 2012 27th Annual ACM/IEEE Symposium on Logic in Computer Science

Logics of Dynamical Systems

(Invited Paper)

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

***Abstract*—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 in 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.**

This is a brief survey of *differential dynamic logic* for specifying and verifying properties of hybrid systems. We explain hybrid system models, differential dynamic logic, its semantics, and its axiomatization for proving logical formulas about hybrid systems. We study *differential invariants*, i.e., induction principles for differential equations. We briefly survey theoretical results, including soundness and completeness and deductive power. Differential dynamic logic has been implemented in automatic and interactive theorem provers and has been used successfully to verify safety-critical applications in automotive, aviation, railway, robotics, and analogue electrical circuits.

***Index Terms*—logic of dynamical systems, dynamic logic, differential dynamic logic, hybrid systems, axiomatization, deduction**

I. Introduction

Dynamical systems study the mathematics of change [44]. 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, aircraft, robots, and power plants are prototypical examples. But dynamical systems are more general and can also describe and analyze chemical processes, biological systems, medical models, 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, continuous dynamical systems described by differential equations [44], and hybrid dynamical systems alias hybrid systems combining discrete and continuous dynamics [2], [9], [18], [30], [39], [47]–[50], [54].

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.

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. In this tutorial, we survey differential dynamic logic (dL) [45], [47]–[50], [54] for studying properties of the behavior of dynamical systems and proof approaches for proving those properties deductively. Dynamic logic [64] has been developed and used very successfully for conventional discrete programs, both for theoretical [28], [29], [43], [72] and practical purposes [8], [28], [67]. 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 exciting and highly nontrivial theoretical results [45], [47]–[55], while, at the same time, enabling the practical verification of complex applications across different fields [4], [37], [38], [41], [48], [50], [59], [61], [68] and inspiring algorithmic approaches based on these logics [48], [50], [57], [58], [60], [62], [68].

We see a number of advantages of the approach we focus on here, which make it attractive for research and applications, including 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 [16], [56]. In logic, *soundness* is easier to achieve, because we just check a small number of elementary proof rules for soundness once. Everything that can be derived from those simple rules, no matter how complicated, is 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. II-D and [54]). More generally, theoretical questions and logically foundational questions, including relative completeness [47], [53], [54] and relative deductive power [49], [55], 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 can improve scalability [38].

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 [49], an induction principle for differential equations, then differential cuts [49], [55], a logical cut principle for differential equations, and finally differential auxiliaries [55]. Differential refinement and differential transformation rules are further extensions [49], [50], but beyond the scope of this article. Temporal logic extensions [46], [50] and extensions to differential-algebraic hybrid systems [49], [50] 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. For space reasons, extensions to quantified differential dynamic logic for distributed hybrid systems [51], [53], and stochastic differential dynamic logic for stochastic hybrid systems [52] are beyond the scope of this brief survey as well.

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 co-semidecidable [6], [12], [30]. Consequently, quite unlike in finite-state systems and timed automata [7], [15], 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 [14] for counterexample-guided abstraction refinement techniques). Some nontrivial applications [50], [57]–[59], [61] 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 [45], [47]–[50], [54], [55]. We provide a very incomplete overview of the approach of logic of dynamical systems here. It is, by no means, possible to handle all material comprehensively in this brief survey. A more comprehensive source on logic of hybrid systems is a book [50] and subsequent extensions [54], [55].

II. Differential Dynamic Logic for Hybrid Systems

In this section, we study *differential dynamic logic* $\mathsf{d}\mathcal{L}$ [47], [54], the *logic of hybrid systems*, i.e., systems with interacting discrete and continuous dynamics.

Hybrid systems [2], [9], [10], [18], [30], [47]–[50], [54] 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* $\mathsf{d}\mathcal{L}$ [45], [47], [50], [54]. The logic $\mathsf{d}\mathcal{L}$ is based on first-order modal logic [11], [34] and dynamic logic [28], [64] 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 $\mathsf{d}\mathcal{L}$ 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 $\mathsf{d}\mathcal{L}$ 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 $\mathsf{d}\mathcal{L}$ provides for modeling hybrid systems (Sect. II-A). Then we explain the logical formulas that $\mathsf{d}\mathcal{L}$ provides for specification and verification purposes (Sect. II-B). We show reasoning principles, axioms, and proof rules for verifying $\mathsf{d}\mathcal{L}$ formulas (Sect. II-C). We subsequently show soundness and relative completeness theorems (Sect. II-D) and investigate stronger proof rules for differential equations (Sect. II-E–II-H). Finally, we briefly discuss an implementation in the theorem prover KeYmaera and applications (Sect. II-I).

A. Regular Hybrid Programs

Differential dynamic logic uses (regular) *hybrid programs* (HP) [45], [47], [50], [54] 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 [36]. 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 [50] for details.

Definition 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 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 ($\geq$0).

These operations can define all classical WHILE programming constructs and all hybrid systems [50]. 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. Other program constructs can be defined easily [50].

[1]The test $?\chi$ means "if χ then *skip* else *abort*". Our results generalize to rich-test $\mathsf{d}\mathcal{L}$, where $?\chi$ is a HP for any $\mathsf{d}\mathcal{L}$ formula χ (Sect. II-B).

HPs have a compositional semantics. We define their semantics by a reachability relation and refer to previous work for their trace semantics [46], [50]. 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 $[\![\theta]\!]_\nu$. We write $\nu \models \chi$ iff first-order formula χ is true in state ν (also defined in Sect. II-B).

Definition 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 [47]
- $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
- $\rho(\alpha; \beta) = \rho(\beta) \circ \rho(\alpha)$
  $= \{(\nu, \omega) : (\nu, \mu) \in \rho(\alpha), (\mu, \omega) \in \rho(\beta)\}$
- $\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n)$ with $\alpha^{n+1} \equiv \alpha^n; \alpha$ and $\alpha^0 \equiv ?true$.

We refer to our book [50] 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 1 (Single car)**.** As an example, consider a simple car control scenario. We denote the position of a car by x, its velocity by v, and its acceleration by a. From Newton's laws of mechanics, we obtain a simple kinematic model for the longitudinal motion of the car on a straight road, which can be described by the differential equation $x' = v, v' = a$. That is, the time-derivative of position is velocity ($x' = v$) and 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:

$$\big(((?\chi; a := A) \cup a := -b);\ x' = v, v' = a \,\&\, v \geq 0\big)^* \quad (1)$$

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 (1) 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; see [50] for better models.

HPs are a program notation for hybrid systems. Hybrid automata [30] 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. The car system in (1) can be represented by the hybrid automaton in Fig. 1. Hybrid automata can be represented as

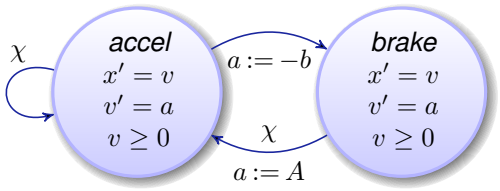

Fig. 1. Hybrid automaton for a simple car

HPs [50] just like finite automata can be implemented in classical WHILE programs.

To avoid technicalities, we consider only polynomial differential equations here and refer to previous work [49], [50] for how to handle hybrid systems with more general differential equations, including differential equations with fractions, differential inequalities [75], differential-algebraic equations, 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 [49], [50]. 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 [46], [50]. We refer to [50] for details.

B. dL Formulas

Differential dynamic logic dL [45], [47], [50], [54] is a dynamic logic [64] for hybrid systems. It combines first-order real arithmetic [73] with first-order modal logic [11], [34] 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 ($[\alpha]$) or some ($\langle\alpha\rangle$) state reachable by following HP α.

Definition 3 (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 $\rightarrow, \leftrightarrow$.

Definition 4 (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 [21], [33], 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 2 (Single car)**.** 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.

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 [50], [57], [58].

C. 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. II-D, 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. II-I).

Our axiomatization of dL is shown in Fig. 2. To highlight the logical essentials, we use our axiomatization from our recent result [54] that is simplified compared to our earlier work [47], which was tuned for automation. The axiomatization we use here is closer to that of Pratt's dynamic logic for conventional discrete programs [29], [64]. 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 [73] 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 Fig. 2. 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 Fig. 2 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 [50] 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Fig. 2. Differential dynamic logic axiomatization

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 [75, Theorem 10.VI]) solution of the symbolic initial-value problem $y'(t)=\theta, y(0)=x$. Given such a solution $y(\cdot)$, continuous evolution along that differential equation can be replaced by a discrete assignment $x:=y(t)$ with an additional quantifier for the evolution time t. It goes without saying that variables like t are fresh in Fig. 2. Notice that conventional initial-value problems are numerical with concrete numbers $x\in\mathbb{R}^n$ as initial values, not symbols x [75]. 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 dL axioms is most noticeable in axiom $[\cup]$. The equivalence axioms of dL 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 dL 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 [33]). This is also an advantage for computing invariants [50], [57], [58].

Sequential compositions are proven using nested modalities in axiom $[;]$. From right to left: If, after all α-runs, all β-runs lead to states satisfying ϕ (i.e., $[\alpha][\beta]\phi$ holds), then also all runs of the sequential composition $\alpha;\beta$ lead to states satisfying ϕ (i.e., $[\alpha;\beta]\phi$ holds). The converse implication uses the fact that if after all α-run all β-runs lead to ϕ (i.e., $[\alpha][\beta]\phi$), then all runs of $\alpha;\beta$ lead to ϕ (that is, $[\alpha;\beta]\phi$), because the runs of $\alpha;\beta$ are exactly those that first do any α-run, followed by any β-run. Again, it is crucial that dL is a full logic that considers reachability statements as modal operators, which can be nested, for then both sides in $[;]$ are dL formulas again (unlike in Hoare logic [33], 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 that need unbounded repetitions of loops.

Axiom K is the modal modus ponens from modal logic [34]. It expresses that, if an implication $\phi\rightarrow\psi$ holds after all runs of α (i.e., $[\alpha](\phi\rightarrow\psi)$) 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., $[\alpha^*](\phi\rightarrow[\alpha]\phi)$), 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.

Axiom B is the Barcan formula of first-order modal logic, characterizing anti-monotonic domains [34]. 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[2] [34, BFC p. 245] and we also call it B.

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 [34]. 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.

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 Fig. 2 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. 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 Fig. 3 for an illustration.

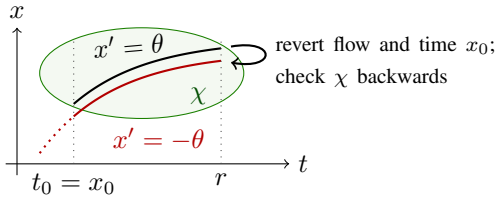

Fig. 3. "There and back again" axiom [&] checks evolution domain along backwards flow over time

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.

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 of the Hilbert calculus in Fig. 2 for automatic verification of hybrid systems, including trains, cars, and aircraft; see Sect. II-I. 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 [50].

Uses of real arithmetic, which, we denote by $\mathbb{R}$, are decidable by quantifier elimination in real-closed fields [73].

Example 3 (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 (1):

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

Formula (2) 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 (2) is valid by applying dL axioms to it. The following dL proof reveals that (2) is valid if $v^2 \leq 2b(m-x)$ holds, see Fig. 4.

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

Fig. 4. A dL proof for safe braking

[2]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.

D. Soundness and Completeness

The dL calculus is *sound* [47], [54], that is, every formula that is provable using the dL axioms and proof rules is valid, i.e., true in all states. That is, for all dL formulas ϕ:

$$\vdash \phi \text{ implies } \models \phi \tag{3}$$

Soundness should be *sine qua non* for formal verification, but, for fundamental reasons [16], [56], is so complex for hybrid systems that it is sometimes inadvertently forsaken. In logic, we ensure soundness just by checking locally once for each axiom and proof rule. Thus, no matter how complicated a proof, the proven dL 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 dL proof calculus *complete*, i.e., can it prove all that is true? That is, does the converse of (3) 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 [47]. In logic, the suitability of an axiomatization can still be established by showing completeness relative to a fragment [17], [29]. 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, also tells us how subproblems are related computationally. It tells us whether one subproblem dominates the others. Standard relative completeness [17], [29], 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 [73].

We have shown that both the original dL sequent calculus [47] and the Hilbert-type calculus in Fig. 2 [54] 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 1 (Relative completeness of dL [47], [54])**.** *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:*

$$\models \phi \quad \textit{iff} \quad \text{Taut}_{FOD} \vdash \phi$$

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 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 is exactly how dL proofs can scale to interesting systems in practice. Theorem 1 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 1 proves (constructively) that every true property of hybrid systems can be proved in the dL 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 dL relative to the discrete fragment of dL, we refer to recent work [54]. That proof is again a constructive reduction, proving that hybrid dynamics, continuous dynamics, and discrete dynamics are proof-theoretically equivalently reducible in the dL 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:

"*hybrid* = *continuous* = *discrete*"

A discussion of this fundamental result about the nature of hybridness is beyond the scope of this paper; we refer to previous work [54].

E. Differential Invariants

The dL axiomatization in Fig. 2 is sound and complete relative to FOD. But Fig. 2 only has a very simple proof rule for differential equations ($[']$) based on computing a solution of the differential equation. For proving more complicated differential equations by induction, dL provides *differential invariants* and *differential variants* [49], which have been introduced in 2008 [49] and later refined to a procedure that computes differential invariants in a fixed-point loop [57], [58]. All premier proof principles for discrete loops are based on some form of induction. Theorem 1 and its discrete converse [54] 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* ($F'^{\theta}_{x'}$, defined in [49]) 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.

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 left side of Fig. 5 for an illustration. Hence, if we start

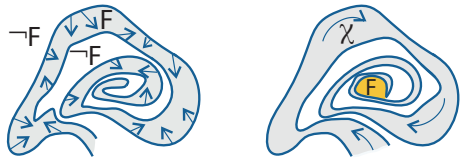

Fig. 5. Differential invariant F for safety and differential variant for progress

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 $F'^{\theta}_{x'}$.

More advanced uses of differential invariants can be found in previous work [49], [50], [55], [57], [59], [61]. 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 [50], [59], advanced safety, reactivity and controllability properties of train control systems with disturbance and PI controllers [50], [61], and properties of electrical circuits [50]. Differential invariants are also the proof technique of choice for differential inequalities, differential-algebraic equations, and differential equations with disturbances [49], [50].

We refer to previous work [49], [55] for details on the structure of differential invariants and a complete investigation of the relative deductive power of several classes of differential invariants; see Fig. 6 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 $\wr$.

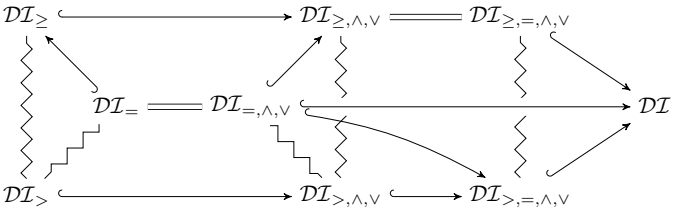

Fig. 6. Differential invariance chart

We also refer to previous work [49], [50] 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.

F. Differential Variants

Differential variants [49] 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 previous work [49].

G. Differential Cuts

Differential cuts [49] are another fundamental proof principle for differential equations. They can be used to strengthen assumptions in a sound way:

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

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. 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.

Differential cuts do not only help in practice, but are a fundamental proof principle. We refuted the *differential cut elimination hypothesis* [49]. 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 [24], standard logical cuts do not change the deductive power and can be eliminated. Unlike standard cuts, however, differential cuts actually increase the deductive power [55]. 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 2 (Differential cut power [55])**.** *The deductive power with differential cuts (rule DC) exceeds the deductive power without differential cuts.*

We refer to previous work [55] for details on the differential cut elimination hypothesis [49], the proof of its refutation [55], and a complete investigation of the relative deductive power of several classes of differential invariants.

H. Differential Auxiliaries

Furthermore, the addition of auxiliary differential variables increases the deductive power.

Theorem 3 (Auxiliary differential variable power [55])**.** *The deductive power with auxiliary differential variables exceeds the deductive power without auxiliary differential variables even in the presence of differential cuts.*

I. Implementation and Applications

Differential dynamic logic [45], [47], [54] and its proof calculus [45], [47], including differential invariants, differential variants, and differential cuts [49] have been implemented in the automatic and interactive theorem prover KeYmaera [60],[3] which is based on KeY [8]. The name KeYmaera is a homophone to Chimæra, the hybrid animal from ancient Greek mythology. KeYmaera implements a sequent calculus version [47] of the axiomatization in Fig. 2, 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 [50], [57], [58] that has been implemented in KeYmaera.

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

Differential dynamic logic and KeYmaera have been used successfully for verifying system-level properties of local lane controllers for highway car traffic [38], car controllers for intersections [37], intelligent speed adaptation for variable speed limit control and incident management by traffic centers on highways [41], CICAS-SLTA left-turn assist controllers for cars at intersections [4], flyable roundabout collision avoidance maneuvers for aircraft [59], the cooperation protocols of the European Train Control System ETCS [61], and analog circuits [50]. 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 [50] 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.

III. Related Work

Hybrid systems is a very active area of research, so a comprehensive overview of all results is impossible. In this tutorial, we focus on logic and proofs for hybrid systems. For background on classical logic and proving, we refer to the literature [20], [28], [34].

Model checking and reachability analysis have been used successfully for hybrid systems. They work by state space exploration and use various abstractions or approximations [2], [30], [32], including numerical approximations [5], [13].

Verification tools are based on logic and proofs [47], [54], [60], polyhedral reachability analysis [22], [31], reachability analysis with support functions [23], [25], interval-constraint propagation [66], and numerical PDE solving [40]. Constraint-based verification approaches [26], [63], [71] have been considered, which are related to differential invariants.

Many languages have been proposed for modeling hybrid systems, including extended duration calculus [76], hybrid automata [30], hybrid programs [45], [47], guarded commands [69], hybrid π-calculus [70], and process algebra χ [74].

Logic has been used successfully for real-time systems [19], [32] and for timed-automata based model checking [3], [7]. Details about real-time systems can be found in [7], [42].

The use of logic has been proposed for hybrid systems, e.g., in a propositional modal μ-calculus [18] or in early work based on phase transition systems [39]. See [18] 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 [18], [65]. The use of theorem provers has been suggested in hybrid systems, including STeP [35] and PVS [1]. 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 [50], [58].

IV. Summary and Outlook

We have surveyed a (small) selection of topics about differential dynamic logic for 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.

Only a small selection of the important results about differential dynamic logics are included in this survey. We still hope to have given the reader a good first overview of logics for dynamical systems. The reader should note that, for space reasons, we could not cover differential-algebraic dynamic logic (DAL) [49] for hybrid systems with differential-algebraic constraints modeled in differential-algebraic program, the temporal extension of differential temporal dynamic logic (dTL) [50], quantified differential dynamic logic (Q$\mathsf{d}\mathcal{L}$) for distributed hybrid systems [51], [53], and stochastic differential dynamic logic (S$\mathsf{d}\mathcal{L}$) for stochastic hybrid systems [52].

The results summarized in this article demonstrate that logic is a powerful tool, not just for studying discrete phenomena, but also continuous phenomena. Extensions to infinite-dimensional and stochastic phenomena are shown elsewhere [51]–[53]. The logic $\mathsf{d}\mathcal{L}$ is a strong logical foundation for hybrid 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.

Acknowledgment

I am grateful to Jan-David Quesel for indispensable help with implementing KeYmaera. 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. This material is based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246, NSF EXPEDITION CNS-0926181, and under Grant 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).

References

[1] E. Ábrahám-Mumm, M. Steffen, and U. Hannemann, "Verification of hybrid systems: Formalization and proof rules in PVS." in *ICECCS*, S. F. Andler and J. Offutt, Eds. Los Alamitos: IEEE Computer Society, 2001, pp. 48–57.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theor. Comput. Sci.*, vol. 138, no. 1, pp. 3–34, 1995.

[3] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[4] N. Aréchiga, S. M. Loos, A. Platzer, and B. H. Krogh, "Using theorem provers to guarantee closed-loop system properties," in *ACC*, D. Tilbury, Ed., 2012.

[5] E. Asarin, T. Dang, and A. Girard, "Reachability analysis of nonlinear systems using conservative approximation," in *HSCC*, ser. LNCS, O. Maler and A. Pnueli, Eds., vol. 2623. Springer, 2003, pp. 20–35.

[6] E. Asarin and O. Maler, "Achilles and the tortoise climbing up the arithmetical hierarchy," *J. Comput. Syst. Sci.*, vol. 57, no. 3, pp. 389–398, 1998.

[7] C. Baier, J.-P. Katoen, and K. G. Larsen, *Principles of Model Checking*. MIT Press, 2008.

[8] B. Beckert, R. Hähnle, and P. H. Schmitt, Eds., *Verification of Object-Oriented Software: The KeY Approach*, ser. LNCS. Springer, 2007, vol. 4334.

[9] M. S. Branicky, "General hybrid dynamical systems: Modeling, analysis, and control," in *Hybrid Systems*, ser. LNCS, R. Alur, T. A. Henzinger, and E. D. Sontag, Eds., vol. 1066. Springer, 1995, pp. 186–200.

[10] M. S. Branicky, V. S. Borkar, and S. K. Mitter, "A unified framework for hybrid control: Model and optimal control theory," *IEEE T. Automat. Contr.*, vol. 43, no. 1, pp. 31–45, 1998.

[11] R. Carnap, "Modalities and quantification," *J. Symb. Log.*, vol. 11, no. 2, pp. 33–64, 1946.

[12] F. Cassez and K. G. Larsen, "The impressive power of stopwatches," in *CONCUR*, 2000, pp. 138–152.

[13] A. Chutinan and B. H. Krogh, "Computational techniques for hybrid system verification," *IEEE T. Automat. Contr.*, vol. 48, no. 1, pp. 64–75, 2003.

[14] E. Clarke, A. Fehnker, Z. Han, B. Krogh, O. Stursberg, and M. Theobald, "Verification of hybrid systems based on counterexample-guided abstraction refinement," in *TACAS 2003*, ser. LNCS, H. Garavel and J. Hatcliff, Eds., no. 2619, 2003, pp. 192–207.

[15] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[16] P. Collins, "Optimal semicomputable approximations to reachable and invariant sets," *Theory Comput. Syst.*, vol. 41, no. 1, pp. 33–48, 2007.

[17] S. A. Cook, "Soundness and completeness of an axiom system for program verification." *SIAM J. Comput.*, vol. 7, no. 1, pp. 70–90, 1978.

[18] J. M. Davoren and A. Nerode, "Logics for hybrid systems," *IEEE*, vol. 88, no. 7, pp. 985–1010, July 2000.

[19] B. Dutertre, "Complete proof systems for first order interval temporal logic," in *LICS*. IEEE Computer Society, 1995, pp. 36–43.

[20] M. Fitting, *First-Order Logic and Automated Theorem Proving*, 2nd ed. New York: Springer, 1996.

[21] R. W. Floyd, "Assigning meanings to programs," in *Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics*, J. T. Schwartz, Ed., vol. 19. Providence: AMS, 1967, pp. 19–32.

[22] G. Frehse, "PHAVer: algorithmic verification of hybrid systems past HyTech," *STTT*, vol. 10, no. 3, pp. 263–279, 2008.

[23] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *CAV*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 379–395.

[24] G. Gentzen, "Untersuchungen über das logische Schließen. II," *Math. Zeit.*, vol. 39, no. 3, pp. 405–431, 1935.

[25] C. L. Guernic and A. Girard, "Reachability analysis of hybrid systems using support functions," in *CAV*, ser. LNCS, A. Bouajjani and O. Maler, Eds., vol. 5643. Springer, 2009, pp. 540–554.

[26] S. Gulwani and A. Tiwari, "Constraint-based approach for analysis of hybrid systems," in *CAV*, ser. LNCS, A. Gupta and S. Malik, Eds., vol. 5123. Springer, 2008, pp. 190–203.

[27] A. Gupta and S. Malik, Eds., *Computer Aided Verification, CAV 2008, Princeton, NJ, USA, Proceedings*, ser. LNCS, vol. 5123. Springer, 2008.

[28] D. Harel, D. Kozen, and J. Tiuryn, *Dynamic logic*. Cambridge: MIT Press, 2000.

[29] D. Harel, A. R. Meyer, and V. R. Pratt, "Computability and completeness in logics of programs (preliminary report)," in *STOC*. ACM, 1977, pp. 261–268.

[30] T. A. Henzinger, "The theory of hybrid automata." in *LICS*. Los Alamitos: IEEE Computer Society, 1996, pp. 278–292.

[31] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HyTech: A model checker for hybrid systems," *STTT*, vol. 1, no. 1-2, pp. 110–122, 1997.

[32] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model checking for real-time systems," in *LICS*. IEEE Computer Society, 1992, pp. 394–406.

[33] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[34] G. E. Hughes and M. J. Cresswell, *A New Introduction to Modal Logic*. Routledge, 1996.

[35] Y. Kesten, Z. Manna, and A. Pnueli, "Verification of clocked and hybrid systems," *Acta Inf.*, vol. 36, no. 11, pp. 837–912, 2000.

[36] D. Kozen, "Kleene algebra with tests," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 3, pp. 427–443, 1997.

[37] S. M. Loos and A. Platzer, "Safe intersections: At the crossing of hybrid systems and verification," in *ITSC*, K. Yi, Ed. Springer, 2011, pp. 1181–1186.

[38] S. M. Loos, A. Platzer, and L. Nistor, "Adaptive cruise control: Hybrid, distributed, and now formally verified," in *FM*, ser. LNCS, M. Butler and W. Schulte, Eds., vol. 6664. Springer, 2011, pp. 42–56.

[39] O. Maler, Z. Manna, and A. Pnueli, "From timed to hybrid systems," in *REX Workshop*, ser. LNCS, J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, Eds., vol. 600. Springer, 1991, pp. 447–484.

[40] I. M. Mitchell and J. A. Templeton, "A toolbox of Hamilton-Jacobi solvers for analysis of nondeterministic continuous and hybrid systems," in *HSCC*, ser. LNCS, M. Morari and L. Thiele, Eds., vol. 3414. Springer, 2005, pp. 480–494.

[41] S. Mitsch, S. M. Loos, and A. Platzer, "Towards formal verification of freeway traffic control," in *ICCPS*, C. Lu, Ed. IEEE, 2012, pp. 171–180.

[42] E.-R. Olderog and H. Dierks, *Real-Time Systems: Formal Specification and Automatic Verification*. Cambridge Univ. Press, 2008.

[43] R. Parikh, "The completeness of propositional dynamic logic," in *MFCS*, ser. LNCS, J. Winkowski, Ed., vol. 64. Springer, 1978, pp. 403–415.

[44] L. Perko, *Differential equations and dynamical systems*, 3rd ed. New York: Springer, 2006.

[45] A. Platzer, "Differential dynamic logic for verifying parametric hybrid systems." in *TABLEAUX*, ser. LNCS, N. Olivetti, Ed., vol. 4548. Springer, 2007, pp. 216–232.

[46] ——, "A temporal dynamic logic for verifying hybrid system invariants." in *LFCS*, ser. LNCS, S. N. Artëmov and A. Nerode, Eds., vol. 4514. Springer, 2007, pp. 457–471.

[47] ——, "Differential dynamic logic for hybrid systems." *J. Autom. Reas.*, vol. 41, no. 2, pp. 143–189, 2008.

[48] ——, "Differential dynamic logics: Automated theorem proving for hybrid systems," Ph.D. dissertation, Department of Computing Science, University of Oldenburg, Dec 2008, appeared with Springer.

[49] ——, "Differential-algebraic dynamic logic for differential-algebraic programs," *J. Log. Comput.*, vol. 20, no. 1, pp. 309–352, 2010.

[50] ——, *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Heidelberg: Springer, 2010.

[51] ——, "Quantified differential dynamic logic for distributed hybrid systems," in *CSL*, ser. LNCS, A. Dawar and H. Veith, Eds., vol. 6247. Springer, 2010, pp. 469–483.

[52] ——, "Stochastic differential dynamic logic for stochastic hybrid programs," in *CADE*, ser. LNCS, N. Bjørner and V. Sofronie-Stokkermans, Eds., vol. 6803. Springer, 2011, pp. 431–445.

[53] ——, "A complete axiomatization of quantified differential dynamic logic for distributed hybrid systems," *Logical Methods in Computer Science*, 2012, special issue for selected papers from CSL'10.

[54] ——, "The complete proof theory of hybrid systems," in *LICS*. IEEE Computer Society, 2012.

[55] ——, "The structure of differential invariants and differential cut elimination," *Logical Methods in Computer Science*, 2012, to appear.

[56] A. Platzer and E. M. Clarke, "The image computation problem in hybrid systems model checking." in *HSCC*, ser. LNCS, A. Bemporad, A. Bicchi, and G. C. Buttazzo, Eds., vol. 4416. Springer, 2007, pp. 473–486.

[57] ——, "Computing differential invariants of hybrid systems as fixedpoints," in *CAV*, ser. LNCS, A. Gupta and S. Malik, Eds., vol. 5123. Springer, 2008, pp. 176–189.

[58] ——, "Computing differential invariants of hybrid systems as fixedpoints," *Form. Methods Syst. Des.*, vol. 35, no. 1, pp. 98–120, 2009, special issue for selected papers from CAV'08.

[59] ——, "Formal verification of curved flight collision avoidance maneuvers: A case study," in *FM*, ser. LNCS, A. Cavalcanti and D. Dams, Eds., vol. 5850. Springer, 2009, pp. 547–562.

[60] A. Platzer and J.-D. Quesel, "KeYmaera: A hybrid theorem prover for hybrid systems." in *IJCAR*, ser. LNCS, A. Armando, P. Baumgartner, and G. Dowek, Eds., vol. 5195. Springer, 2008, pp. 171–178.

[61] ——, "European Train Control System: A case study in formal verification," in *ICFEM*, ser. LNCS, K. Breitman and A. Cavalcanti, Eds., vol. 5885. Springer, 2009, pp. 246–265.

[62] A. Platzer, J.-D. Quesel, and P. Rümmer, "Real world verification," in *CADE*, ser. LNCS, R. A. Schmidt, Ed., vol. 5663. Springer, 2009, pp. 485–501.

[63] S. Prajna, A. Jadbabaie, and G. J. Pappas, "A framework for worst-case and stochastic safety verification using barrier certificates," *IEEE T. Automat. Contr.*, vol. 52, no. 8, pp. 1415–1429, 2007.

[64] V. R. Pratt, "Semantical considerations on Floyd-Hoare logic," in *FOCS*. IEEE, 1976, pp. 109–121.

[65] ——, "A decidable mu-calculus: Preliminary report," in *FOCS*. IEEE Computer Society, 1981, pp. 421–427.

[66] S. Ratschan and Z. She, "Safety verification of hybrid systems by constraint propagation-based abstraction refinement," *Trans. on Embedded Computing Sys.*, vol. 6, no. 1, p. 8, 2007.

[67] W. Reif, G. Schellhorn, and K. Stenzel, "Proving system correctness with KIV 3.0," in *CADE*, ser. LNCS, W. McCune, Ed., vol. 1249. Springer, 1997, pp. 69–72.

[68] D. W. Renshaw, S. M. Loos, and A. Platzer, "Distributed theorem proving for distributed hybrid systems," in *ICFEM*, ser. LNCS, S. Qin and Z. Qiu, Eds., vol. 6991. Springer, 2011, pp. 356–371.

[69] M. Rönkkö, A. P. Ravn, and K. Sere, "Hybrid action systems." *Theor. Comput. Sci.*, vol. 290, no. 1, pp. 937–973, 2003.

[70] W. C. Rounds and H. Song, "The ϕ-calculus: A language for distributed control of reconfigurable embedded systems," in *HSCC*, ser. LNCS, vol. 2623, 2003, pp. 435–449.

[71] S. Sankaranarayanan, H. B. Sipma, and Z. Manna, "Constructing invariants for hybrid systems," *Form. Methods Syst. Des.*, vol. 32, no. 1, pp. 25–55, 2008.

[72] K. Segerberg, "A completeness theorem in the modal logic of programs," *Notices AMS*, vol. 24, p. 522, 1977.

[73] A. Tarski, *A Decision Method for Elementary Algebra and Geometry*, 2nd ed. Berkeley: University of California Press, 1951.

[74] D. A. van Beek, K. L. Man, M. A. Reniers, J. E. Rooda, and R. R. H. Schiffelers, "Syntax and consistent equation semantics of hybrid Chi," *J. Log. Algebr. Program.*, vol. 68, no. 1-2, pp. 129–210, 2006.

[75] W. Walter, *Ordinary Differential Equations*. Springer, 1998.

[76] C. Zhou, A. P. Ravn, and M. R. Hansen, "An extended duration calculus for hybrid real-time systems." in *Hybrid Systems*, ser. LNCS, R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, Eds., vol. 736. Springer, 1992, pp. 36–59.