*. PhD thesis, Department of Computing Science, University of Oldenburg, Dec 2008. Appeared with Springer.

[12] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. ISBN 978-3-642-14508-7. doi:10.1007/978-3-642-14509-4.

[13] André Platzer. Logics of dynamical systems. In LICS [120], pages 13–24. ISBN 978-1-4673-2263-8. doi:10.1109/LICS.2012.13.

[14] André Platzer. The complete proof theory of hybrid systems. In LICS [120], pages 541–550. ISBN 978-1-4673-2263-8. doi:10.1109/LICS.2012.64.

[15] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HyTech: The next generation. In *IEEE Real-Time Systems Symposium*, pages 56–65, 1995.

[16] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *Trans. on Embedded Computing Sys.*, 6 (1):8, 2007. ISSN 1539-9087.

[17] André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 171–178. Springer, 2008. ISBN 978-3-540-71069-1. doi:10.1007/978-3-540-71070-7_15.

[18] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV*, volume 6806 of *LNCS*, pages 379–395. Springer, 2011. ISBN 978-3-642-22109-5.

[19] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 258–263. Springer, 2013. ISBN 978-3-642-39798-1. doi:10.1007/978-3-642-39799-8_18.

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

[21] Henri Poincaré. Sur les courbes définies par une équation différentielle. *Oeuvres*, 1, 1892. Paris.

[22] Morris W. Hirsch, Stephen Smale, and Robert L. Devaney. *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. Academic Press, 2 edition, 2003.

[23] Oded Galor. *Discrete Dynamical Systems*. Springer, 2010.

[24] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-03270-8.

[25] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of Model Checking*. MIT Press, 2008. ISBN 978-0262026499.

[26] Krzysztof R. Apt, Frank S. de Boer, and Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. Springer, 3rd edition, 2010.

[27] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation*. Springer, 1998. ISBN 0-387-98281-7.

[28] Marian Boykan Pour-El and Ian Richards. *Computability in Analysis and Physics*. Springer, 1989.

[29] Klaus Weihrauch. *Computable Analysis*. Springer, 2005. ISBN 978-3-540-26179-7.

[30] Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[31] André Platzer and Edmund M. Clarke. The image computation problem in hybrid systems model checking. In Alberto Bemporad, Antonio Bicchi, and Giorgio Buttazzo, editors, *HSCC*, volume 4416 of *LNCS*, pages 473–486. Springer, 2007. ISBN 978-3-540-71492-7. doi:10.1007/978-3-540-71493-4_37.

[32] Lawrence Perko. *Differential equations and dynamical systems*. Springer, New York, 3 edition, 2006. ISBN 978-0387951164.

[33] Akitoshi Kawamura and Stephen A. Cook. Complexity theory for operators in analysis. In Leonard J. Schulman, editor, *STOC*, pages 495–502. ACM, 2010. ISBN 978-1-4503-0050-6. doi:10.1145/1806689.1806758.

[34] Wolfgang Walter. *Ordinary Differential Equations*. Springer, 1998. ISBN 978-0387984599.

[35] Olivier Bournez, Manuel Lameiras Campagnolo, Daniel S. Graça, and Emmanuel Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *Journal of Complexity*, 23:317–335, 2007.

[36] Claude Elwood Shannon. Mathematical theory of the differential analyzer. *J. Math. Phys.*, 20:337–354, 1941.

[37] V. Bush. The differential analyzer. a new machine for solving differential equations. *Journal of the Franklin Institute*, 212(4):447 – 488, 1931. ISSN 0016-0032.

[38] Daniel Silva Graça and José Félix Costa. Analog computers and recursive functions over the reals. *J. Complexity*, 19(5):644–664, 2003.

[39] Cristopher Moore. Recursion theory on the reals and continuous-time computation. *Theor. Comput. Sci.*, 162(1):23–44, 1996. doi:10.1016/0304-3975(95)00248-0.

[40] Thomas Chadzelek and Günter Hotz. Analytic machines. *Theor. Comput. Sci.*, 219 (1-2):151–167, 1999.

[41] Alexander Moshe Rabinovich. Automata over continuous time. *Theor. Comput. Sci.*, 300(1-3):331–363, 2003. doi:10.1016/S0304-3975(02)00331-6.

[42] Boris A. Trakhtenbrot. Automata, circuits, and hybrids: Facets of continuous time. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *ICALP*, volume 2076 of *LNCS*, pages 4–23. Springer, 2001. ISBN 3-540-42287-0. doi:10.1007/3-540-48224-5_2.

[43] Otomar Hájek. Discontinuous differential equations, I. *Journal of Differential Equations*, 32(2):149 – 170, 1979. ISSN 0022-0396. doi:10.1016/0022-0396(79)90056-1.

[44] Gilles Dowek. The physical Church–Turing thesis and non-deterministic computation over the real numbers. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 370(1971):3349–3358, 2012. doi:10.1098/rsta.2011.0322.

[45] Jean-Pierre Aubin and Arrigo Cellina. *Differential Inclusions: Set-Valued Maps and Viability Theory*. Springer, 1984.

[46] Sarah M. Loos, André Platzer, and Ligia Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In Michael Butler and Wolfram Schulte, editors, *FM*, volume 6664 of *LNCS*, pages 42–56. Springer, 2011. ISBN 978-3-642-21436-3. doi:10.1007/978-3-642-21437-0_6.

[47] Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. Hybrid dynamical systems. *IEEE Control Systems Magazine*, 29(2):28–93, 2009.

[48] 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.

[49] André Platzer. A complete axiomatization of differential game logic for hybrid games. Technical Report CMU-CS-13-100R, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, January, Revised and extended in July 2013.

[50] Anil Nerode and Wolf Kohn. Models for hybrid systems: Automata, topologies, controllability, observability. In *Hybrid Systems*, pages 317–356, London, UK, UK, 1993. Springer-Verlag. ISBN 3-540-57318-6.

[51] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Grossman et al. [119], pages 209–229. ISBN 3-540-57318-6.

[52] Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. An approach to the description and analysis of hybrid systems. In Grossman et al. [119], pages 149–178. ISBN 3-540-57318-6. doi:10.1007/3-540-57318-6_28.

[53] Lucio Tavernini. Differential automata and their discrete simulators. *Non-Linear Anal.*, 11(6):665–683, 1987. ISSN 0362-546X.

[54] Jan A. Bergstra and C. A. Middelburg. Process algebra for hybrid systems. *Theor. Comput. Sci.*, 335(2-3):215–280, 2005.

[55] D. A. van Beek, Ka L. Man, Michel A. Reniers, J. E. Rooda, and Ramon R. H. Schiffelers. Syntax and consistent equation semantics of hybrid Chi. *J. Log. Algebr. Program.*, 68(1-2):129–210, 2006.

[56] René David and Hassane Alla. On hybrid petri nets. *Discrete Event Dynamic Systems*, 11(1-2):9–40, 2001. doi:10.1023/A:1008330914786.

[57] Franck Cassez and Kim Guldstrand Larsen. The impressive power of stopwatches. In *CONCUR*, pages 138–152, 2000.

[58] Goran Frehse. PHAVer: algorithmic verification of hybrid systems past HyTech. *STTT*, 10(3):263–279, 2008.

[59] Akash Deshpande, Aleks Göllü, and Pravin Varaiya. SHIFT: A formalism and a programming language for dynamic networks of hybrid automata. In Antsaklis et al. [121], pages 113–133. ISBN 3-540-63358-8.

[60] William C. Rounds. A spatial logic for the hybrid π-calculus. In Rajeev Alur and George J. Pappas, editors, *HSCC*, volume 2993 of *LNCS*, pages 508–522. Springer, 2004. ISBN 3-540-21259-0. doi:10.1007/978-3-540-24743-2_34.

[61] Fabian Kratz, Oleg Sokolsky, George J. Pappas, and Insup Lee. R-Charon, a modeling language for reconfigurable hybrid systems. In Hespanha and Tiwari [122], pages 392–406. ISBN 3-540-33170-0.

[62] José Meseguer and Raman Sharykin. Specification and analysis of distributed object-based stochastic hybrid systems. In Hespanha and Tiwari [122], pages 460–475. ISBN 3-540-33170-0.

[63] Seth Gilbert, Nancy Lynch, Sayan Mitra, and Tina Nolte. Self-stabilizing robot formations over unreliable networks. *ACM Trans. Auton. Adapt. Syst.*, 4(3):1–29, 2009. ISSN 1556-4665.

[64] André Platzer. Quantified differential dynamic logic for distributed hybrid systems. In Anuj Dawar and Helmut Veith, editors, *CSL*, volume 6247 of *LNCS*, pages 469–483. Springer, 2010. ISBN 978-3-642-15204-7. doi:10.1007/978-3-642-15205-4_36.

[65] André Platzer. A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *Logical Methods in Computer Science*, 8(4):1–44, 2012. doi:10.2168/LMCS-8(4:17)2012. Special issue for selected papers from CSL'10.

[66] Taylor T. Johnson and Sayan Mitra. A small model theorem for rectangular hybrid automata networks. In Holger Giese and Grigore Rosu, editors, *FORTE/FMOODS*, LNCS. Springer, 2012. To appear.

[67] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[68] Paul C. Attie and Nancy A. Lynch. Dynamic input/output automata: A formal model for dynamic systems. In Kim Guldstrand Larsen and Mogens Nielsen, editors, *CONCUR*, volume 2154 of *LNCS*, pages 137–151. Springer, 2001. ISBN 3-540-42497-0.

[69] Mark H. A. Davis. Piecewise-deterministic Markov processes: A general class of non-diffusion stochastic models. *Journal of the Royal Statistical Society. Series B*, 46(3):358–388, 1984.

[70] Mrinal K. Ghosh, Aristotle Arapostathis, and Steven I. Marcus. Ergodic control of switching diffusions. *SIAM J. Control Optim.*, 35(6):1952–1988, 1997. ISSN 0363-0129.

[71] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In Nancy A. Lynch and Bruce H. Krogh, editors, *HSCC*, volume 1790 of *LNCS*, pages 160–173. Springer, 2000. ISBN 3-540-67259-1.

[72] Manuela L. Bujorianu and John Lygeros. Towards a general theory of stochastic hybrid systems. In Henk A. P. Blom and John Lygeros, editors, *Stochastic Hybrid Systems: Theory and Safety Critical Applications*, volume 337 of *Lecture Notes Contr. Inf.*, pages 3–30. Springer, 2006.

[73] Christos G. Cassandras and John Lygeros, editors. *Stochastic Hybrid Systems*. CRC, 2006. ISBN 978-0849390838.

[74] Xenofon D. Koutsoukos and Derek Riley. Computational methods for verification of stochastic hybrid systems. *IEEE T. Syst. Man, Cy. A*, 38(2):385–396, 2008.

[75] Martin Fränzle, Tino Teige, and Andreas Eggers. Engineering constraint solvers for automatic analysis of probabilistic hybrid automata. *J. Log. Algebr. Program.*, 79 (7):436–466, 2010.

[76] André Platzer. Stochastic differential dynamic logic for stochastic hybrid programs. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *LNCS*, pages 431–445. Springer, 2011. ISBN 978-3-642-22437-9. doi:10.1007/978-3-642-22438-6_34.

[77] Ioannis Karatzas and Steven Shreve. *Brownian Motion and Stochastic Calculus*. Graduate Texts in Mathematics. Springer, 1991. ISBN 978-0387976556.

[78] Bernt Øksendal. *Stochastic Differential Equations: An Introduction with Applications*. Springer, 2007. ISBN 978-3540047582.

[79] Peter E. Kloeden and Eckhard Platen. *Numerical Solution of Stochastic Differential Equations*. Springer, New York, 2010. ISBN 978-3642081071.

[80] Anil Nerode, Jeffrey B. Remmel, and Alexander Yakhnis. Hybrid system games: Extraction of control automata with small topologies. In Antsaklis et al. [121], pages 248–293. ISBN 3-540-63358-8. doi:10.1007/BFb0031565.

[81] Claire Tomlin, George J. Pappas, and Shankar Sastry. Conflict resolution for air traffic management: a study in multi-agent hybrid systems. *IEEE T. Automat. Contr.*, 43(4):509–521, 1998.

[82] Thomas A. Henzinger, Benjamin Horowitz, and Rupak Majumdar. Rectangular hybrid games. In Jos C. M. Baeten and Sjouke Mauw, editors, *CONCUR*, volume 1664 of *LNCS*, pages 320–335. Springer, 1999. ISBN 3-540-66425-4. doi:10.1007/3-540-48320-9_23.

[83] Claire J. Tomlin, John Lygeros, and Shankar Sastry. A game theoretic approach to controller design for hybrid systems. *Proc. IEEE*, 88(7):949–970, 2000.

[84] S. Dharmatti and M. Ramaswamy. Zero-sum differential games involving hybrid controls. *Journal of Optimization Theory and Applications*, 128(1):75–102, 2006. ISSN 0022-3239. doi:10.1007/s10957-005-7558-x.

[85] Patricia Bouyer, Thomas Brihaye, and Fabrice Chevalier. O-minimal hybrid reachability games. *Logical Methods in Computer Science*, 6(1), 2010.

[86] Vladimeros Vladimerou, Pavithra Prabhakar, Mahesh Viswanathan, and Geir E. Dullerud. Specifications for decidable hybrid games. *Theor. Comput. Sci.*, 412(48): 6770–6785, 2011. doi:10.1016/j.tcs.2011.08.036.

[87] Jan-David Quesel and André Platzer. Playing hybrid games with KeYmaera. In Bernhard Gramlich, Dale Miller, and Ulrike Sattler, editors, *IJCAR*, volume 7364 of *LNCS*, pages 439–453. Springer, 2012. ISBN 978-3-642-31364-6. doi:10.1007/978-3-642-31365-3_34.

[88] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. In Nicola Olivetti, editor, *TABLEAUX*, volume 4548 of *LNCS*, pages 216–232. Springer, 2007. ISBN 978-3-540-73098-9. doi:10.1007/978-3-540-73099-6_17.

[89] Dexter Kozen. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19(3): 427–443, 1997.

[90] Ken Thompson. Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968. doi:10.1145/363347.363387.

[91] 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. ISBN 978-3-540-72732-3. doi:10.1007/978-3-540-72734-7_32.

[92] Konstantin Sergeevich Sibirsky. *Introduction to Topological Dynamics*. Noordhoff, Leyden, 1975.

[93] Michael S. Branicky. *Studies in Hybrid Systems: Modeling, Analysis, and Control*. PhD thesis, Dept. Elec. Eng. and Computer Sci., Massachusetts Inst. Technol., Cambridge, MA, 1995.

[94] Vaughan R. Pratt. Semantical considerations on Floyd-Hoare logic. In *FOCS*, pages 109–121. IEEE, 1976.

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

[96] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, 2nd edition, 1951.

[97] Rudolf Carnap. Modalities and quantification. *J. Symb. Log.*, 11(2):33–64, 1946.

[98] G. E. Hughes and M. J. Cresswell. *A New Introduction to Modal Logic*. Routledge, 1996. ISBN 978-0415125994.

[99] Robert W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics*, volume 19, pages 19–32, Providence, 1967. AMS.

[100] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[101] André Platzer. The structure of differential invariants and differential cut elimination. *Logical Methods in Computer Science*, 8(4):1–38, 2012. ISSN 1860-5974. doi:10.2168/LMCS-8(4:16)2012.

[102] André Platzer. A differential operator approach to equational differential invariants. In Lennart Beringer and Amy Felty, editors, *ITP*, volume 7406 of *LNCS*, pages 28–48. Springer, 2012. ISBN 978-3-642-32346-1. doi:10.1007/978-3-642-32347-8_3.

[103] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX Workshop*, volume 600 of *LNCS*, pages 447–484. Springer, 1991. ISBN 3-540-55564-1. doi:10.1007/BFb0032003.

[104] Anil Nerode. Logic and control. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *CiE*, volume 4497 of *LNCS*, pages 585–597. Springer, 2007. ISBN 978-3-540-73000-2.

[105] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Mon. hefte Math. Phys.*, 38:173–198, 1931.

[106] Cristopher Moore. Unpredictability and undecidability in dynamical systems. *Phys. Rev. Lett.*, 64:2354–2357, May 1990. doi:10.1103/PhysRevLett.64.2354.

[107] Michael S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theor. Comput. Sci.*, 138(1):67–100, 1995.

[108] Daniel Silva Graça, Manuel L. Campagnolo, and Jorge Buescu. Computability with polynomial differential equations. *Advances in Applied Mathematics*, 2007.

[109] Pieter Collins and Daniel S. Graça. Effective computability of solutions of differential inclusions the ten thousand monkeys approach. *J. UCS*, 15(6):1162–1185, 2009. doi:10.3217/jucs-015-06-1162.

[110] André Platzer. Logical analysis of hybrid systems: A complete answer to a complexity challenge. *Journal of Automata, Languages and Combinatorics*, 17(2-4): 265–275, 2012.

[111] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *LNCS*, pages 176–189. Springer, 2008. ISBN 978-3-540-70543-7. doi:10.1007/978-3-540-70545-1_17.

[112] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Form. Methods Syst. Des.*, 35(1):98–120, 2009. ISSN 0925-9856. doi:10.1007/s10703-009-0079-8. Special issue for selected papers from CAV'08.

[113] André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In Ana Cavalcanti and Dennis Dams, editors, *FM*, volume 5850 of *LNCS*, pages 547–562. Springer, 2009. ISBN 978-3-642-05088-6. doi:10.1007/978-3-642-05089-3_35.

[114] André Platzer and Jan-David Quesel. European Train Control System: A case study in formal verification. In Karin Breitman and Ana Cavalcanti, editors, *ICFEM*, volume 5885 of *LNCS*, pages 246–265. Springer, 2009. ISBN 978-3-642-10372-8. doi:10.1007/978-3-642-10373-5_13.

[115] Stefan Mitsch, Khalil Ghorbal, and André Platzer. On provably safe obstacle avoidance for autonomous robotic ground vehicles. In Paul Newman, Dieter Fox, and David Hsu, editors, *Robotics: Science and Systems*, 2013. ISBN 978-981-07-3937-9.

[116] Khalil Ghorbal and André Platzer. Characterizing algebraic invariants by differential radical invariants. In Erika Ábrahám and Klaus Havelund, editors, *TACAS*, volume 8413 of *LNCS*, pages 279–294. Springer, 2014. ISBN 978-3-642-54861-1. doi:10.1007/978-3-642-54862-8_19.

[117] Khalil Ghorbal, Andrew Sogokon, and André Platzer. Invariance of conjunctions of polynomial equalities for algebraic differential equations. In Markus Müller-Olm and Helmut Seidl, editors, *SAS*, volume 8723 of *LNCS*, pages 151–167. Springer, 2014. ISBN 978-3-319-10935-0. doi:10.1007/978-3-319-10936-7_10.

[118] André Platzer. Differential game logic. *CoRR*, abs/1408.1980, 2014.

[119] Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors. *Hybrid Systems*, volume 736 of *LNCS*, 1993. Springer. ISBN 3-540-57318-6.

[120] LICS. *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*, 2012. IEEE. ISBN 978-1-4673-2263-8.

[121] Panos J. Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors. *Hybrid Systems IV*, volume 1273 of *LNCS*, 1997. Springer. ISBN 3-540-63358-8.

[122] João P. Hespanha and Ashish Tiwari, editors. *Hybrid Systems: Computation and Control, 9th International Workshop, HSCC 2006, Santa Barbara, CA, USA, March 29-31, 2006, Proceedings*, volume 3927 of *LNCS*, 2006. Springer. ISBN 3-540-33170-0.