Dynamic Logics of Dynamical Systems

ANDRÉ PLATZER, Carnegie Mellon University

We study the logic of dynamical systems, that is, logics and proof principles for properties of dynamical systems. *Dynamical systems* are mathematical models describing how the state of a system evolves over time. They are important for modeling and understanding many applications, including embedded systems and cyber-physical systems. In *discrete dynamical systems*, the state evolves in discrete steps, one step at a time, as described by a difference equation or discrete state transition relation. In *continuous dynamical systems*, the state evolves continuously along a function, typically described by a differential equation. Hybrid dynamical systems or *hybrid systems* combine both discrete and continuous dynamics. *Distributed hybrid systems* combine distributed systems with hybrid systems, i.e., they are multi-agent hybrid systems that interact through remote communication or physical interaction. *Stochastic hybrid systems* combine stochastic dynamics with hybrid systems.

We survey *dynamic logics* for specifying and verifying properties for each of those classes of dynamical systems. A dynamic logic is a first-order modal logic with a pair of parametrized modal operators for each dynamical system to express necessary or possible properties of their transition behavior. Due to their full basis of first-order modal logic operators, dynamic logics can express a rich variety of system properties, including safety, controllability, reactivity, liveness, and quantified parametrized properties, even about relations between multiple dynamical systems. In this survey, we focus on some of the representatives of the family of *differential dynamic logics*, which share the ability to express properties of dynamical systems having continuous dynamics described by various forms of differential equations.

We explain the dynamical system models, dynamic logics of dynamical systems, their semantics, their axiomatizations, and proof calculi for proving logical formulas about these dynamical systems. We study *differential invariants*, i.e., induction principles for differential equations. We survey theoretical results, including soundness and completeness and deductive power. Differential dynamic logics have been implemented in automatic and interactive theorem provers and have been used successfully to verify safety-critical applications in automotive, aviation, railway, robotics, and analogue electrical circuits.

Categories and Subject Descriptors: F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic; D.2.4 [**Software Engineering**]: Software/Program Verification; C.1.m [**Processor Architectures**]: Hybrid Systems; G.1.4 [**Numerical Analysis**]: Ordinary Differential Equations; C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.4.7 [**Organization and Design**]: Distributed Systems; G.3 [**Probability and Statistics**]: Stochastic Processes

General Terms: Theory, Verification

Additional Key Words and Phrases: Logic of dynamical systems, dynamic logic, differential dynamic logic, hybrid systems, distributed hybrid systems, stochastic hybrid systems, axiomatization, deduction

---

This material is an extended version of [Platzer 2012c] and based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246, NSF EXPEDITION CNS-0926181, and under Grant Nos. CNS-1035800 and CNS-0931985, by the ONR award N00014-10-1-0188, by the Army Research Office under Award No. W911NF-09-1-0273, 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).

Author's addresses: A. Platzer, Computer Science Department, Carnegie Mellon University.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

ACM Reference Format:
Platzer, A. YYYY. Dynamic logics of dynamical systems. ACM Trans. Comput. Logic V, N, Article A (YYYY), 73 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Dynamical systems study the mathematics of change [Hirsch et al. 2003; Perko 2006]. Dynamical systems are mathematical models for describing how the state of a system evolves over time in a state space. They can describe, for example, the temporal evolution of the state of an embedded system or of a cyber-physical system, i.e., a system combining and integrating cyber (computation and/or communication) with physical effects. Cars [Deshpande et al. 1996], aircraft [Tomlin et al. 1998], robots [Plaku et al. 2009], and power plants [Fourlas et al. 2004] are prototypical examples. But dynamical systems are more general and can also describe and analyze chemical processes [Riley et al. 2010; Kerkez et al. 2010], biological systems [Tiwari 2011], medical models [Grosu et al. 2011; Kim et al. 2011], and many other behavioral phenomena. Since dynamical systems occur in so many different contexts, different variations of dynamical system models are relevant for applications, including discrete dynamical systems described by difference equations or discrete transitions relations [Galor 2010], continuous dynamical systems described by differential equations [Hirsch et al. 2003; Perko 2006], hybrid dynamical systems alias hybrid systems combining discrete and continuous dynamics [Maler et al. 1991; Alur et al. 1995; Branicky 1995; Henzinger 1996; Branicky et al. 1998; Davoren and Nerode 2000; Alur et al. 2000; Platzer 2008a; Platzer 2010a; Platzer 2008b; Platzer 2010b; Platzer 2012b], distributed hybrid systems or multi-agent hybrid systems [Deshpande et al. 1996; Rounds 2004; Kratz et al. 2006; Gilbert et al. 2009; Platzer 2010c; Platzer 2012a], and stochastic hybrid systems that take stochastic effects into account [Davis 1984; Ghosh et al. 1997; Hu et al. 2000; Bujorianu and Lygeros 2006; Cassandras and Lygeros 2006; Meseguer and Sharykin 2006; Koutsoukos and Riley 2008; Fränzle et al. 2010; Platzer 2011b].

For many of the applications that can be understood as dynamical systems, we are interested in analyzing and predicting their behavior, e.g., because the applications are safety-critical or performance-critical. For car control systems, for example, it is important to verify that the controllers choose only safe control choices that can never lead to collisions with other traffic participants at any later point in time [Deshpande et al. 1996; Loos et al. 2011].

This illustrates a central point about the analysis of dynamical systems. Whether a *current* control choice is safe or unsafe in a dynamical system depends on whether the states that the dynamical system could reach after this control choice *in the future* will be safe or unsafe. Whether a dynamical system is safe or unsafe depends on whether it will *always* choose safe control choices *at all times*. Whether we can find that out depends on whether we can find a *proof* that the dynamical system is safe or whether we can find a proof that it is unsafe.

What we can accept as a proof or other form of evidence depends on how critical it is that the answer is right. If the answer is that the dynamical system is unsafe, then a test scenario demonstrating one bad behavior is good evidence, because it can be used for debugging purposes. If the dynamical system is suspected unsafe, then an expert's engineering judgment can be good evidence, because that would already prevent premature manufacturing and/or deployment of a potentially unsafe system design. If the answer is that the dynamical system is safe, we prefer stronger evidence than a series of successful test scenarios. After all, most dynamical systems have large or even (uncountably) infinite state spaces, so that no finite set of tests alone could demonstrate that the system will be safe in the infinitely many other possible situations that could

not be tested. This issue is particularly daunting for the complex systems found in practical applications, e.g., because they follow complex control logic or many of their features interact or because their physical interactions are difficult etc.

For those reasons, we pursue the question of what constitutes a proof about a dynamical system and how we can systematically obtain proofs to show whether the system is safe or unsafe. Safety, in this introductory discussion, should be broadly construed, because the approaches we study in this article work for much more complicated properties than classical safety properties as well, including liveness, controllability, reactivity, quantified parametrized properties and so on.

Our technical vehicle for answering these questions from a logically foundational perspective is our study of logics of dynamical systems. We survey logics for studying properties of the behavior of dynamical systems and proof approaches for proving those properties deductively. Dynamic logic [Pratt 1976] has been developed and used very successfully for conventional discrete programs, both for theoretical [Harel et al. 1977; Segerberg 1977; Parikh 1978; Fischer and Ladner 1979; Harel 1979; Kozen and Parikh 1981; Meyer and Parikh 1981; Peleg 1987; Istrail 1989; Harel et al. 2000; Leivant 2006] and practical purposes [Reif et al. 1997; Harel et al. 2000; Beckert et al. 2007]. We consider extensions of dynamic logic to dynamical systems, including logic for hybrid systems [Platzer 2007b; Platzer 2008a; Platzer 2010a; Platzer 2008b; Platzer 2010b; Platzer 2012b], logic for distributed hybrid systems [Platzer 2010c; Platzer 2012a], and logic for stochastic hybrid systems [Platzer 2011b]. We emphasize that the logic of dynamical systems approach we survey in this article lends itself to many interesting theoretical investigations as witnessed by a number of highly nontrivial theoretical results [Platzer 2007b; Platzer 2008a; Platzer 2010a; Platzer 2008b; Platzer 2010b; Platzer 2010c; Platzer 2011b; Platzer 2012d; Platzer 2012b; Platzer 2012a], while, at the same time, enabling the practical verification of complex applications across different fields [Platzer 2008b; Platzer and Clarke 2009b; Platzer and Quesel 2009; Platzer 2010b; Loos et al. 2011; Loos and Platzer 2011; Renshaw et al. 2011; Mitsch et al. 2012; Aréchiga et al. 2012] and inspiring algorithmic approaches based directly on these logics [Platzer and Clarke 2008; Platzer 2008b; Platzer and Clarke 2009a; Platzer and Quesel 2008; Platzer et al. 2009; Platzer 2010b; Renshaw et al. 2011].

We remind the reader that this is not an isolated phenomenon. Logics have been used very successfully in many different ways, including deduction and model checking, for verifying several other classes of systems, including finite-state systems [Clarke et al. 1999; Baier et al. 2008], programs [Pratt 1976; Harel et al. 2000; Beckert et al. 2007; Bradley and Manna 2007; Apt et al. 2010], and real-time systems [Dutertre 1995; Zhou and Hansen 2004; Olderog and Dierks 2008; Baier et al. 2008]. Hybrid systems verification, for example, has generally received significant attention by the research community, including a number of verification tools [Henzinger et al. 1997; Mitchell and Templeton 2005; Ratschan and She 2007; Frehse 2008; Platzer and Quesel 2008; Renshaw et al. 2011; Frehse et al. 2011]; see Sect. 6 for an overview. Each verification approach has benefits and tradeoffs. It is promising to combine ideas from approaches rooted in different traditions to leverage the specific advantages of each. For instance, fixpoint loops, which are a driving force behind model checking [Clarke et al. 1999; Baier et al. 2008], have been used as a proof strategy to find deductive proofs in the proof calculus of differential dynamic logic [Platzer 2008a]. Both can be used to compute invariants and differential invariants of the system [Platzer and Clarke 2008; Platzer and Clarke 2009a]. The study of the logic of dynamical systems combines many areas of science, including mathematical logic, automated theorem proving, proof theory, model checking, and decision procedures, as well as differential algebra, computer algebra, algebraic geometry, analysis, stochastic calculus, and numerical approximation.

We see a number of advantages of the approach we focus on here, which make it attractive for research and applications, with the most important being soundness, completeness, compositionality, and extendability. Because dynamical systems can capture very complex behavior, their analysis can become very challenging and it is surprisingly difficult to get the reasoning sound [Collins 2007; Platzer and Clarke 2007]. In logic, *soundness* is easier to achieve, because we just check a small number of elementary proof rules for soundness once and for all. Then everything that can be derived from those simple rules, no matter how complicated, is going to be correct. Soundness (everything we prove is true) and completeness (we can prove everything that is true) are separated by design. In logic, *completeness* is a meaningful question to ask, not just in practice but also in theory, and has been answered in detail for logic of dynamical systems (Sect. 3.5 and [Platzer 2012b]). More generally, theoretical questions and logically foundational questions, including relative completeness [Platzer 2008a; Platzer 2012b; Platzer 2012a] and relative deductive power [Platzer 2010a; Platzer 2012d], become meaningful in a logical setting.

The logics and proof systems we consider are *compositional*. That is, the logics have a perfectly compositional, denotational semantics, in which the semantics of a model and the meaning of a formula are simple functions of the respective semantics of their parts. Furthermore, the proof systems are compositional, i.e., they exploit this compositional semantics and systematically reduce a property of a complex systems to a number of properties about simpler systems by structural decomposition. This makes it possible to understand complex dynamical systems in terms of their parts, which are often much easier than the full system. In fact, completeness results prove that decomposition is always successful. This result translates into practice, where systems that are designed according to good engineering practice adhering to modularity principles are easier to verify than those that are not. Smart decompositions can have a tremendous impact on the practical verification complexity and improve scalability [Loos et al. 2011].

Another beneficial phenomenon in logics of dynamical systems is that they are easy to *extend*. Verification is based on a proof calculus, which is a collection of simple proof rules (and axioms). In order to verify a feature in a different way, we can simply add new proof rules, which will improve the verification since the previous proof rules are kept as alternatives. We will exercise this a number of times in this article, particularly when we are adding more and more proof rules to handle various sophisticated aspects of differential equations. We start with simple rules using solutions of differential equations, then study differential invariants [Platzer 2010a], an induction principle for differential equations, then differential cuts [Platzer 2010a; Platzer 2012d], a logical cut principle for differential equations, and finally differential auxiliaries [Platzer 2012d]. Differential refinement and differential transformation rules are further extensions [Platzer 2010a; Platzer 2010b], but beyond the scope of this article. Temporal logic extensions [Platzer 2010b; Platzer 2007c] and extensions to differential-algebraic hybrid systems [Platzer 2010a; Platzer 2010b] are other illustrations of how the logic and proof calculus can be extended easily just by adding rules to cover more advanced temporal properties and systems with more complex dynamics.

In this article we focus on the logic of hybrid systems and we illustrate two more invasive extensions that change the logic of dynamical systems in fundamental ways by changing the characteristic of relevant dynamical aspects. In Sect. 4, we consider the logic of distributed hybrid systems [Platzer 2010c; Platzer 2012a], which changes the state space in fundamental ways from fixed finite-dimensional state spaces to evolving and infinite-dimensional state spaces of arbitrarily many hybrid system agents interacting with each other through remote communication and physical interaction. This extension is as radical as that from propositional logic to first-order logic, except

that it happens in the dynamics, not just the propositions. In Sect. 5, we consider the logic of stochastic hybrid systems [Platzer 2011b], which changes the dynamics in fundamental ways to incorporate discrete and continuous stochastic effects changing the semantics from deterministic boolean truth to the randomness of stochastic processes. While both extensions are radical, catapulting us into fundamentally different classes of dynamical systems, we will see that the changes in the proof calculi are surprisingly moderate additions of proof rules for new dynamical features and refinements, e.g., to adapt to a stochastic semantics.

Another helpful aspect of logic is that it produces proofs that can serve as readable evidence for the correctness of a system for certification purposes. Concerns that are sometimes voiced in the context of classical discrete systems about theorem proving compared to model checking involve the degree of automation and the ability to find counterexamples. They are less relevant for general dynamical systems. Even the verification of very simple classes of hybrid systems is neither semidecidable nor cosemidecidable [Asarin and Maler 1998; Henzinger 1996; Cassez and Larsen 2000]. Consequently, quite unlike in finite-state systems and timed automata [Clarke et al. 1999; Baier et al. 2008], exhaustive exploration of all states, even in bisimulation quotients, does not terminate in general, so that approximations and abstractions have to be used during the reachability analysis, and counterexamples are no longer reliable (see [Clarke et al. 2003a] for counterexample-guided abstraction refinement techniques). Some nontrivial applications [Platzer and Clarke 2008; Platzer and Clarke 2009a; Platzer and Clarke 2009b; Platzer and Quesel 2009; Platzer 2010b] have been proved fully automatically with the approach we survey here. Improving automation and scalability is, nevertheless, a permanently promising challenge in verification. For complex systems, we find it advantageous that proving is amenable to human guidance, because the designer can specify the critical invariants of his system design, which helps finding proofs when current automation techniques fail.

In this article, we take a view that we call *multi-dynamical systems*, i.e., the principle to understand complex systems as a combination of multiple elementary dynamical aspects. This approach helps us tame the complexity of complex systems by understanding that their complexity just comes from combining lots of simple dynamical aspects with one another. The overall system itself is still as complicated as the whole application. But since differential dynamic logics and proofs are compositional, we can leverage the fact that the individual parts of a system are simpler than the whole, and we can prove correctness properties about the whole system by reduction to simpler proofs about their parts. This approach demonstrates that the whole can be greater than the sum of all parts. The whole system is complicated, but we can still tame its complexity by an analysis of its parts, which are simpler. Completeness results are the theoretical justification why this multi-dynamical systems principle works.

The results reported in this paper are based on previous research on logics of dynamical systems [Platzer 2007b; Platzer 2008a; Platzer 2010a; Platzer 2008b; Platzer 2010b; Platzer 2010c; Platzer 2011a; Platzer 2011b; Platzer 2012b; Platzer 2012d; Platzer 2012a]. The results presented here are new in that we show significantly simplified Hilbert-type axiomatizations and, consequently, simplified semantics in comparison to the earlier presentations, which were more tuned for automation. This setting enables us to identify connections between the approaches for the different classes of dynamical systems. We provide an overview of the approach of logic of dynamical systems here, but it is, by no means, possible to handle all material comprehensively in this survey. A more comprehensive source on logic of hybrid systems is a book [Platzer 2010b] and subsequent extensions [Platzer 2012d; Platzer 2012b]. Details about the logic of distributed hybrid systems [Platzer 2010c; Platzer 2011a; Platzer 2012a] and about logic of stochastic hybrid systems [Platzer 2011b] can be found in previous work.

More information about algorithmic aspects can be found in related papers [Platzer 2007a; Platzer and Clarke 2008; Platzer and Clarke 2009a; Platzer and Quesel 2008] and applications [Platzer and Clarke 2009b; Platzer and Quesel 2009; Loos et al. 2011; Loos and Platzer 2011; Renshaw et al. 2011; Mitsch et al. 2012]. Complementary extensions to differential temporal dynamic logic [Platzer 2010b; Platzer 2007c] and extensions to differential-algebraic hybrid systems with complex dynamics [Platzer 2010a; Platzer 2010b] are very useful, but beyond the scope of this article.

In Sect. 2, we briefly summarize the dynamical aspects of various classes of dynamical systems before we study their models, logics, and proofs in more detail in subsequent sections. In Sect. 3, we study the logic of hybrid systems, which includes the logic of discrete dynamical systems and the logic of continuous dynamical systems as fragments. In Sect. 4, we study the logic of distributed hybrid systems, extending the results from Sect. 3 to multi-agent scenarios. We study the logic of stochastic hybrid systems in Sect. 5. We discuss related work in Sect. 6 and give pointers to the literature. Section 7 concludes with a summary and an outlook for future research opportunities.

2. DYNAMICAL SYSTEMS

In this section, we briefly recall the basic principles behind a number of classes of dynamical systems, for which we study models, logics, and proof approaches in subsequent sections. We also illustrate our multi-dynamical systems view on these dynamical systems, which we detail in subsequent sections.

Formally, a *dynamical system* is an action of a monoid T (time) on a state space $\mathcal{X}$. That is a dynamical system is described by a function φ, whose value $\varphi_t(x) \in \mathcal{X}$ at time $t \in T$ denotes the state that the system has at time t, provided that it started in the initial state $x \in \mathcal{X}$ at time 0. It starts at $\varphi_0(x) = x$ 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 $x \in \mathcal{X}$. That is, if the dynamical system evolves for time t and, from the state $\varphi_t(x)$ that it reached then, for time s, then it reaches the same state by simply evolving for time $t+s$ starting from x right away. For different choices of T and $\mathcal{X}$, we get different classes of dynamical systems. For computational analysis purposes, it is also crucial to choose a sufficiently computational description of the dynamical system φ.

2.1. Discrete Dynamical Systems

Discrete dynamical systems have an integer notion of time (e.g., $T = \mathbb{N}$ or $T = \mathbb{Z}$) so that the state evolves in discrete steps, one step at a time, as typically described by a difference equation or discrete state transition function. The *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 the initial state. Equivalently, when defining $h(x) := f(x) - x$ the discrete dynamical system (1) can be described by the *difference equation*

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

Computation processes can be described by discrete dynamical systems, for example. The system starts in an initial state $\varphi_0(x) = x$ at a time 0, performs 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. Program models and automata models have been used to describe discrete dynamical systems and have been used very

successfully in verification [Clarke et al. 1999; Baier et al. 2008; Apt et al. 2010]. The behavior of systems with a discrete state transition relation $R \subseteq \mathcal{X} \times \mathcal{X}$ is nondeterministic, but can still be captured as a discrete dynamical system using the powerset $2^{\mathcal{X}}$ as the state space instead of $\mathcal{X}$:

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

when starting from a set $X \subseteq \mathcal{X}$ of initial states.

Discrete dynamical systems cannot, however, 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 [Platzer and Clarke 2007] and similarities [Platzer 2012b].

2.2. 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. 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. Depending on the duration of the solution of the above differential equation, the continuous system may only be defined on a subinterval of $\mathbb{R}$. The time-derivative $\frac{\mathsf{d}}{\mathsf{d}t}$ is only well-defined under additional assumptions, e.g., that $\mathcal{X}$ is a Euclidean space $\mathbb{R}^n$ or a differentiable manifold [Hirsch et al. 2003; Perko 2006].

Many physical processes are continuous dynamical systems described by differential equations. The movement of the longitudinal position of a car of velocity v down a straight road from initial position p_0, for example, 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 $t \in \mathbb{R}$. We refer to the literature for more details and many more examples of continuous dynamical systems [Hirsch et al. 2003; Perko 2006]. Continuous dynamical systems cannot represent discrete transitions easily; see, however, Sect. 3.5. Discrete transitions lead to discontinuities, which lead to interesting but very complicated generalized notions of solutions, including Carathéodory solutions [Walter 1998] or Filippov solutions [Aubin and Cellina 1984].

2.3. Hybrid Systems

Hybrid dynamical systems alias *hybrid systems* [Alur et al. 1995; Branicky 1995; Henzinger 1996; Branicky et al. 1998; Davoren and Nerode 2000; Alur et al. 2000; Platzer 2008a; Platzer 2010a; Platzer 2008b; Platzer 2010b; Platzer 2012b] 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., 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, e.g., when changing acceleration, or when the continuous dynamics determines the values of sen-

sor readings for the discrete decisions, e.g., the velocity). Embedded systems and cyber-physical systems are often modeled as hybrid systems, because they involve both discrete control and physical effects.

A typical example is a car that drives on a road according to a differential equation for the physical movement, but 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 1 shows 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). The situation in Figure 1 illustrates a bad control choice, 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) could 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.

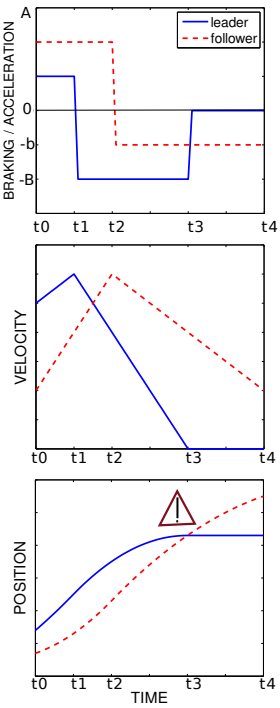

Fig. 1. Local car crash

Notice that the state space $\mathcal{X}$ has no bearing on whether a system is a hybrid system or not. It is the notion of time and dynamics that determines hybrid systems. For example, a system that has both discrete-valued state variables from a discrete set $\{1, 2, 3, 4\}$ and continuous-valued state variables from a continuous set like $\mathbb{R}$ is still a discrete dynamical system if all its variables only change in discrete steps (Sect. 2.1).

In hybrid systems, we follow our multi-dynamical systems philosophy and model each part of the system by the most appropriate dynamics, whether discrete or continuous, instead of having to model everything discrete, uniformly, for the whole system as in discrete dynamical systems or to model everything continuous, uniformly, as in continuous dynamical systems. The overall system behavior can still be very complicated, if the system under investigation is complex, but at least each part of the system has an easier, more natural model.

For example, when using hybrid systems, there neither is 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 [Walter 1998] or Filippov solutions [Aubin and Cellina 1984] to understand jumps in continuous processes, because discrete jumps are allowed directly as separate elements in hybrid systems. The overall system behavior can still be as complicated, and, in fact, a study of some behaviors in terms of Carathéodory and Filippov solutions can be insightful. But the individual parts of the hybrid system have a simpler behavior that can be understood and analyzed by easier means. In our model for hybrid systems, the dynamical affects have separate atomic programs that can be combined in flexible ways by program combinators (Sect. 3).

We exploit the multi-dynamical systems philosophy in our analysis approach, because the logics we explain in the subsequent sections of this article have a fully com-

positional semantics and fully compositional proof principles. Thus, since our proof approach works by reasoning by parts, all the individual reasoning steps get easier, because hybrid systems combine many but simpler dynamical aspects instead of requiring a single inscrutable effect. Consequently, we can reason separately about the individual parts of the hybrid systems.

2.4. Distributed Hybrid Systems

Distributed hybrid systems [Deshpande et al. 1996; Rounds 2004; Kratz et al. 2006; Meseguer and Sharykin 2006; Gilbert et al. 2009; Platzer 2010c; Platzer 2012a; Johnson and Mitra 2012] are dynamical systems that combine distributed systems [Lynch 1996; Attie and Lynch 2001; Apt et al. 2010] with hybrid systems (and their discrete and continuous dynamics). Again, they are not just combined side by side, but can interact.

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. A typical example is a distributed car control scenario (see Figure 2), in which multiple cars drive on a road and use sensing and/or communication to inform each other of their respective positions and velocities and control intentions in order to coordinate their actions to prevent collisions. Distributed hybrid systems become crucial, e.g., when we do not know how many agents are going to be involved exactly, or when there are more agents than hybrid systems analysis could handle. Consequently, unlike in classical hybrid systems, the state space of distributed hybrid systems is usually an infinite-dimensional vector space $\mathcal{X}$. Because of their importance in practical applications, many modeling approaches have been pursued for distributed hybrid systems [Deshpande et al. 1996; Rounds 2004; Kratz et al. 2006; Meseguer and Sharykin 2006], including SHIFT [Deshpande et al. 1996], R-Charon [Kratz et al. 2006], and the process algebra χ [van Beek et al. 2006].

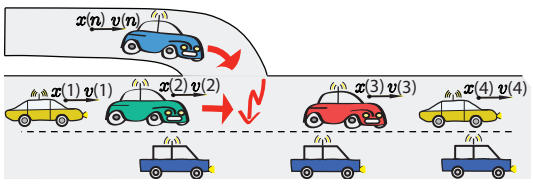

Fig. 2. Distributed car control

In distributed hybrid systems, we follow our multi-dynamical systems philosophy and model each part of the system by the most appropriate dynamical aspect, whether discrete or continuous or structural (e.g., changes in the communication topology or changes in the physical configuration) or dimensional (e.g., appearance or disappearance of cars on the street). In our model for distributed hybrid systems, the dynamical affects have separate atomic programs that can be combined in flexible ways by program combinators (Sect. 4). We exploit the multi-dynamical systems philosophy in our analysis, logic, and proofs, so that we can reason separately about the individual parts of a distributed hybrid system.

2.5. Stochastic Hybrid Systems

Stochastic hybrid systems [Davis 1984; Ghosh et al. 1997; Hu et al. 2000; Bujorianu and Lygeros 2006; Cassandras and Lygeros 2006; Meseguer and Sharykin 2006; Koutsoukos and Riley 2008; Fränzle et al. 2010; Platzer 2011b] are dynamical systems that combine the dynamics of stochastic processes [Karatzas and Shreve 1991; Øksendal

2007; Kloeden and Platen 2010] with hybrid systems. Again, they are not just combined side by side, but can interact.

There is more than one way in which stochasticity has been added into hybrid systems models; see, e.g., Figure 3. Stochasticity might be restricted to the discrete dynamics, as in piecewise deterministic Markov decision processes [Davis 1984], restricted to the continuous and switching behavior as in switching diffusion processes [Ghosh et al. 1997], or allowed in many parts as in so-called General Stochastic Hybrid Systems; see [Bujorianu and Lygeros 2006; Cassandras and Lygeros 2006] for an overview. Stochastic hybrid systems models have the desire in common to add stochastic information about uncertainties into the system dynamics. Hybrid systems and distributed hybrid systems are limited to nondeterministic views and can only encode simple probabilistic effects in their hybrid dynamics. For stochastic hybrid systems, the state space is more complicated, because it has to be rich enough to define stochastic process transitions. But the time domain is still such that some transitions are in continuous time, others are discrete steps in time.

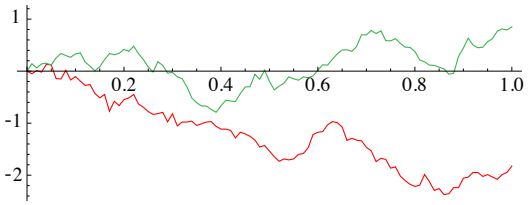

Fig. 3. Two samples from a switched continuous stochastic process

In stochastic hybrid systems, we follow our multi-dynamical systems philosophy and model each part of the system by the most appropriate dynamical aspect, whether discrete or continuous, whether stochastic or not. In particular, there is no need to represent the system dynamics with interesting but complicated concepts like semimartingales [Karatzas and Shreve 1991; Protter 2010]. The overall system behavior can still be as complicated, and a study of some behaviors in terms of semimartingales can be insightful. But the individual parts of the stochastic hybrid system have a simpler behavior that can be understood and analyzed by easier means. In our model for stochastic hybrid systems, the dynamical effects have separate atomic programs that can be combined by program combinators (Sect. 5). We exploit the multi-dynamical systems philosophy in our analysis, logic, and proofs, so that we can reason separately about the individual parts of a stochastic hybrid system.

3. DIFFERENTIAL DYNAMIC LOGIC FOR HYBRID SYSTEMS

In this section, we study *differential dynamic logic* $\mathsf{d}\mathcal{L}$ [Platzer 2007b; Platzer 2008a; Platzer 2012b], the *logic of hybrid systems*, i.e., systems with interacting discrete and continuous dynamics.

Hybrid systems [Alur et al. 1995; Branicky 1995; Henzinger 1996; Branicky et al. 1998; Davoren and Nerode 2000; Alur et al. 2000; Platzer 2008a; Platzer 2010a; Platzer 2008b; Platzer 2010b; Platzer 2012b] are a fusion of continuous dynamical systems and discrete dynamical systems. They freely combine dynamical features from both worlds and play an important role, e.g., in modeling systems that use computers to control physical systems. Hybrid systems feature (iterated) difference equations for discrete dynamics and differential equations for continuous dynamics. They, further, combine conditional switching, nondeterminism, and repetition.

As a specification and verification language for hybrid systems, we have introduced *differential dynamic logic* $\mathsf{d}\mathcal{L}$ [Platzer 2007b; Platzer 2008a; Platzer 2008b; Platzer 2010b; Platzer 2012b]. The logic $\mathsf{d}\mathcal{L}$ is based on first-order modal logic [Carnap 1946; Hughes and Cresswell 1996] and dynamic logic [Pratt 1976; Harel et al. 2000] and internalizes operational models of hybrid systems as first-class citizens, so that correctness statements about the transition behavior of hybrid systems can be expressed

as logical formulas. In addition to all operators of first-order real arithmetic, the logic dL provides parametrized modal operators $[\alpha]$ and $\langle\alpha\rangle$ that refer to the states reachable by hybrid system α and can be placed in front of any formula. The dL formula $[\alpha]\phi$ expresses that all states reachable by hybrid system 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.

We first explain the system model of hybrid programs that dL provides for modeling hybrid systems (Sect. 3.1). Then we explain the logical formulas that dL provides for specification and verification purposes (Sect. 3.2). For reference, we provide a short exposition of hybrid automata (Sect. 3.3) and relate them to hybrid programs. Then, we explain reasoning principles, axioms, and proof rules for verifying dL formulas (Sect. 3.4). We subsequently show soundness and relative completeness theorems (Sect. 3.5) and investigate stronger proof rules for differential equations (Sect. 3.6–3.9). Finally, we briefly discuss an implementation in the theorem prover KeYmaera and applications (Sect. 3.10).

3.1. Regular Hybrid Programs

Differential dynamic logic uses (regular) *hybrid programs* (HP) [Platzer 2007b; Platzer 2008a; Platzer 2010b; Platzer 2012b] as hybrid system models. HPs are a program notation for hybrid systems and combine differential equations with conventional program constructs and discrete assignments. HPs form a Kleene algebra with tests [Kozen 1997]. Atomic HPs are instantaneous discrete jump *assignments* $x:=\theta$, *tests* $?\chi$ of a first-order formula[1] χ of real arithmetic, and *differential equation (systems)* $x'=\theta\,\&\,\chi$ for a continuous evolution restricted to the domain of evolution χ, 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 (*). We use polynomials with rational coefficients as terms here, but divisions can be allowed as well when guarding against singularities of divisions by zero; see [Platzer 2008a; Platzer 2010b] 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 χ a formula of first-order logic of real arithmetic):

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

The first three cases are called atomic HPs, the last three compound. The *test* action $?\chi$ is used to define conditions. Its effect is that of a *no-op* if the formula χ is true in the current state; otherwise, like *abort*, it allows no transitions. That is, if the test succeeds because formula χ holds in the current state, then the state does not change, and the system execution continues normally. If the test fails because formula χ does not hold in the current state, then the system execution cannot continue, is cut off, 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

[1] The test $?\chi$ means "if χ then *skip* else *abort*". Our results generalize to rich-test dL, where $?\chi$ is a HP for any dL formula χ (Sect. 3.2).

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).

These operations can define all classical WHILE programming constructs and all hybrid systems [Platzer 2010b]. We, e.g., write $x' = \theta$ for the unrestricted differential equation $x' = \theta \,\&\, true$. We allow differential equation systems and use vectorial notation. Vectorial assignments are definable from scalar assignments and ; using auxiliary variables.[2] Other program constructs can be defined easily [Platzer 2010b]. For example, nondeterministic assignments of any real value to x, if-then-else statements, and while loops can be defined as follows:

$$\begin{aligned} x := * &\equiv x' = 1 \cup x' = -1 \\ \text{if } (\chi) \text{ then } \alpha \text{ else } \beta \text{ fi} &\equiv (?\chi;\alpha) \cup (?\neg\chi;\beta) \\ \text{if } (\chi) \text{ then } \alpha &\equiv (?\chi;\alpha) \cup ?\neg\chi \\ \mathsf{while}(\chi)\,\alpha &\equiv (?\chi;\alpha)^*;?\neg\chi \end{aligned} \tag{2}$$

HPs have a compositional semantics. We define their semantics by a reachability relation and refer to previous work for their trace semantics [Platzer 2007c; Platzer 2010b]. 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 first-order formula χ is true in state ν (defined in Sect. 3.2).

Definition 3.2 (*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 $[\![x]\!]_\omega = [\![\theta]\!]_\nu\}$
— $\rho(?\chi) = \{(\nu,\nu) \,:\, \nu \models \chi\}$
— $\rho(x' = \theta \,\&\, \chi) = \{(\varphi(0),\varphi(r)) \;:\; \varphi(t) \models x' = \theta$ and $\varphi(t) \models \chi$ for all $0 \leq t \leq r$ for a solution $\varphi : [0,r] \to \mathcal{S}$ of any duration $r\}$; i.e., with $\varphi(t)(x') \overset{\text{def}}{=} \frac{\mathsf{d}\varphi(\zeta)(x)}{\mathsf{d}\zeta}(t)$, φ solves the differential equation and satisfies χ at all times [Platzer 2008a]
— $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
— $\rho(\alpha;\beta) = \rho(\beta) \circ \rho(\alpha) = \{(\nu,\omega) : (\nu,\mu) \in \rho(\alpha), (\mu,\omega) \in \rho(\beta)\}$
— $\rho(\alpha^*) = \bigcup_{n\in\mathbb{N}} \rho(\alpha^n)$ with $\alpha^{n+1} \equiv \alpha^n;\alpha$ and $\alpha^0 \equiv ?true$.

We refer to our book [Platzer 2010b] for a comprehensive background and for an elaboration how the case $r = 0$ (in which the only condition is $\varphi(0) \models \chi$) is captured by the above definition. Time itself is not special but implicit. If a clock variable t is needed in a HP, it can be axiomatized by $t' = 1$.

Example 3.3 (*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

[2] A vectorial assignment $x_1 := \theta_1, \ldots, x_n := \theta_n$ is definable by $\dot{x}_1 := x_1; \ldots; \dot{x}_n := x_n; x_1 := \grave{\theta}_1; \ldots; x_n := \grave{\theta}_n$ where $\grave{\theta}_i$ is θ_i with x_j replaced by $\dot{x}_j$ for all j.

$x' = v, v' = a$. 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 $a := A \cup a := -b$ describes a controller that can choose nondeterministically to accelerate or brake. Accelerating will only sometimes be a safe control decision, so the discrete controller in the following HP requires a test $?\chi$ to be passed in the acceleration choice:

$$\mathit{car}_s \equiv \big(((?\chi; a := A) \cup a := -b);\ x' = v, v' = a \,\&\, v \geq 0\big)^* \tag{3}$$

This HP, which we abbreviate by car_s, first allows a nondeterministic choice of acceleration (if the test χ succeeds) or braking, 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 (3) can nondeterministically select to proceed with $?\chi; a := A$ or with $a := -b$. Yet the first choice can only continue if, indeed, formula χ is true about the current state (then both choices are possible). Otherwise only the braking choice will run successfully. With this principle, HPs elegantly separate the fundamental principles of (nondeterministic) choice from conditional execution (tests).

Which formula is suitable for χ depends on the control objective or property we care about. A simple guess for χ like $v \leq 20$ has the effect that the controller can only choose to accelerate at lower speeds. This condition alone is insufficient for most control purposes. We will refine χ in Example 3.7.

HPs are a program notation for hybrid systems. Hybrid automata [Alur et al. 1995; Henzinger 1996] are an automaton notation for hybrid systems. Hybrid automata correspond to finite automata with guards and reset relations annotated at edges and with differential equations and evolution domain constraints annotated at nodes (defined in detail in Sect. 3.3). The car system in (3) can be represented by the hybrid automaton in Figure 4. All hybrid automata can be represented as HPs [Platzer 2010b]

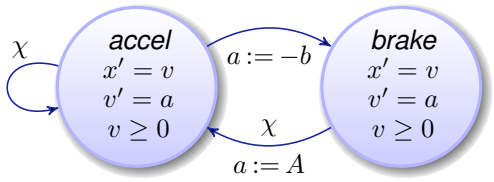

Fig. 4. Hybrid automaton for a simple car

just like finite automata can be implemented in classical WHILE programs (Sect. 3.3).

An important phenomenon is that the evolution domain constraint in (3) and Figure 4 is too lax for many purposes. It does not specify when the continuous evolution stops. Many systems are unsafe if the continuous evolution evolves forever without giving the controller a chance to react. To model *event-triggered systems*, we would augment the evolution domain constraint with a formula that prevents the continuous evolution from missing important events. For example, we could add the evolution domain constraint $v \leq 22$ into the differential equation in (3) to ensure that the continuous evolutions stop and the discrete controllers will react before the velocity increases

beyond 22:

$$\big(((?\chi; a := A) \cup a := -b);\ x' = v, v' = a \,\&\, v \geq 0 \wedge v \leq 22\big)^*$$

In *time-triggered systems*, we would, instead, replace the continuous evolution in (3) by $t := 0;\ x' = v, v' = a, t' = 1 \,\&\, v \geq 0 \wedge t \leq \varepsilon$ with a clock t with slope $t' = 1$ that is reset by a discrete assignment ($t := 0$) before the continuous evolution and whose value is bounded ($t \leq \varepsilon$ in the evolution domain constraint) by a symbolic parameter for the maximum reaction time $\varepsilon > 0$. Then, the continuous evolution stops at the latest after ε time units so that the discrete controllers have a chance to react to situation changes. Without such a bound on the reaction time, systems are rarely safe. The time-triggered version of (3) is the following HP, which we abbreviate by car_ε:

$$\mathit{car}_\varepsilon \equiv \big(((?\chi; a := A) \cup a := -b);\ t := 0;\ x' = v, v' = a, t' = 1 \,\&\, v \geq 0 \wedge t \leq \varepsilon\big)^* \tag{4}$$

Time-triggered models are closer to the implementation, because event-triggered models require permanent sensing. Event-triggered models are usually easier to verify but time-triggered models are easier to implement and reveal important timing effects.

Observe that, at this point, we could try to investigate the reachability question whether from a given state ν we can reach a state ω along car model car_s from (3), i.e., $(\nu, \omega) \in \rho(\mathit{car}_s)$, at which $\omega(x)$ is at a certain goal position. We could also study the safety question whether for all states ω with $(\nu, \omega) \in \rho(\mathit{car}_s)$ it is the case that $\omega(v) < 10$ is true. Instead of studying each of those questions with one ad-hoc notion for each question, we follow a more principled approach and define a logic in which those and many more general properties of hybrid systems can be expressed and verified. We first discuss another instructive example, however.

Example 3.4 (*Bouncing ball*). Another intuitive example of a hybrid system is the bouncing ball [Egerstedt et al. 1999]; see Figure 5. The bouncing ball is let loose in

$$\big(\, h' = v, v' = -g \,\&\, h \geq 0;\ \text{if } (h = 0) \text{ then } v := -cv \text{ fi}\big)^*$$

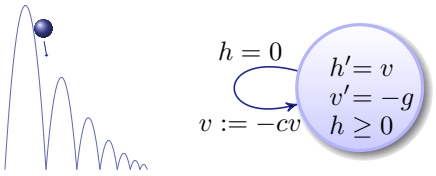

Fig. 5. Hybrid program, plot, and hybrid automaton of a bouncing ball

the air and is falling towards the ground. When it hits the ground, the ball bounces back up and climbs until gravity wins and it starts to fall again. The bouncing ball follows the continuous dynamics of physical movement by gravity. It can be understood naturally as a hybrid system, because its continuous movement switches from falling to climbing by reversing its velocity whenever the ball hits the ground and bounces back. Let us denote the height of the ball by h and the current velocity of the ball by v. The bouncing ball is affected by gravity of force $g > 0$, so its height follows the differential equation $h'' = -g$, i.e., the second time derivative of height equals the negative gravity force. The ball bounces back from the ground (which is at height $h = 0$) after an elastic deformation. At every bounce, the ball loses energy according to a damping factor $0 \leq c < 1$. Figure 5 depicts a HP, an illustration of the system dynamics, and a representation as a hybrid automaton.

The first line of the HP describes the continuous dynamics along the differential equation $h' = v, v' = -g$ (which is equivalent to $h'' = -g$) restricted to (written $\&$) the evolution domain $h \geq 0$ above the floor. In particular, the bouncing ball never falls

through the floor. After the sequential composition (;), an if-then statement resets velocity v to $-cv$ by assignment $v := -cv$ if $h = 0$ holds at the current state. This assignment will change the direction from falling (the velocity v was negative before) to climbing (the velocity $-cv$ is nonnegative again) after dampening the velocity v by c. Recall (2) for how if-then is defined. Finally, the sequence of continuous and discrete statements can be repeated arbitrarily often, as indicated by the regular-expression-style repetition operator (*) at the end.

The hybrid automaton on the right of Figure 5 represents the same system as a HP. It has one node: falling along the differential equation system $h' = v, v' = -g$ restricted to evolution domain $h \geq 0$, above the floor. The hybrid automaton has one jump edge: on the ground ($h = 0$), it can reset the velocity v to $-cv$ and continue in the same node.

Note one strange phenomenon in the bouncing ball. It seems like the bouncing ball will bounce over and over again, switching its direction in shorter and shorter periods of time as indicated in Figure 5 (unless $c = 0$, which means that the ball will just lie flat right away). Even worse, the ball will end up switching directions infinitely often in a short amount of time. This controversial phenomenon is called *Zeno behavior*.

In reality, the ball bounces a couple of times and can then come to a standstill when its remaining kinetic energy is insufficient. To model this phenomenon without the need to have a precise physical model for all physical forces and frictions, we can allow for the damping factor c to change at each bounce by adding $c := *;\ ?(0 \leq c < 1)$ before $v := -cv$. HP $c := *$ represents an uncountably infinite nondeterministic choice for c as a nondeterministic assignment. Recall (2) for its definition. The subsequent test $?(0 \leq c < 1)$ restricts the arbitrary choices for c to choices in the half-open interval $[0, 1)$ and discards all other choices. Now the bouncing ball can stop. This particular model still allows a Zeno execution when each choice of c is $c > 0$, which can be removed by imposing additional restrictions on the permitted choices of c.

To avoid technicalities, we consider only polynomial differential equations here and refer to previous work [Platzer 2010a; Platzer 2010b] for how to handle hybrid systems with more general differential equations, including differential equations with fractions, differential inequalities [Walter 1998], differential-algebraic equations [Kunkel and Mehrmann 2006], and differential-algebraic constraints with disturbances. Those more general hybrid systems can be modeled by differential-algebraic programs, for which there is an extension of dL called *differential-algebraic dynamic logic* DAL [Platzer 2010a; Platzer 2010b]. There also is an extension of dL to temporal properties that gives hybrid programs a trace semantics. This extension is called *differential temporal dynamic logic* dTL [Platzer 2010b; Platzer 2007c]. We refer to [Platzer 2010b] for details.

3.2. dL Formulas

Differential dynamic logic dL [Platzer 2007b; Platzer 2008a; Platzer 2010b; Platzer 2012b] is a dynamic logic [Pratt 1976] for hybrid systems. It combines first-order real arithmetic [Tarski 1951] with first-order modal logic [Carnap 1946; Hughes and Cresswell 1996] and dynamic logic [Pratt 1976] 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. 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.

Definition 3.5 (dL *formula*). The *formulas of differential dynamic logic* (dL) are defined by the grammar (where ϕ, ψ are dL formulas, θ_1, θ_2 terms, x a variable, α a HP):

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

The operator $\langle\alpha\rangle$ dual to $[\alpha]$ is defined by $\langle\alpha\rangle\phi \equiv \neg[\alpha]\neg\phi$. Operators $>, \leq, <, \vee, \rightarrow, \leftrightarrow, \exists x$ can be defined as usual, e.g., $\exists x\, \phi \equiv \neg\forall x\, \neg\phi$. 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, $\neg A \wedge B \vee C \rightarrow D \vee E \wedge F \equiv (((\neg A) \wedge B) \vee C) \rightarrow (D \vee (E \wedge F))$.

Definition 3.6 (dℒ *semantics*). The *satisfaction relation* $\nu \models \phi$ for dℒ 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 $[\![\theta_1]\!]_\nu = [\![\theta_2]\!]_\nu$.
— $\nu \models (\theta_1 \geq \theta_2)$ iff $[\![\theta_1]\!]_\nu \geq [\![\theta_2]\!]_\nu$.
— $\nu \models \neg\phi$ iff it is not the case that $\nu \models \phi$.
— $\nu \models \phi \wedge \psi$ iff $\nu \models \phi$ and $\nu \models \psi$.
— $\nu \models \forall x\, \phi$ iff $\nu^d_x \models \phi$ for all $d \in \mathbb{R}$.
— $\nu \models \exists x\, \phi$ iff $\nu^d_x \models \phi$ for some $d \in \mathbb{R}$.
— $\nu \models [\alpha]\phi$ iff $\omega \models \phi$ for all ω with $(\nu, \omega) \in \rho(\alpha)$.
— $\nu \models \langle\alpha\rangle\phi$ iff $\omega \models \phi$ for some ω with $(\nu, \omega) \in \rho(\alpha)$.

If $\nu \models \phi$, then we say that ϕ is true at ν. A dℒ formula ϕ is *valid*, written $\vDash \phi$, iff $\nu \models \phi$ for all states ν.

A dℒ formula of the form $A \rightarrow [\alpha]B$ corresponds to a Hoare triple [Floyd 1967; Hoare 1969], but for hybrid systems. It is valid if, for all states: if dℒ formula A holds (in the initial state), then dℒ formula B holds for all states reachable by following 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 3.7 (*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 (3) in Example 3.3 that always allows braking but acceleration only when $\chi \equiv v \leq 20$ holds:

$$v \geq 0 \wedge A \geq 0 \wedge b > 0 \rightarrow [\mathit{car}_s]v \geq 0$$

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 (3). Because it is implied by the precondition, $v \geq 0$ 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.

A more interesting valid dℒ formula is the following, where car_ε denotes the time-triggered HP from (4) with the choice $\chi \equiv 2b(x - m) \geq v^2 + \big(A + b\big)\big(A\varepsilon^2 + 2\varepsilon v\big)$ as

acceleration constraint:

$$v^2 \leq 2b(m-x) \land A \geq 0 \land b > 0 \rightarrow [\mathit{car}_\varepsilon]x \leq m \tag{5}$$

This dL formula expresses that if, initially, the velocity is not too large ($v^2 \leq 2b(m-x)$) compared to the braking b and remaining distance $m-x$ to a stoplight m, and if $A \geq 0 \land b > 0$, then all states reachable by following HP car_ε satisfy the postcondition $x \leq m$, i.e., the car never passes the stoplight at position m.

The dL formula (5) expresses a safety property, because it says that car_ε always remains safely before the stoplight. But that would be the case for car controllers that never move. Yet, we can also express and prove liveness in dL by showing that the car is able to (note the $\langle\cdot\rangle$ modality) pass every point p by an appropriate choice of the stoplight m:

$$\varepsilon > 0 \land -b > 0 \land A \geq 0 \rightarrow \forall p\, \exists m\, \langle \mathit{car}_\varepsilon \rangle\, x \geq p \tag{6}$$

Statements of this type give alternations of quantifiers and of modalities. See [Platzer 2010b, Sections 2.9 and 7.3] and [Loos and Platzer 2011; Mitsch et al. 2012] for details about models in which m changes dynamically as permission to move changes over time and for details on the proof of dL formula (6).

Example 3.8 (*Single car, multiple modalities*). The fact that dL formula (5) is valid shows that its assumption about the initial state is sufficient for safety. The logic dL can be used to state and prove constraints that are both necessary and sufficient for dynamical properties [Platzer 2010b, Chapter 7]. First, we consider what the proper assumptions about the initial state should be for car control. The HP car_ε in (5), which originates from (4), is a specific car control model deciding under which circumstance to choose which control action. It would not make sense to require a controller to remain safe even in circumstances where no safe control choice is left, e.g., when not even immediate braking would be safe anymore. In dL, we can easily state that we want car_ε to always remain safe at least from those states where braking remains safe:

$$v \geq 0 \land A \geq 0 \land b > 0 \land [x' = v, v' = -b]x \leq m \rightarrow [\mathit{car}_\varepsilon]x \leq m \tag{7}$$

This valid (and provable) dL formula says that if, initially, $v \geq 0 \land A \geq 0 \land b > 0$ holds and if $x \leq m$ would always hold if the decision were to brake immediately (i.e. $[x' = v, v' = -b]x \leq m$), then the more permissive control model car_ε also always remains safe (i.e. $[\mathit{car}_\varepsilon]x \leq m$), because it may accelerate instead, but, due to the choice of constraint χ, will start braking in due time before m. This principle of using modal formulas about simpler dynamical systems to describe states can be very useful for systematically designing controllers. Formula (7) is very intuitive: if braking would be safe, then car_ε will be safe, because it will notice in time when acceleration would not be safe any longer.

The same principle can be used to design how to choose the constraint χ in car_ε. Constraint χ is a design choice that determines under which circumstance the car controller is allowed to choose to accelerate. Since, according to (4), the car controller may possibly not have a chance to react again for up to ε time units, the car controller can only choose to accelerate, if it would be safe to accelerate for ε time units, and, after that, the car still has enough distance to brake (from its then faster velocity) before reaching the stoplight m. This behavior can be expressed by the following dL formula with two nested modalities, the first one for the acceleration for up to time ε, the second one for subsequent braking:

$$[t := 0;\ x' = v, v' = a, t' = 1 \,\&\, v \geq 0 \land t \leq \varepsilon][x' = v, v' = -b]x \leq m$$

In rich-test dL, we can directly use dL formulas with such modalities as tests inside HPs. These are useful and instructive for designing systems, but harder to implement,

because they refer to future states reached when following a dynamics. This is perfect for model-predictive control, but tests on static quantifier-free first-order arithmetic formulas without modalities are easier to implement by simple arithmetic checks for the concrete values of the current state.

The following equivalence shows that the assumption about the initial state in (5) is necessary and sufficient and also explains how dℒ formulas (5) and (7) are related:

$$v \geq 0 \wedge b > 0 \rightarrow \big(v^2 \leq 2b(m-x) \leftrightarrow [x' = v, v' = -b]x \leq m\big)$$

This valid dℒ formula expresses that, if the initial velocity is nonnegative and the braking constant is $b > 0$, then the car will always remain before the stoplight when braking if and only if $v^2 \leq 2b(m-x)$ holds for the initial state. Note that this dℒ formula relates a dynamic statement ($[x' = v, v' = -b]x \leq m$) about the behavior at all future states of a dynamical system to a static statement about its present state. Because the dℒ formula is an equivalence ($\leftrightarrow$), it characterizes all states from which the car can be controlled to remain safely before the stoplight. We refer to [Platzer 2010b, Chapter 7] for more details on equivalent characterizations of dynamical constraints and how they can be used for systematic design.

Example 3.9 (*Bouncing ball*). Consider the bouncing ball (with or without variable damping coefficient c) from Example 3.4 and denote this HP by *ball*. The intuitive property that, if gravity g is positive, the bouncing ball never bounces higher than its initial height H is expressed by the following dℒ formula:

$$h = H \wedge h \geq 0 \wedge g > 0 \rightarrow [\mathit{ball}](0 \leq h \leq H)$$

This dℒ formula may be intuitive, but it is not valid, because the postcondition $0 \leq h \leq H$ will be violated if the initial velocity is positive (climbing). Assuming $v \leq 0$ holds initially,

$$v \leq 0 \wedge h = H \wedge h \geq 0 \wedge g > 0 \rightarrow [\mathit{ball}](0 \leq h \leq H)$$

will, however, still not lead to a valid formula, because the ball would then start falling, but, if it is initially falling very fast (e.g., when dribbling a basket ball), then it will jump back higher than its initial height, despite the damping coefficient c. We refer to [Platzer 2010b] for more details and for properties of bouncing balls that are valid (and provable).

The logic dℒ also supports more complicated nested properties and quantifiers like $\exists p\,[\alpha]\langle\beta\rangle\phi$ which says that there is a choice of parameter p (expressed by $\exists p$) such that for all behaviors of HP α (expressed by $[\alpha]$) there is a reaction of HP β (i.e., $\langle\beta\rangle$) that ensures that ϕ holds in the resulting state. Likewise, $\exists p\,([\alpha]\phi \wedge [\beta]\psi)$ says that there is a choice of parameter p that makes both $[\alpha]\phi$ and $[\beta]\psi$ true, simultaneously, i.e., that makes the conjunction $[\alpha]\phi \wedge [\beta]\psi$ true, saying that formula ϕ holds for all states reachable by runs of HP α and, independently, ψ holds after all runs of HP β. This results in a very flexible logic for specifying and verifying even sophisticated properties of hybrid systems, including the ability to refer to multiple hybrid systems at once in a single formula. This flexibility is useful for computing invariants and differential invariants [Platzer and Clarke 2008; Platzer and Clarke 2009a; Platzer 2010b].

3.3. Hybrid Automata

In this subsection, we discuss hybrid automata and show their close relation to hybrid programs. Besides hybrid programs, hybrid automata [Alur et al. 1995; Henzinger 1996] are another popular notation for hybrid systems and there is a close connection between both models, which we have seen in Examples 3.3 and 3.4. There are numerous slightly different notions of hybrid automata or automata-based models for hybrid

systems [Tavernini 1987; Alur et al. 1992; Nicollin et al. 1992; Alur et al. 1995; Branicky 1995; Henzinger 1996; Alur et al. 1996; Branicky et al. 1998; Lafferriere et al. 1999; Davoren and Nerode 2000; Piazza et al. 2005; Damm et al. 2006]. We review a hybrid automata model close to Henzinger's [Henzinger 1996], yet with polynomial differential equations even if the theoretical model allows others and even if verification tools for hybrid automata focus on subclasses of hybrid automata, e.g., constant [Henzinger et al. 1997; Frehse 2008] or linear dynamics.

Hybrid automata are graph models with two kinds of transitions: discrete jumps in the state space caused by mode switches (edges in the graph), and continuous evolution along continuous flows within a mode (vertices in the graph). Recall the automata in Figure 4 from Example 3.3 and in Figure 5 from Example 3.4 as typical examples.

Definition 3.10 (*Hybrid automaton*). A *hybrid automaton* A consists of

— a finite set $X = \{x_1, \ldots, x_n\}$ of real-valued state variables, where $n \in \mathbb{N}$ is the dimension of A;
— a finite directed multigraph (i.e., graph that may have multiple edges between the same pair of vertices) with vertices Q (as *modes*) and edges E (as *control switches*);
— flow conditions flow_q in mode $q \in Q$, i.e., differential equations $x_1' = \theta_1, \ldots, x_n' = \theta_n$ that determine the relationship of the continuous state variables x_i and their time derivative x_i' during continuous evolution in mode $q \in Q$;
— evolution domain constraints dom_q, which are first-order real arithmetic formulas over X that have to be true of the continuous state while in mode $q \in Q$;
— initial conditions init_q, which are first-order real arithmetic formulas over X that are true of the continuous state if the system starts in mode $q \in Q$;
— guard conditions guard_e, i.e., which are first-order real arithmetic formulas over X that determine whether the automaton can follow edge $e \in E$ depending on whether guard_e is true of the current state value;
— resets reset_e along edge $e \in E$, which are lists of equalities $x_1^+ = \theta_1, \ldots, x_n^+ = \theta_n$ where θ_i is a term over X that determines x_i^+, which denotes the new value of the continuous state variable x_i after following edge $e \in E$;

It is crucial to work with a computational representation [Henzinger 1996], e.g., as first-order real arithmetic formulas, instead of just arbitrary initial set of states $\mathit{init}_q \subseteq \mathbb{R}^n$ and an arbitrary relation $\mathit{flow}_q \subseteq \mathbb{R}^n \times \mathbb{R}^n$ of variables and their derivatives to describe the dynamics. Otherwise computational analysis becomes impossible. If init_q is an undecidable set, it may already be undecidable whether 0 is an initial state ($0 \in \mathit{init}_q$).

Definition 3.11 (*Transition semantics of hybrid automata*). The *transition system* of a hybrid automaton A is a transition relation $\rightsquigarrow$ defined as follows

— $S := \{(q, x) \in Q \times \mathbb{R}^n \,:\, x \models \mathit{dom}_q\}$ is the state space;
— $S_0 := \{(q, x) \in S \,:\, x \models \mathit{init}_q\}$ is the set of initial states;
— $\rightsquigarrow \,\subseteq S \times S$ is the transition relation defined as the union $\bigcup_{e \in E} \overset{e}{\rightsquigarrow} \,\cup\, \bigcup_{q \in Q} \overset{q}{\rightsquigarrow}$ where
  (1) $(q, x) \overset{e}{\rightsquigarrow} (\tilde{q}, \tilde{x})$ iff $e \in E$ is an edge from $q \in Q$ to $\tilde{q} \in Q$ in the hybrid automaton A and $x \models \mathit{guard}_e$ and, further, $\tilde{x}_i = [\![\theta_i]\!]_x$ for $i = 1, \ldots, n$. (*discrete transition*).
  (2) $(q, x) \overset{q}{\rightsquigarrow} (q, \tilde{x})$ iff $q \in Q$ and there is a function $\varphi : [0, r] \to \mathbb{R}^n$ that has a time derivative $\varphi' : (0, r) \to \mathbb{R}^n$ such that $\varphi(0) = x, \varphi(r) = \tilde{x}$ and such that $\varphi_i'(\zeta) = [\![\theta_i]\!]_{\varphi(\zeta)}$ at each $\zeta \in (0, r)$ and for $i = 1, \ldots, n$, where φ_i is the projection of φ to the i-th component. Further, $\varphi(\zeta) \models \mathit{dom}_q$ has to hold for each $\zeta \in [0, r]$. (*continuous transition*).

State $\sigma \in S$ is *reachable* from state $\sigma_0 \in S_0$, denoted by $\sigma_0 \rightsquigarrow^* \sigma$, iff, for some $n \in \mathbb{N}$, there is a sequence of states $\sigma_1, \sigma_2, \ldots, \sigma_n = \sigma \in S$ such that $\sigma_{i-1} \rightsquigarrow \sigma_i$ for $1 \leq i \leq n$.

Just like for classical discrete systems, where every finite automaton can be implemented as a WHILE program, every hybrid automaton can be represented as a hybrid program with a similar construction [Platzer 2010b]. NaÏve compilation introduces additional coding variables, however, which may make verification unnecessarily tedious compared to a direct natural representation as a hybrid program. The following HP has been compiled from the hybrid automaton in Figure 4:

$$
\begin{array}{ll}
q := \textit{accel}; & \text{/* initial mode is node accel */} \\
\big(& (?q = \textit{accel};\ \ x' = v, v' = a \,\&\, v \geq 0) \\
\cup & (?q = \textit{accel};\ \ a := -b;\ \ q := \textit{brake};\ \ ?v \geq 0) \\
\cup & (?q = \textit{accel} \wedge \chi;\ \ q := \textit{accel};\ \ ?v \geq 0) \\
\cup & (?q = \textit{brake};\ \ x' = v, v' = a \,\&\, v \geq 0) \\
\cup & (?q = \textit{brake} \wedge \chi;\ \ a := A;\ \ q := \textit{accel};\ \ ?v \geq 0)\big)^*
\end{array}
$$

Note the difference of this HP compared to the natural HP in Example 3.3. Line 1 represents that, in the beginning, the current node q of the system is the initial node *accel*. The HP represents each discrete and continuous transition of the automaton as a sequence of statements with a nondeterministic choice ($\cup$) between these transitions. Line 2 represents a continuous transition of the automaton. It tests if the current node q is *accel*, and then (i.e., if the test was successful) follows the differential equation system $x' = v, v' = a$ restricted to the evolution domain $v \geq 0$. Line 3 characterizes a discrete transition of the automaton. It tests whether the automaton is in node *accel*, resets $a := -b$ and then switches q to node *brake*. By the semantics of hybrid automata, an automaton in node *accel* is only allowed to make a transition to node *brake* if the evolution domain restriction of *brake* is true when entering the node, which is expressed by the additional test $?v \geq 0$ at the end of line 3. Observe that this test of the evolution domain restriction generally needs to be checked as the last operation after the guard and reset, because a reset like $v := v - 1$ could affect the outcome of the evolution domain region test. In order to obtain a fully compositional model, HPs make all these implicit side conditions explicit. Line 4 represents the discrete transition for the self-loop at *accel* of the automaton. It tests the guard χ when in node *accel*, and, if successful, switches q back to node *accel*, and checks the evolution domain constraint $v \geq 0$ of *accel*. Line 5 represents the continuous transition when staying in node *brake* and following the differential equation system $x' = v, v' = a$ restricted to the evolution domain $v \geq 0$. Line 6 represents the discrete transition from node *brake* of the automaton to node *accel*, again testing the guard in the beginning and testing the evolution domain constraint of *accel* at the end.

Lines 2–6 cannot run unless their tests succeed. In particular, at any state, the nondeterministic choice ($\cup$) among lines 2–6 reduces de facto to a nondeterministic choice between either lines 2–4 or between lines 5–6. At any state, q can have value either *accel* or *brake* (assuming these are different constants), not both. Consequently, when $q =$ *brake*, a nondeterministic choice of lines 2–4 would immediately fail the tests in the beginning and not run any further. The only remaining choices that have a chance to succeed are lines 5–6 then. In fact, only the single successful choice of line 5 would remain if the second conjunct χ of the test in line 6 does not hold for the current state. Note that, still, all four choices in lines 2–6 are available, but at least two of these nondeterministic choices will always be unsuccessful. Note that executions of line 3,4, or 6 would fail if the respective test at the end of those lines fails. Since $v \geq 0$ is in the evolution domain constraints of all nodes, however, the system gets stuck if $v < 0$, which can only happen initially in this system. Finally, the repetition operator (*) at

the end of the HP expresses that the transitions of a hybrid automaton, as represented by lines 2–6, can repeat arbitrarily often, possibly taking different nondeterministic choices between lines 2–6 at every repetition.

We could have defined differential dynamic logic for hybrid automata instead of for hybrid programs, because hybrid automata can be compiled to hybrid programs. The primary reason why we chose a hybrid program representation instead of an automata representation for our logic is because our verification works by structural decomposition and hybrid programs have a perfectly compositional semantics, which enables us to use perfectly compositional proof rules (Sect. 3.4).

3.4. Axiomatization

We do not only use dL for specification purposes but also for verification of hybrid systems. That is, we use dL formulas to specify what properties of hybrid systems we are interested in, and then use dL proof rules to verify them. The axioms and proof rules of dL are syntactic, which means that we can use them to verify properties of hybrid systems without having to recourse to their mathematical semantics. In Sect. 3.5, we show that the semantics and proof rules of dL match completely, so we are not losing anything by taking on a syntactic perspective on verification. Syntactic proof rules are crucial, because they can be implemented and used computationally in a computer (Sect. 3.10).

Our axiomatization of dL is shown in Figure 6. To highlight the logical essentials, we use our axiomatization from our recent result [Platzer 2012b] that is simplified compared to our earlier work [Platzer 2008a], which was tuned for automation. The axiomatization we use here is closer to that of Pratt's dynamic logic for conventional discrete programs [Pratt 1976; Harel et al. 1977]. We use the first-order Hilbert calculus (modus ponens MP and $\forall$-generalization rule $\forall$) as a basis and allow all instances of valid formulas of first-order real arithmetic as axioms. The first-order theory of real-closed fields is decidable [Tarski 1951] by quantifier elimination. We write $\vdash \phi$ iff dL formula ϕ can be *proved* with dL rules from dL axioms (including first-order rules and axioms); see Figure 6. 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 G, MP, $\forall$, whose premises (above the rule bar) are all provable. Our axiomatization in Figure 6 is phrased in terms of $[\cdot]$. Corresponding axioms hold for $\langle\cdot\rangle$ by the defined duality $\langle\alpha\rangle\phi \equiv \neg[\alpha]\neg\phi$; see [Platzer 2010b] for explicit $\langle\cdot\rangle$ rules.

Axiom $[:=]$ is Hoare's assignment rule. It uses substitutions to axiomatize discrete assignments. To show that $\phi(x)$ is true after a discrete assignment, axiom $[:=]$ shows that it has been true before, when substituting the affected variable x with its new value θ. Formula $\phi(\theta)$ is obtained from $\phi(x)$ by *substituting* θ for x, provided x does not occur in the scope of a quantifier or modality binding x or a variable of θ. All substitutions in this paper require this admissibility condition. A modality $[\alpha]$ containing $z :=$ or z' *binds* z (written $z \in BV(\alpha)$ for bound variable). Only variables that are bound by HP α can possibly be changed when running α.

Tests are proven by assuming that the test succeeds with an implication in axiom $[?]$, because test $?\chi$ can only make a transition when condition χ actually holds true. From left to right, axiom $[?]$ for dL formula $[?\chi]\phi$ assumes that formula χ holds true (otherwise there is no transition and thus nothing to show) and shows that ϕ holds after the resulting no-op. The converse implication from right to left is by case distinction. Either χ is false, then $?\chi$ cannot make a transition and there is nothing to show. Or χ is true, but then also ϕ is true.

In axiom $[']$, $y(\cdot)$ is the (unique [Walter 1998, Theorem 10.VI]) solution of the symbolic initial-value problem $y'(t) = \theta, y(0) = x$. Given such a solution $y(\cdot)$, continuous

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

$$[?]\quad [?\chi]\phi \leftrightarrow (\chi \rightarrow \phi)$$

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

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

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

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

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

$$\mathbf{K}\quad [\alpha](\phi\rightarrow\psi) \rightarrow ([\alpha]\phi \rightarrow [\alpha]\psi)$$

$$\mathbf{I}\quad [\alpha^*](\phi \rightarrow [\alpha]\phi) \rightarrow (\phi \rightarrow [\alpha^*]\phi)$$

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

$$\mathbf{B}\quad \forall x\,[\alpha]\phi \rightarrow [\alpha]\forall x\,\phi \qquad (x \notin \alpha)$$

$$\mathbf{V}\quad \phi \rightarrow [\alpha]\phi \qquad (FV(\phi)\cap BV(\alpha)=\emptyset)$$

$$\mathbf{G}\quad \frac{\phi}{[\alpha]\phi}$$

$$\mathbf{MP}\quad \frac{\phi\rightarrow\psi \quad \phi}{\psi}$$

$$\forall\quad \frac{\phi}{\forall x\,\phi}$$

Fig. 6. Differential dynamic logic axiomatization

evolution along differential equation $x'=\theta$ 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 Figure 6. Notice that conventional initial-value problems are numerical with concrete numbers $x \in \mathbb{R}^n$ as initial values, not symbols x [Walter 1998]. 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.

Nondeterministic choices split into their alternatives in axiom $[\cup]$. From right to left: If all α runs lead to states satisfying ϕ (i.e., $[\alpha]\phi$ holds) and all β runs lead to states satisfying ϕ (i.e., $[\beta]\phi$ holds), then all runs of HP $\alpha\cup\beta$, which may choose between following α and following β, also lead to states satisfying ϕ (i.e., $[\alpha\cup\beta]\phi$ holds). The converse implication from left to right holds, because $\alpha\cup\beta$ can run all runs of α and all runs of β. A general principle behind the $\mathsf{d}\mathcal{L}$ axioms is most noticeable in axiom $[\cup]$. The equivalence axioms of $\mathsf{d}\mathcal{L}$ are primarily intended to be used by reducing the formula on the left to the (structurally simpler) formula on the right. With such a reduction, we symbolically decompose a property of a more complicated system into separate properties of easier fragments α and β. This decomposition makes the problem tractable and is good for scalability purposes. For these symbolic structural decompositions, it is very helpful that $\mathsf{d}\mathcal{L}$ is a full logic that is closed under all logical operators, including

disjunction and conjunction, for then both sides in $[\cup]$ are dL formulas again (unlike in Hoare logic [Hoare 1969]). This is also an advantage for computing invariants [Platzer and Clarke 2008; Platzer and Clarke 2009a; Platzer 2010b].

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 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-runs 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 (unlike in Hoare logic [Hoare 1969], 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 $[^*]$ is the iteration axiom, which partially unwinds loops. It uses the fact that ϕ always holds after repeating α (i.e., $[\alpha^*]\phi$), if ϕ holds at the beginning (for ϕ holds after zero repetitions then), and if, after one run of α, ϕ holds after every number of repetitions of α, including zero repetitions (i.e., $[\alpha][\alpha^*]\phi$). So axiom $[^*]$ expresses that $[\alpha^*]\phi$ holds iff ϕ holds immediately and after one or more repetitions of α. Bounded model checking corresponds to unwinding loops N times by axiom $[^*]$ and simplifying the resulting formula in the dL calculus. If the formula is invalid, a bug has been found, otherwise N increases. We use induction axioms I and C for proving formulas with unbounded repetitions of loops.

Axiom K is the modal modus ponens from modal logic [Kripke 1959; Kripke 1963; Hughes and Cresswell 1996]. It expresses that, if an implication $\phi\to\psi$ holds after all runs of α (i.e., $[\alpha](\phi\to\psi)$) and ϕ holds after all runs of α (i.e., $[\alpha]\phi$), then ψ holds after all runs of α (i.e., $[\alpha]\psi$), because ψ is a consequence in each state reachable by α.

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 $[\alpha^*]\phi$.

Axiom C, in which v does not occur in α (written $v\notin\alpha$), is a variation of Harel's convergence rule, suitably adapted to hybrid systems over $\mathbb{R}$. Axiom C expresses that, if, after any number of repetitions of α, $\varphi(v)$ can decrease after some run of α by 1 (or another positive real constant) when $v>0$, then, if $\varphi(v)$ holds for any v, then the variant $\varphi(v)$ holds for some real number $v\leq 0$ after repeating α sufficiently often (i.e., $\langle\alpha^*\rangle\exists v{\leq}0\,\varphi(v)$). This axiom shows that positive progress with respect to $\varphi(v)$ can be achieved by running α. Note that positive progress is only sufficient if it is bounded from below, otherwise progress could converge to zero before reaching the destination.

Axiom B is the Barcan formula of first-order modal logic, characterizing anti-monotonic domains [Hughes and Cresswell 1996]. In order for it to be sound for dL, x must not occur in α. It expresses that, if, from all initial values of x, all runs of α lead to states satisfying ϕ, then, after all runs of α, ϕ holds for all values of x, because the value of x cannot affect the runs of α, nor can x change during runs of α, since $x\notin\alpha$. The converse of B is provable[3] [Hughes and Cresswell 1996, BFC p. 245] and called B.

[3] From $\forall x\,\phi\to\phi$, derive $[\alpha](\forall x\,\phi\to\phi)$ by G, from which K and propositional logic derive $[\alpha]\forall x\,\phi\to[\alpha]\phi$. Then, first-order logic derives $[\alpha]\forall x\,\phi\to\forall x\,[\alpha]\phi$, as x is not free in the antecedent.

Axiom V is for vacuous modalities and requires that no free variable of ϕ (written $FV(\phi)$) is bound by α, because α then cannot change any of the free variables of ϕ. It expresses that, if ϕ holds in a state, then it holds after all runs of α, because, by $FV(\phi) \cap BV(\alpha) = \emptyset$, no variable that α can change occurs free in ϕ. The converse of V holds, but we do not need it. Note that, unlike the other axioms, B, V, and $[^*]$ are not strictly required for proving dL formulas.

Rule G is Gödel's necessitation rule for modal logic [Hughes and Cresswell 1996]. It expresses that, if ϕ is valid, i.e., true in all states, then $[\alpha]\phi$ is valid. Note that, quite unlike rule G, axiom V crucially requires the variable condition that ensures that the value of ϕ is not affected by running α [Platzer 2012b].

Rules MP and $\forall$ are as in first-order logic. Modus ponens (MP) expresses that if we know that both $\phi \to \psi$ and ϕ are valid, then ψ is a valid consequence. The $\forall$-generalization rule ($\forall$) expresses that if ϕ is valid, then so is $\forall x\, \phi$.

The dL axiomatization in Figure 6 uses a modular dL 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 [Platzer 2012b]. It checks χ 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 Figure 7 for an illustration. To simplify notation, we assume that the (vector) differential equation $x' = \theta$ in axiom [&] already includes a clock $x_0' = 1$ for tracking time. The idea behind axiom [&] is that the fresh variable t_0 remembers the initial time x_0, then x evolves forward along $x' = \theta$ for any amount of time. Afterwards, ϕ has to hold if, for all ways of evolving backwards along $x' = -\theta$ for any amount of time, $x_0 \geq t_0 \to \chi$ holds, i.e., χ holds at all previous times that are later than the initial time t_0. Thus, ϕ is not required to hold after a forward evolution if the evolution domain constraint χ can be left by evolving backwards for less time than the forward evolution took.

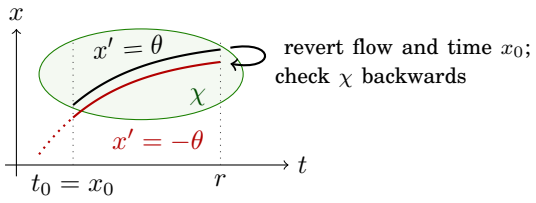

Fig. 7. "There and back again" axiom [&] checks evolution domain along backwards flow over time

The following loop invariant rule ind derives from G and I. Convergence rule con derives from $\forall$-generalization, G, and C (like in C, v does not occur in α):

$$(ind)\quad \frac{\phi \to [\alpha]\phi}{\phi \to [\alpha^*]\phi} \qquad (con)\quad \frac{\varphi(v) \wedge v > 0 \to \langle\alpha\rangle\varphi(v-1)}{\varphi(v) \to \langle\alpha^*\rangle\exists v{\leq}0\, \varphi(v)}$$

While this is not the focus of this paper, we note that we have successfully used a refined sequent calculus variant [Platzer 2008a] of the Hilbert calculus in Figure 6 for automatic verification of hybrid systems, including trains, cars, and aircraft; see Sect. 3.10. Several different verification paradigms can be formulated for the dL calculus by choosing in which order to use the axioms, including proving by symbolic execution, proving by forward image computation, proving by backward image computation, proving by fixpoint loops, and full deduction [Platzer 2010b].

Uses of real arithmetic, which, we denote by $\mathbb{R}$, are decidable by quantifier elimination in real-closed fields [Tarski 1951].

Definition 3.12 (*Quantifier elimination*). A first-order theory admits *quantifier elimination* if, with each formula ϕ, a quantifier-free formula $\mathrm{QE}(\phi)$ can be associated effectively that is equivalent (i.e., $\phi \leftrightarrow \mathrm{QE}(\phi)$ is valid) and has no additional free variables or function symbols. The operation QE is further assumed to evaluate formulas

without variables, yielding a decision procedure for closed formulas of this theory (i.e., formulas without free variables).

Quantifier elimination is decidable in the first-order logic of real-closed fields [Tarski 1951]. It exploits the special structure of real arithmetic to express quantified arithmetic formulas equivalently without quantifiers.

Example 3.13 (*Quantifier elimination*). QE yields the equivalence:

$$\mathrm{QE}(\exists x\,(ax^2+bx+c=0)) \equiv (a\neq 0 \wedge b^2-4ac\geq 0) \vee (a=0 \wedge (b=0 \rightarrow c=0))$$

In this particular case, the equivalence can be found by using the generic condition for solvability of quadratic equations over the reals plus special cases when coefficients are zero.

Quantifier elimination gives a decision procedure for real arithmetic [Tarski 1951]. Implementations use partial cylindrical algebraic decomposition [Collins and Hong 1991], virtual substitution [Weispfenning 1997], semidefinite programming relaxations [Parrilo 2003; Harrison 2007] for Stengle's Positivstellensatz [Stengle 1973], or Gröbner bases for the real Nullstellensatz [Platzer et al. 2009], which combine Gröbner bases [Buchberger 1965] with Stengle's real Nullstellensatz [Stengle 1973] and semidefinite programming [Boyd and Vandenberghe 2004].

For the purposes of this survey, we denote the use of decidable real arithmetic and quantifier elimination in proofs by $\mathbb{R}$. More constructive deduction modulo proof rules, which can be used to combine first-order real arithmetic with the proof calculus presented here and that are suitable for automation, have been reported in previous work [Platzer 2008a; Platzer 2010a; Platzer 2010b]. Those are based on real-valued free variables, Skolemization, Deskolemization, and the following lifting of quantifier elimination in real-closed fields [Tarski 1951; Collins 1975].

LEMMA 3.14 (QUANTIFIER ELIMINATION LIFTING [PLATZER 2008A]). *Quantifier elimination can be lifted to instances of formulas of first-order theories that admit quantifier elimination, i.e., to formulas that result from the base theory by substitution.*

Example 3.15 (*Single car*). In order to illustrate how the dL calculus can be used to prove dL formulas and identify parameter constraints required for them to be valid, we consider a dL formula for the braking case of HP (3):

$$v\geq 0 \wedge x\leq m \rightarrow [a:=-b; x'=v, v'=a]x\leq m \tag{8}$$

Formula (8) claims a car would never run a stoplight if it starts before the stoplight ($x\leq m$) and is applying the brakes. Since braking is the safest operation for cars, we might think that car control would always be safe in this most conservative scenario. But that is not the case. If the car starts off too fast compared to the remaining distance to the stoplight, then not even braking can prevent a crash. We can easily find out, however, under which circumstance the dL formula (8) is valid by applying dL axioms to it. The following dL proof reveals that (8) is valid if $v^2\leq 2b(m-x)$ holds initially:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{v\geq 0 \wedge x\leq m \rightarrow v^2\leq 2b(m-x)}{v\geq 0 \wedge x\leq m \rightarrow \forall t\geq 0\,(\frac{-b}{2}t^2+vt+x\leq m)}\,\mathbb{R}}{v\geq 0 \wedge x\leq m \rightarrow [a:=-b]\forall t\geq 0\,(\frac{a}{2}t^2+vt+x\leq m)}\,[:=]}{v\geq 0 \wedge x\leq m \rightarrow [a:=-b]\forall t\geq 0\,[x:=\frac{a}{2}t^2+vt+x]x\leq m}\,[:=]}{v\geq 0 \wedge x\leq m \rightarrow [a:=-b][x'=v,v'=a]x\leq m}\,[']}{v\geq 0 \wedge x\leq m \rightarrow [a:=-b; x'=v, v'=a]x\leq m}\,[;]$$

$C \to [a := -b][x'' = a]E$	1
$[a := -b; x'' = a]E \leftrightarrow [a := -b][x'' = a]E$	2: by [;]
$([a := -b; x'' = a]E \leftrightarrow [a := -b][x'' = a]E) \to ((C \to [a := -b][x'' = a]E) \to (C \to [a := -b; x'' = a]E))$	3: by $\mathbb{R}$
$(C \to [a := -b][x'' = a]E) \to (C \to [a := -b; x'' = a]E)$	4: MP(2,3)
$C \to [a := -b; x'' = a]E$	5: MP(4,1)

Fig. 8. dℒ Hilbert proof corresponding to bottom proof step in Example 3.15, abbreviating $v \geq 0 \land x \leq m$ by C, $x \leq m$ by E, and $x' = v, v' = a$ by $x'' = a$

We follow the conventions in sequent calculus and read proofs bottom-up, from the desired conclusion at the bottom to the premises at the top; see previous work [Platzer 2008a] for more details on a sequent calculus for dℒ. Here, we first apply the axiom [;] to reduce the sequential composition equivalently to a nested modality. Then we use axiom [′] to reduce the differential equation to an assignment with the solution and a quantifier $\forall t$ for its duration. Even though it is a quantifier over a real variable, we cannot use the decision procedure of quantifier elimination for real-closed fields [Tarski 1951] to handle it, because we do not have a formula of first-order real arithmetic, but still a dℒ formula with a modality expressing a property of all reachable states. Instead, we first use axiom [:=] twice to equivalently substitute in the effect of the assignments. Finally, we use equivalences of real arithmetic (using quantifier elimination, denoted $\mathbb{R}$) to discover the constraint $v^2 \leq 2b(m-x)$, which is required to make (8) valid. Indeed, if we add this so-discovered constraint about the initial state, the following dℒ formula is provable in the dℒ calculus by a minor variation of the above dℒ proof:

$$v^2 \leq 2b(m-x) \land v \geq 0 \land x \leq m \to [a := -b; x' = v, v' = a]x \leq m$$

This construction explains why $v^2 \leq 2b(m-x)$ has to be assumed in dℒ formula (5), because braking is one of the choices for its HP car_ε. Constructions based on this principle turn out to be very effective for discovering invariants [Platzer and Clarke 2008; Platzer and Clarke 2009a; Platzer 2010b] and constraints on the free parameters of the system [Platzer 2008a; Platzer and Quesel 2009; Platzer 2010b], or design constraints for closed-loop properties [Aréchiga et al. 2012]. Another interesting observation is that parameter constraints discovered in this way from precise proofs about simplified models are useful for deriving design decisions about the full system, even for aspects that have not been modeled, like camera resolutions and frame rates for video-based car safety technology [Mitsch et al. 2012; Platzer 2010b].

The reader should note that we use an abbreviated notation for proofs here. Without abbreviations, the bottom-most proof step [:=] in the above proof expands to the Hilbert-style dℒ proof shown in Figure 8, in which each line corresponds to a dℒ axiom or the result of a dℒ proof rule applied to previous lines as indicated in the column on the right. In this paper, we abbreviate Hilbert proofs by rewriting formulas using the dℒ axioms directly, e.g., by replacing instances of the left-hand side of an equivalence in a dℒ axiom by the corresponding (structurally) simpler right-hand side. This abbreviated style can be understood systematically in a sequent calculus formulation [Platzer 2008a] of the dℒ proof calculus.

For typesetting purposes for a dℒ proof for dℒ formula (5), let us abbreviate $x' = v, v' = a, t' = 1 \,\&\, v \geq 0 \land t \leq \varepsilon$ in the HP car_ε from (5) by $x'' = a \,\&\, t \leq \varepsilon$. Further-

more, we abbreviate $v^2 \leq 2b(m-x) \wedge A \geq 0 \wedge b > 0$ by ϕ.

$$\phi \rightarrow (\chi \rightarrow [a := A][t := 0; x'' = a \,\&\, t \leq \varepsilon]\phi) \wedge [a := -b][t := 0; x'' = a \,\&\, t \leq \varepsilon]\phi$$

$$[?]\ \frac{\phi \rightarrow [?\chi][a := A][t := 0; x'' = a \,\&\, t \leq \varepsilon]\phi \wedge [a := -b][t := 0; x'' = a]\phi}{\phi \rightarrow [?\chi; a := A][t := 0; x'' = a \,\&\, t \leq \varepsilon]\phi \wedge [a := -b][t := 0; x'' = a \,\&\, t \leq \varepsilon]\phi}$$

$$[;]\ \frac{\phi \rightarrow [?\chi; a := A][t := 0; x'' = a \,\&\, t \leq \varepsilon]\phi \wedge [a := -b][t := 0; x'' = a \,\&\, t \leq \varepsilon]\phi}{\phi \rightarrow [(?\chi; a := A) \cup a := -b][t := 0; x'' = a \,\&\, t \leq \varepsilon]\phi}$$

$$[\cup]\ \frac{\phi \rightarrow [(?\chi; a := A) \cup a := -b][t := 0; x'' = a \,\&\, t \leq \varepsilon]\phi}{\phi \rightarrow [((?\chi; a := A) \cup a := -b); t := 0; x'' = a \,\&\, t \leq \varepsilon]\phi}$$

$$[;]\ \frac{\phi \rightarrow [((?\chi; a := A) \cup a := -b); t := 0; x'' = a \,\&\, t \leq \varepsilon]\phi}{ind\ \phi \rightarrow [((?\chi; a := A) \cup a := -b); t := 0; x'' = a \,\&\, t \leq \varepsilon)^*]\phi}$$

$$\mathrm{K,G}\ \frac{\phi \rightarrow [((?\chi; a := A) \cup a := -b); t := 0; x'' = a \,\&\, t \leq \varepsilon)^*]x \leq m}{\phi \rightarrow [((?\chi; a := A) \cup a := -b); t := 0; x'' = a \,\&\, t \leq \varepsilon)^*]x \leq m}$$

The first (bottom-most) step uses that $\phi \rightarrow x \leq m$ is provable in arithmetic. From this, G proves $[\mathit{car}_\varepsilon](\phi \rightarrow x \leq m)$, and, thus, K proves $[\mathit{car}_\varepsilon]\phi \rightarrow [\mathit{car}_\varepsilon]x \leq m$. The right conjunct in the top-most premise can be proven by a minor variation of the d$\mathcal{L}$ proof for (8). The d$\mathcal{L}$ proof for the left conjunct in the premise works similarly, but requires slightly more involved arithmetic and the choice $\chi \equiv 2b(x-m) \geq v^2 + \big(A+b\big)\big(A\varepsilon^2 + 2\varepsilon v\big)$. A full proof about a similar system can be found in our book [Platzer 2010b, Section 2.9].

3.5. Soundness and Completeness

The d$\mathcal{L}$ calculus is *sound* [Platzer 2008a; Platzer 2012b], that is, every formula that is provable using the d$\mathcal{L}$ axioms and proof rules is valid, i.e., true in all states. That is, for all d$\mathcal{L}$ formulas ϕ:

$$\vdash \phi \text{ implies } \models \phi \tag{9}$$

Soundness should be *sine qua non* for formal verification, but, for fundamental reasons [Platzer and Clarke 2007; Collins 2007], is so complex for hybrid systems that it is sometimes inadvertently forsaken. In logic, we ensure soundness easily just by checking it locally once for each axiom and proof rule. Thus, no matter how complicated a proof, the proven d$\mathcal{L}$ formula is valid, because it is a (complicated) consequence of lots simple valid proof steps.

More intriguingly, however, our logical setting also enables us to ask the converse: is the d$\mathcal{L}$ proof calculus *complete*, i.e., can it prove all that is true? That is, does the converse of (9) hold? A simple corollary to Gödel's incompleteness theorem shows that already the fragments for discrete dynamical systems and for continuous dynamical systems are incomplete.

THEOREM 3.16 (INCOMPLETENESS [PLATZER 2008A]). *Both the discrete fragment and the continuous fragment of* d$\mathcal{L}$ *are* not effectively axiomatizable*, i.e., they have no sound and complete effective calculus, because natural numbers are definable in both fragments.*

PROOF. Gödel's incompleteness theorem [Gödel 1931] applies to the discrete fragment of d$\mathcal{L}$, because natural numbers are definable in that fragment by repeated addition:

$$\mathit{nat}(n) \;\leftrightarrow\; \langle x := 0; (x := x + 1)^* \rangle\, x = n$$

Gödel's incompleteness theorem [Gödel 1931] applies to the continuous fragment of d$\mathcal{L}$, because an isomorphic copy of the natural numbers is definable using linear differential equations, which characterize solutions $\sin$ and $\cos$, whose zeros (see Figure 9), as detected by τ, correspond to natural numbers, scaled by π:

$$\mathit{nat}(n) \;\leftrightarrow\; \exists s\, \exists c\, \exists \tau\, (s = 0 \wedge c = 1 \wedge \tau = 0 \wedge \langle s' = c, c' = -s, \tau' = 1 \rangle (s = 0 \wedge \tau = n))$$

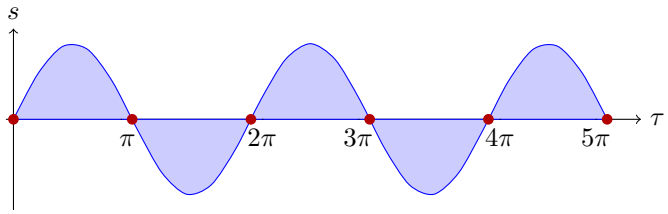

Fig. 9. Characterization of $\mathbb{N}$ as zeros of solutions of differential equations

□

Incompleteness is not unexpected since hybrid systems contain a computationally complete sublanguage and because reachability of hybrid systems is not semidecidable [Henzinger 1996]. Yet, it is instructive to understand the above simple proof based on a classical result about logic.

In logic, the suitability of an axiomatization can still be established by showing completeness relative to a fragment [Cook 1978; Harel et al. 1977]. This *relative completeness*, in which we assume we were able to prove valid formulas in a fragment and prove that we can then prove all others, tells us how subproblems are related computationally. It tells us whether one subproblem dominates the others. Standard relative completeness [Cook 1978; Harel et al. 1977], however, which works relative to the data logic, is inadequate for hybrid systems, whose complexity comes from the dynamics, not the data logic, first-order real arithmetic, which is perfectly decidable [Tarski 1951].

We have shown that both the original dℒ sequent calculus [Platzer 2008a] and the Hilbert-type calculus in Figure 6 [Platzer 2012b] are sound and complete axiomatizations of dℒ relative to the continuous fragment (FOD). FOD is the *first-order logic of differential equations*, i.e., first-order real arithmetic augmented with formulas expressing properties of differential equations, that is, dℒ formulas of the form $[x' = \theta]F$ with a first-order formula F. Note that axioms B and V are not needed for the proof of the following theorem.

THEOREM 3.17 (RELATIVE COMPLETENESS OF dℒ [PLATZER 2008A; PLATZER 2012B]). *The* dℒ *calculus is a* sound and complete axiomatization *of hybrid systems relative to FOD, i.e., every valid* dℒ *formula can be derived from FOD tautologies:*

$$\vDash \phi \ \textit{iff}\ \mathrm{Taut}_{FOD} \vdash \phi$$

This central result shows that we can prove properties of hybrid systems in the dℒ calculus exactly as good as properties of differential equations can be proved. One direction is obvious, because differential equations are part of hybrid systems, so we can only understand hybrid systems to the extent that we can reason about their differential equations. We have shown the other direction by proving that all true properties of hybrid systems can be reduced effectively in the dℒ calculus to elementary properties of differential equations. Moreover, the dℒ proof calculus for hybrid systems can perform this reduction constructively and, vice versa, provides a provably perfect lifting of every approach for differential equations to hybrid systems.

Another important consequence of this result is that decomposition can be successful in taming the complexity of hybrid systems. The dℒ proof calculus is strictly compositional. All proof rules prove logical formulas or properties of HPs by reducing them to structurally simpler dℒ formulas. As soon as we understand that the hybrid systems complexity comes from a combination of several simpler aspects, we can, hence, tame the system complexity by reducing it to analyzing the dynamical effects of simpler parts. This decomposition principle makes it possible for dℒ proofs to scale to interest-

ing systems in practice. Theorem 3.17 gives the theoretical evidence why this principle works in general, not just in the case studies we have considered so far. This is a good illustration of our principle of multi-dynamical systems and even a proof that the decompositions behind the multi-dynamical systems approach are successful. Note that, even though Theorem 3.17 proves (constructively) that every true property of hybrid systems can be proved in the $\mathsf{d}\mathcal{L}$ calculus by decomposition from elementary properties of differential equations, it is still an interesting question which decompositions are most efficient.

For an even more surprising "converse" result proving a sound and complete axiomatization of $\mathsf{d}\mathcal{L}$ relative to the discrete fragment of $\mathsf{d}\mathcal{L}$, we refer to recent work [Platzer 2012b]. That proof is again a constructive reduction, proving that hybrid dynamics, continuous dynamics, and discrete dynamics are proof-theoretically equivalently reducible in the $\mathsf{d}\mathcal{L}$ calculus. Even though the nature of each kind of dynamics is fundamentally different, they still enjoy a perfect proof-theoretical correspondence. In a nutshell, we have shown that we can proof-theoretically equate:

$$\text{“}\textit{hybrid} = \textit{continuous} = \textit{discrete}\text{”}$$

A discussion of this fundamental result about the nature of hybridness is beyond the scope of this paper; we refer to previous work [Platzer 2012b].

3.6. Differential Invariants

The $\mathsf{d}\mathcal{L}$ axiomatization in Figure 6 is sound and complete relative to FOD. But Figure 6 only has a very simple proof rule for differential equations ($[']$) based on computing a solution of the differential equation; we refer to previous work for discretization techniques [Platzer 2012b]. For proving more complicated differential equations by induction, $\mathsf{d}\mathcal{L}$ provides *differential invariants* and *differential variants* [Platzer 2010a], which have been introduced in 2008 [Platzer 2010a] and later refined to a procedure that computes differential invariants in a fixed-point loop [Platzer and Clarke 2008; Platzer and Clarke 2009a]. All premier proof principles for discrete loops are based on some form of induction. Theorem 3.17 and its discrete converse [Platzer 2012b] prove that verification techniques that are successful for discrete systems generalize to continuous and hybrid systems and vice versa. Differential invariants and differential variants can be considered as one (of many possible) constructive and practical consequences of this result. Differential induction defines induction for differential equations. It resembles induction for discrete loops (rule ind) but works for differential equations instead and uses a *differential formula* ($F'^{\theta}_{x'}$, which we develop below) for the induction step.

$$\text{(DI)}\quad \frac{\chi \rightarrow F'^{\theta}_{x'}}{F \rightarrow [x' = \theta \,\&\, \chi]F}$$

This *differential induction* rule is a natural induction principle for differential equations. The difference compared to ordinary induction for discrete loops is that the evolution domain constraint χ is assumed in the premise (because the continuous evolution is not allowed to leave its evolution domain constraint) and that the induction step uses the differential formula $F'^{\theta}_{x'}$ corresponding to formula F and the differential equation $x' = \theta$ instead of a statement that the loop body preserves the invariant. Intuitively, the *differential formula* $F'^{\theta}_{x'}$ captures the infinitesimal change of formula F over time along $x' = \theta$, and expresses the fact that F is only getting more true when following the differential equation $x' = \theta$. The semantics of differential equations is defined in a mathematically precise but computationally intractable way using analytic differentiation and limit processes at infinitely many points in time. The key

point about differential invariants is that they replace this precise but computationally intractable semantics with a computationally effective, algebraic and syntactic total derivative F' along with simple substitution of differential equations. The valuation of the resulting computable formula $F'^{\theta}_{x'}$ along differential equations coincides with analytic differentiation.

Definition 3.18 (*Derivation*). The operator D that is defined as follows on terms is called *syntactic (total) derivation*:

$$D(r) = 0 \qquad \text{for numbers } r \in \mathbb{Q} \tag{10a}$$
$$D(x) = x' \qquad \text{for variable } x \tag{10b}$$
$$D(a+b) = D(a) + D(b) \tag{10c}$$
$$D(a-b) = D(a) - D(b) \tag{10d}$$
$$D(a \cdot b) = D(a) \cdot b + a \cdot D(b) \tag{10e}$$
$$D(a/b) = (D(a) \cdot b - a \cdot D(b))/b^2 \tag{10f}$$

We extend it to (quantifier-free) first-order real-arithmetic formulas F as follows:

$$D(F \wedge G) \equiv D(F) \wedge D(G) \tag{11a}$$
$$D(F \vee G) \equiv D(F) \wedge D(G) \tag{11b}$$
$$D(a \geq b) \equiv D(a) \geq D(b) \qquad \text{accordingly for } <, >, \leq, = \tag{11c}$$

We abbreviate $D(F)$ by F' and define $F'^{\theta}_{x'}$ as the result of substituting θ for x' in F', which is a Lie-type operator [Lie 1893].

The conditions (10) define a derivation operator on terms that (11) lifts conjunctively to logical formulas. It is important for the soundness of DI to define $(F \vee G)'$ as $F' \wedge G'$, because both subformulas need to satisfy the induction step, it is not enough if F satisfies the induction step F' and G holds initially; see [Platzer 2010a; Platzer 2010b] for details and alternatives. We assume for simplicity that formulas use dualities like $\neg(a \geq b) \equiv a < b$ to avoid negations; see [Platzer 2010a; Platzer 2010b] for a discussion of this and the $\neq$ operator. For a discussion why this definition of differential invariants gives a sound approach and many other attempts would be unsound, we refer to previous work [Platzer 2010a; Platzer 2010b]. We also refer to previous work [Platzer 2010b] for discussions about which weaker conditions like $D(a > b) \equiv D(a) \geq D(b)$ are sound. It is crucial, however, to realize that it would generally be unsound to assume F or the boundary of F in the premise. Otherwise, we could draw the counterfactual conclusion that $-(x-y)^2 \geq 0$ is an invariant of $x' = 1, y' = y$. See previous work [Platzer 2010a; Platzer 2010b; Platzer 2012d] for an explanation under which circumstances this assumption would be sound.

The basic idea behind rule DI is that the premise of DI shows that the total derivative F' holds within evolution domain χ when substituting the differential equations $x' = \theta$ into F'. If F holds initially (antecedent of conclusion), then F itself always stays true (succedent of conclusion). Intuitively, the premise gives a condition showing that, within χ, the total derivative F' along the differential constraints is pointing inwards or transversally to F but never outwards to $\neg F$; see Figure 10 for an illustration. Hence, if we start in F and, as indicated by F', the local dynamics never points outside F, then the system always stays in F when following the dynamics. Observe that, unlike F', the premise of DI is a well-formed formula, because all differential expressions are replaced by non-differential terms when forming $F'^{\theta}_{x'}$. It is

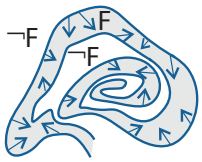

Fig. 10. Differential invariant F for safety

possible and insightful to give a meaning to the differential formula F' itself in differential states [Platzer 2010a]. Crucial for soundness is the result that the valuation of syntactic derivatives along differential equations coincides with analytic differentiation. This derivation lemma plays a role similar to the substitution lemma in classical logic, but for derivations and differential equations. For simplicity, we report a derivation lemma that already combines differential substitution [Platzer 2010a, Lemma 2] with the derivation lemma [Platzer 2010a, Lemma 1]. These results form the basis for more general differential transformations [Platzer 2010b, Sect. 3.5].

LEMMA 3.19 (DERIVATION LEMMA [PLATZER 2010A]). *Let $x' = \theta \,\&\, \chi$ be a differential equation with evolution domain constraint χ and let $\varphi : [0,r] \to (V \to \mathbb{R})$ be a corresponding solution of duration $r > 0$, where V is the set of variables. Then for all terms c and all $\zeta \in [0,r]$:*

$$\frac{\mathsf{d}\,[\![c]\!]_{\varphi(t)}}{\mathsf{d}t}(\zeta) = [\![c'^{\theta}_{x'}]\!]_{\varphi(\zeta)} \quad .$$

In particular, $[\![c]\!]_{\varphi(t)}$ is continuously differentiable.

Example 3.20 (*Rotational dynamics*). The rotational dynamics $x' = y, y' = -x$ is complicated to the extent that the solution involves trigonometric functions, which are generally outside decidable classes of arithmetic (see proof of Theorem 3.16). Yet, we can easily prove properties about the solution using DI and decidable polynomial arithmetic. As a simple example, we can prove that $x^2 + y^2 \geq p^2$ is a differential invariant of the dynamics using the following $\mathsf{d}\mathcal{L}$ proof:

$$\cfrac{\cfrac{\cfrac{}{true}}{2xy + 2y(-x) \geq 0}\,\mathbb{R}}{(2xx' + 2yy' \geq 0)^{y}_{x',\,y'^{-x}}}\,\text{DI}}{x^2 + y^2 \geq p^2 \rightarrow [x' = y, y' = -x]x^2 + y^2 \geq p^2}$$

Example 3.21 (*Quartic dynamics*). The following simple $\mathsf{d}\mathcal{L}$ proof uses DI to prove an invariant of a quartic dynamics.

$$\cfrac{\cfrac{\cfrac{}{true}}{a \geq 0 \rightarrow 2x^2((x-3)^4 + a) \geq 0}\,\mathbb{R}}{a \geq 0 \rightarrow (2x^2x' \geq 0)^{(x-3)^4+a}_{x'}}\,\text{DI}}{x^3 \geq -1 \rightarrow [x' = (x-3)^4 + a \,\&\, a \geq 0]x^3 \geq -1}$$

Observe that rule DI directly makes the evolution domain constraint $a \geq 0$ available as an assumption in the premise, because the continuous evolution is never allowed to leave it. This is useful if we have a strong evolution domain constraint or can make it strong during the proof, which is what we consider in Sect. 3.8.

Counterexample 3.22 (*Negative equations*). It is crucial for soundness that we do not define $D(a \neq b)$ to be $D(a) \neq D(b)$. Otherwise we could draw the wrong conclusion that $x \neq 0$ is an invariant of $x' = 1$ from the fact that $D(x) \neq 0$, i.e., $1 \neq 0$. This would be counterfactual, because variable x can reach $x = 0$ without its derivative ever being 0. In fact, x can only reach $x = 0$ from an initial state $x \neq 0$ if its derivative is nonzero at some point. Instead, we could define $D(a \neq b) \equiv D(a) = D(b)$ if needed. Intuitively, if a and b have the same derivative, then, $a \neq b$ is an invariant if it holds initially. This definition is already included, because we can equivalently rewrite $a \neq b$ to $a > b \vee a < b$ and use the weaker condition $D(a > b) \equiv D(a) \geq D(b)$ to obtain

$D(a > b \vee a < b) \equiv D(a) \geq D(b) \wedge D(a) \leq D(b)$, which is equivalent to $D(a) = D(b)$. Sometimes, it can be more efficient to split the reasoning into two cases, the case where $a > b$ and the case where $a < b$ and consider them separately to prove that $a > b$ is an invariant if it was true initially and that $a < b$ is an invariant if that was true initially, instead of proving the stronger condition $D(a \neq b) \equiv D(a) = D(b)$.

More advanced uses of differential invariants can be found in previous work [Platzer 2010a; Platzer and Clarke 2008; Platzer 2010b; Platzer and Clarke 2009b; Platzer and Quesel 2009; Platzer 2012d]. Differential dynamic logic proofs with differential invariants have been instrumental in enabling the verification of more complicated hybrid systems, including separation properties in complex curved flight collision avoidance maneuvers for air traffic control [Platzer 2010b; Platzer and Clarke 2009b], advanced safety, reactivity and controllability properties of train control systems with disturbance and PI controllers [Platzer 2010b; Platzer and Quesel 2009], and properties of electrical circuits [Platzer 2010b]. Differential invariants are also the proof technique of choice for differential inequalities, differential-algebraic equations, and differential equations with disturbances [Platzer 2010a; Platzer 2010b].

Differential invariants enjoy closure properties, e.g.:

LEMMA 3.23 (CLOSURE PROPERTIES OF DIFFERENTIAL INVARIANTS [PLATZER 2010A]). *Differential invariants are closed under conjunction, differentiation, and propositional equivalences.*

We refer to previous work [Platzer 2012d; Platzer 2010a] for more details on the structure of differential invariants and a complete investigation of the relative deductive power of several classes of differential invariants; see Figure 11 for an overview of classes of differential invariants restricted to the operators as indicated, where strict inclusions of the deductive power are indicated by $\hookrightarrow$, equivalences of deductive power are indicated by $=$, and incomparable deductive powers are indicated by $\curlywedge$.

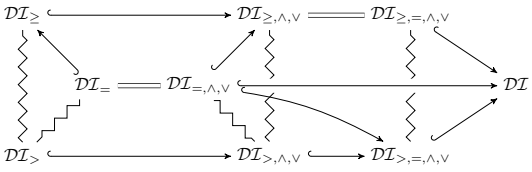

Fig. 11. Differential invariance chart

We also refer to previous work [Platzer 2010a; Platzer 2010b] for the technique of *differential axiomatization*, which is useful for transforming sophisticated non-polynomial differential equations into polynomial differential equations by introducing new variables. This is beneficial because, even though the solutions of the resulting polynomial differential equations are still equally complicated, we never need the solutions when working with differential invariants. Differential invariants depend on the right-hand side of the differential equations, which is then polynomial and, thus, leads to decidable arithmetic.

3.7. Differential Variants

Differential variants [Platzer 2010a] use ideas similar to those behind differential invariants, except that they use progress arguments so that differential variants can be used to prove formulas of the form $\langle x' = \theta \,\&\, \chi\rangle F$. That is, differential variants prove that the system can make progress along $x' = \theta$ to finally reach F without having left χ before; see the right side of Figure 10 for an illustration.

$$\text{(DV)}\quad \frac{\exists \varepsilon{>}0\, \forall x\, (\neg F \wedge \chi \rightarrow (F' \geq \varepsilon)^{\theta}_{x'})}{[x' = \theta \,\&\, {\sim}F]\chi \rightarrow \langle x' = \theta \,\&\, \chi\rangle F}$$

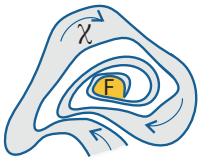

Fig. 12. Differential variant for progress

In rule DV, $F' \geq \varepsilon$ is a mnemonic notation for replacing all occurrences of inequalities $a \geq b$ in F' with $a \geq b + \varepsilon$ and $a > b$ by $a > b + \varepsilon$ (accordingly for $\leq, >, <$). Intuitively, the premise expresses that, wherever χ holds but F does not yet hold, the total derivative is pointing towards F; see right side of Figure 12. Especially, $F' \geq \varepsilon$ guarantees a minimum progress rate of ε towards F along the dynamics. To further ensure that the continuous evolution towards F remains within χ, the antecedent of the conclusion shows that χ holds *until* F is attained, which can again be proven using DI. Overall, the premise of rule DV shows that the dynamics makes progress (at least some ε) toward F, and the antecedent shows that the dynamics does not leave the evolution domain restriction χ on the way to F. In this context, $\sim F$ is a shorthand notation for *weak negation*, i.e., the operation that behaves like $\neg$, except that $\sim(a \geq b) \equiv a \leq b$ and $\sim(a > b) \equiv a \leq b$. Unlike negation $\neg F$, weak negation $\sim F$ retains the boundary of F, which is required in DV as χ needs to continue to hold (including the boundary of F) until F is reached. Especially, for rule DV, invariant χ is not required to hold after F has been reached successfully. The operations $F' \geq \varepsilon$ and $\sim F$ are defined accordingly for other inequalities (in rule DV, we do not permit F to contain equalities, because they could lead to unsoundness). We refer to previous work [Platzer 2010a; Platzer 2010b] for details. Note that the order of quantifiers in DV is crucial for soundness [Platzer 2010a; Platzer 2010b] to avoid Zeno progress that never reaches F despite always getting closer.

Example 3.24 (*Progress discovery*). Consider the simple property $\langle x' = a \rangle x \geq b$, i.e., that we can finally reach region $x \geq b$, when following the dynamics $x' = a$ long enough. We analyze this dL formula in the following dL proof:

$$\dfrac{\dfrac{\dfrac{a > 0}{\exists \varepsilon{>}0\, \forall x\, (x \leq b \rightarrow a \geq \varepsilon)}\,\mathbb{R}}{\exists \varepsilon{>}0\, \forall x\, (x \leq b \rightarrow (x' \geq \varepsilon)^{a}_{x'})}}{\langle x' = a \rangle x \geq b}\,\text{DV}$$

As the dL proof reveals, the dL formula is valid if $a > 0$. This makes sense, because the system dynamics is then evolving towards $x \geq b$; otherwise it is evolving away from $x \geq b$ (if $a < 0$) or is constant ($a = 0$). This proof constructs the answer that $a > 0$ is the required condition, which illustrates how constraints on parameters can be found by dL proofs. For the above proof, we do not need to solve the differential equations. Solving the differential equation would be trivial here for a constant a, but is more involved when a is an arbitrary term. With DV, we just form the total differential of $F \equiv x \geq b$, which gives $F' = x' \geq b'$. When we substitute in the differential equations $x' = a$, we obtain ${F'}^{a}_{x'} \equiv a \geq 0$. Consequently, $(F' \geq \varepsilon)^{a}_{x'}$ gives $a \geq \varepsilon$. If we prove $a \geq \varepsilon$ holds for one common minimum progress $\varepsilon > 0$, then the system makes some minimum progress towards the goal and will reach it in finite time. This even holds if we restrict the progress condition to all x that have not yet reached $x \geq b$ or are on the boundary of $x \geq b$, which is the assumption $x \leq b$ in the premise.

3.8. Differential Cuts

Differential cuts [Platzer 2010a] are another fundamental proof principle for differential equations. They can be used to strengthen assumptions in a sound way:

$$\text{(DC)}\quad \frac{F \rightarrow [x' = \theta \,\&\, \chi]C \qquad F \rightarrow [x' = \theta \,\&\, (\chi \wedge C)]F}{F \rightarrow [x' = \theta \,\&\, \chi]F}$$

$$\dfrac{\dfrac{\dfrac{}{5y^4y^2 \geq 0}\,\mathbb{R}}{(5y^4y' \geq 0)^{(x-3)^4+y^5\ y^2}_{x'\qquad y'}}\,\text{DI}}{y^5 \geq 0 \rightarrow [x' = (x-3)^4 + y^5, y' = y^2]y^5 \geq 0}\qquad \dfrac{\dfrac{\dfrac{}{y^5 \geq 0 \rightarrow 2x^2((x-3)^4+y^5) \geq 0}\,\mathbb{R}}{y^5 \geq 0 \rightarrow (2x^2x' \geq 0)^{(x-3)^4+y^5\ y^2}_{x'\qquad y'}}\,\text{DI}}{x^3 \geq -1 \rightarrow [x' = (x-3)^4 + y^5, y' = y^2 \,\&\, y^5 \geq 0]x^3 \geq -1}$$

$$\text{DC}\quad \dfrac{}{x^3 \geq -1 \wedge y^5 \geq 0 \rightarrow [x' = (x-3)^4 + y^5, y' = y^2]x^3 \geq -1}$$

Fig. 13. Differential cut proof for multidimensional nonlinear dynamics

The differential cut rule works like a cut, but for differential equations. In the right premise, rule DC restricts the system evolution to the subdomain $\chi \wedge C$ of χ, which restricts the system dynamics to a subdomain but this change is a pseudo-restriction, because the left premise proves that the extra restriction C on the system evolution is an invariant anyhow (e.g. using rule DI). Rule DC is special in that it changes the dynamics of the system (it adds a constraint to the system evolution domain region that the resulting system is never allowed to leave by Def. 3.2), but it is still sound, because this change does not reduce the reachable set. The benefit of rule DC is that C will (soundly) be available as an extra assumption for all subsequent DI uses on the right premise of DC. The differential cut rule DC can be used to strengthen the right premise with more and more auxiliary differential invariants C that cut down the state space and will be available as extra assumptions to prove the right premise, once they have been proven to be differential invariants in the left premise.

Example 3.25 (*Multidimensional nonlinear dynamics*). The proof in Example 3.21 depends on evolution domain constraint $a \geq 0$. If we replace variable a in Example 3.21 by a term, say y^5, then we first need to use DC with the choice $C \equiv y^5 \geq 0$ and prove that $y^5 \geq 0$ is indeed an invariant of the dynamics before we can use the proof from Example 3.21, which depends on the evolution domain constraint $y^5 \geq 0$. In Example 3.21, this is trivial since y has no differential equation, so that we assume $y' = 0$ by convention. Figure 13 shows a $\mathsf{d}\mathcal{L}$ proof for a more interesting case, where y changes during the continuous evolution. The proof uses DC once and DI twice.

Besides DI, the following simple proof rule for *differential weakening* also benefits from strengthening evolution domain constraints via DC:

$$\text{(DW)}\quad \dfrac{\chi \rightarrow F}{F \rightarrow [x' = \theta \,\&\, \chi]F}$$

This rule is obviously sound, because the system $x' = \theta \,\&\, \chi$, by definition, is never allowed to leave the evolution domain constraint χ anyhow, hence, if χ implies F, then F is an invariant, no matter what the dynamics $x' = \theta$ does. The simple rule DW alone cannot prove very interesting properties, because it only works when χ is very informative. It can, however, be useful in combination with DC with which we can soundly restrict the evolution domain constraint to subregions.

Using differential cuts repeatedly in a process called *differential saturation* has turned out to be extremely useful in practice and even simplifies the invariant search, because it leads to several simpler invariants to find and prove instead of a single complex property [Platzer and Clarke 2008; Platzer and Clarke 2009a; Platzer 2010b].

Differential cuts do not only help in practice, but are a fundamental proof principle in theory. We have refuted the *differential cut elimination hypothesis* [Platzer 2010a]. Differential cuts have a simple intuition. Similar to a cut in first-order logic, they can be used to first prove a lemma and then use it. By the seminal cut elimination theorem of Gentzen [Gentzen 1935a; Gentzen 1935b], standard logical cuts do not change the deductive power and can be eliminated, even if that may have significant cost [Boolos

1984]. Unlike standard cuts, however, differential cuts actually increase the deductive power [Platzer 2012d]. There are properties of differential equations that can only be proven using differential cuts, not without them. Hence, differential cuts are a fundamental proof principle for differential equations.

THEOREM 3.26 (DIFFERENTIAL CUT POWER [PLATZER 2012D]). *The deductive power with differential cuts (rule DC) exceeds the deductive power without differential cuts.*

We refer to previous work [Platzer 2012d] for details on the differential cut elimination hypothesis [Platzer 2010a], the proof of its refutation [Platzer 2012d], and a complete investigation of the relative deductive power of several classes of differential invariants.

3.9. Differential Auxiliaries

It is well-known that auxiliary variables may be necessary to conduct proofs about conventional discrete programs. We have studied auxiliary differential variables, and have shown that some differential equation systems can only be proven after introducing auxiliary differential variables into the dynamics [Platzer 2012d]. That is, the addition of auxiliary differential variables increases the deductive power. This is captured in the following proof rule *differential auxiliaries* (DA) for introducing auxiliary differential variables [Platzer 2012d]:

$$\text{(DA)}\quad \frac{\phi \leftrightarrow \exists y\, \psi \quad \psi\rightarrow[x'=\theta, y'=\vartheta \,\&\, \chi]\psi}{\phi\rightarrow[x'=\theta \,\&\, \chi]\phi}$$

Rule DA is applicable if y is a new variable (or vector of variables) and the new differential equation $y'=\vartheta$ has global solutions on χ (e.g., because term ϑ satisfies a Lipschitz condition [Walter 1998, Proposition 10.VII], which is definable in first-order real arithmetic and thus decidable). Without a condition like this, adding $y'=\vartheta$ could limit the duration of system evolutions incorrectly. In fact, it would be sufficient for the domains of definition of the solutions of $y'=\vartheta$ to be no shorter than those of x.

Intuitively, rule DA can help proving properties, because it may be easier to characterize how x changes in relation to auxiliary variables y that co-evolve with a suitable differential equation ($y'=\vartheta$). We have proved that the addition of auxiliary differential variables increases the deductive power, even in the presence of differential cuts [Platzer 2012d]. That is, there are system properties that can only be proven using auxiliary differential variables in the dynamics. Hence, auxiliary differential variables are also a fundamental proof principle for differential equations.

THEOREM 3.27 (AUXILIARY DIFFERENTIAL VARIABLE POWER [PLATZER 2012D]). *The deductive power with auxiliary differential variables (rule DA) exceeds the deductive power without auxiliary differential variables even in the presence of differential cuts.*

3.10. Implementation and Applications

Differential dynamic logic [Platzer 2007b; Platzer 2008a; Platzer 2012b] and its proof calculus [Platzer 2007b; Platzer 2008a], including differential invariants, differential variants, and differential cuts [Platzer 2010a] have been implemented in the automatic and interactive theorem prover KeYmaera [Platzer and Quesel 2008],[4] which is based on KeY [Beckert et al. 2007]. The name KeYmaera is a homophone to Chimæra, the

[4]Available at `http://symbolaris.com/info/KeYmaera.html`

hybrid animal from ancient Greek mythology. KeYmaera implements a sequent calculus version [Platzer 2008a] of the axiomatization in Figure 6, because the sequent calculus is more suitable for automation. Differential dynamic logic forms the basis for an automatic proof search procedure searching for invariants and differential invariants [Platzer and Clarke 2008; Platzer and Clarke 2009a; Platzer 2010b] that has been implemented in KeYmaera.

Differential dynamic logic and KeYmaera have been used successfully for verifying system-level properties of local lane controllers for highway car traffic [Loos et al. 2011], car controllers for intersections [Loos and Platzer 2011], intelligent speed adaptation for variable speed limit control and incident management by traffic centers on highways [Mitsch et al. 2012], CICAS-SLTA left-turn assist controllers for cars at intersections [Aréchiga et al. 2012], flyable roundabout collision avoidance maneuvers for aircraft [Platzer and Clarke 2009b], the cooperation protocols of the European Train Control System ETCS [Platzer and Quesel 2009], and analog circuits [Platzer 2010b]. KeYmaera has been used to prove safety requirements of a distributed elevator controller, medical robotic surgery systems, robotic factories, and to study biological models. Properties proved about these systems using $\mathsf{d}\mathcal{L}$ include safety, controllability, reactivity, liveness, and characterization properties. More details about $\mathsf{d}\mathcal{L}$ and some of its applications are described in a book [Platzer 2010b] about the theory, practice, and applications of $\mathsf{d}\mathcal{L}$ and its extensions DAL for differential-algebraic hybrid systems and dTL for temporal properties.

4. QUANTIFIED DIFFERENTIAL DYNAMIC LOGIC FOR DISTRIBUTED HYBRID SYSTEMS

In this section, we study *quantified differential dynamic logic* $\mathsf{Qd}\mathcal{L}$ [Platzer 2010c; Platzer 2012a], the *logic of distributed hybrid systems*, i.e., systems that combine the dynamics of distributed systems with the discrete and continuous dynamics of hybrid systems.

Not all cyber-physical systems and certainly not all dynamical systems can be modeled faithfully as hybrid systems. Cyber-physical systems typically combine *communication, computation, and control*. They may even form dynamic distributed networks, where neither structure nor dimension stay the same while the system follows hybrid dynamics.

Combining computation and control leads to *hybrid systems*, whose behavior involves both discrete and continuous dynamics originating, e.g., from discrete control decisions and differential equations of movement (Sect. 3). Combining communication and computation leads to *distributed systems* [Lynch 1996; Attie and Lynch 2001; Apt et al. 2010], whose dynamics are discrete transitions of system parts that communicate with each other. They may form *dynamic distributed systems*, where the structure of the system is not fixed but evolves over time and agents may appear or disappear during the system evolution.

Combinations of all three aspects (communication, computation, and control) are used in sophisticated applications, e.g., cooperative distributed car control [Hsu et al. 1991] and decentralized aircraft control [Pallottino et al. 2007]. Neither the structure nor dimension of the system stay the same, because new cars can appear on the street or leave it; see Figure 2 on p. 9. These systems are *(dynamic) distributed hybrid systems* [Deshpande et al. 1996; Rounds 2004; Kratz et al. 2006; Meseguer and Sharykin 2006; Gilbert et al. 2009; Platzer 2010c; Platzer 2012a]. More generally, distributed hybrid systems are multi-agent hybrid systems that interact through remote communication or physical interaction. They cannot be considered just as a distributed system (because, e.g., the continuous evolution of positions and velocities matters crucially for collision freedom in car control) nor just as a hybrid system (because the evolving system structure and appearance of new agents or structural changes in the system can

make an otherwise collision-free system unsafe). It is generally not possible to split the analysis of distributed hybrid systems soundly into an analysis of a distributed system (without continuous movement) and an analysis of a hybrid system (without structural changes or appearance), because all kinds of dynamics interact. Just like hybrid systems that are very difficult to analyze from a purely discrete or a purely continuous perspective [Henzinger 1996]. See previous work [Platzer 2012b] for a complete discussion of the relationship between discrete and continuous dynamics in hybrid systems.

As a formal logic for specifying and verifying correctness properties of distributed hybrid systems, we have introduced *quantified differential dynamic logic* (Qd$\mathcal{L}$) [Platzer 2010c; Platzer 2012a]. Qd$\mathcal{L}$ extends d$\mathcal{L}$ to distributed hybrid systems. Qd$\mathcal{L}$ combines dynamic logic for reasoning about all ($[\alpha]\phi$) or some ($\langle\alpha\rangle\phi$) system runs of a system α [Harel et al. 2000] with many-sorted first-order logic for reasoning about all ($\forall i:C\ \phi$) or some ($\exists i:C\ \phi$) objects of a sort C, e.g., the sort of all cars.

The most important defining characteristic of Qd$\mathcal{L}$ is that α can be a distributed hybrid system, because the Qd$\mathcal{L}$ system model of *quantified hybrid programs* (QHP) supports quantified operations that affect *all* objects of a sort C at once. If C is the sort of cars, the quantified assignment $\forall i:C\ a(i):=a(i)+1$ increases the respective accelerations $a(i)$ of *all cars* i at once by a single instantaneous discrete jump. It can be used to model simultaneous discrete changes in multiple agents at once. Discrete changes where only some of the cars change their acceleration, others do not, are easy to model with quantified assignments by masking. The quantified differential equation $\forall i:C\ v(i)'=a(i)$ represents a continuous evolution of the respective velocities $v(i)$ of *all cars* i at the same time according to their acceleration by their respective differential equations $v(i)'=a(i)$. Again, continuous evolutions where only some of the cars evolve, others remain stopped, are easy to model with quantified differential equations by masking. These quantified assignments and quantified differential equation systems of QHPs are crucial for representing distributed hybrid systems where an unbounded number of objects co-evolve simultaneously, because no finite set of classical assignments and classical differential equations could represent that. Note that, because of the close semantical relationship, we use the same quantifier notation $\forall i:C$ for quantified operations in programs and for quantifiers in logical formulas, instead of a separate notation $\Pi_{i:C}$ for parallel products in programs.

Interaction by communication can be modeled by (possibly quantified) discrete assignments to share data between agents i and j in QHPs. Physical interaction, instead, may be modeled either by (possibly quantified) discrete assignments when an agent i activates a response in agent j by an instantaneous discrete action (e.g., pushing a physical button) or by a (possibly quantified) differential equation involving multiple agents i and j when they come into physical contact and act jointly over a (nonzero) period of time (e.g., both agents jointly lifting and pulling on a rigid object). Observe that the cyber structure of the system reconfigures dynamically when discrete communication topologies change, whereas the physical structure reconfigures dynamically when agents engage in physical contact. QHPs for the latter case may involve structural changes in the quantified differential equation.

We model the appearance of new participants in the distributed hybrid system, e.g., new cars entering the road, by a program $n:=\mathsf{new}\,C$. It creates a new object of type C, thereby extending the range of all subsequent quantified assignments or quantified differential equations ranging over created objects of type C. With quantifiers and function terms, new can be defined and handled in an entirely modular way; see previous work [Platzer 2010c; Platzer 2012a] for details. Overall, d$\mathcal{L}$, which we considered in Sect. 3, is for finite-dimensional hybrid systems, but Qd$\mathcal{L}$ can handle evolving or infinite-dimensional distributed hybrid systems. Qd$\mathcal{L}$ and QHPs provide first-order state variables and quantifiers (also in the dynamics) over the arguments of first-order

function symbols, which is the fundamental enabling technique for distributed hybrid systems. This difference of QdL compared to dL is as fundamental as that of first-order logic compared to propositional logic, except that it also affects the dynamics not just the formulas. The logic dL and HPs only support primitive variables x, v, a of type $\mathbb{R}$, whereas QdL supports first-order function symbols $x(i), v(i), a(i), d(i,j)$ and quantifiers, e.g., over i and j in the dynamics, so that an unbounded (instead of a statically fixed finite) number of agents can be described by the dynamics.

The model of QHPs is of independent interest as a formal model for distributed hybrid systems. Inside a QHP, logical formulas can occur in state tests for conditional execution. We thus explain logical formulas, terms, and sorts first. Conversely, however, a QHP α occurs inside the modalities ($[\alpha]$ and $\langle\alpha\rangle$) of QdL formulas, which state properties of the behavior of α. Hence, QHPs may occur inside QdL formulas yet formulas may occur inside QHPs. The subsequent definitions of QdL and QHP are thus to be understood by simultaneous induction.

We first explain the logical formulas that QdL provides for specification and verification (Sect. 4.1) and then explain the system model of quantified hybrid programs that QdL provides for modeling distributed hybrid systems (Sect. 4.2). We define the semantics (Sect. 4.3) and then explain reasoning principles, axioms, and proof rules for verifying QdL formulas (Sect. 4.4). We then show soundness and relative completeness theorems (Sect. 4.5) and investigate stronger proof rules for quantified differential equations (Sect. 4.6). Finally, we briefly discuss an implementation in the theorem prover KeYmaeraD and applications (Sect. 4.7).

4.1. QdL Formulas

We have introduced quantified differential dynamic logic (QdL) [Platzer 2010c; Platzer 2012a], which is the first formal logic for specifying and verifying correctness properties of distributed hybrid systems. QdL is a combination of many-sorted first-order logic with dynamic logic, generalized to a system model (QHPs) for distributed hybrid systems.

4.1.1. Sorts. QdL supports a (finite) number of object sorts, e.g., the sort of all cars and that of all aircraft. For continuous quantities of distributed hybrid systems like positions or velocities, we add the sort $\mathbb{R}$ of real numbers. It would be easy to add subtyping of sorts; see previous work [Beckert and Platzer 2006] for details. We refrain from doing so, because that just obscures the logical essence of our approach.

The primary purpose of the sorts is to distinguish different kinds of objects in multi-agent hybrid systems in which different kinds of agents occur, e.g., cars of sort C, traffic lights of sort T, lanes of sort L, and aircraft of sort A.

4.1.2. Terms. QdL terms are built from a set of (sorted) function and variable symbols as in many-sorted first-order logic. Each function symbol f has a fixed type $C_1 \times \cdots \times C_n \to D$ for some $n \in \mathbb{N}$ and some sorts $D, C_1, \ldots, C_n$ such that f only accepts argument terms $\theta_1, \ldots, \theta_n$ of the respective sorts $C_1, \ldots, C_n$ and then $f(\theta_1, \ldots, \theta_n)$ is a term of sort D. We use these function symbols to represent the state of the system or other parameters. In a car control scenario like that in Figure 2, for example, we could use function symbol x to represent the positions of cars, i.e., the term $x(i)$ could represent the position of car i and $x(j)$ the position of car j. Similarly, the term $v(i)$ could represent the velocity of car i and $a(i)$ its acceleration. These terms have sort $\mathbb{R}$, whereas a term $l(i)$ that represents the car in front of car i has sort C.

Unlike in first-order logic, the interpretation of function symbols can change when transitioning from one state to the other while following the dynamics of a distributed hybrid system. The value of position $x(i)$ will change over time as car i drives down the street. The value of $x(i)$ also changes if the argument term i changes its value and now

refers to a different car than before. Objects may appear or disappear as the distributed hybrid system evolves. We use function symbol $\mathbb{E}(\cdot)$ to distinguish between objects i that actually exist ($\mathbb{E}(i)=1$) and those that have not been created yet or exist no longer ($\mathbb{E}(i)=0$), depending on the value of $\mathbb{E}(i)$, which may change its interpretation from state to state. We use $0,1,+,-,\cdot$ with the usual notation and fixed semantics for real arithmetic. For $n \geq 0$ we abbreviate $f(s_1,\ldots,s_n)$ by $f(\vec{s})$ using vectorial notation and we use $\vec{s}=\vec{t}$ for component-wise equality.

4.1.3. Formulas. The formulas of QdL are defined as in first-order dynamic logic plus many-sorted first-order logic

Definition 4.1 (QdL *formula*). The formulas of QdL are defined by the following grammar (ϕ,ψ are formulas, θ_1,θ_2 are terms of the same sort, i is a variable of sort C, and α is a QHP as defined in Sect. 4.2):

$$\phi,\psi \;::=\; \theta_1=\theta_2 \mid \theta_1 \geq \theta_2 \mid \neg\phi \mid \phi\wedge\psi \mid \forall i:C\ \phi \mid [\alpha]\phi$$

We use standard abbreviations to define $\leq,>,<,\vee,\rightarrow,\exists$. The operator $\langle\alpha\rangle$ dual to $[\alpha]$ is again defined by $\langle\alpha\rangle\phi \equiv \neg[\alpha]\neg\phi$. Similarly, $\exists i:C\ \phi \equiv \neg\forall i:C\ \neg\phi$. Sorts $C\neq\mathbb{R}$ have no ordering and only $\theta_1=\theta_2$ is allowed, not $\theta_1\geq\theta_2$. For sort $\mathbb{R}$, we abbreviate $\forall x:\mathbb{R}\ \phi$ by $\forall x\,\phi$ and $\exists x:\mathbb{R}\ \phi$ by $\exists x\,\phi$. All QdL formulas and terms have to be well-typed. For instance, $x(i)=l(i)$ is no formula if x has type $C\rightarrow\mathbb{R}$ and l has type $C\rightarrow C$ for a sort $C\neq\mathbb{R}$ or if i has a sort $D\neq C$. QdL formula $[\alpha]\phi$ expresses that *all states* reachable by QHP α satisfy formula ϕ. Likewise, $\langle\alpha\rangle\phi$ expresses that *there is at least one state* reachable by α for which ϕ holds.

For short notation, we allow *conditional terms* of the form $\mathsf{if}\,\phi\,\mathsf{then}\,\theta_1\,\mathsf{else}\,\theta_2\,\mathsf{fi}$ (where θ_1 and θ_2 have the same sort). This term evaluates to θ_1 if the formula ϕ is true and to θ_2 otherwise. We generally consider formulas with conditional terms as abbreviations, e.g., $\psi(\mathsf{if}\,\phi\,\mathsf{then}\,\theta_1\,\mathsf{else}\,\theta_2\,\mathsf{fi})$ abbreviates $(\phi\rightarrow\psi(\theta_1))\wedge(\neg\phi\rightarrow\psi(\theta_2))$. Conditional terms can be understood as an additional operator for terms and formulas as well.

Example 4.2 (*Distributed car control*). If i is a term of type C (for cars), let $x(i)$ denote the position of car i, $v(i)$ its current velocity, and $a(i)$ its current acceleration; see Figure 2 on p. 9. A car control system is collision-free at a state if all cars are at different positions ($\forall i{\neq}j:C\ x(i){\neq}x(j)$). Without a quantifier we could not describe that all cars on a highway are in a collision-free state, because there is a large number of cars on the highway and we may not know how many. The car control system is globally collision-free if it will always stay collision-free. The following QdL formula expresses that a distributed car control system *DCCS* (we develop QHP models for *DCCS* later) controls cars such that they are always collision-free:

$$(\forall i,j:C\ \mathcal{M}(i,j)) \;\rightarrow\; [\mathit{DCCS}]\ \forall i{\neq}j:C\ x(i){\neq}x(j) \tag{12}$$

It says that cars following the distributed hybrid systems dynamics of *DCCS* are always collision-free (postcondition), provided that *DCCS* starts in an initial state satisfying a formula $\mathcal{M}(i,j)$ for all cars i,j (precondition). In particular, the modality $[\mathit{DCCS}]$ expresses that all states reachable by following the distributed hybrid system *DCCS* satisfy the postcondition. The simple-most choice for the formula $\mathcal{M}(i,j)$ in the precondition is a formula that characterizes a simple compatibility condition: for different cars $i\neq j$, the car that is further down the road (i.e., with greater position) neither moves slower nor accelerates slower than the other car, i.e. $\mathcal{M}(i,j)$ is

$$\begin{aligned}\mathcal{M}(i,j) \equiv i\neq j \rightarrow \big((x(i)<x(j)\wedge v(i)\leq v(j)\wedge a(i)\leq a(j))\\ \vee\ (x(i)>x(j)\wedge v(i)\geq v(j)\wedge a(i)\geq a(j))\big)\end{aligned} \tag{13}$$

Based on a generalization of the dL formulas in Example 3.7, we can also choose the following more permissive formula for $\mathcal{M}(i,j)$, which allows cars to drive more aggressively with different accelerations, if only the respective safety distances are compatible with the different velocities:

$$\begin{aligned}\mathcal{M}(i,j) \equiv i \neq j \rightarrow v(i) \geq 0 \wedge v(j) \geq 0 \wedge \\ \big((x(i) < x(j) \wedge v(i)^2 < v(j)^2 + 2b(x(j) - x(i))) \\ \vee (x(i) > x(j) \wedge v(j)^2 < v(i)^2 + 2b(x(i) - x(j)))\big)\end{aligned} \tag{14}$$

This formula characterizes that, for different cars $i \neq j$, the car that is further down the road is at a sufficient distance to allow its follower car to adapt its velocity by braking to be no faster than the car that is already further down the road. With this choice of $\mathcal{M}(i,j)$, observe the relationship of QdL formula (12) to dL formula (5) in Example 3.7. For a more detailed and exhaustive study of possible initial conditions and invariants for distributed car control, we refer to previous work [Platzer 2010c; Platzer 2012a; Loos et al. 2011].

The reader should note that more sophisticated combinations of nested quantifiers and modalities like the ones we considered for dL (e.g., Example 3.7) are possible with QdL and its axiomatization in Sect. 4.4 as well.

4.2. Quantified Hybrid Programs

As a formal model for distributed hybrid systems, we have introduced *quantified hybrid programs* (QHPs) [Platzer 2010c; Platzer 2012a]. These are regular programs from dynamic logic [Harel et al. 2000] to which we add quantified assignments and quantified differential equation systems for *distributed* hybrid dynamics. From these quantified assignments and quantified differential equations, QHPs are built like a Kleene algebra with tests [Kozen 1997].

Definition 4.3 (*Quantified hybrid program*). QHPs are defined by the following grammar (α, β are QHPs, i a variable of sort C, f is a function symbol, $\vec{s}$ is a vector of terms with sorts compatible to the arguments of f, θ is a term with sort compatible to the result of f, and χ is a formula of many-sorted first-order logic):

$$\alpha, \beta \ ::= \ \forall i:C\ f(\vec{s}) := \theta \mid ?\chi \mid \forall i:C\ f(\vec{s})' = \theta \,\&\, \chi \mid \alpha \cup \beta \mid \alpha;\beta \mid \alpha^*$$

In order to simplify technical difficulties, we impose regularity assumptions on $f(\vec{s})$ in quantified assignments and quantified differential equations. We assume $\vec{s}$ to be either a vector of length 0 or that the mapping from the quantified variable i to $\vec{s}$ is *injective*. That is, each value of $\vec{s}$ can be exhibited by at most one choice of i. A system is injective, e.g., when at least one component of $\vec{s}$ is the quantified variable i. These assumptions can be relaxed, but are sufficient for our purposes; see Sect. 4.3 for a discussion of injectivity. For quantified differential equations, we further assume that f is an $\mathbb{R}$-valued function symbol so that derivatives can be defined.

4.2.1. Quantified State Change. The effect of *quantified assignment* $\forall i:C\ f(\vec{s}) := \theta$ is an instantaneous discrete jump assigning θ to $f(\vec{s})$ simultaneously for all objects i of sort C. Hence all $f(\vec{s})$ that are affected by $\forall i:C\ f(\vec{s}) := \theta$ will change their value to the respective θ simultaneously for all choices of i in a single discrete instant of time. Usually, i occurs in term θ, but does not have to. The effect of *quantified differential equation* $\forall i:C\ f(\vec{s})' = \theta \,\&\, \chi$ is a continuous evolution where, for all objects i of sort C, all differential equations $f(\vec{s})' = \theta$ hold at the same time and formula χ holds throughout the evolution (the state always remains in the region described by χ, i.e., the evolution stops at any arbitrary time before it leaves χ). Again, i usually occurs in term θ. For the

trivial evolution domain restriction $\chi \equiv true$, which is always satisfied, we also write $\forall i:C\ f(\vec{s})' = \theta$ instead of $\forall i:C\ f(\vec{s})' = \theta \,\&\, true$.

The dynamics of QHPs changes the interpretation of terms over time: $f(\vec{s})'$ is intended to denote the derivative of the interpretation of the term $f(\vec{s})$ over time during continuous evolution, not the derivative of $f(\vec{s})$ by its argument $\vec{s}$. For $f(\vec{s})'$ to be defined, we assume f is an $\mathbb{R}$-valued function symbol. Although our approach can be extended, we assume that f does not occur in $\vec{s}$. The most common choice of $\vec{s}$ in quantified assignments and quantified differential equations is just i. Other choices are possible for $\vec{s}$, e.g., $\vec{s} = (i, f(i))$ in $\forall i:C\ d(i,f(i)) := \frac{1}{2}a(i) + \frac{1}{2}a(f(i))$. The latter QHP could be used to model that, for each car i, the average acceleration of a car i and its follower $f(i)$ is assigned to a data field $d(i,f(i))$ that car i and its follower communicate to determine their safe distance.

Time itself is not special but implicit. If a clock variable t is needed in a QHP, it can be axiomatized by $t' = 1$, which is equivalent to $\forall i:C\ t' = 1$ where i does not occur in t. For such *vacuous quantification* (i does not occur anywhere), we may omit $\forall i:C$ from assignments and differential equations, which are then classical assignments and ordinary differential equations as in HPs (Sect. 3). We may omit vectors $\vec{s}$ of length 0.

4.2.2. Regular Programs. The *test* $?\chi$ of a first-order formula χ of real arithmetic is as in HPs except that χ is a formula of many-sorted first-order logic. Compound QHPs are generated from atomic QHPs by nondeterministic choice ($\cup$), sequential composition ($;$), and Kleene's nondeterministic repetition (*), just like in Sect. 3.1. The (decisive) difference of QHPs of Qd$\mathcal{L}$ compared to HPs of d$\mathcal{L}$ is that QHPs can contain *quantified* assignments and *quantified* differential equations with first-order functions.

QHPs (with their semantics and our proof rules) can be extended to systems of quantified differential equations, systems of simultaneous assignments to multiple functions f, g, and statements with multiple quantifiers ($\forall i:C\ \forall j:D\ \ldots$) similar to vectorial generalizations in discrete programs [Beckert and Platzer 2006; Rümmer 2006].

Example 4.4 (*Distributed car control*). Continuous movement of position $x(i)$ of car i with acceleration $a(i)$ is expressed by differential equation $x(i)'' = a(i)$, which corresponds to the first-order differential equation system $x(i)' = v(i), v(i)' = a(i)$ where $v(i)$ is the velocity of car i. Simultaneous movement of all cars with their respective accelerations $a(i)$ is expressed by the quantified differential equation $\forall i:C\ (x(i)'' = a(i))$ where quantifier $\forall i:C$ ranges over all cars, such that all cars co-evolve along their respective differential equations at the same time.

In addition to continuous dynamics, cars have discrete control. In the following QHP, discrete and continuous dynamics interact (repeatedly because of the * repetition operator):

$$\big(\forall i:C\ (a(i) := \mathsf{if}\ \forall j:C\ \mathit{far}(i,j)\ \mathsf{then}\ A\ \mathsf{else}\ -b\ \mathsf{fi});\ \forall i:C\ (x(i)'' = a(i))\big)^* \tag{15}$$

First, all cars i control their acceleration $a(i)$. Each car i chooses maximum acceleration $A \geq 0$ for $a(i)$ if its distance to all other cars j is far enough (some condition $\mathit{far}(i,j)$ that depends on the velocities and either on the acceleration of j or on reaction times ε as in Example 3.7). Otherwise, i chooses full braking $-b < 0$. After all accelerations have been set, all cars move continuously along $\forall i:C\ (x(i)'' = a(i))$. Accelerations may change repeatedly, because the repetition operator * can repeat the QHP after the continuous evolution stops, which it can do at any time. When *DCCS* denotes QHP (15), the Qd$\mathcal{L}$ formula (12) from Example 4.2 is valid, when choosing (13) for $\mathcal{M}(i,j)$. For more elaborate car models, verification results, and formal proofs, including verification results about distributed car control with dynamic appearance and disappearance of cars on highways with arbitrarily many cars on arbitrarily many lanes including onramps and exits, we refer to previous work [Platzer 2010c; Platzer

2012a; Loos et al. 2011]. Observe that the QHP (15) requires car i to check $far(i,j)$ for all other cars j, which is easy to model, but hard to implement. We refer to previous work [Loos et al. 2011] for QdL proofs for distributed car control models that are globally safe even though each car reaches its control decisions solely based on local sensor/communication input and local control decisions.

Note that the presence of the function argument i in $x(i), v(i), a(i)$ is a decisive difference when comparing the QHP in Example 4.4 to the HP in Example 3.3 and when comparing the QdL formula in Example 4.2 to the dL formula in Example 3.7. In hybrid systems, we are limited to using variables x, v, a of a single car. If we want to add a second car to a hybrid system model, we need to add new state variables y, w, c, new dynamics $y' = w, w' = c$, and new control for the second car. We can keep on adding any fixed finite number of state variables that way, but we need to know exactly how many cars there are on the street. This does not work when we want to model and verify situations with arbitrarily many cars or in distributed car control scenarios like Figure 2, where new cars appear or disappear during the evolution of the system. A quantified differential equation like $\forall i : C\ (x(i)' = v(i), v(i)' = a(i))$, for example, cannot be expressed in hybrid systems, because we do not know how many cars i ranges over. If i did range over exactly 3 cars, called 1, 2, and 3, we could replace it by

$$x(1)' = v(1), v(1)' = a(1), x(2)' = v(2), v(2)' = a(2), x(3)' = v(3), v(3)' = a(3)$$

and change notation to obtain primitive state variables $x_1, v_1, a_1, x_2, v_2, a_2, x_3, v_3, a_3$ in an ordinary differential equation system

$$x_1' = v_1, v_1' = a_1, x_2' = v_2, v_2' = a_2, x_3' = v_3, v_3' = a_3$$

But this replacement does not work unless we know exactly how many cars are in the system. Even for systems with a fixed known but large number of participants, such flat representations as (non-distributed) hybrid systems are inefficient, because the system dimension is exponential in the number of participants and all reasoning needs to be repeated for each participant, or even for each pair of participants (collision freedom requires each pair of cars to remain safely separated).

In QdL formulas and in QHP models, we can leverage the distributed structure in systems, make the models more expressive, and make the reasoning more efficient by exploiting their first-order structure. Only in QdL can properties of distributed hybrid systems with an unknown or evolving number of participants be proved. See previous work [Loos et al. 2011] for a detailed practical illustration of those phenomena in verification of local and of distributed car control.

4.3. Semantics

The QdL semantics is a *constant domain Kripke semantics [Fitting and Mendelsohn 1999] with first-order structures as states* that associate total functions of appropriate type with function symbols. In constant domain, all states share the same domain for quantifiers. We choose to represent object creation not by changing the domain of states, but by changing the interpretation of the createdness flag $\exists(i)$ of the object denoted by i. With $\exists(i)$, object creation is definable in a modular way by masking the effect of QHPs to objects i with $\exists(i) = 1$ [Platzer 2010c; Platzer 2012a].

4.3.1. States. A *state* ν associates an (infinite) set $\nu(C)$ of objects with each sort C, and it associates a function $\nu(f)$ of appropriate type with each function symbol f, including $\exists(\cdot)$. For simplicity, ν also associates a value $\nu(i)$ of appropriate type with each variable i. The domain of $\mathbb{R}$ and the interpretation of $0, 1, +, -, \cdot$ is that of real arithmetic. We assume *constant domain* for each sort C: all states ν, ω share the same domains $\nu(C) = \omega(C)$ for C. Sorts $C \neq D$ are disjoint: $\nu(C) \cap \nu(D) = \emptyset$. The set of all states

is again denoted by $\mathcal{S}$, but different from the set of states of $\mathsf{d}\mathcal{L}$. The state ν_i^e agrees with ν except for the interpretation of variable i, which is changed to e. We assume $\mathsf{E}(\cdot)$ to have (unbounded but) finite support, i.e., each state only has a finite number of positions i at which $\mathsf{E}(i) = 1$. This makes sense in practice, because there is a varying and possibly large but still finite numbers of participants (e.g., cars).

4.3.2. Formulas. We use $[\![\theta]\!]_\nu$ to denote the value of term θ at state ν, which is defined as in first-order logic. Especially, $[\![\theta]\!]_{\nu_i^e}$ denotes the value of θ in state ν_i^e, i.e., in state ν with i interpreted as e. Further, $\rho(\alpha)$ denotes the state transition relation of QHP α, which we define below.

Definition 4.5 (QdL *semantics*). The *interpretation* $\nu \models \phi$ of QdL formula ϕ with respect to state ν is defined inductively as:

— $\nu \models (\theta_1 = \theta_2)$ iff $[\![\theta_1]\!]_\nu = [\![\theta_2]\!]_\nu$.
— $\nu \models (\theta_1 \geq \theta_2)$ iff $[\![\theta_1]\!]_\nu \geq [\![\theta_2]\!]_\nu$.
— $\nu \models \neg\phi$ iff it is not the case that $\nu \models \phi$.
— $\nu \models \phi \wedge \psi$ iff $\nu \models \phi$ and $\nu \models \psi$.
— $\nu \models \forall i : C\ \phi$ iff $\nu_i^e \models \phi$ for all objects $e \in \nu(C)$.
— $\nu \models \exists i : C\ \phi$ iff $\nu_i^e \models \phi$ for some object $e \in \nu(C)$.
— $\nu \models [\alpha]\phi$ iff $\omega \models \phi$ for all states ω with $(\nu, \omega) \in \rho(\alpha)$.
— $\nu \models \langle\alpha\rangle\phi$ iff $\omega \models \phi$ for some ω with $(\nu, \omega) \in \rho(\alpha)$.

If $\nu \models \phi$, then we say that ϕ is true at ν. QdL formula ϕ is *valid*, written $\vDash \phi$, iff $\nu \models \phi$ for all ν.

4.3.3. Programs. The transition semantics of QHPs is defined similar to the transition semantics of HPs, except that the quantified assignments and quantified differential equations need to be defined.

Definition 4.6 (*Transition semantics of QHPs*). The *transition relation,* $\rho(\alpha) \subseteq \mathcal{S} \times \mathcal{S}$, of QHP α specifies which state ω is reachable from ν by running QHP α. It is defined inductively:

(1) $(\nu, \omega) \in \rho(\forall i : C\ f(\vec{s}) := \theta)$ iff state ω is identical to ν except that at each position $\vec{o}$ of f: if $[\![\vec{s}]\!]_{\nu_i^e} = \vec{o}$ for some object $e \in \nu(C)$, then $\omega(f)\big([\![\vec{s}]\!]_{\nu_i^e}\big) = [\![\theta]\!]_{\nu_i^e}$. If there are multiple objects e giving the same position $[\![\vec{s}]\!]_{\nu_i^e} = \vec{o}$, then all of the resulting states ω are reachable.
(2) $(\nu, \omega) \in \rho(\forall i : C\ f(\vec{s})' = \theta\ \&\ \chi)$ iff there is a function $\varphi{:}[0, r] \to \mathcal{S}$ for some $r \geq 0$ with $\varphi(0) = \nu$ and $\varphi(r) = \omega$ satisfying the following conditions. At each time $t \in [0, r]$, state $\varphi(t)$ is identical to ν, except that at each position $\vec{o}$ of f: if $[\![\vec{s}]\!]_{\nu_i^e} = \vec{o}$ for some object $e \in \nu(C)$, then, at each time $\zeta \in [0, r]$:
— All differential equations hold and corresponding derivatives exist (trivial for $r = 0$):

$$\frac{\mathsf{d}\,([\![f(\vec{s})]\!]_{\varphi(t)_i^e})}{\mathsf{d}t}(\zeta) = [\![\theta]\!]_{\varphi(\zeta)_i^e}$$

— The evolution domain is respected: $\varphi(\zeta)_i^e \models \chi$.
If there are multiple objects e giving the same position $[\![\vec{s}]\!]_{\nu_i^e} = \vec{o}$, then all of the resulting states ω are reachable.
(3) $\rho(?\chi) = \{(\nu, \nu)\ :\ \nu \models \chi\}$
(4) $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
(5) $\rho(\alpha; \beta) = \rho(\beta) \circ \rho(\alpha)$

(6) $\rho(\alpha^*) = \bigcup_{n\in\mathbb{N}} \rho(\alpha^n)$ with $\alpha^{n+1} \equiv \alpha^n;\alpha$ and $\alpha^0 \equiv ?true$.

The semantics is *explicit change*: nothing changes unless an assignment or differential equation specifies how. In cases 1–2, only f changes and only at positions of the form $[\![\vec{s}]\!]_{\nu_i^e}$ for some interpretation $e \in \nu(C)$ of i. If there are multiple such e that affect the same position $\vec{o}$, any of those changes can take effect by a nondeterministic choice. QHP $\forall i:C\ x := a(i)$ may change x to *any* $a(i)$. Hence, $[\forall i:C\ x := a(i)]\phi(x) \equiv \forall i:C\ \phi(a(i))$, because that modality considers *all* possibilities of changing x to *any* $a(i)$. In contrast, $\langle\forall i:C\ x := a(i)\rangle\phi(x) \equiv \exists i:C\ \phi(a(i))$, because that modality considers *some* possibility of changing x to *any* $a(i)$. Similarly, x can evolve along $\forall i:C\ x' = a(i)$ with any of the slopes $a(i)$. But evolutions cannot start with slope $a(c)$ and then switch to a different slope $a(d)$ later. Any choice for the quantified variable i is possible but i remains unchanged during each evolution.

We call a quantified assignment $\forall i:C\ f(\vec{s}) := \theta$ or a quantified differential equation $\forall i:C\ f(\vec{s})' = \theta \,\&\, \chi$ *injective* iff there is at most one e satisfying cases 1–2. For injective quantified assignments and injective quantified differential equations, conditions 1–2 can be simplified as follows:

(1) $(\nu,\omega) \in \rho(\forall i:C\ f(\vec{s}) := \theta)$ iff state ω is identical to ν except that for each $e \in \nu(C)$: $\omega(f)\big([\![\vec{s}]\!]_{\nu_i^e}\big) = [\![\theta]\!]_{\nu_i^e}$.

(2) $(\nu,\omega) \in \rho(\forall i:C\ f(\vec{s})' = \theta \,\&\, \chi)$ iff there is a function $\varphi{:}[0,r] \to \mathcal{S}$ for some $r \geq 0$ with $\varphi(0) = \nu$ and $\varphi(r) = \omega$ such that for each $e \in \nu(C)$ and each time $\zeta \in [0,r]$:
   — All differential equations hold and corresponding derivatives exist (trivial for $r = 0$):
   $$\frac{\mathsf{d}\,([\![f(\vec{s})]\!]_{\varphi(t)_i^e})}{\mathsf{d}t}(\zeta) = [\![\theta]\!]_{\varphi(\zeta)_i^e}$$
   — The evolution domain is respected: $\varphi(\zeta)_i^e \models \chi$.

We call quantified assignments and quantified differential equations *schematic* iff $\vec{s}$ is i (thus injective) and the only arguments to function symbols in θ are i. Schematic quantified differential equations like $\forall i:C\ f(i)' = a(i) \,\&\, \chi$ are very common, because distributed hybrid systems often have a family of similar differential equations replicated for multiple participants i. Their synchronization often comes from discrete communication on top of their continuous dynamics. Physically coupled differential equations are possible as well. They correspond to continuous physical interactions, e.g., if a car bumps into another car from the side, it radically changes the structure of the differential equations that determine its movement. Either case can be represented in QHPs, even if the schematic case is more common.

4.4. Axiomatization

Our axiomatization of Qd$\mathcal{L}$ is shown in Figure 14. To again highlight the logical essentials, we use an axiomatization that is significantly simplified compared to our earlier work [Platzer 2010c; Platzer 2012a]. The axiomatization we use here is in the spirit of our simpler d$\mathcal{L}$ axiomatization that we show in Sect. 3.4. We use the first-order Hilbert calculus (modus ponens MP and $\forall$-generalization rule $\forall$) as a basis and allow all instances of valid formulas of many-sorted first-order logic and first-order real arithmetic as axioms. The first-order theory of real-closed fields is decidable [Tarski 1951]. More constructive deduction modulo rules, which can be used to combine first-order real arithmetic of many-sorted first-order logic with the proof calculus presented here and are suitable for automation, have been reported in previous work [Platzer 2010c;

$$[:] \quad [\forall i:C\ f(\vec{s}):=\theta]\,\Upsilon(\vec{u}) \leftrightarrow \Upsilon([\forall i:C\ f(\vec{s}):=\theta]\vec{u}) \qquad (f \neq \Upsilon)$$

$$[:=] \quad \phi([\forall i:C\ f(\vec{s}):=\theta]f(\vec{u})) \leftrightarrow \mathsf{if}\ \exists i:C\ \vec{s}=[\mathcal{A}]\vec{u}\ \mathsf{then}\ \forall i:C\ (\vec{s}=[\mathcal{A}]\vec{u}\to\phi(\theta))\ \mathsf{else}\ \phi(f([\mathcal{A}]\vec{u}))\ \mathsf{fi}$$

$$[:=]_s \quad \phi([\forall i:C\ f(i):=\theta]f(u)) \leftrightarrow \forall i:C\ (i=[\forall i:C\ f(i):=\theta]u\to\phi(\theta))$$

$$[:*] \quad [\forall j:C\ n:=\theta]\phi(n) \leftrightarrow \forall j:C\ \phi(\theta)$$

$$[?] \quad [?\chi]\phi \leftrightarrow (\chi\to\phi)$$

$$['] \quad [\forall i:C\ f(\vec{s})'=\theta]\phi \leftrightarrow \forall t{\geq}0\,[\forall i:C\ f(\vec{s}):=y_{\vec{s}}(t)]\phi \qquad (y'_{\vec{s}}(t)=\theta\,\forall i)$$

$$[\&] \quad [\forall i:C\ f(\vec{s})'=\theta\,\&\,\chi]\phi \leftrightarrow \forall t_0{=}x_0\,[\forall i:C\ f(\vec{s})'=\theta]\big([\forall i:C\ f(\vec{s})'=-\theta](x_0\geq t_0\to\chi)\to\phi\big)$$

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

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

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

$$\in \quad \exists n:C\ \in(n)=0 \qquad (C\neq\mathbb{R})$$

$$\mathbf{K} \quad [\alpha](\phi\to\psi)\to([\alpha]\phi\to[\alpha]\psi)$$

$$\mathbf{I} \quad [\alpha^*](\phi\to[\alpha]\phi)\to(\phi\to[\alpha^*]\phi)$$

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

$$\mathbf{B} \quad \forall x:C\ [\alpha]\phi\to[\alpha]\forall x:C\ \phi \qquad (x\notin\alpha)$$

$$\mathbf{V} \quad \phi\to[\alpha]\phi \qquad (FV(\phi)\cap BV(\alpha)=\emptyset)$$

$$\mathbf{G} \quad \frac{\phi}{[\alpha]\phi}$$

$$\mathbf{MP} \quad \frac{\phi\to\psi \quad \phi}{\psi}$$

$$\forall \quad \frac{\phi}{\forall x:C\ \phi}$$

Fig. 14. Quantified differential dynamic logic axiomatization

Platzer 2012a]. Note that the combination of first-order real arithmetic augmented with many-sorted function symbols is more challenging than the decidable first-order arithmetic of real-closed fields used as a basis for $\mathsf{d}\mathcal{L}$. It can still be handled with a combination of free variables, instantiation, requantification, and quantifier elimination [Platzer 2010c; Platzer 2012a], which we use to lift quantifier elimination to the context of $\mathsf{d}\mathcal{L}$ (Lemma 3.14) using real-valued free variables, Skolemization, and Deskolemization for automation purposes [Platzer 2008a].

We write $\vdash \phi$ iff $\mathsf{QD}\mathcal{L}$ formula ϕ can be *proved* with $\mathsf{QD}\mathcal{L}$ rules from $\mathsf{QD}\mathcal{L}$ axioms (including first-order rules and axioms); see Figure 14.

The $\mathsf{QD}\mathcal{L}$ axioms $[?]$, $[\cup]$, $[;]$, $[^*]$, K, I, C, B, V, and rule G are as for $\mathsf{d}\mathcal{L}$ in Sect. 3.4, because $\mathsf{QD}\mathcal{L}$ is a modular extension of $\mathsf{d}\mathcal{L}$ and the operators $?, \cup, ;, ^*$ have the same compositional semantics as in $\mathsf{d}\mathcal{L}$. We use the same first-order rules MP and $\forall$, except that $\forall$ applies to variables x of any sort C. The axioms $[:=]$, $[:=]_s$, $[:*]$, $[:]$ for quantified

assignments, [′], [&] for quantified differential equations, and ∈ for object creation are specific to QdL. Observe that, despite the radical semantical generalization, an important principle of generalizing dL to QdL is modularity. One local, but decisive change is from dL's primitive variables to QdL's first-order variables. The other changes are modular in the syntax, semantics, and axiomatization and consist in adding new cases for quantified assignments and for quantified differential equations (and object creation).

Axiom [:] characterizes the fact that quantified assignments to f have no effect on all other operators $\Upsilon \neq f$ (including other function symbols, $\wedge$, if then else fi), so that Υ will not be affected by the quantified assignment and can be skipped over. The argument $\vec{u}$ may still be affected by the quantified assignment, hence [:] prefixes $\vec{u}$ (componentwise) by $\forall i:C\ f(\vec{s}) := \theta$. Thus, the [:] axiom maps a quantified assignment over all arguments homomorphically. For example, if Υ is an operator taking two arguments and is not the function symbol f, then axiom [:] derives the proof step

$$[:]\frac{\Upsilon([\forall i:C\ f(\vec{s}) := \theta]u_1, [\forall i:C\ f(\vec{s}) := \theta]u_2)}{[\forall i:C\ f(\vec{s}) := \theta]\,\Upsilon(u_1,u_2)}$$

Axiom [:=] characterizes how a quantified assignment to f affects the value of a term $f(\vec{u})$. The effect depends on whether the quantified assignment $\forall i:C\ f(\vec{s}) := \theta$ *matches* $f(\vec{u})$, i.e., there is a choice for i such that $f(\vec{u})$ is affected by the assignment, because $\vec{u}$ is of the form $\vec{s}$ for some i. Whether it matches or not cannot always be decided statically, because it may depend on the particular interpretations. Hence, axiom [:=] makes a case distinction on matching by yielding an if-then-else formula. The formula if ϕ then ϕ_1 else ϕ_2 fi is short notation for $(\phi \to \phi_1) \wedge (\neg\phi \to \phi_2)$. If the quantified assignment does not match (else part), the occurrence of f in $\phi(f(\vec{u}))$ will be left unchanged, because f is not changed at position $\vec{u}$. If it matches (then part), the term θ assigned to $f(\vec{s})$ is used instead of $f(\vec{u})$, for all possible $i:C$ that match $f(\vec{u})$. In all cases, the original quantified assignment $\forall i:C\ f(\vec{s}) := \theta$, which we abbreviate by $\mathcal{A}$ in [:=], will be applied to $\vec{u}$ in the premise, because the value of argument $\vec{u}$ may also be affected by $\mathcal{A}$, recursively. Recall that axioms [:=] and [:] assume $\forall i:C\ f(\vec{s}) := \theta$ to be injective or vacuous.

Axiom $[:=]_s$ is an important special case of [:=] that applies to the schematic case where $\vec{s}$ is of the form i, which matches trivially. If f does not occur in u, then [:] simplifies this further:

$$[:=]_s,[:]\frac{\forall i:C\ (i = u \to \phi(\theta))}{\phi([\forall i:C\ f(i) := \theta]f(u))}$$

When f does not occur in u, standard first-order reasoning can simplify further (θ_i^u is the term θ with i replaced by u):

$$[:=]\frac{\phi(\theta_i^u)}{\phi([\forall i:C\ f(i) := \theta]f(u))}$$

Together with [:] to propagate the change to both arguments of $\neq$, this derived rule proves, e.g., the following proof step:

$$[:=],[:]\frac{\forall j\neq k\,(-\frac{b}{2}s^2 + v(j)s + x(j) \neq -\frac{b}{2}s^2 + v(k)s + x(k))}{\forall j\neq k\,[\forall i\,x(i) := -\frac{b}{2}s^2 + v(i)s + x(i)]\,x(j)\neq x(k)}$$

Axiom [:*] reduces nondeterministic assignments to universal quantification. For the handling of other general nondeterministic assignments and nondeterministic differential equations, we refer to previous work [Platzer 2010a; Platzer 2010b].

Axioms [:=],[:] also apply for assignments without quantifiers, which correspond to vacuous quantification $\forall i:C$ where i does not occur anywhere; see previous work

[Platzer 2010c; Platzer 2012a]. That case amounts to a notational variant of the $[:=]$ axiom of dL from Figure 6. Vectorial extensions to systems of quantified assignments and systems of quantified differential equations with multiple function symbols are possible as well [Platzer 2010c; Platzer 2012a].

Axiom $[']$ handles continuous evolutions for quantified differential equations with first-order definable solutions. The difference compared to the axiom $[']$ of dL is that QdL handles infinite-dimensional quantified differential equation systems or quantified differential equation systems with evolving dimensions. Their solutions are no longer expressible as assignments, but need quantified assignments. Given a solution for the quantified differential equation system with symbolic initial values $f(\vec{s})$, continuous evolution along differential equations can be replaced with a quantified assignment $\forall i : C\ f(\vec{s}) := y_{\vec{s}}(t)$ corresponding to the simultaneous solution (of the differential equations $\forall i : C\ f(\vec{s})' = \theta$ with $f(\vec{s})$ as symbolic initial values) and an additional quantifier for all evolution durations $t \geq 0$.

For schematic cases like $\forall i : C\ f(i)' = a(i)$, first-order definable solutions can be obtained by adding argument i to first-order definable solutions of the deparametrized version $f' = a$. For example, the following proof step uses axiom $[']$ to turn a quantified differential equation system into a quantified assignment with an extra quantifier for the common duration t of the evolution.

$$[']\frac{\forall t{\geq}0\,[\forall i\, x(i) := -\frac{b}{2}t^2 + v(i)t + x(i)]\,\forall j{\neq}k\, x(j){\neq}x(k)}{[\forall i\, x(i)' = v(i), v(i)' = -b]\,\forall j{\neq}k\, x(j){\neq}x(k)}$$

The quantified assignment $\forall i\, x(i) := -\frac{b}{2}t^2 + v(i)t + x(i)$ solving the above quantified differential equation system can be obtained easily from the solution $x := -\frac{b}{2}t^2 + vt + x$ of the deparametrized differential equation system $x' = v, v' = -b$, just by adding the parameter i back in and checking whether this gives the solution.

The modular "there and back again" axiom $[\&]$ that reduces quantified differential equations with evolution domain constraints to quantified differential equations without them works as in dL (Sect. 3.4). For an explanation how quantified differential equations have unique solutions as required for this to be sound, we refer to previous work [Platzer 2010c; Platzer 2012a]. The QdL axioms $[?]$, $[\cup]$, $[;]$, $[^*]$, K, I, C, B, V, and rule G are as for dL, even if B and rule $\forall$ are now many-sorted.

Axiom $\exists$ expresses that, for sort $C \neq \mathbb{R}$, there always is a new object n that has not been created yet ($\exists(n) = 0$), because domains are infinite. This is the only place where we are using the assumption about infinite domains. The primary purpose is to simplify technicalities that would arise if object creation could run out of objects and may thus fail if, e.g., no more cars can be created; see previous work [Platzer 2010c; Platzer 2012a] for details on object/agent creation.

Example 4.7 (*Distributed car control*). To illustrate how the QdL proof calculus works, we use QdL axioms as shown in Figure 15 to identify when the QdL formula at the bottom is valid. The QdL formula that we consider here follows the pattern of the running example formula (12). We only consider the braking case for typesetting reasons and refer to [Platzer 2012a] for a full proof. The case we consider is the distributed hybrid systems analog of dL formula (8) that we proved in Example 3.15. The other difference compared to (12) is that the formula in Figure 15 has a weaker assumption. It only assumes that cars start from different positions ($\forall i{\neq}j\, x(i){\neq}x(j)$), not that they respect the compatibility constraint $\forall i, j : C\ \mathcal{M}(i, j)$. Like in Example 3.15, we are using the QdL axioms to find out by the derivation in Figure 15 how we need to choose $\mathcal{M}(i, j)$ to ensure collision freedom.

$$\mathbb{R}\ \frac{\forall i{\neq}j\, x(i){\neq}x(j) \to \forall j{\neq}k\,(x(j) \leq x(k) \land v(j) \leq v(k) \lor x(j) \geq x(k) \land v(j) \geq v(k))}{\forall i{\neq}j\, x(i){\neq}x(j) \to \forall j{\neq}k\, \forall t{\geq}0\,(-\frac{b}{2}t^2 + v(j)t + x(j) \neq -\frac{b}{2}t^2 + v(k)t + x(k))}$$

$$\mathbb{R}\ \frac{\forall i{\neq}j\, x(i){\neq}x(j) \to \forall t{\geq}0\, \forall j{\neq}k\,(-\frac{b}{2}t^2 + v(j)t + x(j) \neq -\frac{b}{2}t^2 + v(k)t + x(k))}{\forall i{\neq}j\, x(i){\neq}x(j) \to \forall t{\geq}0\, \forall j{\neq}k\,[\forall i\, x(i) := -\frac{b}{2}t^2 + v(i)t + x(i)]\, x(j){\neq}x(k)}\ [:=]_s$$

$$\frac{\forall i{\neq}j\, x(i){\neq}x(j) \to \forall t{\geq}0\,[\forall i\, x(i) := -\frac{b}{2}t^2 + v(i)t + x(i)]\, \forall j{\neq}k\, x(j){\neq}x(k)}{\forall i{\neq}j\, x(i){\neq}x(j) \to \forall t{\geq}0\,[\forall i\, x(i) := -\frac{b}{2}t^2 + v(i)t + x(i)]\, \forall j{\neq}k\, x(j){\neq}x(k)}\ [:]$$

$$\frac{}{\forall i{\neq}j\, x(i){\neq}x(j) \to [\forall i\, x(i)' = v(i), v(i)' = -b]\, \forall j{\neq}k\, x(j){\neq}x(k)}\ [']$$

Fig. 15. Example of a QdL prove about collision-freedom of simple distributed car control

We start with the conjecture at the bottom of Figure 15 and successively reduce it by using QdL axioms and proof rules. First, we use axiom $[']$ to turn the quantified differential equation system into a quantified assignment with an extra quantifier for the duration t of the evolution. The quantified differential equation system is easy to solve. The quantified assignment $\forall i\, x(i) := -\frac{b}{2}t^2 + v(i)t + x(i)$ solving it can be obtained easily from the solution $x := -\frac{b}{2}t^2 + vt + x$ of the deparametrized differential equation system $x' = v, v' = -b$, just by adding the parameter i back in and checking that the resulting terms solve the quantified differential equation. The premise of the use of $[']$ has a quantifier $\forall t$ as the top-most logical operator in the succedent. Even though it is a quantifier over a real variable, we cannot use the decision procedure of quantifier elimination for real-closed fields [Tarski 1951] to handle it, because we do not have a formula of first-order real arithmetic, but still a QdL formula with a modality expressing a property of all reachable states. Instead, we use axiom $[:]$ to skip over the quantifier $\forall j{\neq}k$ and then use axiom $[:=]_s$ or $[:=]$ to let the quantified assignment to $x(i)$ (for all i) take effect on the postcondition $x(j) \neq x(k)$ by skipping over the $\neq$ with axiom $[:]$ (not shown in Figure 15) and then affecting $x(j)$ and $x(k)$ subsequently by rule $[:=]_s$.

At this point (premise of the top-most use of axiom $[:=]_s$ in Figure 15), we already have a first-order formula and it looks like we could apply quantifier elimination for real-closed fields [Tarski 1951] to the quantified real variable t. This would not work, however, because quantifier elimination works from inside out and will try to eliminate the inner quantifier $\forall j{\neq}k$ before the outer quantifier $\forall s$. Yet, this formula is not an instance of first-order real arithmetic (not even when using Lemma 3.14), but of many-sorted first-order logic with quantified variables j, k of sort C, because there are dependencies on the quantified variables j, k in function arguments, which is fundamentally more difficult [Platzer 2010b]. Instead, the proof in Figure 15 uses a tautology of first-order logic (marked $\mathbb{R}$) to commute the quantifiers. Now, we can apply quantifier elimination for real-closed fields [Tarski 1951] as an equivalence in first-order logic (written $\mathbb{R}$), even though the formula is still not in first-order real arithmetic, because function symbols like $v(j)$ occur. However, it is an instance ($v(j)$ for V and $x(j)$ for X and $v(k)$ for W and $x(k)$ for Y) of the following formula of first-order real arithmetic:

$$\forall s{\geq}0\left(j{\neq}k \to -\frac{b}{2}s^2 + Vs + X \neq -\frac{b}{2}s^2 + Ws + Y\right) \tag{16}$$

and, thus, quantifier elimination can be lifted by Lemma 3.14. The result of quantifier elimination is an instance (with the same instantiation as above) of the result of applying QE to (16). Finally, we could use real arithmetic on the top-most formula, as an instance of the following formula of plain first-order real arithmetic (with the instantiation $x(j)$ for X, $v(j)$ for V, $x(k)$ for Y, and $v(k)$ for W)

$$j \neq k \to X \leq Y \land V \leq W \lor X \geq Y \land V \geq W$$

However, the formula is not valid, so the proof does not close. This is good news for soundness, however, because the conjecture at the bottom of Figure 15 is not valid, unless the constraints at the top hold about the relation of the velocities and positions of the cars. The constraints at the top of Figure 15 can be used to construct the constraints required for safety, which coincide with $\mathcal{M}(j,k)$ that we have shown in (13).

A comparison of the QdL proof in Example 4.7 compared to the dL proofs in Example 3.15 shows that the interplay of instantiation in many-sorted first-order logic and quantifier elimination in first-order real arithmetic is an important challenge when reasoning about distributed hybrid systems. We refer to previous work [Platzer 2010c; Platzer 2012a; Platzer 2008a; Renshaw et al. 2011] for details on how this can be automated in the QdL calculus.

4.5. Soundness and Completeness

Distributed hybrid systems have several independent sources of undecidability: discrete dynamics, continuous dynamics, and structural/dimensional dynamics [Platzer 2010c; Platzer 2012a].

THEOREM 4.8 (INCOMPLETENESS OF QdL [PLATZER 2010C; PLATZER 2012A]). *The discrete fragment of* QdL*, the continuous fragment of* QdL*, and the fragment of* QdL *with structural and dimension-changing dynamics are* not effectively axiomatizable*, i.e., they have no sound and complete effective calculus, because natural numbers are definable in each of those fragments.*

PROOF. Incompleteness of the discrete fragment and of the continuous fragment follows from Theorem 3.16. Gödel's incompleteness theorem [Gödel 1931] applies to the fragment with only structural and dimensional dynamics, because natural numbers are definable in that fragment by chains of links along the values of a function p, encoding zero by constant symbol z:

$$\mathit{nat}(n) \;\leftrightarrow\; \langle (?n \neq z;\; n := p(n))^* \rangle\, n = z.$$

For details on the characterization of addition and multiplication, we refer to the full proof [Platzer 2012a]. The idea behind addition is shown in Figure 16: create a new

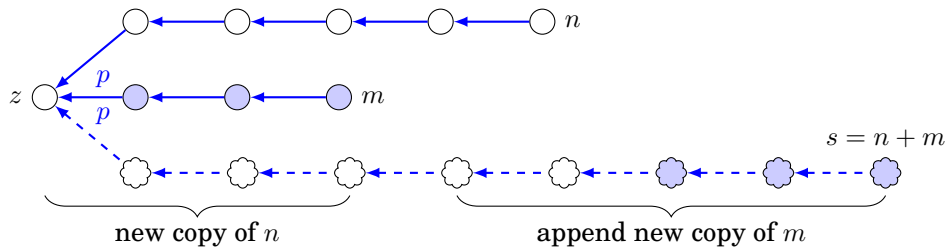

Fig. 16. Characterization of $\mathbb{N}$ addition with p links in dimensional dynamics

chain of links along the values of p by first creating exactly as many links as we can follow along p when starting from n, and then continue creating exactly as many links as we can follow along p when starting from m, instead. The number of links of the result s is the sum of the respective numbers of links of n and m. □

We have shown that the original QdL calculus [Platzer 2010c; Platzer 2012a] is a sound and complete axiomatization of QdL relative to the continuous fragment (FOQD). FOQD is the *first-order logic of quantified differential equations*, i.e.,

(many-sorted) first-order logic with real arithmetic augmented with formulas expressing properties of quantified differential equations, that is, QdL formulas of the form $[\forall i:C\ f(\vec{s})' = \theta \,\&\, \chi]F$. The dual formula $\langle \forall i:C\ f(\vec{s})' = \theta \,\&\, \chi\rangle F$ is expressible as $\neg[\forall i:C\ f(\vec{s})' = \theta \,\&\, \chi]\neg F$. A combination of the original proof [Platzer 2010c; Platzer 2012a] and the proof of Theorem 3.17 can be used to show that the simplified QdL calculus in Figure 14 is sound and complete relative to FOQD.

THEOREM 4.9 (RELATIVE COMPLETENESS OF QdL [PLATZER 2010C; PLATZER 2012A]). *The QdL calculus is a* sound and complete axiomatization *of distributed hybrid systems relative to FOQD, i.e., every valid QdL formula can be derived from FOQD tautologies:*

$$\vDash \phi \ \textit{iff}\ \mathrm{Taut}_{FOQD} \vdash \phi$$

This central result shows that properties of distributed hybrid systems can be proven to exactly the same extent to which properties of quantified differential equations can be proven. Proof-theoretically, the QdL calculus completely lifts verification techniques for quantified continuous dynamics to distributed hybrid dynamics. Even though distributed hybrid systems have numerous independent sources of undecidability, we have shown that all true QdL formulas can be proven in our QdL calculus, if only we manage to tame the complexity of the continuous dynamics. Despite these new independent sources of undecidability, we have shown that QdL can still be axiomatized completely relative to differential equations, only now they are quantified differential equations.

Another important consequence of this result is that decomposition is successful in taming the complexity of distributed hybrid systems. The QdL proof calculus is strictly compositional. All QdL axioms and proof rules prove logical formulas or properties of QHPs by reducing them to structurally simpler QdL formulas. As soon as we understand that the distributed hybrid systems complexity comes from a combination of several simpler aspects, we can, hence, tame the system complexity by reducing it to analyzing the dynamical effects of simpler parts. This decomposition principle is exactly how QdL proofs can scale to interesting systems in practice. The relative completeness theorem 4.9 gives the theoretical evidence why this principle works in general. This is yet another illustration of our principle of multi-dynamical systems and even a proof that the decompositions behind the multi-dynamical systems approach are successful.

4.6. Quantified Differential Invariants

Differential invariants (Sect. 3.6) are the premier proof technique for proving properties of complicated differential equations, but they only work for hybrid systems. Their generalization to quantified differential equations of distributed hybrid systems is called *quantified differential invariant* [Platzer 2011a]. For quantified differential equations, one of the extra challenges is that the system does not have a fixed finite dimension but can be arbitrary-dimensional and even of evolving dimensions. Consequently, there is not even a finite vector space in which the local directions of the vector field of the differential equations can be described and checked. Instead we need a criterion based on implicit properties of the local dynamics at uncountably many points in an essentially infinite-dimensional vector field of evolving dimensions. We lift the syntactic derivation operator from Sect. 3.6 to first-order for that purpose.

Definition 4.10 (*Quantified derivation*). The operator D that is defined as follows on terms is called *(quantified) syntactic (total) derivation*:

$$D(r) = 0 \qquad \text{for } r \in \mathbb{Q} \tag{17a}$$
$$D(x(s)) = x(s)' \quad \text{for function symbol } x : C \to \mathbb{R} \text{ with } C \neq \mathbb{R} \text{ discrete} \tag{17b}$$
$$D(a + b) = D(a) + D(b) \tag{17c}$$
$$D(a - b) = D(a) - D(b) \tag{17d}$$
$$D(a \cdot b) = D(a) \cdot b + a \cdot D(b) \tag{17e}$$
$$D(a/b) = (D(a) \cdot b - a \cdot D(b))/b^2 \tag{17f}$$

We extend D to first-order formulas F in prefix disjunctive normal form as follows:

$$\begin{aligned} D(\forall i : C\ F) &\equiv \forall i : C\ D(F) \\ D(\exists i : C\ F) &\equiv \forall i : C\ D(F) \\ D(F \wedge G) &\equiv D(F) \wedge D(G) \\ D(F \vee G) &\equiv D(F) \wedge D(G) \\ D(a \geq b) &\equiv D(a) \geq D(b) \quad \text{accordingly for } <, >, \leq, = . \end{aligned}$$

The *quantified differential induction* rule is a natural induction principle for quantified differential equations:

$$\text{(DI)} \quad \frac{\chi \to [\forall i : C\ f(\vec{i})' := \theta] D(F)}{F \to [\forall i : C\ f(\vec{i})' = \theta\ \&\ \chi] F}$$

For a formula F if we can prove the premise of rule DI, i.e., that, after a differential substitution $[\forall i : C\ f(\vec{i})' := \theta]$, the total derivative $D(F)$ is valid in the evolution domain region χ, then the conclusion of DI is valid, i.e., the system always stays in region F when it starts in F (left assumption in conclusion). It is important that we add the quantified assignment $\forall i : C\ f(\vec{i})' := \theta$ for quantified differential symbol $f(\vec{i})'$ as a *quantified differential substitution* in the premise, because, otherwise, the premise of DI is not a logical formula that would have a well-defined semantics when evaluated in a state. Unlike F, the total derivative $D(F)$ contains differential function symbols like $f(i)'$, which do not have a semantics in isolated states but only along a flow (an insightful differential semantics can be defined but is beyond the scope of this article). The quantified assignment defines a value for those differential function symbols, which has a well-defined correspondence to the local dynamics of quantified differential equations based on a differential substitution property [Platzer 2011a, Lemma 2]. The conjunctive definition of $D(F \vee G)$ is crucial (recallSect. 3.6) and so is the universal definition of $D(\exists i : C\ F)$. Differential cuts DC (see Sect. 3.8) generalize to quantified differential equations immediately [Platzer 2011a] and are as crucial as they are for hybrid systems. A generalization of the derivation lemma (Lemma 3.19) to quantified differential equations is a key argument to relate analytic differentiation and syntactic derivations [Platzer 2011a, Lemma 1]. For details, see previous work [Platzer 2011a].

4.7. Implementation and Applications

Quantified differential dynamic logic and a sequent calculus variation of its proof calculus [Platzer 2010c; Platzer 2012a], including quantified differential invariants and quantified differential cuts [Platzer 2011a] have been implemented in the LCF-style tactics-based theorem prover KeYmaeraD [Renshaw et al. 2011].[5] The name

[5]Available at `http://symbolaris.com/info/KeYmaeraD.html`

KeYmaeraD extends KeYmaera with a D for distributed, which represents the fact that KeYmaeraD can prove distributed hybrid systems and the fact that KeYmaeraD is implemented with an architecture that supports distributed and parallel proving of both distributed and non-distributed hybrid systems.

Quantified differential dynamic logic and KeYmaeraD have been used successfully to prove collision-freedom properties for distributed car controllers for arbitrarily many cars on arbitrarily many lanes on straight highways [Platzer 2010c; Platzer 2012a; Loos et al. 2011; Renshaw et al. 2011] and safe separation properties for roundabout collision avoidance maneuvers for arbitrarily many aircraft with dynamic appearance of aircraft during collision avoidance [Platzer 2011a]. They have also been used to prove properties of advanced flight control protocols and medical robotic applications.

5. STOCHASTIC DIFFERENTIAL DYNAMIC LOGIC FOR STOCHASTIC HYBRID SYSTEMS

In this section, we study *stochastic differential dynamic logic* Sd$\mathcal{L}$ [Platzer 2011b], the *logic of stochastic hybrid systems*, i.e., systems that combine stochastic dynamics with the discrete and continuous dynamics of hybrid systems.

In the previous sections, we have seen that logic of dynamical systems is a powerful tool for analyzing and verifying dynamical systems, including hybrid systems (Sect. 3) and distributed hybrid systems (Sect. 4). Some applications also exhibit a stochastic behavior, however, either because of fundamental properties of nature, uncertain environments, or simplifications to overcome complexity. Uncertainties can sometimes be modeled well by taking a nondeterministic perspective where anything could happen and we are not concerned with how probable which outcome is. Nondeterminism can be used to model uncertainty in hybrid systems (Sect. 3) and distributed hybrid systems (Sect. 4). That is useful, for example, if a car is uncertain about whether the car in front of it will brake or accelerate, so that the follower car has to be prepared to handle either choice safely. In other situations, however, stochastic models are needed to model uncertainty. For example, a nondeterministic model of a lossy communication channel would consider it possible for a message to arrive and possible for a message to disappear during transmission. But then one possible resolution of the nondeterminisms would cause all messages to disappear from all communication attempts, which is certainly possible, just extremely unlikely, except in adversarial situations. Whenever our analysis would be too imprecise with a nondeterministic view or whenever we have good stochastic models, probabilistic resolutions of uncertainties are more appropriate. This is what we study in this section.

Discrete probabilistic systems have been studied successfully using logic [Kozen 1981; Kozen 1985; Feldman and Harel 1984; McIver and Morgan 2004]. In this section, we present our dynamic logic of stochastic hybrid systems [Platzer 2011b], i.e., systems with interacting discrete, continuous, and stochastic dynamics. Our results indicate that logic is a promising tool for understanding stochastic hybrid systems and can help taming some of their complexity.

Classical logic is about boolean yes/no truth. That makes it tricky to use logic for systems with stochastic effects. Logic has reached out into probabilistic extensions at least for discrete programs [Kozen 1981; Kozen 1985; Feldman and Harel 1984; McIver and Morgan 2004] and for first-order logic over a finite domain [Richardson and Domingos 2006]. Logic has been used for the purpose of specifying system properties in model checking finite Markov chains [Younes et al. 2006] and probabilistic timed automata [Kwiatkowska et al. 2007].

Stochastic hybrid systems [Bujorianu and Lygeros 2006; Cassandras and Lygeros 2006; Hu et al. 2000; Platzer 2011b] are more general systems with interacting discrete, continuous, and stochastic dynamics. There is not just one canonical way to add stochastic behavior to a system model. Stochasticity might be restricted to the discrete

dynamics, as in piecewise deterministic Markov decision processes [Davis 1984], restricted to the continuous and switching behavior as in switching diffusion processes [Ghosh et al. 1997], or allowed in many parts as in so-called General Stochastic Hybrid Systems; see [Bujorianu and Lygeros 2006; Cassandras and Lygeros 2006] for an overview. Several different forms of combinations of probabilities with hybrid systems and continuous systems have been considered, both for model checking [Cassandras and Lygeros 2006; Koutsoukos and Riley 2008; Fränzle et al. 2010] and for simulation-based validation [Meseguer and Sharykin 2006; Zuliani et al. 2010].

We have introduced a very different approach [Platzer 2011b] that is based on logic for dynamical systems. We consider logic and proofs for stochastic hybrid systems[6] to transfer the success that logic has had in other domains. Our approach is partially inspired by probabilistic PDL [Kozen 1985] by barrier certificates for continuous dynamics [Prajna et al. 2007]. We follow the arithmetical view that Kozen identified as suitable for probabilistic logic [Kozen 1985]. Simple probabilistic effects can already be encoded in real variables of $\mathsf{d}\mathcal{L}$ and $\mathsf{Qd}\mathcal{L}$, but general stochastic dynamics requires the approach presented in this section.

Classical analysis is provably inadequate [Kloeden and Platen 2010] for analyzing even simple continuous stochastic processes. We heavily draw on both stochastic calculus and logic. It is not possible to present all mathematical background exhaustively here. We provide basic definitions and intuition and refer to the literature for more details and proofs [Platzer 2011b; Karatzas and Shreve 1991; Øksendal 2007; Kloeden and Platen 2010].

We show the model of *stochastic hybrid programs* (SHPs) [Platzer 2011b] that combine discrete stochastic dynamics and stochastic differential equations for continuous stochastic dynamics, and we define a compositional semantics of SHP runs in terms of stochastic processes. We have proved that the semantic processes are adapted, almost surely have càdlàg paths, and that their natural stopping times are Markov times. We have introduced *stochastic differential dynamic logic* ($\mathsf{Sd}\mathcal{L}$) for specifying and verifying properties of SHPs [Platzer 2011b]. We define a semantics and have proved that the semantics is measurable such that probabilities are well-defined and probabilistic questions become meaningful. We present proof rules for $\mathsf{Sd}\mathcal{L}$ and have proved their soundness [Platzer 2011b]. $\mathsf{Sd}\mathcal{L}$ makes the rich semantical complexity and deep theory of stochastic hybrid systems accessible in a simple syntactic language. This makes the verification of stochastic hybrid systems possible with elementary syntactic proof principles.

We first briefly recall the basics of stochastic processes and stochastic differential equations (Sect. 5.1). Then we explain the system model of stochastic hybrid programs that $\mathsf{Sd}\mathcal{L}$ provides for modeling stochastic hybrid systems (Sect. 5.2) and define their semantics. We define the terms and logical formulas that $\mathsf{Sd}\mathcal{L}$ provides for specification and verification (Sect. 5.3), show measurability results (Sect. 5.4) and then provide reasoning principles, axioms, and proof rules for verifying $\mathsf{Sd}\mathcal{L}$ formulas (Sect. 5.5). We then show soundness theorems (Sect. 5.6) and investigate proof rules for stochastic differential equations (Sect. 5.7).

5.1. Preliminaries: Stochastic Differential Equations

We fix a dimension $d \in \mathbb{N}$ for the Euclidean state space $\mathbb{R}^d$ equipped with its *Borel σ-algebra* $\mathcal{B}$, i.e., the σ-algebra generated by all open subsets. A *σ-algebra* on a set Ω is a nonempty set $\mathcal{F} \subseteq 2^\Omega$ that is closed under complement ($E \in \mathcal{F}$ implies $\Omega \setminus E \in$

[6]Note that there is a specific class of models called Stochastic Hybrid Systems [Hu et al. 2000]. We do not mean this specific model in the narrow sense but refer to stochastic hybrid systems as the broader class of systems that share discrete, continuous, and stochastic dynamics.

$\mathcal{F}$) and countable union ($E_i \in \mathcal{F}$ implies $\bigcup_{i=1}^{\infty} E_i \in \mathcal{F}$). Even though it could also be constructed, we axiomatically fix a *probability space* $(\Omega, \mathcal{F}, P)$ with a σ-algebra $\mathcal{F} \subseteq 2^\Omega$ of events on space Ω and a probability measure P on $\mathcal{F}$. Function $P : \mathcal{F} \to [0,1]$ is a *probability measure* on $\mathcal{F}$ if P is countable additive (i.e., $P(\bigcup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} P(E_i)$ when $E_i \cap E_j = \emptyset$ for all $i \neq j$) and $P \geq 0, P(\Omega) = 1$. We assume the probability space has been *completed*, i.e., every subset of a null set (i.e., $P(A) = 0$) is measurable. A property holds *P-almost surely* (*a.s.*) if it holds with probability 1. A *filtration* is a family $(\mathcal{F}_t)_{t\geq 0}$ of σ-algebras that is increasing, i.e., $\mathcal{F}_s \subseteq \mathcal{F}_t$ for all $s < t$. Intuitively, $\mathcal{F}_t$ are the events that can be discriminated at time t. We always assume a filtration $(\mathcal{F}_t)_{t\geq 0}$ that has been completed to include all null sets and that is *right-continuous*, i.e., $\mathcal{F}_t = \bigcap_{u>t} \mathcal{F}_u$ for all t. We generally assume the compatibility condition that $\mathcal{F}$ coincides with the σ-algebra $\mathcal{F}_\infty := \sigma\left(\bigcup_{t\geq 0} \mathcal{F}_t\right)$, i.e., the σ-algebra generated by all $\mathcal{F}_t$.

For a σ-algebra Σ on a set D and the Borel σ-algebra $\mathcal{B}$ on $\mathbb{R}^d$, function $f : D \to \mathbb{R}^d$ is *measurable* iff $f^{-1}(B) \in \Sigma$ for all $B \in \mathcal{B}$ (or, equivalently, for all open $B \subseteq \mathbb{R}^d$). An $\mathbb{R}^d$-valued *random variable* is an $\mathcal{F}$-measurable function $X : \Omega \to \mathbb{R}^d$. All sets and functions definable in first-order logic over real arithmetic are Borel-measurable [Tarski 1951]. A *stochastic process* X is a collection $\{X_t\}_{t\in T}$ of $\mathbb{R}^d$-valued random variables X_t indexed by some set T for time. That is, $X : T \times \Omega \to \mathbb{R}^d$ is a function such that at all $t \in T$, $X_t = X(t,\cdot) : \Omega \to \mathbb{R}^d$ is a random variable. Process X is *adapted* to filtration $(\mathcal{F}_t)_{t\geq 0}$ if X_t is $\mathcal{F}_t$-measurable for each t. That is, the process does not depend on future events. We consider only adapted processes (which can be ensured, e.g., by using the completion of the natural filtration of a process or the completion of the optional σ-algebra for $\mathcal{F}$, see [Karatzas and Shreve 1991]). A process X is *càdlàg* iff its *paths* $t \mapsto X_t(\omega)$ (for each $\omega \in \Omega$) are càdlàg a.s., i.e., *right-continuous* ($\lim_{s\searrow t} X_s(\omega) = X_t(\omega)$) and have *left limits* ($\lim_{s\nearrow t} X_s(\omega)$ exists).

We further need an e-dimensional *Brownian motion* W [Karatzas and Shreve 1991; Øksendal 2007; Kloeden and Platen 2010], that is, W is a stochastic process starting at 0 ($W_0 = 0$) that is almost surely continuous and has independent increments that are normally distributed with mean 0 and variance equal to the time difference, i.e., $W_t - W_s \sim \mathcal{N}(0, t-s)$. Brownian motion is mathematically extremely complex. Its paths are a.s. continuous everywhere but a.s. differentiable nowhere and a.s. of unbounded variation. Thus, Brownian motion is a.s. not of finite variation, hence, standard integral notions are inapplicable. Brownian motion is also a.s. nonmonotonic on every interval. Intuitively, W can be understood as the limit of a random walk. We denote the Euclidean vector norm by $|x|$ and use the Frobenius norm $|\sigma| := \sqrt{\sum_{i,j} \sigma_{ij}^2}$ for matrices $\sigma \in \mathbb{R}^{d\times e}$.

We use stochastic differential equations [Øksendal 2007; Kloeden and Platen 2010] to describe stochastic continuous system dynamics. They are like ordinary differential equations but have an additional diffusion term that varies the state stochastically. Stochastic differential equations are of the form $dX_t = b(X_t)dt + \sigma(X_t)dW_t$. We consider Itō stochastic differential equations, whose solutions are defined by the stochastic Itō integral [Øksendal 2007; Kloeden and Platen 2010], which is a stochastic process. Like in an ordinary differential equation, the drift coefficient $b(X_t)$ determines the deterministic part of how X_t changes systematically over time as a function of its current value. As a function of X_t, the diffusion coefficient $\sigma(X_t)$ determines the stochastic influence by integration with respect to the Brownian motion process W_t. See Figure 17 for sample paths. Ordinary differential equations are retained for $\sigma = 0$. We focus on the time-homogeneous case, where b and σ are time-independent, because time could be added as an extra state variable if needed.

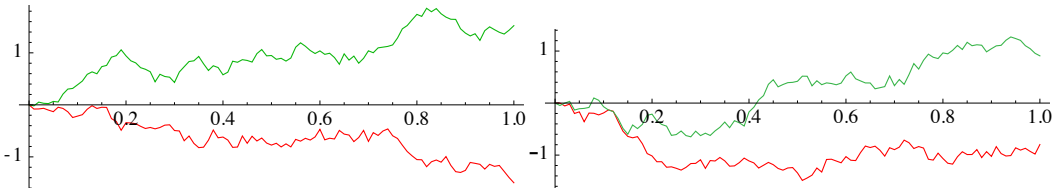

Fig. 17. Sample paths with $b = 1, \sigma = 1$ (top) and $b = 0, \sigma = 1$ (bottom)

Definition 5.1 (*Stochastic differential equation*). A stochastic process $X : [0, \infty) \times \Omega \to \mathbb{R}^d$ solves the (Itō) *stochastic differential equation*

$$dX_t = b(X_t)dt + \sigma(X_t)dW_t \tag{18}$$

with $X_0 = Z$, if $X_t = Z + \int b(X_t)dt + \int \sigma(X_t)dW_t$, where $\int \sigma(X_t)dW_t$ is an Itō integral process [Øksendal 2007; Kloeden and Platen 2010]. We assume $b : \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma : \mathbb{R}^d \to \mathbb{R}^{d\times e}$ to be measurable and locally Lipschitz-continuous, i.e., for all N there is a C such that for all x, y with $|x|, |y| \leq N$:

$$|b(x) - b(y)| \leq C|x - y| \text{ and } |\sigma(x) - \sigma(y)| \leq C|x - y|$$

As an integral of an a.s. continuous process, solution X has a.s. continuous paths [Øksendal 2007]. Local Lipschitz-continuity guarantees that the a.s. continuous solution X is pathwise unique [Kloeden and Platen 2010, Ch 4.5] and enables us to compute the infinitesimal generator of X from its differential generator (see Sect. 5.7). Process X is a strong Markov process for each initial value x [Øksendal 2007, Theorem 7.2.4]. We focus on the time-homogeneous case, where b and σ are time-independent, because time could be added as an extra state variable.

5.2. Stochastic Hybrid Programs

As a system model for stochastic hybrid systems, we have introduced stochastic hybrid programs (SHPs) [Platzer 2011b]. SHPs combine stochastic differential equations for describing the stochastic continuous system dynamics with program operations to describe the discrete stochastic choices, discrete switching, and jumps. These primitive dynamics can be combined programmatically in flexible ways. All basic terms in stochastic hybrid programs and stochastic differential dynamic logic are polynomial terms built over real-valued variables and rational constants. Extensions to rational functions are possible.

Definition 5.2 (*Stochastic hybrid program*). *Stochastic hybrid programs (SHPs)* are formed by the following grammar (where x_i is a variable, x a vector of variables, θ a basic term, b a vector of basic terms, σ a matrix of basic terms, χ is a quantifier-free first-order real arithmetic formula, $\lambda, \nu \geq 0$ are rational numbers):

$$\alpha, \beta \ ::= \ x_i := \theta \mid x_i := * \mid ?\chi \mid dx = bdt + \sigma dW \,\&\, \chi \mid \lambda\alpha \ \oplus \ \nu\beta \mid \alpha; \beta \mid \alpha^*$$

Assignment $x_i := \theta$ deterministically assigns term θ to variable x_i instantaneously. *Random assignment* $x_i := *$ randomly updates variable x_i, but unlike in classical dynamic logic [Pratt 1976] and differential dynamic logic, we assume a probability distribution for x. As one example for a probability distribution, we consider uniform distribution in the interval [0,1], but other distributions can be used as long as they are computationally tractable, e.g., definable in first-order real arithmetic.

Most importantly, $dx = bdt + \sigma dW \,\&\, \chi$ represents a *stochastic continuous evolution* along a stochastic differential equation, restricted to the evolution domain region χ, i.e., the stochastic process will stop when it leaves χ. Unlike in $\mathsf{d}\mathcal{L}$ and $\mathsf{Qd}\mathcal{L}$, where

we take a nondeterministic view, stochastic continuous evolutions do not stop prematurely, but exactly when they leave χ. The time when evolutions stop is still random, but now described by the stochastic continuous evolution and its evolution domain constraint instead of by nondeterminism. In particular, χ represents control decisions when to interrupt the continuous stochastic process. Because of that, we can show that the random variable describing when the stochastic continuous evolution stops is a Markov time (Theorem 5.9). We assume that $dx = bdt + \sigma dW$ satisfies the assumptions of stochastic differential equations from Def. 5.1.

Test $?\chi$ represents a stochastic process that fails (disappears into an absorbing state) if χ is not satisfied yet continues unmodified otherwise. *Linear combination* $\lambda\alpha \oplus \nu\beta$ evolves like α with probability λ and like β otherwise. For conceptual simplicity, we assume $\lambda + \nu = 1$, but other linear combinations are possible when taking care to ensure that the result still gives probabilities [Kozen 1985], e.g., when using complementary tests in the definition $\mathsf{if}(\chi)\,\alpha\,\mathsf{else}\,\beta \equiv (?\chi;\alpha) \oplus (?\neg\chi;\beta)$. It is possible to extend this to the case where λ, μ are terms. Linear combination alias probabilistic choice $\lambda\alpha \oplus \nu\beta$ is the counterpart of the nondeterministic choice $\alpha \cup \beta$ from Sect. 3, but gives information about the probability with which the respective choices are taken. *Sequential composition* $\alpha;\beta$ and *repetition* α^* work similarly to differential dynamic logic, except that they combine SHPs instead of non-stochastic HPs.

The semantics of a SHP is the stochastic process that it generates. The semantics $[\![\alpha]\!]$ of a SHP α consists of a function $[\![\alpha]\!] : (\Omega \to \mathbb{R}^d) \to ([0,\infty) \times \Omega \to \mathbb{R}^d)$ that maps any $\mathbb{R}^d$-valued random variable Z describing the initial state to a stochastic process $[\![\alpha]\!]^Z$ together with a function $(\!|\alpha|\!) : (\Omega \to \mathbb{R}^d) \to (\Omega \to \mathbb{R})$ that maps any $\mathbb{R}^d$-valued random variable Z describing the initial state to a (random) stopping time $(\!|\alpha|\!)^Z$ indicating when to stop $[\![\alpha]\!]^Z$. Often, an $\mathcal{F}_0$-measurable random variable Z or deterministic state is used to describe the initial state. We assume independence of Z from subsequent stochastic processes like Brownian motions occurring in the definition of $[\![\alpha]\!]^Z$. For an $\mathbb{R}^d$-valued random variable Z, we denote the stochastic process $\hat{Z} : \{0\} \times \Omega \to \mathbb{R}^d; (0,\omega) \mapsto \hat{Z}_0(\omega) := Z(\omega)$ that is stuck at Z by $\hat{Z}$. We write $\hat{x}$ for the random variable Z that is a deterministic state $Z(\omega) := x$ for all $\omega \in \Omega$. We write $[\![\alpha]\!]^x$ and $(\!|\alpha|\!)^x$ for $[\![\alpha]\!]^Z$ and $(\!|\alpha|\!)^Z$ in that case.

In order to simplify notation, we assume that all variables are uniquely identified by an index, i.e., the only occurring variables are $x_1, x_2, \ldots, x_d$. We write $Z(\omega) \models \chi$ if state $Z(\omega)$ satisfies first-order real arithmetic formula χ and $Z(\omega) \not\models \chi$ otherwise. In the semantics we use a family of random variables $\{U_i\}_{i\in I}$ that are distributed uniformly in $[0,1]$ and independent of other U_j and of all other random variables and stochastic processes in the semantics. Hence, U satisfies $P(\{\omega \in \Omega \,:\, U(\omega) \leq s\}) = \int_{-\infty}^{s} \mathcal{I}_{[0,1]} dt$ with the usual extensions to other Borel subsets of $\mathbb{R}$. To describe this situation, we just say that "$U \sim \mathcal{U}(0,1)$ is i.i.d. (independent and identically distributed)", meaning that U is furthermore independent of all other random variables and stochastic processes in the semantics. We denote the *characteristic function* of a set S by $\mathcal{I}_S$, defined by $\mathcal{I}_S(x) := 1$ if $x \in S$ and $\mathcal{I}_S(x) := 0$ if $x \notin S$.

Definition 5.3 (*Process semantics of SHPs*). The *semantics* of SHP α is defined by

$$[\![\alpha]\!] : (\Omega \to \mathbb{R}^d) \to ([0,\infty) \times \Omega \to \mathbb{R}^d); Z \mapsto [\![\alpha]\!]^Z = ([\![\alpha]\!]^Z_t)_{t\geq 0}$$
$$(\!|\alpha|\!) : (\Omega \to \mathbb{R}^d) \to (\Omega \to \mathbb{R} \cup \{\infty\}); Z \mapsto (\!|\alpha|\!)^Z$$

These functions are inductively defined for random variable $Z : (\Omega \to \mathbb{R}^d)$ by

(1) $[\![x_i := \theta]\!]^Z = \hat{Y}$ where $Y(\omega)_i = [\![\theta]\!]^{Z(\omega)}$ and $Y_j = Z_j$ for all $j \neq i$, and $(\!|x_i := \theta|\!)^Z = 0$.

(2) $[\![x_i := *]\!]^Z = \hat{U}$ where $U_j = Z_j$ for all $j \neq i$, and $U_i \sim \mathcal{U}(0,1)$ is i.i.d. and $\mathcal{F}_0$-measurable. Further, $(\!|x_i := *|\!)^Z = 0$.

(3) $[\![?\chi]\!]^Z = \hat{Z}$ on the event $\{Z \models \chi\}$ and $(\!|?\chi|\!)^Z = 0$ (on all events $\omega \in \Omega$). Note that $[\![?\chi]\!]^Z$ is not defined on the event $\{Z \not\models \chi\}$.

(4) $[\![dx = bdt + \sigma dW \,\&\, \chi]\!]^Z$ is the stochastic process $X : [0,\infty) \times \Omega \to \mathbb{R}^d$ that solves the (Itō) stochastic differential equation $dX_t = [\![b]\!]^{X_t} dt + [\![\sigma]\!]^{X_t} dB_t$ with $X_0 = Z$ on the event $\{Z \models \chi\}$, where B_t is a fresh e-dimensional Brownian motion if σ has e columns. We assume that Z is independent of the σ-algebra generated by $(B_t)_{t\geq 0}$. Further, $(\!|dx = bdt + \sigma dW \,\&\, \chi|\!)^Z = \inf\{t \geq 0 : X_t \notin \chi\}$. Note that X is not defined on the event $\{Z \not\models \chi\}$.

(5) $$[\![\lambda\alpha \oplus \nu\beta]\!]^Z = \mathcal{I}_{U\leq\lambda}[\![\alpha]\!]^Z + \mathcal{I}_{U>\lambda}[\![\beta]\!]^Z = \begin{cases} [\![\alpha]\!]^Z & \text{on the event } \{U \leq \lambda\} \\ [\![\beta]\!]^Z & \text{on the event } \{U > \lambda\} \end{cases}$$
$$(\!|\lambda\alpha \oplus \nu\beta|\!)^Z = \mathcal{I}_{U\leq\lambda}(\!|\alpha|\!)^Z + \mathcal{I}_{U>\lambda}(\!|\beta|\!)^Z$$
where $U \sim \mathcal{U}(0,1)$ is i.i.d. and $\mathcal{F}_0$-measurable.

(6) $$[\![\alpha;\beta]\!]^Z_t = \begin{cases} [\![\alpha]\!]^Z_t & \text{on the event } \{(\!|\alpha|\!)^Z > t\} \\ [\![\beta]\!]^{[\![\alpha]\!]^Z_{(\!|\alpha|\!)^Z}}_{t-(\!|\alpha|\!)^Z} & \text{on the event } \{(\!|\alpha|\!)^Z \leq t\} \end{cases}$$
$$(\!|\alpha;\beta|\!)^Z = (\!|\alpha|\!)^Z + (\!|\beta|\!)^{[\![\alpha]\!]^Z_{(\!|\alpha|\!)^Z}}$$

(7) $[\![\alpha^*]\!]^Z_t = [\![\alpha^n]\!]^Z_t$ on the event $\{(\!|\alpha^n|\!)^Z > t\}$
$(\!|\alpha^*|\!)^Z = \lim_{n\to\infty} (\!|\alpha^n|\!)^Z$
where $\alpha^0 \equiv ?true$, $\alpha^1 \equiv \alpha$, and $\alpha^{n+1} \equiv \alpha;\alpha^n$.

For case 7, note that $(\!|\alpha^n|\!)^Z$ is monotone in n, hence the limit $(\!|\alpha^*|\!)^Z$ exists and is finite if the sequence is bounded. The limit is ∞ otherwise. Note that $[\![\alpha^*]\!]^Z_t$ is independent of the choice of n on the event $\{(\!|\alpha^n|\!)^Z > t\}$ (but not necessarily independent of n on the event $\{(\!|\alpha^n|\!)^Z \geq t\}$, because α might start with a jump after α^n). Observe that $[\![\alpha^*]\!]^Z_t$ is not defined on the event $\{\forall n\, (\!|\alpha^n|\!)^Z \leq t\}$, which happens, e.g., for Zeno executions violating divergence of time. It would still be possible to give a semantics in this case, e.g., at $t = (\!|\alpha^n|\!)^Z$, but we do not gain much from introducing those technicalities. Note that the choice of inequalities in cases 6 and 7 is important to obtain a càdlàg process.

In the semantics of $[\![\alpha]\!]^Z$, time is allowed to end. We explicitly consider $[\![\alpha]\!]^Z_t$ as not defined for a realization ω if a part of this process is not defined, because of failed tests in α. The process is explicitly not defined when $(\!|\alpha|\!)^Z < t$. Explicitly being not defined can be viewed as being in a special absorbing state that can never be left again, as in killed processes. The stochastic process $[\![\alpha]\!]^Z$ is only intended to be used until time $(\!|\alpha|\!)^Z$. We stop using $[\![\alpha]\!]^Z$ after time $(\!|\alpha|\!)^Z$, which is a random variable.

Example 5.4 (*Jumping rotational Brownian motion*). Consider a simple example illustrating how SHPs combine discrete and stochastic continuous dynamics to form stochastic hybrid systems. Consider the following SHP:

$$?x^2+y^2\leq\frac{1}{3};\ x := \frac{x}{2};\ dx = \frac{-x}{2}dt - ydW, dy = \frac{-y}{2}dt + xdW \,\&\, \chi$$

It starts with a test $?x^2+y^2\leq\frac{1}{3}$ checking whether the state variables x, y are in a ball of radius $\sqrt{\frac{1}{3}}$ around 0. Only if it succeeds can the SHP continue. Then, the SHP performs an instantaneous discrete jump reducing the value of x to $x := \frac{x}{2}$. Finally, the SHP follows a stochastic differential equation for rotational Brownian motion restricted to the evolution domain region χ that we define as $\chi \equiv x^2 + y^2 < 9$. This gives rotational dynamics from the diffusion terms (compare Example 3.20) and an exponentially contracting drift.

5.3. SdL Formulas

Formulas of stochastic differential dynamic logic are built out of SdL function terms.

Definition 5.5 (SdL *term*). *Function terms* of *stochastic differential dynamic logic* SdL are formed by the grammar (F is a primitive measurable function definable in first-order real arithmetic, e.g., the characteristic function $\mathcal{I}_S$ of a set S definable in first-order real arithmetic, B is a boolean combination of such characteristic functions using operators $\wedge, \vee, \neg, \rightarrow$ from Figure 18, λ, ν are rational numbers):

$$f, g \ ::= \ F \mid \lambda f + \nu g \mid Bf \mid \langle\alpha\rangle f$$

One typical choice for a primitive measurable function F is the characteristic function $\mathcal{I}_S$ of a set S definable in first-order real arithmetic, which is then measurable [Tarski 1951]. The SdL term $\lambda f + \nu g$ is a *linear combination* of SdL terms. The SdL term Bf is a boolean product with a boolean combination B of characteristic functions of measurable sets. SdL term $\langle\alpha\rangle f$ represents the supremal value of f along the process belonging to α. The syntactic abbreviations in Figure 18 can be useful, especially for convenient operators on boolean combinations of characteristic functions. Formulas of SdL are simple, because SdL function terms are powerful. SdL *formulas* express equational and inequality relations between SdL function terms f, g.

$$\begin{aligned} 0 &\equiv \mathcal{I}_\emptyset \\ 1 &\equiv \mathcal{I}_{\mathbb{R}^d} \\ \neg f &\equiv 1 - f \\ A \wedge B &\equiv AB \\ A \vee B &\equiv A + B - AB \\ A \rightarrow B &\equiv 1 - A + AB \\ [\alpha]f &\equiv \neg\langle\alpha\rangle\neg f \end{aligned}$$

Fig. 18. Common SdL and SHP abbreviations

Definition 5.6 (SdL *formula*). The *formulas of* SdL are defined by the following grammar (where f, g are SdL function terms):

$$\phi \ ::= \ f \leq g \mid f = g$$

The semantics of classical logics maps an interpretation to a truth-value. This does not work for stochastic logic, because the state evolution of SHPs contained in SdL formulas is stochastic, not deterministic. Instead, the semantics of an SdL function term is a generator for a random variable.

Definition 5.7 (SdL *semantics*). The *semantics* $[\![f]\!]$ of SdL function term f is a function $[\![f]\!] : (\Omega \to \mathbb{R}^d) \to (\Omega \to \mathbb{R})$ that maps any $\mathbb{R}^d$-valued random variable Z describing the current state to a random variable $[\![f]\!]^Z$. It is defined inductively by

(1) $[\![F]\!]^Z = F^\ell(Z)$, i.e., $[\![F]\!]^Z(\omega) = F^\ell(Z(\omega))$ where F^ℓ is the function denoted by function symbol F
(2) $[\![\lambda f + \nu g]\!]^Z = \lambda[\![f]\!]^Z + \nu[\![g]\!]^Z$
(3) $[\![Bf]\!]^Z = [\![B]\!]^Z \cdot [\![f]\!]^Z$, i.e., pathwise multiplication $[\![Bf]\!]^Z(\omega) = [\![B]\!]^Z(\omega) \cdot [\![f]\!]^Z(\omega)$
(4) $[\![\langle\alpha\rangle f]\!]^Z = \sup\{[\![f]\!]^{[\![\alpha]\!]^Z_t} \ : \ 0 \leq t \leq (\!|\alpha|\!)^Z\}$

When Z is not defined (results from a failed test), then $[\![f]\!]^Z$ is not defined.

If f is a characteristic function of a measurable set, then $[\![\langle\alpha\rangle f]\!]^Z$ corresponds to a random variable that reflects the supremal f value that α can reach at least once during its evolution until stopping time $(\!|\alpha|\!)^Z$ when starting in a state corresponding to random variable Z. Then $P([\![\langle\alpha\rangle f]\!]^Z = 1)$ is the probability with which α reaches f at least once. Expected values of SdL terms are well-defined, e.g., $E([\![\langle\alpha\rangle(f+g)]\!]^Z)$ is an expected value, given Z. This includes the special case where Z is a deterministic state $Z(\omega) := x$ for all $\omega \in \Omega$.

We say that SdL formula $f \leq g$ is *valid*, written $\vDash f \leq g$, if for all $\mathbb{R}^d$-valued random variables Z:

$$[\![f]\!]^Z \leq [\![g]\!]^Z, \text{ i.e., } ([\![f]\!]^Z)(\omega) \leq ([\![g]\!]^Z)(\omega) \text{ for all } \omega \in \Omega$$

Validity of SdL formula $f = g$ is defined accordingly, hence, $\vDash f = g$ iff $\vDash f \leq g$ and $\vDash g \leq f$. As consequence relation on SdL formulas, we use the *(global) consequence* that we define as follows (similarly when some of the formulas are $f_i = g_i$):

$$\begin{aligned} f_1 \leq g_1, \ldots, f_n \leq g_n &\vDash f \leq g \\ \text{iff } \vDash f_1 \leq g_1, \ldots, &\vDash f_n \leq g_n \text{ implies } \vDash f \leq g \end{aligned}$$

The (global) consequence $f_1 \leq g_1, \ldots, f_n \leq g_n \vDash f \leq g$ holds *pathwise* if it holds for each $\omega \in \Omega$.

Example 5.8 (*Jumping rotational Brownian motion*). Let α denote the SHP from Example 5.4. The SdL term $\langle\alpha\rangle x^2+y^2$ represents the supremal value of the square, x^2+y^2, of the Euclidean norm along the process α. The semantics $[\![\langle\alpha\rangle x^2+y^2]\!]^Z$ of this SdL term starting from an initial random variable Z is a random variable. The SdL formula $\langle\alpha\rangle x^2+y^2 \geq 1$ expresses that this supremal value is greater or equal 1. The following states that this happens (SdL formula $\langle\alpha\rangle x^2+y^2 \geq 1$ holds) with probability at most $\frac{1}{3}$:

$$P(\langle\alpha\rangle x^2+y^2 \geq 1) \leq \frac{1}{3} \tag{19}$$

5.4. Measurability

The semantics of SHPs and SdL needs to satisfy measurability properties to make sure that probabilities are well-defined and probabilistic questions become meaningful.

A *Markov time* (a.k.a. stopping time) is a non-negative random variable τ such that $\{\tau \leq t\} \in \mathcal{F}_t$ for all t (i.e., it does not depend on the future). For a Markov time τ and a stochastic process X_t, the following process that is stuck after time τ is called *stopped process* X^τ

$$X^\tau_t := X_{t \sqcap \tau} = \begin{cases} X_t & \text{if } t < \tau \\ X_\tau & \text{if } t \geq \tau \end{cases} \quad \text{where} \quad t \sqcap \tau := \min\{t, \tau\}$$

A class $\mathcal{C}$ of processes is *stable under stopping* if $X \in \mathcal{C}$ implies $X^\tau \in \mathcal{C}$ for every Markov time τ. Right continuous adapted processes, and processes satisfying the strong Markov property are stable under stopping [Dynkin 1965, Theorem 10.2].

We have proved that the SHP semantics is well-defined. This includes that the natural stopping times $(\!|\alpha|\!)^Z$ are Markov times so that it is meaningful to stop process $[\![\alpha]\!]^Z$ at $(\!|\alpha|\!)^Z$ and so that useful properties of $[\![\alpha]\!]^Z$ inherit to the stopped process $[\![\alpha]\!]^Z_{t \sqcap (\!|\alpha|\!)^Z}$. Furthermore, we have shown that the process $[\![\alpha]\!]^Z$ is adapted (does not look into the future) and is a.s. càdlàg (right-continuous and has left limits), which is important to define a semantics for SdL formulas.

THEOREM 5.9 (ADAPTIVE CÀDLÀG PROCESS WITH MARKOV TIMES [PLATZER 2011B]). *For each SHP α and any $\mathbb{R}^d$-valued random variable Z, $[\![\alpha]\!]^Z$ is a.s. a càdlàg process and adapted (to the completed filtration $(\mathcal{F}_t)_{t\geq 0}$ generated by Z and the constituent Brownian motions $(B_s)_{s\leq t}$ and uniform processes U) and $(\!|\alpha|\!)^Z$ is a Markov time (for $(\mathcal{F}_t)_{t\geq 0}$). In particular, the end value $[\![\alpha]\!]^Z_{(\!|\alpha|\!)^Z}$ is again measurable.*

Note in particular, that the event $\{(\!|\alpha^n|\!)^Z \geq t\}$ is $\mathcal{F}_t$-measurable, thus, by [Karatzas and Shreve 1991, Prop 1.2.3], the event $\{(\!|\alpha^n|\!)^Z > t\}$ in case 7 of the semantics of SHPs is $\mathcal{F}_t$-measurable. As a corollary to Theorem 5.9, $[\![\alpha]\!]^Z$ is progressively measurable [Karatzas and Shreve 1991, Prop 1.1.13], which implies that the stopped processes are measurable.

We have proved that the Sd$\mathcal{L}$ semantics is well-defined and that $[\![f]\!]^Z$ is, indeed, a random variable, i.e., measurable. Without this, probabilistic questions about the value of Sd$\mathcal{L}$ terms and formulas would not be well-defined, because they are not measurable with respect to probability space $(\Omega, \mathcal{F}, P)$.

THEOREM 5.10 (MEASURABILITY [PLATZER 2011B]). *For any $\mathbb{R}^d$-valued random variable Z, the semantics $[\![f]\!]^Z$ of function term f is a random variable (i.e., $\mathcal{F}$-measurable).*

In particular, for each Borel-measurable set S, the probability $P([\![f]\!]^Z \in S)$ is well-defined so that probabilistic questions have a well-defined answer. Note that well-definedness of case 4 of the semantics of Sd$\mathcal{L}$ uses Theorem 5.9.

5.5. Axioms

Stochastic hybrid systems are a very expressive model and can even represent systems with a very complicated behavior. Just like the systems they model, however, they still expose their rich semantical complexity. In order to make this rich semantical complexity and the deep theory behind the stochastic dynamics accessible in a form that is amenable to a computational approach, we have introduced a proof calculus for Sd$\mathcal{L}$ [Platzer 2011b]. Just like the proof calculi for d$\mathcal{L}$ and Qd$\mathcal{L}$, the proof rules for Sd$\mathcal{L}$ are syntactic so that only the justification of their soundness requires the deep theory of stochastic processes and stochastic hybrid systems, their use does not. This makes it possible to computerize stochastic calculus in the form of syntactic Sd$\mathcal{L}$ proofs.

We show axioms and proof rules that can be used to prove Sd$\mathcal{L}$ formulas in Figure 19. First we present proof rules that are sound pathwise, i.e., satisfy the global consequence relation pathwise for each $\omega \in \Omega$. Axiom $\langle := \rangle$ corresponds to Hoare's assignment rule for deterministic discrete assignment dynamics. Axiom $\langle ? \rangle$ is the test axiom from dynamic logic, just using the product notation of Sd$\mathcal{L}$ instead of conjunctions.

Axiom $\langle ; \rangle$ is the Sd$\mathcal{L}$ form of the sequential composition axiom. By $\sqcup$ we denote the binary maximum operator. Operator $\sqcup$ coincides with $\vee$ for values in $\{0,1\}$, e.g., for Sd$\mathcal{L}$ terms built using operators $\wedge, \vee, \neg, \langle\alpha\rangle$ from characteristic functions. As a supremum, $\langle\alpha\rangle B$ only takes on values $\{0,1\}$ if B does. Note that the two sides of $\langle ; \rangle$ are generally not equal, because α has to run to completion before β starts in $\langle\alpha;\beta\rangle f$, but α can stop early in $\langle\alpha\rangle(f \sqcup \langle\beta\rangle f)$ and β can then start already.

Axiom $\langle\rangle\lambda$ and $\langle\rangle+$ represent scalar multiplication and (sub)linearity for scalars λ, ν. The two sides of axiom $\langle\rangle+$ are not equal if the suprema $\langle\alpha\rangle f$ and $\langle\alpha\rangle g$ are at different times. Axiom $\mathcal{I}$ expresses idempotence and range constraints on boolean combinations (Figure 18) of characteristic functions.

$$\langle:=\rangle \quad \langle x:=\theta\rangle f(x) \;=\; f(\theta)$$

$$\langle ?\rangle \quad \langle ?\chi\rangle f \;=\; \chi f$$

$$\langle ;\rangle \quad \langle \alpha;\beta\rangle f \;\leq\; \langle\alpha\rangle(f \sqcup \langle\beta\rangle f)$$

$$\langle\rangle\lambda \quad \langle\alpha\rangle(\lambda f) \;=\; \lambda\langle\alpha\rangle f$$

$$\langle\rangle+ \quad \langle\alpha\rangle(\lambda f+\nu g) \;\leq\; \lambda\langle\alpha\rangle f+\nu\langle\alpha\rangle g$$

$$\mathcal{I} \quad 0\leq B=BB\leq 1 \qquad (B \text{ boolean from characteristic functions})$$

$$\mathrm{R} \quad f\leq g \;\vDash\; \langle\alpha\rangle f\leq\langle\alpha\rangle g$$

$$\mathrm{DW} \quad \overline{\chi}\to f\leq\lambda \;\vDash\; \langle dx=b\,dt+\sigma\,dW \,\&\, \chi\rangle f\leq\lambda \qquad (\lambda\in\mathbb{Q})$$

$$\mathrm{ind} \quad \langle\alpha\rangle g\leq g \;\vDash\; \langle\alpha^*\rangle g\leq g$$

Fig. 19. Pathwise proof rules for Sd$\mathcal{L}$

Rule R is the generalization rule of regular modal logic C expressing monotonicity. It is the counterpart of a corresponding regular generalization rule that is easily derivable from K and G in d$\mathcal{L}$. Rule DW is the weakening rule for differential equations, yet generalized to stochastic differential equations. Note that formula $\overline{\chi}\to f\leq\lambda$ in DW is equivalent to $\overline{\chi}f\leq\overline{\chi}\lambda$ but easier to read. If f is continuous, rule DW is sound when replacing the topological closure $\overline{\chi}$ (which is computable by quantifier elimination) by χ, because the inequality is a weak inequality. Rule ind is an induction rule. It corresponds to rule ind of d$\mathcal{L}$ from p. 24.

Other rules are derivable from Figure 19:

$$(pos) \quad 0\leq f \;\vDash\; 0\leq\langle\alpha\rangle f$$

$$(\langle;\rangle_2) \quad \langle\alpha;\beta\rangle f \;\leq\; \langle\alpha\rangle\langle\beta\rangle f \qquad (\vDash f\leq\langle\beta\rangle f)$$

$$(\langle;\rangle_3) \quad \langle\alpha;\beta\rangle f \;\leq\; \langle\alpha\rangle(f+\langle\beta\rangle f) \qquad (0\leq f)$$

Rule pos expresses positivity (if f is positive then $\langle\alpha\rangle f$ is) and is derivable[7] from $\langle\rangle\lambda$ and R. The simpler sequential composition principle $\langle;\rangle_2$ is derivable[8] from $\langle;\rangle$ and R. Its assumption $\vDash f\leq\langle\beta\rangle f$ holds in particular if β is continuous at 0 a.s.. A sufficient condition for SHP β to be a.s. continuous at 0 is that, on all paths, the first atomic operation that is not a test is a stochastic differential equation, not an assignment or random assignment. Formula $\langle;\rangle_3$ is derivable[9] from $\langle;\rangle$, pos, and R. Consequently the operator $\sqcup$ can either be added into the language or approximated conservatively by $+$ as in $\langle;\rangle_3$.

Not all reasoning for stochastic hybrid systems is sound pathwise. We have introduced other Sd$\mathcal{L}$ axioms and proof rules that do not hold pathwise, but are still sound in distribution. Rule $\langle\oplus\rangle$ relates probabilities of linear combinations (alias probabilistic choices):

[7] By R, $0\leq f\vDash\langle\alpha\rangle 0\leq\langle\alpha\rangle f$. By $\langle\rangle\lambda$, $\langle\alpha\rangle 0=\langle\alpha\rangle(0*0)=0\langle\alpha\rangle 0=0$.
[8] By $\mathbb{R}$, $f\leq\langle\beta\rangle f$ entails $f\sqcup\langle\beta\rangle f=\langle\beta\rangle f$, from which R derives $\langle\alpha\rangle(f\sqcup\langle\beta\rangle f)\leq\langle\alpha\rangle\langle\beta\rangle f$. Thus, $\langle\alpha;\beta\rangle f\leq\langle\alpha\rangle\langle\beta\rangle f$ by $\langle;\rangle$.
[9] By pos, $0\leq f$ entails $0\leq\langle\beta\rangle f$. Thus, R and the semantics of $\sqcup$, yield $\langle\alpha\rangle(f\sqcup\langle\beta\rangle f)\leq\langle\alpha\rangle(f+\langle\beta\rangle f)$ by $0\leq f$, which derives $\langle;\rangle_3$ together with $\langle\alpha;\beta\rangle f\leq\langle\alpha\rangle(f\sqcup\langle\beta\rangle f)$, which holds by $\langle;\rangle$.

$$(\langle\oplus\rangle)\ P(\langle\lambda\alpha\oplus\nu\beta\rangle f\in S)\ =\ \lambda P(\langle\alpha\rangle f\in S)+\nu P(\langle\beta\rangle f\in S)$$

How to prove properties about a random assignment $x_i := *$ depends on the distribution that we use for the random assignment. For a uniform distribution in [0,1], e.g., we obtain the following proof rule that is sound in distribution:

$$(\langle*\rangle)\quad P(\langle x_i:=*\rangle f\in S)\ =\ \textstyle\int_0^1 \mathcal{I}_{\langle x_i:=r\rangle f\in S}dr$$

The integrand is measurable for measurable S by Theorem 5.10. The rule is applicable when f has been simplified enough using other proof rules such that the integral can be computed after using $\langle:=\rangle$ to simplify the integrand.

The Sd$\mathcal{L}$ rules and axioms generalize to probabilistic assumptions by the rule of partition, i.e., using

$$P(C)=P(C|A)P(A)+P(C|\neg A)P(\neg A)$$

to consider the case where assumption A holds separately from the case where A does not hold (giving less information about conclusion C, but the probability can by bounded).

5.6. Soundness

The most critical result about the Sd$\mathcal{L}$ axioms and proof rules is that they are sound, so that their simple syntactic reasoning always gives correct results about the semantical behavior of stochastic hybrid systems. This result splits into Sd$\mathcal{L}$ axioms and proof rules that are sound pathwise (i.e., the global consequence relation between premises and conclusion holds pathwise) and those that are sound in distribution (i.e., the indicated probability relations hold). All proof rules that are sound pathwise are sound in distribution but not vice versa.

THEOREM 5.11 (PATHWISE SOUNDNESS OF Sd$\mathcal{L}$ [PLATZER 2011B]). *The* Sd$\mathcal{L}$ *axioms and proof rules in Figure 19 are globally sound pathwise.*

THEOREM 5.12 (SOUNDNESS IN DISTRIBUTION OF Sd$\mathcal{L}$ [PLATZER 2011B]). *The* Sd$\mathcal{L}$ *proof rules* $\langle\oplus\rangle$ *and* $\langle*\rangle$ *are sound in distribution.*

5.7. Stochastic Differential Invariants

Proving properties of differential equations is very challenging. Differential invariants (Sect. 3.6) are the primary proof technique for more complicated differential equations that have no computable solutions. Differential invariants are the "only" choice for differential equations with disturbances [Platzer 2010a; Platzer 2010b], because those do not have a single unique solution but depend on input functions. In that respect, stochastic differential equations are like differential equations with disturbances, because both have noise terms in the right hand sides, which make solutions non-unique, depending on input functions and noise. The difference is that stochastic differential equations have a stochastic model of the noise, whereas classical differential equations with disturbances have a nondeterministic model. Solving stochastic differential equations is, for the most part, limited to sampling, and even that is difficult [Øksendal 2007; Kloeden and Platen 2010].

For Sd$\mathcal{L}$, we lift the ideas behind differential invariants for differential equations to *stochastic differential invariants* for stochastic differential equations [Platzer 2011b]. One critical element behind differential invariants is Lemma 3.19, which relates (computable) syntactic derivatives to (semantic) analytic differentiation. The stochastic

analogue of analytic differentiation are infinitesimal generators, the analogue of syntactic derivatives are differential generators.

Definition 5.13 (*Infinitesimal generator*). The *(infinitesimal) generator* of an a.s. right continuous strong Markov process (e.g., a solution from Def. 5.1) is the operator A that maps a function $f : \mathbb{R}^d \to \mathbb{R}$ to the function $Af : \mathbb{R}^d \to \mathbb{R}$ defined as

$$Af(x) := \lim_{t \searrow 0} \frac{E^x f(X_t) - f(x)}{t}$$

We say that Af is defined if this limit exists for all $x \in \mathbb{R}^d$. By Dynkin's formula [Dynkin 1965, p. 133], infinitesimal generators can be used to determine, without solving the stochastic differential equation, the expected value of a function when following the process until a Markov time.

THEOREM 5.14 (DYNKIN'S FORMULA [ØKSENDAL 2007, THEOREM 7.4.1]). *Let X_t an a.s. right continuous strong Markov process (e.g., a solution from Def. 5.1). If $f \in C^2(\mathbb{R}^d, \mathbb{R})$ has compact support and τ is a Markov time with $E^x\tau < \infty$, then*

$$E^x f(X_\tau) = f(x) + E^x \int_0^\tau Af(X_s)ds$$

Dynkin's formula is very useful, but only if we can compute the infinitesimal generator and its integral. The generator A is a stochastic expression in the sense that A is defined in terms of a limit of an expectation of a stochastic process. It has been shown, however, that, under fairly mild assumptions, it is equal to a deterministic expression of x called the differential generator.

THEOREM 5.15 (DIFFERENTIAL GENERATOR [ØKSENDAL 2007, THEOREM 7.3.3]). *Consider a solution X_t of a stochastic differential equation from Def. 5.1. If $\mathsf{f} \in C^2(\mathbb{R}^d, \mathbb{R})$ is compactly supported, then $A\mathsf{f}$ is defined and equal to the* differential generator $L\mathsf{f}$ *of* f*:*

$$L\mathsf{f}(x) := \sum_i b_i(x)\frac{\partial \mathsf{f}}{\partial x_i}(x) + \frac{1}{2}\sum_{i,j}(\sigma(x)\sigma(x)^T)_{i,j}\frac{\partial^2 \mathsf{f}}{\partial x_i \partial x_j}(x)$$

Observe that this deterministic differential generator can be computed syntactically, yet is still used for a stochastic process in Theorem 5.14. We turn this principle into a fully syntactic proof rule for stochastic differential invariants.

THEOREM 5.16 (SOUNDNESS OF STOCHASTIC DIFFERENTIAL INVARIANTS). *If function $\mathsf{f} \in C^2(\mathbb{R}^d, \mathbb{R})$ has compact support on χ (which holds for all $\mathsf{f} \in C^2(\mathbb{R}^d, \mathbb{R})$ if χ represents a bounded set), then the proof rule $\langle' \rangle$ is sound for $\lambda > 0, p \geq 0$*

$$(\langle' \rangle) \quad \frac{\langle\alpha\rangle(\chi \to \mathsf{f}) \leq \lambda p \quad \chi \to \mathsf{f} \geq 0 \quad \chi \to L\mathsf{f} \leq 0}{P(\langle\alpha\rangle\langle dx = bdt + \sigma dW \,\&\, \chi\rangle \mathsf{f} \geq \lambda) \leq p}$$

See [Platzer 2011b] for a proof of Theorem 5.16. The implications in the premises of $\langle' \rangle$ can be understood like that in DW. Let χ be given by first-order real arithmetic formulas. If f is polynomial and, thus, $\mathsf{f} \in C^2(\mathbb{R}^d, \mathbb{R})$, then the second and third premise of $\langle' \rangle$ are in first-order real arithmetic, hence decidable.

Example 5.17 (*Jumping rotational Brownian motion*).
With $\chi \equiv x^2 + y^2 < 9$, the probabilistic statement about the Sd$\mathcal{L}$ formula (19) from Example 5.8 can be proved easily in the Sd$\mathcal{L}$ calculus. The first step is to use $\langle;\rangle_2$ as

follows:

$$P\Big(\Big\langle ?x^2+y^2\leq\frac{1}{3};\ x:=\frac{x}{2};\ dx=\frac{-x}{2}dt-ydW, dy=\frac{-y}{2}dt+xdW\ \&\ \chi\Big\rangle x^2+y^2\geq 1\Big)$$
$$\overset{\langle;\rangle_2}{\leq} P\Big(\Big\langle ?x^2+y^2\leq\frac{1}{3};\ x:=\frac{x}{2}\Big\rangle\Big\langle dx=\frac{-x}{2}dt-ydW, dy=\frac{-y}{2}dt+xdW\ \&\ \chi\Big\rangle x^2+y^2\geq 1\Big)\leq\frac{1}{3}$$

The last inequality ($\leq\frac{1}{3}$) is by rule $\langle'\rangle$. The second premise of $\langle'\rangle$ is proved by $\mathsf{f}\equiv x^2+y^2\geq 0$. The third premise is proved as follows. The stochastic differential equation has the form

$$d\begin{pmatrix}x\\y\end{pmatrix}=bdt+\sigma dW=\begin{pmatrix}-\frac{x}{2}\\-\frac{y}{2}\end{pmatrix}dt+\begin{pmatrix}-y\\x\end{pmatrix}dW$$

Thus, in order to determine $L\mathsf{f}$, we compute

$$\sigma\sigma^T=\begin{pmatrix}-y\\x\end{pmatrix}(-y\ \ x)=\begin{pmatrix}y^2&-xy\\-xy&x^2\end{pmatrix}$$

and obtain by Theorem 5.15:

$$\begin{aligned}L\mathsf{f}=&-\frac{x}{2}\frac{\partial\mathsf{f}}{\partial x}-\frac{y}{2}\frac{\partial\mathsf{f}}{\partial y}+\frac{1}{2}\left(y^2\frac{\partial^2\mathsf{f}}{\partial x^2}-2xy\frac{\partial^2\mathsf{f}}{\partial x\partial y}+x^2\frac{\partial^2\mathsf{f}}{\partial y^2}\right)\\=&-\frac{x}{2}2x-\frac{y}{2}2y+\frac{1}{2}\left(2y^2-0xy+2x^2\right)\leq 0\end{aligned}$$

The first premise of $\langle'\rangle$ can be proved in the Sd$\mathcal{L}$ calculus by the following chain of inequalities, which are justified by Sd$\mathcal{L}$ axioms and arithmetic as indicated:

$$\begin{aligned}&\langle ?x^2+y^2\leq\frac{1}{3};\ x:=\frac{x}{2}\rangle(\chi\to\mathsf{f})\\&\overset{\langle;\rangle}{\leq}\langle ?x^2+y^2\leq\frac{1}{3}\rangle\Big((\chi\to\mathsf{f})\sqcup\langle x:=\frac{x}{2}\rangle(\chi\to\mathsf{f})\Big)\\&\overset{\langle:=\rangle}{=}\langle ?x^2+y^2\leq\frac{1}{3}\rangle\Big((\chi\to\mathsf{f})\sqcup(\langle x:=\frac{x}{2}\rangle\chi\to\Big(\frac{x}{2}\Big)^2+y^2)\Big)\\&\overset{\mathbb{R}}{=}\langle ?x^2+y^2\leq\frac{1}{3}\rangle(\chi\to\mathsf{f})\\&\overset{\langle?\rangle}{=}\Big(x^2+y^2\leq\frac{1}{3}\Big)(\chi\to\mathsf{f})\\&\overset{\mathbb{R}}{=}\Big(x^2+y^2\leq\frac{1}{3}\Big)(x^2+y^2<9\to x^2+y^2)\leq\frac{1}{3}\end{aligned}$$

Together with $\chi\to\mathsf{f}\geq 0$ and $\chi\to L\mathsf{f}\leq 0$, this inequality entails Sd$\mathcal{L}$ formula (19) by Sd$\mathcal{L}$ proof rule $\langle'\rangle$ and $\langle;\rangle_2$.

Observe that the d$\mathcal{L}$ and Qd$\mathcal{L}$ calculus use equivalences and implications of d$\mathcal{L}$ and Qd$\mathcal{L}$ axioms together with proof rules to prove d$\mathcal{L}$ and Qd$\mathcal{L}$ formulas. The Sd$\mathcal{L}$ calculus uses equations and inequalities of Sd$\mathcal{L}$ axioms together with Sd$\mathcal{L}$ proof rules to prove Sd$\mathcal{L}$ formulas. Some of this reasoning is in the scope of a probability operator (e.g., when using axiom $\langle\oplus\rangle$), other inequalities and equations can be determined without a probability operator.

6. RELATED WORK

Since dynamical systems, hybrid systems, and their extensions are very active areas of research, a comprehensive overview of all results is impossible. In this survey, we

focus on logic and proofs for dynamical systems. For more details on hybrid systems, we refer to the literature [Henzinger 1996; Alur 2011]. For background on classical logic and proving, we refer to the literature [Hughes and Cresswell 1996; Fitting 1996; Harel et al. 2000].

Model checking and reachability analysis have been used successfully for hybrid systems. They work by state space exploration and use various abstractions or approximations [Henzinger 1996; Henzinger et al. 1992; Alur et al. 1995; Henzinger 1996; Fränzle 1999; Anai and Weispfenning 2001; Clarke et al. 2003b; Tiwari 2003; Mysore et al. 2005; Alur et al. 2006b; Alur et al. 2006a], including numerical approximations [Chutinan and Krogh 2003; Asarin et al. 2003]. Lafferriere et al. [Lafferriere et al. 1999; Lafferriere et al. 2000; Lafferriere et al. 2001] have shown that finite-state bisimulations, which generally do not exist for hybrid systems [Henzinger 1996], still work for o-minimal hybrid automata and classes of linear dynamics with a homogeneous eigenstructure, provided the discrete and continuous dynamics are completely decoupled. Surveys of model checking techniques for hybrid systems appeared elsewhere [Henzinger 1996; Doyen et al. 2012].

Discretizations have been used for linear systems [Guernic and Girard 2009], to obtain abstractions of fragments of hybrid systems [Alur et al. 2000; Alur et al. 2006b; Tiwari 2008], and to approximate nonlinear systems by hybrid systems [Henzinger et al. 1998] or by piecewise linear dynamics [Asarin et al. 2003]. Constraint-based verification approaches [Prajna et al. 2007; Sankaranarayanan et al. 2008; Gulwani and Tiwari 2008] have been considered, which are related to differential invariants. Verification tools are based on logic and proofs [Platzer 2008a; Platzer and Quesel 2008; Platzer 2012b], polyhedral reachability analysis [Henzinger et al. 1997; Frehse 2008], reachability analysis with support functions [Guernic and Girard 2009; Frehse et al. 2011], interval-constraint propagation [Ratschan and She 2007], and numerical PDE solving [Mitchell and Templeton 2005].

Many languages have been proposed for modeling hybrid systems, including extended duration calculus [Zhou et al. 1992], hybrid automata [Henzinger 1996], hybrid programs [Platzer 2007b; Platzer 2008a], guarded commands [Rönkkö et al. 2003], hybrid π-calculus [Rounds and Song 2003], and process algebra χ [van Beek et al. 2006].

Logic has been used successfully for real-time systems [Henzinger et al. 1992; Dutertre 1995; Schobbens et al. 2002; Zhou and Hansen 2004; Olderog and Dierks 2008] and for timed-automata based model checking [Alur and Dill 1994; Alur 1999; Comon and Jurski 1999; Larsen et al. 1997; Baier et al. 2008]. More details about real-time systems can be found in books [Olderog and Dierks 2008; Baier et al. 2008].

The importance of understanding dynamic / reconfigurable distributed hybrid systems was recognized in modeling languages SHIFT [Deshpande et al. 1996] and R-Charon [Kratz et al. 2006] for simulation and compilation [Deshpande et al. 1996] or semantical considerations [Kratz et al. 2006]. For distributed hybrid systems, even giving a formal semantics is very challenging [Chaochen et al. 1995; Rounds 2004; Kratz et al. 2006; van Beek et al. 2006]. Random simulation has been proposed for general dynamical systems [Meseguer and Sharykin 2006].

Discrete programs with random number generators have been studied in the literature, including [Kozen 1981; Feldman and Harel 1984; Kozen 1985; McIver and Morgan 2004]. Probabilities and logic have been considered in many contexts, e.g., [Richardson and Domingos 2006]. Discrete probabilistic systems and finite Markov chains, have been studied using probabilistic model checking [Baier et al. 1997; Baier et al. 2003; Courcoubetis and Yannakakis 1995] and statistical model checking [Sen et al. 2005; Younes et al. 2006; Jha et al. 2009]. Extensions to probabilistic timed automata [Kwiatkowska et al. 2007] and probabilistic hybrid automata [Zuliani et al. 2010; Zhang et al. 2010] have been considered too.

For an overview of model checking techniques for various classes of stochastic hybrid systems, we refer to a survey [Cassandras and Lygeros 2006]. Most verification techniques for stochastic hybrid systems use discretizations, approximations, or assume discrete time and bounded horizon [Koutsoukos and Riley 2008; Cassandras and Lygeros 2006; Abate et al. 2008; Hu et al. 2000]. Barrier certificates have been extended to the stochastic case [Prajna et al. 2007].

The use of logic has been proposed for hybrid systems, e.g., in a propositional modal μ-calculus [Davoren and Nerode 2000] or in early work based on phase transition systems [Maler et al. 1991]. See [Davoren and Nerode 2000] for an excellent overview. We consider the first-order case, i.e., how to model and prove systems with concrete differential equations like $x' = v, v' = a$ and concrete control decisions like $a := -b$, instead of abstract propositional actions A, B, C of unknown effects that propositional modal μ-calculi consider [Pratt 1981; Davoren and Nerode 2000]. The use of theorem provers has been suggested in hybrid systems, including STeP [Manna and Sipma 1998; Kesten et al. 2000] and PVS [Ábrahám-Mumm et al. 2001]. Their working principles are different from what we show here. They separate hybridness from the logic and proof by compiling a given global system invariant for a hybrid automaton into a single verification condition expressing that the invariant is preserved under all transitions of the hybrid automaton.

In our approach, we, instead, take logic and hybridness at face value by developing and studying logics for hybrid systems, which directly integrate the logic and the hybrid dynamics (or extensions) within a single language. That makes it easier to identify the core logical reasoning principles and transform formulas soundly in an entirely local way even for more general properties than invariance checking. This view enables the study of logically more foundational questions, including completeness, deductive power, and relationships of differential invariants and differential cuts. Benefits for automation of proofs and for computing invariants and differential invariants have been discussed elsewhere [Platzer and Clarke 2009a; Platzer 2010b].

7. SUMMARY AND OUTLOOK

We have surveyed logics of dynamical systems, including hybrid systems, distributed hybrid systems, and stochastic hybrid systems. The logic of discrete dynamical systems and the logic of continuous dynamical systems are fragments of the logic of hybrid systems. We have surveyed differential dynamic logic (d$\mathcal{L}$) for hybrid systems, quantified differential dynamic logic (Qd$\mathcal{L}$) for distributed hybrid systems, and stochastic differential dynamic logic (Sd$\mathcal{L}$) for stochastic hybrid systems. We have recalled dynamical system models, dynamic logics, their semantics, their axiomatizations, and proof calculi for each of those dynamical systems. We have surveyed important theoretical results, including soundness and completeness, and results about the relative deductive power of differential cuts and of differential auxiliaries. The differential dynamic logics and their induction techniques for differential equations, which are captured in various forms of differential invariants and differential variants, have been instrumental in proving properties for more advanced dynamical systems. While the use of the theorem provers implementing differential dynamic logics is beyond the scope of this article, we have given references to more information, including a number of applications and case studies.

Not all important results about logics of dynamical systems and about differential dynamic logics are included in this survey. We still hope to have given the reader a good overview of logics for dynamical systems, and point out relationships and similarities among the techniques. The reader should note that, for space reasons, not all important members of the family of differential dynamic logics have been presented in

this article. Prominent cases that are missing from this survey include differential-algebraic dynamic logic (DAL) for hybrid systems with differential-algebraic constraints modeled in differential-algebraic program and the temporal extension of differential temporal dynamic logic (dTL).

The results summarized in this article demonstrate that logic is a powerful tool, not just for studying discrete phenomena, but also continuous phenomena, infinite-dimensional phenomena, and stochastic phenomena. These logics set a strong logical foundation for dynamical systems, including logical foundations for cyber-physical systems. Such stable foundations for the relatively young area of logic of dynamical systems make it a very promising direction for future research, including theoretical, practical, and applied research. Given the tremendous progress that logic for programs has made since its conception, we expect to see no less from the area of logics for dynamical systems.

ACKNOWLEDGMENTS

I am grateful to Jan-David Quesel for indispensable help with implementing KeYmaera and to David Renshaw for implementing KeYmaeraD. My thanks go to Sicun Gao, David Harel, David Henriques, Oded Maler, João Martins, Sayan Mitra, Vaughan Pratt, and Sriram Sankaranarayanan for feedback on this article.

REFERENCES

ABATE, A., PRANDINI, M., LYGEROS, J., AND SASTRY, S. 2008. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica 44,* 11, 2724–2734.

ÁBRAHÁM-MUMM, E., STEFFEN, M., AND HANNEMANN, U. 2001. Verification of hybrid systems: Formalization and proof rules in PVS. In *ICECCS*, S. F. Andler and J. Offutt, Eds. IEEE Computer Society, Los Alamitos, 48–57.

ALUR, R. 1999. Timed automata. In *CAV*, N. Halbwachs and D. Peled, Eds. LNCS Series, vol. 1633. Springer, 8–22.

ALUR, R. 2011. Formal verification of hybrid systems. See Chakraborty et al. [2011], 273–278.

ALUR, R., COURCOUBETIS, C., HALBWACHS, N., HENZINGER, T. A., HO, P.-H., NICOLLIN, X., OLIVERO, A., SIFAKIS, J., AND YOVINE, S. 1995. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci. 138,* 1, 3–34.

ALUR, R., COURCOUBETIS, C., HENZINGER, T. A., AND HO, P.-H. 1992. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. See Grossman et al. [1993], 209–229.

ALUR, R., DANG, T., AND IVANCIC, F. 2006a. Counterexample-guided predicate abstraction of hybrid systems. *Theor. Comput. Sci. 354,* 2, 250–271.

ALUR, R., DANG, T., AND IVANCIC, F. 2006b. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trans. Embedded Comput. Syst. 5,* 1, 152–199.

ALUR, R. AND DILL, D. L. 1994. A theory of timed automata. *Theor. Comput. Sci. 126,* 2, 183–235.

ALUR, R., HENZINGER, T., LAFFERRIERE, G., AND PAPPAS, G. J. 2000. Discrete abstractions of hybrid systems. *Proc. IEEE 88,* 7, 971–984.

ALUR, R., HENZINGER, T. A., AND HO, P.-H. 1996. Automatic symbolic verification of embedded systems. *IEEE T. Software Eng. 22,* 3, 181–201.

ALUR, R., HENZINGER, T. A., AND SONTAG, E. D., Eds. 1996. *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop, October 22-25, 1995, Rutgers University, New Brunswick, NJ, USA*. LNCS Series, vol. 1066. Springer.

ANAI, H. AND WEISPFENNING, V. 2001. Reach set computations using real quantifier elimination. In *HSCC*, M. D. D. Benedetto and A. L. Sangiovanni-Vincentelli, Eds. LNCS Series, vol. 2034. Springer, 63–76.

APT, K. R., DE BOER, F. S., AND OLDEROG, E.-R. 2010. *Verification of Sequential and Concurrent Programs* 3rd Ed. Springer.

ARÉCHIGA, N., LOOS, S. M., PLATZER, A., AND KROGH, B. H. 2012. Using theorem provers to guarantee closed-loop system properties. In *ACC*, D. Tilbury, Ed.

ASARIN, E., DANG, T., AND GIRARD, A. 2003. Reachability analysis of nonlinear systems using conservative approximation. See Maler and Pnueli [2003], 20–35.

ASARIN, E. AND MALER, O. 1998. Achilles and the tortoise climbing up the arithmetical hierarchy. *J. Comput. Syst. Sci. 57,* 3, 389–398.

ATTIE, P. C. AND LYNCH, N. A. 2001. Dynamic input/output automata: A formal model for dynamic systems. In *CONCUR*, K. G. Larsen and M. Nielsen, Eds. LNCS Series, vol. 2154. Springer, 137–151.

AUBIN, J.-P. AND CELLINA, A. 1984. *Differential Inclusions: Set-Valued Maps and Viability Theory*. Springer.

BAIER, C., CLARKE, E. M., HARTONAS-GARMHAUSEN, V., KWIATKOWSKA, M. Z., AND RYAN, M. 1997. Symbolic model checking for probabilistic processes. In *ICALP*, P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, Eds. LNCS Series, vol. 1256. Springer, 430–440.

BAIER, C., HAVERKORT, B. R., HERMANNS, H., AND KATOEN, J.-P. 2003. Model-checking algorithms for continuous-time markov chains. *IEEE Trans. Software Eng. 29,* 6, 524–541.

BAIER, C., KATOEN, J.-P., AND LARSEN, K. G. 2008. *Principles of Model Checking*. MIT Press.

BECKERT, B., HÄHNLE, R., AND SCHMITT, P. H., Eds. 2007. *Verification of Object-Oriented Software: The KeY Approach*. LNCS Series, vol. 4334. Springer.

BECKERT, B. AND PLATZER, A. 2006. Dynamic logic with non-rigid functions: A basis for object-oriented program verification. In *IJCAR*, U. Furbach and N. Shankar, Eds. LNCS Series, vol. 4130. Springer, 266–280.

BOOLOS, G. 1984. Don't eliminate cut. *Journal of Philosophical Logic*.

BOYD, S. AND VANDENBERGHE, L. 2004. *Convex Optimization*. Cambridge University Press.

BRADLEY, A. R. AND MANNA, Z. 2007. *The Calculus of Computation: Decision Procedures with Applications to Verification*. Springer.

BRANICKY, M. S. 1995. General hybrid dynamical systems: Modeling, analysis, and control. See Alur et al. [1996], 186–200.

BRANICKY, M. S., BORKAR, V. S., AND MITTER, S. K. 1998. A unified framework for hybrid control: Model and optimal control theory. *IEEE T. Automat. Contr. 43,* 1, 31–45.

BUCHBERGER, B. 1965. An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. Ph.D. thesis, University of Innsbruck.

BUJORIANU, M. L. AND LYGEROS, J. 2006. Towards a general theory of stochastic hybrid systems. In *Stochastic Hybrid Systems: Theory and Safety Critical Applications*, H. A. P. Blom and J. Lygeros, Eds. Lecture Notes Contr. Inf. Series, vol. 337. Springer, 3–30.

CARNAP, R. 1946. Modalities and quantification. *J. Symb. Log. 11,* 2, 33–64.

CASSANDRAS, C. G. AND LYGEROS, J., Eds. 2006. *Stochastic Hybrid Systems*. CRC.

CASSEZ, F. AND LARSEN, K. G. 2000. The impressive power of stopwatches. In *CONCUR*. 138–152.

CHAKRABORTY, S., JERRAYA, A., BARUAH, S. K., AND FISCHMEISTER, S., Eds. 2011. *Proceedings of the 11th International Conference on Embedded Software, EMSOFT 2011, part of the Seventh Embedded Systems Week, ESWeek 2011, Taipei, Taiwan, October 9-14, 2011*. ACM.

CHAOCHEN, Z., JI, W., AND RAVN, A. P. 1995. A formal description of hybrid systems. See Alur et al. [1996], 511–530.

CHUTINAN, A. AND KROGH, B. H. 2003. Computational techniques for hybrid system verification. *IEEE T. Automat. Contr. 48,* 1, 64–75.

CLARKE, E., FEHNKER, A., HAN, Z., KROGH, B., STURSBERG, O., AND THEOBALD, M. 2003a. Verification of hybrid systems based on counterexample-guided abstraction refinement. In *TACAS 2003*, H. Garavel and J. Hatcliff, Eds. Number 2619 in LNCS. 192–207.

CLARKE, E. M., FEHNKER, A., HAN, Z., KROGH, B. H., OUAKNINE, J., STURSBERG, O., AND THEOBALD, M. 2003b. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. J. Found. Comput. Sci. 14,* 4, 583–604.

CLARKE, E. M., GRUMBERG, O., AND PELED, D. A. 1999. *Model Checking*. MIT Press, Cambridge, MA, USA.

COLLINS, G. E. 1975. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, H. Barkhage, Ed. LNCS Series, vol. 33. Springer, 134–183.

COLLINS, G. E. AND HONG, H. 1991. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput. 12,* 3, 299–328.

COLLINS, P. 2007. Optimal semicomputable approximations to reachable and invariant sets. *Theory Comput. Syst. 41,* 1, 33–48.

COMON, H. AND JURSKI, Y. 1999. Timed automata and the theory of real numbers. In *CONCUR*, J. C. M. Baeten and S. Mauw, Eds. LNCS Series, vol. 1664. Springer, 242–257.

COOK, S. A. 1978. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput. 7,* 1, 70–90.

COURCOUBETIS, C. AND YANNAKAKIS, M. 1995. The complexity of probabilistic verification. *J. ACM 42,* 4, 857–907.

DAMM, W., HUNGAR, H., AND OLDEROG, E.-R. 2006. Verification of cooperating traffic agents. *International Journal of Control 79,* 5, 395–421.

DAVIS, M. H. A. 1984. Piecewise-deterministic Markov processes: A general class of non-diffusion stochastic models. *Journal of the Royal Statistical Society. Series B 46,* 3, 358–388.

DAVOREN, J. M. AND NERODE, A. 2000. Logics for hybrid systems. *IEEE 88,* 7, 985–1010.

DESHPANDE, A., GÖLLÜ, A., AND VARAIYA, P. 1996. SHIFT: A formalism and a programming language for dynamic networks of hybrid automata. In *Hybrid Systems*, P. J. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, Eds. LNCS Series, vol. 1273. Springer, 113–133.

DOYEN, L., PAPPAS, G. J., AND PLATZER, A. 2012. Verification of hybrid systems. In *Handbook of Model Checking*, E. M. Clarke, T. H. Henzinger, and H. Veith, Eds. Springer, Chapter 28. To appear.

DUTERTRE, B. 1995. Complete proof systems for first order interval temporal logic. In *LICS*. IEEE Computer Society, 36–43.

DYNKIN, E. B. 1965. *Markov Processes*. Springer.

EGERSTEDT, M., JOHANSSON, K. H., SASTRY, S., AND LYGEROS, J. 1999. On the regularization of Zeno hybrid automata. *Systems and Control Letters 38*, 141–150.

ETESSAMI, K. AND RAJAMANI, S. K., Eds. 2005. *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*. LNCS Series, vol. 3576. Springer.

FELDMAN, Y. A. AND HAREL, D. 1984. A probabilistic dynamic logic. *J. Comput. Syst. Sci. 28,* 2, 193–215.

FISCHER, M. J. AND LADNER, R. E. 1979. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci. 18,* 2, 194–211.

FITTING, M. 1996. *First-Order Logic and Automated Theorem Proving* 2nd Ed. Springer, New York.

FITTING, M. AND MENDELSOHN, R. L. 1999. *First-Order Modal Logic*. Kluwer, Norwell, MA, USA.

FLOYD, R. W. 1967. Assigning meanings to programs. In *Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics*, J. T. Schwartz, Ed. Vol. 19. AMS, Providence, 19–32.

FOURLAS, G. K., KYRIAKOPOULOS, K. J., AND VOURNAS, C. D. 2004. Hybrid systems modeling for power systems. *Circuits and Systems Magazine, IEEE 4,* 3, 16 – 23.

FRÄNZLE, M. 1999. Analysis of hybrid systems: An ounce of realism can save an infinity of states. In *CSL*, J. Flum and M. Rodríguez-Artalejo, Eds. LNCS Series, vol. 1683. Springer, 126–140.

FRÄNZLE, M., TEIGE, T., AND EGGERS, A. 2010. Engineering constraint solvers for automatic analysis of probabilistic hybrid automata. *J. Log. Algebr. Program. 79,* 7, 436–466.

FREHSE, G. 2008. PHAVer: algorithmic verification of hybrid systems past HyTech. *STTT 10,* 3, 263–279.

FREHSE, G., GUERNIC, C. L., DONZÉ, A., COTTON, S., RAY, R., LEBELTEL, O., RIPADO, R., GIRARD, A., DANG, T., AND MALER, O. 2011. SpaceEx: Scalable verification of hybrid systems. See Gopalakrishnan and Qadeer [2011], 379–395.

GALOR, O. 2010. *Discrete Dynamical Systems*. Springer.

GENTZEN, G. 1935a. Untersuchungen über das logische Schließen. I. *Math. Zeit. 39,* 2, 176–210.

GENTZEN, G. 1935b. Untersuchungen über das logische Schließen. II. *Math. Zeit. 39,* 3, 405–431.

GHOSH, M. K., ARAPOSTATHIS, A., AND MARCUS, S. I. 1997. Ergodic control of switching diffusions. *SIAM J. Control Optim. 35,* 6, 1952–1988.

GILBERT, S., LYNCH, N., MITRA, S., AND NOLTE, T. 2009. Self-stabilizing robot formations over unreliable networks. *ACM Trans. Auton. Adapt. Syst. 4,* 3, 1–29.

GÖDEL, K. 1931. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Mon. hefte Math. Phys. 38*, 173–198.

GOPALAKRISHNAN, G. AND QADEER, S., Eds. 2011. *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. LNCS Series, vol. 6806. Springer.

GROSSMAN, R. L., NERODE, A., RAVN, A. P., AND RISCHEL, H., Eds. 1993. *Hybrid Systems*. LNCS Series, vol. 736. Springer.

GROSU, R., BATT, G., FENTON, F. H., GLIMM, J., GUERNIC, C. L., SMOLKA, S. A., AND BARTOCCI, E. 2011. From cardiac cells to genetic regulatory networks. See Gopalakrishnan and Qadeer [2011], 396–411.

GUERNIC, C. L. AND GIRARD, A. 2009. Reachability analysis of hybrid systems using support functions. In *CAV*, A. Bouajjani and O. Maler, Eds. LNCS Series, vol. 5643. Springer, 540–554.

GULWANI, S. AND TIWARI, A. 2008. Constraint-based approach for analysis of hybrid systems. See Gupta and Malik [2008], 190–203.

GUPTA, A. AND MALIK, S., Eds. 2008. *Computer Aided Verification, CAV 2008, Princeton, NJ, USA, Proceedings*. LNCS Series, vol. 5123. Springer.

HAREL, D. 1979. *First-Order Dynamic Logic*. Springer, New York.

HAREL, D., KOZEN, D., AND TIURYN, J. 2000. *Dynamic logic*. MIT Press, Cambridge.

HAREL, D., MEYER, A. R., AND PRATT, V. R. 1977. Computability and completeness in logics of programs (preliminary report). In *STOC*. ACM, 261–268.

HARRISON, J. 2007. Verifying nonlinear real formulas via sums of squares. In *TPHOLs*, K. Schneider and J. Brandt, Eds. LNCS Series, vol. 4732. Springer, 102–118.

HENZINGER, T. A. 1996. The theory of hybrid automata. In *LICS*. IEEE Computer Society, Los Alamitos, 278–292.

HENZINGER, T. A., HO, P.-H., AND WONG-TOI, H. 1997. HyTech: A model checker for hybrid systems. *STTT 1,* 1-2, 110–122.

HENZINGER, T. A., HO, P.-H., AND WONG-TOI, H. 1998. Algorithmic analysis of nonlinear hybrid systems. *IEEE T. Automat. Contr. 43*, 540–554.

HENZINGER, T. A., NICOLLIN, X., SIFAKIS, J., AND YOVINE, S. 1992. Symbolic model checking for real-time systems. In *LICS*. IEEE Computer Society, 394–406.

HESPANHA, J. P. AND TIWARI, A., Eds. 2006. *Hybrid Systems: Computation and Control, 9th International Workshop, HSCC 2006, Santa Barbara, CA, USA, March 29-31, 2006, Proceedings*. LNCS Series, vol. 3927. Springer.

HIRSCH, M. W., SMALE, S., AND DEVANEY, R. L. 2003. *Differential Equations, Dynamical Systems, and an Introduction to Chaos* 2 Ed. Academic Press.

HOARE, C. A. R. 1969. An axiomatic basis for computer programming. *Commun. ACM 12,* 10, 576–580.

HSU, A., ESKAFI, F., SACHS, S., AND VARAIYA, P. 1991. Design of platoon maneuver protocols for IVHS. PATH Research Report UCB-ITS-PRR-91-6, Institute of Transportation Studies, University of California, Berkeley.

HU, J., LYGEROS, J., AND SASTRY, S. 2000. Towards a theory of stochastic hybrid systems. In *HSCC*, N. A. Lynch and B. H. Krogh, Eds. LNCS Series, vol. 1790. Springer, 160–173.

HUGHES, G. E. AND CRESSWELL, M. J. 1996. *A New Introduction to Modal Logic*. Routledge.

ISTRAIL, S. 1989. An arithmetical hierarchy in propositional dynamic logic. *Inf. Comput. 81,* 3, 280–289.

JHA, S. K., CLARKE, E., LANGMEAD, C., LEGAY, A., PLATZER, A., AND ZULIANI, P. 2009. A Bayesian approach to model checking biological systems. In *CMSB*, P. Degano and R. Gorrieri, Eds. LNCS Series, vol. 5688. Springer, 218–234.

JOHANSSON, K. H. AND YI, W., Eds. 2010. *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, April 12-15, 2010*. ACM.

JOHNSON, T. T. AND MITRA, S. 2012. A small model theorem for rectangular hybrid automata networks. In *FORTE/FMOODS*, H. Giese and G. Rosu, Eds. LNCS. Springer. To appear.

KARATZAS, I. AND SHREVE, S. 1991. *Brownian Motion and Stochastic Calculus*. Graduate Texts in Mathematics. Springer.

KERKEZ, B., GLASER, S. D., DRACUP, J. A., AND BALES, R. C. 2010. A hybrid system model of seasonal snowpack water balance. See Johansson and Yi [2010], 171–180.

KESTEN, Y., MANNA, Z., AND PNUELI, A. 2000. Verification of clocked and hybrid systems. *Acta Inf. 36,* 11, 837–912.

KIM, B., AYOUB, A., SOKOLSKY, O., LEE, I., JONES, P. L., ZHANG, Y., AND JETLEY, R. P. 2011. Safety-assured development of the gpca infusion pump software. See Chakraborty et al. [2011], 155–164.

KLOEDEN, P. E. AND PLATEN, E. 2010. *Numerical Solution of Stochastic Differential Equations*. Springer, New York.

KOUTSOUKOS, X. D. AND RILEY, D. 2008. Computational methods for verification of stochastic hybrid systems. *IEEE T. Syst. Man, Cy. A 38,* 2, 385–396.

KOZEN, D. 1981. Semantics of probabilistic programs. *J. Comput. Syst. Sci. 22,* 3, 328–350.

KOZEN, D. 1985. A probabilistic PDL. *J. Comput. Syst. Sci. 30,* 2, 162–178.

KOZEN, D. 1997. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst. 19,* 3, 427–443.

KOZEN, D. AND PARIKH, R. 1981. An elementary proof of the completeness of PDL. *Theor. Comp. Sci. 14*, 113–118.

KRATZ, F., SOKOLSKY, O., PAPPAS, G. J., AND LEE, I. 2006. R-Charon, a modeling language for reconfigurable hybrid systems. See Hespanha and Tiwari [2006], 392–406.

KRIPKE, S. 1959. A completeness theorem in modal logic. *J. Symb. Log. 24,* 1, 1–14.

KRIPKE, S. A. 1963. Semantical considerations on modal logic. *Acta Philosophica Fennica 16*, 83–94.

KUNKEL, P. AND MEHRMANN, V. 2006. *Differential-Algebraic Equations: Analysis and Numerical Solution*. European Mathematical Society.

KWIATKOWSKA, M. Z., NORMAN, G., SPROSTON, J., AND WANG, F. 2007. Symbolic model checking for probabilistic timed automata. *Inf. Comput. 205,* 7, 1027–1077.

LAFFERRIERE, G., PAPPAS, G. J., AND SASTRY, S. 2000. O-minimal hybrid systems. *Mathematics of Control, Signals, and Systems 13,* 1, 1–21.

LAFFERRIERE, G., PAPPAS, G. J., AND YOVINE, S. 1999. A new class of decidable hybrid systems. In *HSCC*, F. W. Vaandrager and J. H. van Schuppen, Eds. LNCS Series, vol. 1569. Springer, 137–151.

LAFFERRIERE, G., PAPPAS, G. J., AND YOVINE, S. 2001. Symbolic reachability computation for families of linear vector fields. *J. Symb. Comput. 32,* 3, 231–253.

LARSEN, K. G., PETTERSSON, P., AND YI, W. 1997. UPPAAL in a nutshell. *STTT 1,* 1-2, 134–152.

LEIVANT, D. 2006. Matching explicit and modal reasoning about programs: A proof theoretic delineation of dynamic logic. In *LICS*. IEEE Computer Society, 157–168.

LICS 2012. *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, June 2528, 2012, Dubrovnik, Croatia*. IEEE Computer Society.

LIE, S. 1893. *Vorlesungen über continuierliche Gruppen mit geometrischen und anderen Anwendungen*. Teubner, Leipzig.

LOOS, S. M. AND PLATZER, A. 2011. Safe intersections: At the crossing of hybrid systems and verification. In *ITSC*, K. Yi, Ed. Springer, 1181–1186.

LOOS, S. M., PLATZER, A., AND NISTOR, L. 2011. Adaptive cruise control: Hybrid, distributed, and now formally verified. In *FM*, M. Butler and W. Schulte, Eds. LNCS Series, vol. 6664. Springer, 42–56.

LYNCH, N. 1996. *Distributed Algorithms*. Morgan Kaufmann.

MALER, O., MANNA, Z., AND PNUELI, A. 1991. From timed to hybrid systems. In *REX Workshop*, J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, Eds. LNCS Series, vol. 600. Springer, 447–484.

MALER, O. AND PNUELI, A., Eds. 2003. *Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3-5, 2003, Proceedings*. LNCS Series, vol. 2623. Springer.

MANNA, Z. AND SIPMA, H. 1998. Deductive verification of hybrid systems using STeP. In *HSCC*, T. A. Henzinger and S. Sastry, Eds. LNCS Series, vol. 1386. Springer, 305–318.

MCIVER, A. AND MORGAN, C. 2004. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer.

MESEGUER, J. AND SHARYKIN, R. 2006. Specification and analysis of distributed object-based stochastic hybrid systems. See Hespanha and Tiwari [2006], 460–475.

MEYER, A. R. AND PARIKH, R. 1981. Definability in dynamic logic. *J. Comput. Syst. Sci. 23,* 2, 279–298.

MITCHELL, I. M. AND TEMPLETON, J. A. 2005. A toolbox of Hamilton-Jacobi solvers for analysis of nondeterministic continuous and hybrid systems. In *HSCC*, M. Morari and L. Thiele, Eds. LNCS Series, vol. 3414. Springer, 480–494.

MITSCH, S., LOOS, S. M., AND PLATZER, A. 2012. Towards formal verification of freeway traffic control. In *ICCPS*, C. Lu, Ed. IEEE, 171–180.

MYSORE, V., PIAZZA, C., AND MISHRA, B. 2005. Algorithmic algebraic model checking II: Decidability of semi-algebraic model checking and its applications to systems biology. In *ATVA*, D. Peled and Y.-K. Tsay, Eds. LNCS Series, vol. 3707. Springer, 217–233.

NICOLLIN, X., OLIVERO, A., SIFAKIS, J., AND YOVINE, S. 1992. An approach to the description and analysis of hybrid systems. See Grossman et al. [1993], 149–178.

ØKSENDAL, B. 2007. *Stochastic Differential Equations: An Introduction with Applications*. Springer.

OLDEROG, E.-R. AND DIERKS, H. 2008. *Real-Time Systems: Formal Specification and Automatic Verification*. Cambridge Univ. Press.

PALLOTTINO, L., SCORDIO, V. G., FRAZZOLI, E., AND BICCHI, A. 2007. Decentralized cooperative policy for conflict resolution in multi-vehicle systems. *IEEE Trans. on Robotics 23,* 6, 1170–1183.

PARIKH, R. 1978. The completeness of propositional dynamic logic. In *MFCS*, J. Winkowski, Ed. LNCS Series, vol. 64. Springer, 403–415.

PARRILO, P. A. 2003. Semidefinite programming relaxations for semialgebraic problems. *Math. Program. 96,* 2, 293–320.

PELEG, D. 1987. Concurrent dynamic logic. *J. ACM 34,* 2, 450–479.

PERKO, L. 2006. *Differential equations and dynamical systems* 3 Ed. Springer, New York.

PIAZZA, C., ANTONIOTTI, M., MYSORE, V., POLICRITI, A., WINKLER, F., AND MISHRA, B. 2005. Algorithmic algebraic model checking I: Challenges from systems biology. See Etessami and Rajamani [2005], 5–19.

PLAKU, E., KAVRAKI, L. E., AND VARDI, M. Y. 2009. Hybrid systems: from verification to falsification by combining motion planning and discrete search. *Form. Methods Syst. Des. 34,* 2, 157–182.

PLATZER, A. 2007a. Combining deduction and algebraic constraints for hybrid system analysis. In *VERIFY'07 at CADE, Bremen, Germany*, B. Beckert, Ed. CEUR Workshop Proceedings Series, vol. 259. CEUR-WS.org, 164–178.

PLATZER, A. 2007b. Differential dynamic logic for verifying parametric hybrid systems. In *TABLEAUX*, N. Olivetti, Ed. LNCS Series, vol. 4548. Springer, 216–232.

PLATZER, A. 2007c. A temporal dynamic logic for verifying hybrid system invariants. In *LFCS*, S. N. Artëmov and A. Nerode, Eds. LNCS Series, vol. 4514. Springer, 457–471.

PLATZER, A. 2008a. Differential dynamic logic for hybrid systems. *J. Autom. Reas. 41,* 2, 143–189.

PLATZER, A. 2008b. Differential dynamic logics: Automated theorem proving for hybrid systems. Ph.D. thesis, Department of Computing Science, University of Oldenburg. Appeared with Springer.

PLATZER, A. 2010a. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput. 20,* 1, 309–352.

PLATZER, A. 2010b. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg.

PLATZER, A. 2010c. Quantified differential dynamic logic for distributed hybrid systems. In *CSL*, A. Dawar and H. Veith, Eds. LNCS Series, vol. 6247. Springer, 469–483.

PLATZER, A. 2011a. Quantified differential invariants. In *HSCC*, E. Frazzoli and R. Grosu, Eds. ACM, 63–72.

PLATZER, A. 2011b. Stochastic differential dynamic logic for stochastic hybrid programs. In *CADE*, N. Bjørner and V. Sofronie-Stokkermans, Eds. LNCS Series, vol. 6803. Springer, 431–445.

PLATZER, A. 2012a. A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems. *Logical Methods in Computer Science*. Special issue for selected papers from CSL'10.

PLATZER, A. 2012b. The complete proof theory of hybrid systems. See LICS [2012].

PLATZER, A. 2012c. Logics of dynamical systems (invited tutorial). See LICS [2012].

PLATZER, A. 2012d. The structure of differential invariants and differential cut elimination. *Logical Methods in Computer Science*. To appear.

PLATZER, A. AND CLARKE, E. M. 2007. The image computation problem in hybrid systems model checking. In *HSCC*, A. Bemporad, A. Bicchi, and G. Buttazzo, Eds. LNCS Series, vol. 4416. Springer, 473–486.

PLATZER, A. AND CLARKE, E. M. 2008. Computing differential invariants of hybrid systems as fixedpoints. See Gupta and Malik [2008], 176–189.

PLATZER, A. AND CLARKE, E. M. 2009a. Computing differential invariants of hybrid systems as fixedpoints. *Form. Methods Syst. Des. 35,* 1, 98–120. Special issue for selected papers from CAV'08.

PLATZER, A. AND CLARKE, E. M. 2009b. Formal verification of curved flight collision avoidance maneuvers: A case study. In *FM*, A. Cavalcanti and D. Dams, Eds. LNCS Series, vol. 5850. Springer, 547–562.

PLATZER, A. AND QUESEL, J.-D. 2008. KeYmaera: A hybrid theorem prover for hybrid systems. In *IJCAR*, A. Armando, P. Baumgartner, and G. Dowek, Eds. LNCS Series, vol. 5195. Springer, 171–178.

PLATZER, A. AND QUESEL, J.-D. 2009. European Train Control System: A case study in formal verification. In *ICFEM*, K. Breitman and A. Cavalcanti, Eds. LNCS Series, vol. 5885. Springer, 246–265.

PLATZER, A., QUESEL, J.-D., AND RÜMMER, P. 2009. Real world verification. In *CADE*, R. A. Schmidt, Ed. LNCS Series, vol. 5663. Springer, 485–501.

PRAJNA, S., JADBABAIE, A., AND PAPPAS, G. J. 2007. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE T. Automat. Contr. 52,* 8, 1415–1429.

PRATT, V. R. 1976. Semantical considerations on Floyd-Hoare logic. In *FOCS*. IEEE, 109–121.

PRATT, V. R. 1981. A decidable mu-calculus: Preliminary report. In *FOCS*. IEEE Computer Society, 421–427.

PROTTER, P. 2010. *Stochastic Integration and Differential Equations*. Springer.

RATSCHAN, S. AND SHE, Z. 2007. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *Trans. on Embedded Computing Sys. 6,* 1, 8.

REIF, W., SCHELLHORN, G., AND STENZEL, K. 1997. Proving system correctness with KIV 3.0. In *CADE*, W. McCune, Ed. LNCS Series, vol. 1249. Springer, 69–72.

RENSHAW, D. W., LOOS, S. M., AND PLATZER, A. 2011. Distributed theorem proving for distributed hybrid systems. In *ICFEM*, S. Qin and Z. Qiu, Eds. LNCS Series, vol. 6991. Springer, 356–371.

RICHARDSON, M. AND DOMINGOS, P. 2006. Markov logic networks. *Machine Learning 62,* 1-2, 107–136.

RILEY, D., KOUTSOUKOS, X., AND RILEY, K. 2010. Reachability analysis of stochastic hybrid systems: A biodiesel production system. *European Journal on Control 16,* 6, 609–623.

RÖNKKÖ, M., RAVN, A. P., AND SERE, K. 2003. Hybrid action systems. *Theor. Comput. Sci. 290,* 1, 937–973.

ROUNDS, W. C. 2004. A spatial logic for the hybrid π-calculus. In *HSCC*, R. Alur and G. J. Pappas, Eds. LNCS Series, vol. 2993. Springer, 508–522.

ROUNDS, W. C. AND SONG, H. 2003. The ϕ-calculus: A language for distributed control of reconfigurable embedded systems. In *HSCC*. LNCS Series, vol. 2623. 435–449.

RÜMMER, P. 2006. Sequential, parallel, and quantified updates of first-order structures. In *LPAR*, M. Hermann and A. Voronkov, Eds. LNCS Series, vol. 4246. Springer, 422–436.

SANKARANARAYANAN, S., SIPMA, H. B., AND MANNA, Z. 2008. Constructing invariants for hybrid systems. *Form. Methods Syst. Des. 32,* 1, 25–55.

SCHOBBENS, P.-Y., RASKIN, J.-F., AND HENZINGER, T. A. 2002. Axioms for real-time logics. *Theor. Comput. Sci. 274,* 1-2, 151–182.

SEGERBERG, K. 1977. A completeness theorem in the modal logic of programs. *Notices AMS 24*, 522.

SEN, K., VISWANATHAN, M., AND AGHA, G. 2005. On statistical model checking of stochastic systems. See Etessami and Rajamani [2005], 266–280.

STENGLE, G. 1973. A Nullstellensatz and a Positivstellensatz in semialgebraic geometry. *Math. Ann. 207,* 2, 87–97.

TARSKI, A. 1951. *A Decision Method for Elementary Algebra and Geometry* 2nd Ed. University of California Press, Berkeley.

TAVERNINI, L. 1987. Differential automata and their discrete simulators. *Non-Linear Anal. 11,* 6, 665–683.

TIWARI, A. 2003. Approximate reachability for linear systems. See Maler and Pnueli [2003], 514–525.

TIWARI, A. 2008. Abstractions for hybrid systems. *Form. Methods Syst. Des. 32,* 1, 57–83.

TIWARI, A. 2011. Logic in software, dynamical and biological systems. In *LICS*. IEEE Computer Society, 9–10.

TOMLIN, C., PAPPAS, G. J., AND SASTRY, S. 1998. Conflict resolution for air traffic management: a study in multi-agent hybrid systems. *IEEE T. Automat. Contr. 43,* 4, 509–521.

VAN BEEK, D. A., MAN, K. L., RENIERS, M. A., ROODA, J. E., AND SCHIFFELERS, R. R. H. 2006. Syntax and consistent equation semantics of hybrid Chi. *J. Log. Algebr. Program. 68,* 1-2, 129–210.

WALTER, W. 1998. *Ordinary Differential Equations*. Springer.

WEISPFENNING, V. 1997. Quantifier elimination for real algebra — the quadratic case and beyond. *Appl. Algebra Eng. Commun. Comput. 8,* 2, 85–101.

YOUNES, H. L. S., KWIATKOWSKA, M. Z., NORMAN, G., AND PARKER, D. 2006. Numerical vs. statistical probabilistic model checking. *STTT 8,* 3, 216–228.

ZHANG, L., SHE, Z., RATSCHAN, S., HERMANNS, H., AND HAHN, E. M. 2010. Safety verification for probabilistic hybrid systems. In *CAV*, T. Touili, B. Cook, and P. Jackson, Eds. LNCS Series, vol. 6174. Springer, 196–211.

ZHOU, C. AND HANSEN, M. R. 2004. *Duration Calculus: A Formal Approach to Real-Time Systems*. Springer.

ZHOU, C., RAVN, A. P., AND HANSEN, M. R. 1992. An extended duration calculus for hybrid real-time systems. See Grossman et al. [1993], 36–59.

ZULIANI, P., PLATZER, A., AND CLARKE, E. M. 2010. Bayesian statistical model checking with application to Simulink/Stateflow verification. See Johansson and Yi [2010], 243–252.

Received May 2012; revised Month YYYY; accepted Month YYYY