Journal of Automata, Languages and Combinatorics **u** (v) w, x–y
© Otto-von-Guericke-Universität Magdeburg

Logical Analysis of Hybrid Systems: A Complete Answer to a Complexity Challenge[1]

André Platzer
Computer Science Department, Carnegie Mellon University
Pittsburgh, USA
e-mail: `aplatzer@cs.cmu.edu`

ABSTRACT
Hybrid systems are systems with interacting discrete and continuous dynamics. They are models for understanding, e.g., computer systems interfacing with the physical environment. Hybrid systems have a complete axiomatization in differential dynamic logic relative to continuous systems. They also have a complete axiomatization relative to discrete systems. Moreover, there is a constructive reduction of properties of hybrid systems to corresponding properties of continuous systems or to corresponding properties of discrete systems. We briefly summarize and discuss some of the implications of these results.

Keywords: survey, differential dynamic logic, hybrid systems, completeness, complexity

1. Overview

Hybrid (dynamical) systems [2, 7, 12] are dynamical systems that combine discrete and continuous dynamics. They are important for modeling embedded systems and cyber-physical systems. Hybrid systems are natural models for many application scenarios, especially because each part of the system can be modeled in the most natural way. Discrete aspects of the system, e.g., discrete switching, computing, and control decisions can be modeled by discrete dynamics. Continuous aspects of the system, e.g., motion or continuous physical processes can be modeled by continuous dynamics. Hybrid systems combine both kinds of dynamics, not just side by side but with interactions.

This flexibility makes hybrid systems very natural for system modeling. Even very complicated systems can be modeled as hybrid systems by recognizing that some parts of the system are simply discrete, others are simply continuous, and the systems themselves are only complicated because both simple pieces interact in complicated ways. Discrete and continuous aspects can be added to the system model on an as needed basis without having to commit to a prior bias on all modeling elements having to be purely discrete without room for continuous phenomena, or, vice versa, having

[1]This brief survey is based on an abstract for an invited talk at DCFS [31].

to commit to purely continuous models of the world without discrete descriptions. Unfortunately, reachability in hybrid systems is undecidable [12]. Even purely discrete systems are already undecidable, as witnessed by the halting problem for Turing machines. Even purely continuous systems are already undecidable [23, Theorem 2] as witnessed by a corollary to Gödel's second incompleteness theorem. Hybrid systems are more useful for modeling purposes than either purely discrete or purely continuous models. But how does the verification problem for hybrid systems compare to that of purely discrete systems and to that of purely continuous systems? The hybrid systems verification problem most certainly could not be any easier than that for discrete systems (which is undecidable) or that for continuous systems (which is undecidable as well). But is the analysis of hybrid systems fundamentally more difficult than the analysis of purely discrete or that of purely continuous systems? Or do hybrid systems only add natural ways of expressing system models without causing additional complexities for verification that are fundamentally more difficult to solve? Are hybrid systems more complex than discrete systems, i.e. more difficult to verify? Are they more complex than continuous systems? And how do the two special fragments of hybrid systems compare? Are continuous systems fundamentally more complex or are discrete systems more complex for analysis purposes?

Hybrid systems combine two *independent* sources of undecidability, discrete and continuous dynamics, because discrete dynamics alone causes undecidability (Turing machines) and because continuous dynamics alone causes undecidability (periodic functions) [5, 23]. Hence, the first intuition may be that hybrid systems should be fundamentally more difficult than either of its two fragments and so higher up in the respective recursive hierarchies. That turns out not to be the case, however, because there are complete proof-theoretical alignments of the discrete dynamics, continuous dynamics, and hybrid dynamics [23, 30]. In this paper, we explain a few of the consequences of these results.

The results surveyed in this paper have been developed by leveraging differential dynamic logic [22–24, 30], which is a logic for hybrid systems. For further background on logic for hybrid systems, we refer to the literature [23, 24, 26, 32]. Dynamic logic [41] has been developed and used very successfully for conventional discrete programs, both for theoretical [8–11, 13–15, 18, 20, 21, 44] and practical purposes [4, 10, 42]. We refer to other sources for a more detailed exposition of dynamic logics for hybrid systems [22–26, 30, 32]. Logic of hybrid systems has been used to obtain interesting theoretical results [22–30, 33], while, at the same time, enabling the practical verification of complex applications across different fields [3, 16, 17, 19, 24, 26, 37, 39, 43] and inspiring algorithmic logic-based verification approaches [24, 26, 35, 36, 38, 40, 43]. Extensions to logic for distributed hybrid systems [27, 29] and logic for stochastic hybrid systems [28] can be found elsewhere.

2. Differential Dynamic Logic

Differential dynamic logic $\mathsf{d}\mathcal{L}$ [22–24, 30, 32] is a dynamic logic [41] for hybrid systems [7, 12]. To set the stage for the results surveyed in this paper, we give a brief introduction to $\mathsf{d}\mathcal{L}$. We refer to previous work [23, 24, 26, 30, 32] for more details.

Regular Hybrid Programs. Differential dynamic logic uses (regular) *hybrid programs* (HP) [23] as hybrid system models. HPs are a program notation for hybrid systems. Their most important feature compared to conventional programming languages is that they allow differential equations to be used for describing the continuous dynamics of a system, e.g., the movement of a car. Other distinguishing features include the generalization of the semantics to that of real Euclidean spaces.

The *atomic HPs* are instantaneous discrete jump *assignments* $x := \theta$, *tests* $?H$ of a first-order formula[2] H of real arithmetic, and *differential equation (systems)* $x' = \theta \,\&\, H$ for a continuous evolution restricted to the domain of evolution described by a first-order formula H, where x' denotes the time-derivative of variable x. *Compound* HPs are generated from these atomic HPs by nondeterministic choice ($\cup$), sequential composition ($;$), and Kleene's nondeterministic repetition (*). For the purpose of this survey, we use polynomials with rational coefficients as terms, even if more general terms are permitted in $\mathsf{d}\mathcal{L}$ [26]. HPs are defined by the following grammar (α, β are HPs, x a variable, θ a term possibly containing x, and H a formula of first-order logic of real arithmetic):

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

The first three cases are called atomic HPs, the last three compound HPs. These operations can define all hybrid systems and all control structures for hybrid systems [26]. For example, hybrid automata can be embedded into hybrid programs in the same way how finite automata can be embedded into WHILE programs. We, e.g., write $x' = \theta$ for the unrestricted differential equation $x' = \theta \,\&\, \mathit{true}$. We allow differential equation systems and use vectorial notation. For various generalizations, e.g., to hybrid systems with differential-algebraic equations [25] and for distributed hybrid systems [29], we refer to the literature. That is also where extensions for special functions can be found [25, 33].

Unlike in conventional programming languages, a state is not an assignment of discrete quantities to program variables, because the current state of a system includes, e.g., positions from a continuous space. A *state* ν is a mapping from variables to $\mathbb{R}$. Hence $\nu(x) \in \mathbb{R}$ is the value of variable x in state ν. The set of states is denoted $\mathcal{S}$. We denote the value of term θ in ν by $\nu[\![\theta]\!]$. Each HP α is interpreted semantically as a binary reachability relation $\rho(\alpha)$ over states, defined inductively by:

- $\rho(x := \theta) = \{(\nu, \omega) : \omega = \nu \text{ except that } \omega[\![x]\!] = \nu[\![\theta]\!]\}$
- $\rho(?H) = \{(\nu, \nu) : \nu \models H\}$
- $\rho(x' = \theta \,\&\, H) = \{(\varphi(0), \varphi(r)) : \varphi(t) \models x' = \theta$ and $\varphi(t) \models H$ for all $0 \le t \le r$ 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 H at all times [23]
- $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
- $\rho(\alpha; \beta) = \rho(\beta) \circ \rho(\alpha)$
- $\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n)$ with $\alpha^{n+1} \equiv \alpha^n; \alpha$ and $\alpha^0 \equiv ?\mathit{true}$.

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

We refer to our book [26] for a comprehensive background. We also refer to [23, 26] for an elaboration how the case $r = 0$ (in which the only condition is $\varphi(0) \models H$) is captured by the above definition.

d$\mathcal{L}$ Formulas. Formulas of d$\mathcal{L}$ are used to specify and verify the relevant properties of HPs. The *formulas of differential dynamic logic* (d$\mathcal{L}$) are defined by the grammar (where ϕ, ψ are d$\mathcal{L}$ formulas, θ_1, θ_2 terms, x a variable, α a HP):

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

The *satisfaction relation* $\nu \models \phi$ is as usual in first-order logic (of real arithmetic) with the addition that $\nu \models [\alpha]\phi$ iff $\omega \models \phi$ for all ω with $(\nu, \omega) \in \rho(\alpha)$. The operator $\langle\alpha\rangle$ dual to $[\alpha]$ is defined by $\langle\alpha\rangle\phi \equiv \neg[\alpha]\neg\phi$. Consequently, $\nu \models \langle\alpha\rangle\phi$ iff $\omega \models \phi$ for some ω with $(\nu, \omega) \in \rho(\alpha)$. Operators $=, >, \leq, <, \vee, \rightarrow, \leftrightarrow, \exists x$ can be defined as usual in first-order logic, e.g., $\exists x\, \phi \equiv \neg\forall x\, \neg\phi$. A d$\mathcal{L}$ formula ϕ is *valid*, written $\vDash \phi$, iff $\nu \models \phi$ for all states ν.

Example 1 For instance, the d$\mathcal{L}$ formula

$$x > 0 \wedge v > 5 \wedge a > 2 \rightarrow [a := a - 1; x' = v, v' = a](x > 0 \wedge v \geq 1) \qquad (1)$$

expresses that the position x will always be positive and the velocity v will always stay greater or equal to 1 if, initially, $x > 0$ and $v > 5$ and the acceleration a is initially greater than 2. The hybrid system dynamics in this formula simply first decreases a by the instantaneous discrete assignment $a := a - 1$ and then follows the differential equation system $x' = v, v' = a$ where the position x changes with time-derivative v and the velocity v changes with time-derivative a. That is, x changes continuously with acceleration a. The d$\mathcal{L}$ formula (1) is obviously valid, i.e., true for all interpretations of its variables.

3. Verifying Hybrid Systems

Differential dynamic logic is not just useful as a specification language for properties of hybrid systems, but also as a verification logic with which truth and validity of properties of hybrid systems can be established. There is a sequent calculus [22–24,26] and a Hilbert calculus [30] for proving the validity of formulas of differential dynamic logic. Both styles of proof calculi follow the same principles. The differences are only that the sequent calculi are more tuned for automation, whereas the Hilbert calculus is tuned for simplicity. The sequent calculi can be implemented directly in theorem provers and have been implemented in KeYmaera [38], whereas the Hilbert calculus allows more flexibility and is less goal-directed.

The proof calculi for d$\mathcal{L}$ basically work by performing a symbolic execution and perform a recursive structural decomposition of the d$\mathcal{L}$ formula and all its constituent hybrid programs. That way, properties of more complex hybrid systems reduce to properties of structurally simpler parts of the system. Because the systems become simpler when performing those recursive decompositions, it is getting structurally easier to verify them. Once a proof of each of the resulting pieces succeeds, the whole system has automatically been verified by an immediate composition of the proofs about all its pieces. The proof calculi for d$\mathcal{L}$ make this recursive decomposition precise and ensure that all arguments in the reasoning process stay correct. It is very easy, for example, to prove formula (1) from Example 1 in the proof calculus for d$\mathcal{L}$ just by reduction to a property of the discrete assignment $a := a - 1$ and a property of the solution of the differential equation $x' = v, v' = a$. In this case, the decompositions that make this happen are completely trivial, because the system is so simple. In more involved cases with actual control loops and repetitions, invariants play a similar role to make the decompositions happen. Whether that is possible is what we discuss in the next section.

4. Complete Relations

Even though hybrid systems are very expressive, they nevertheless have a complete axiomatization in differential dynamic logic d$\mathcal{L}$ [23, 30] relative to elementary properties of differential equations. The completeness notions are inspired by those of Cook [6] and Harel et al. [11], yet different, because the data logic of hybrid systems has a decidable validity problem (first-order real arithmetic) [45]. In classical program cases, the data logic of first-order integer arithmetic, instead, is undecidable.

From Hybrid to Continuous. Using the proof calculus of d$\mathcal{L}$, the problem of proving properties of hybrid systems reduces to proving properties of elementary continuous systems [23]. 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, d$\mathcal{L}$ formulas of the form $[x' = \theta]F$ with a first-order formula F. We have shown that the d$\mathcal{L}$ calculus is a sound and complete axiomatization relative to FOD.

Theorem 1 (Continuous relative completeness of d$\mathcal{L}$ [23, 30]) *The* d$\mathcal{L}$ *calculus is a* sound and complete axiomatization *of hybrid systems relative to its continuous fragment FOD, i.e., every valid* d$\mathcal{L}$ *formula can be derived from FOD tautologies:*

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

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

Since the proof of Theorem 1 is constructive, there is even a complete constructive reduction of properties of hybrid systems to corresponding properties of continuous systems. The d$\mathcal{L}$ calculus can prove hybrid systems properties exactly as good as properties of the corresponding continuous systems can be verified. One important step in the proof of Theorem 1 shows that all required invariants and variants for repetitions can be expressed in the logic d$\mathcal{L}$. Furthermore, the d$\mathcal{L}$ calculus defines a decision procedure for d$\mathcal{L}$ sentences (i.e., closed formulas) relative to an oracle for FOD [30].

This result implies that the continuous dynamics dominates the discrete dynamics since, once the continuous dynamics is handled, all discrete and hybrid dynamics can be handled as well. Therefore, verification of hybrid systems is not more complex than the verification of continuous systems. In particular, discrete systems verification is not more complex than the verification of continuous systems. This is reassuring, because we get the challenges of discrete dynamics solved for free (by the d$\mathcal{L}$ proof calculus) once we address continuous dynamics.

One way of doing practical proof search and generation of invariants has been addressed in previous work [35,36]. But many other ways could be useful to generate invariants more efficiently in practice. In fact, in many circumstances, it is quite obvious that significant advances can be made to the efficiency in practice.

From Hybrid to Discrete. In a certain sense, it may appear to be more complicated to handle continuous dynamics than discrete dynamics. After all, most computers themselves are discrete, so mechanized proofs and any other verification technique on computers will ultimately need to understand continuous effects from a purely discrete perspective. If the continuous dynamics are not just subsuming discrete dynamics and if they were inherently more, then that could be understood as an indicator that hybrid systems verification is fundamentally impossible with discrete means. Of course, if this were the case, the argument would not even be quite so simple, because meta-proofs may still enable discrete finitary proof objects to entail infinite continuous object-properties. In fact, they do, because finite d$\mathcal{L}$ proof objects already entail properties about uncountable continuous spaces.

Fortunately, we can settle all worries about the insufficiency of discrete ways of understanding continuous phenomena once and for all by studying the proof-theoretical relationship between discrete and continuous dynamics. We have shown not only that the axiomatization of d$\mathcal{L}$ is complete relative to the continuous fragment, but that it is also complete relative to the discrete fragment [30]. The *discrete fragment* of d$\mathcal{L}$ is denoted by DL, i.e., the fragment without differential equations. It is, in fact, sufficient to restrict DL to the operators $:=, {}^*$ and allow either ; or vector assignments.

Theorem 2 (Discrete relative completeness of d$\mathcal{L}$ [30]) *The* d$\mathcal{L}$ *calculus is a* sound and complete axiomatization *of hybrid systems relative to its discrete fragment DL, i.e., every valid* d$\mathcal{L}$ *formula can be derived from DL tautologies.*

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

Thus, the d$\mathcal{L}$ calculus can also prove properties of hybrid systems exactly as good as properties of discrete systems can be proved. Again, the proof of Theorem 2 is constructive, entailing that there is a constructive way of reducing properties of hybrid systems to properties of discrete systems using the d$\mathcal{L}$ calculus. Furthermore, the d$\mathcal{L}$ calculus defines a decision procedure for d$\mathcal{L}$ sentences relative to an oracle for DL [30].

As a corollary to Theorems 1 and 2, we can proof-theoretically and constructively equate

hybrid = continuous = discrete

Even though each kind of dynamics comes from fundamentally different principles, they all meet in terms of their proof problems being interreducible, even constructively. The complexity of the proof problem of hybrid systems, the complexity of the proof problem of continuous systems, and the complexity of the proof problem of discrete systems are, thus, equivalent. Any proof technique for one of these classes of systems completely lifts to proof techniques for the other class of systems.

Since the proof problems interreduce constructively, every technique that is successful for one kind of dynamics lifts to the other kind of dynamics through the $\mathsf{d}\mathcal{L}$ calculus in a provably perfect way. Induction, for example, is the primary technique for proving properties of discrete systems. Hence, by Theorem 2, there is a corresponding induction technique for continuous systems and for hybrid systems. And, indeed, *differential invariants* [25, 33] are such an induction technique for differential equations that has been used very successfully for verifying hybrid systems with more advanced differential equations [26, 35–37, 39]. In fact, differential invariants had already been introduced in 2008 [25] before Theorem 2 was proved [30], but Theorem 2 implies that a differential invariant induction technique has to exist. These results also show that there are sound ways of using discretization for differential equations [30] and that numerical integration schemes like, e.g., Euler's method or more elaborate methods can be used for hybrid systems verification, which is not at all clear a priori due to inherent numerical approximation errors, which may blur decisions either way [34].

5. Conclusions and Future Work

We have summarized recent results about complete axiomatizations of hybrid systems relative to continuous systems and relative to discrete systems. These axiomatizations equate the proof problems for all three classes of systems and align the complexity of the their proof problems. Practical consequences of this result include differential invariants and the utility of discretization schemes, but many other consequences are just waiting to be discovered.

At the same time, the complete axiomatizations open up interesting questions for further study. The constructive proofs prove upper bounds for the complexity of the respective interreductions, but do not provide lower bounds. In fact, a number of improved upper bounds can be read off directly from the proof constructions for a number of cases, including better bounds for fragments of the logic. This includes, e.g., the case of closed properties proved for systems that do not even leave the interiors of those regions, or for purely discrete parts of the transition dynamics. It is an interesting question to study these cases in more detail. Identifying other practical consequences of the constructive completeness results is another interesting question for further research.

Practical proof procedures have been devised that make the completeness results practical by fixedpoint loops in proof search space. Yet, at the same time, significantly more efficient procedures can be obtained in practice, which is a very important area for future research.

Acknowledgements

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, by DARPA under agreement number FA8750-12-2-0291, by the US Department of Transportation's University Transportation Center's TSET award DTRT12GUTC11, 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] *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012*. IEEE, 2012.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, The Algorithmic Analysis of Hybrid Systems. *Theor. Comput. Sci.* **138** (1995) 1, 3–34.

[3] N. Aréchiga, S. M. Loos, A. Platzer, B. H. Krogh, Using Theorem Provers to Guarantee Closed-Loop System Properties. In: D. Tilbury (ed.), *ACC*. 2012, 3573–3580.

[4] B. Beckert, R. Hähnle, P. H. Schmitt (eds.), *Verification of Object-Oriented Software: The KeY Approach*. LNCS 4334, Springer, 2007.

[5] O. Bournez, M. L. Campagnolo, D. S. Graça, E. Hainry, Polynomial differential equations compute all real computable functions on computable compact intervals. *Journal of Complexity* **23** (2007), 317–335.

[6] S. A. Cook, Soundness and Completeness of an Axiom System for Program Verification. *SIAM J. Comput.* **7** (1978) 1, 70–90.

[7] J. M. Davoren, A. Nerode, Logics for Hybrid Systems. *IEEE* **88** (2000) 7, 985–1010.

[8] M. J. Fischer, R. E. Ladner, Propositional Dynamic Logic of Regular Programs. *J. Comput. Syst. Sci.* **18** (1979) 2, 194–211.

[9] D. Harel, *First-Order Dynamic Logic*. Springer, New York, 1979.

[10] D. Harel, D. Kozen, J. Tiuryn, *Dynamic logic*. MIT Press, 2000.

[11] D. Harel, A. R. Meyer, V. R. Pratt, Computability and Completeness in Logics of Programs (Preliminary Report). In: *STOC*. ACM, 1977, 261–268.

[12] T. A. Henzinger, The Theory of Hybrid Automata. In: *LICS*. IEEE Computer Society, Los Alamitos, 1996, 278–292.

[13] S. Istrail, An Arithmetical Hierarchy in Propositional Dynamic Logic. *Inf. Comput.* **81** (1989) 3, 280–289.

[14] D. Kozen, R. Parikh, An Elementary Proof of the Completeness of PDL. *Theor. Comp. Sci.* **14** (1981), 113–118.

[15] D. Leivant, Matching Explicit and Modal Reasoning about Programs: A Proof Theoretic Delineation of Dynamic Logic. In: *LICS*. IEEE Computer Society, 2006, 157–168.

[16] S. M. Loos, A. Platzer, Safe Intersections: At the Crossing of Hybrid Systems and Verification. In: K. Yi (ed.), *ITSC*. Springer, 2011, 1181–1186.

[17] S. M. Loos, A. Platzer, L. Nistor, Adaptive Cruise Control: Hybrid, Distributed, and Now Formally Verified. In: M. Butler, W. Schulte (eds.), *FM*. 6664, Springer, 2011, 42–56.

[18] A. R. Meyer, R. Parikh, Definability in Dynamic Logic. *J. Comput. Syst. Sci.* **23** (1981) 2, 279–298.

[19] S. Mitsch, S. M. Loos, A. Platzer, Towards Formal Verification of Freeway Traffic Control. In: C. Lu (ed.), *ICCPS*. IEEE, 2012, 171–180.

[20] R. Parikh, The Completeness of Propositional Dynamic Logic. In: J. Winkowski (ed.), *MFCS*. 64, Springer, 1978, 403–415.

[21] D. Peleg, Concurrent dynamic logic. *J. ACM* **34** (1987) 2, 450–479.

[22] A. Platzer, Differential Dynamic Logic for Verifying Parametric Hybrid Systems. In: N. Olivetti (ed.), *TABLEAUX*. 4548, Springer, 2007, 216–232.

[23] A. Platzer, Differential Dynamic Logic for Hybrid Systems. *J. Autom. Reas.* **41** (2008) 2, 143–189.

[24] A. Platzer, *Differential Dynamic Logics: Automated Theorem Proving for Hybrid Systems*. Ph.D. thesis, Department of Computing Science, University of Oldenburg, 2008.

[25] A. Platzer, Differential-Algebraic Dynamic Logic for Differential-Algebraic Programs. *J. Log. Comput.* **20** (2010) 1, 309–352.

[26] A. Platzer, *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010.

[27] A. Platzer, Quantified Differential Dynamic Logic for Distributed Hybrid Systems. In: A. Dawar, H. Veith (eds.), *CSL*. 6247, Springer, 2010, 469–483.

[28] A. Platzer, Stochastic Differential Dynamic Logic for Stochastic Hybrid Programs. In: N. Bjørner, V. Sofronie-Stokkermans (eds.), *CADE*. 6803, Springer, 2011, 431–445.

[29] A. Platzer, A Complete Axiomatization of Quantified Differential Dynamic Logic for Distributed Hybrid Systems. *Logical Methods in Computer Science* **8** (2012) 4, 1–44.

[30] A. Platzer, The Complete Proof Theory of Hybrid Systems. In: *LICS* [1], 2012, 541–550.

[31] A. Platzer, Logical Analysis of Hybrid Systems: A Complete Answer to a Complexity Challenge. In: M. Kutrib, N. Moreira, R. Reis (eds.), *DCFS*. 7386, Springer, 2012, 43–49.

[32] A. Platzer, Logics of Dynamical Systems. In: *LICS* [1], 2012, 13–24.

[33] A. Platzer, The Structure of Differential Invariants and Differential Cut Elimination. *Logical Methods in Computer Science* **8** (2012) 4, 1–38.

[34] A. Platzer, E. M. Clarke, The Image Computation Problem in Hybrid Systems Model Checking. In: A. Bemporad, A. Bicchi, G. C. Buttazzo (eds.), *HSCC*. 4416, Springer, 2007, 473–486.

[35] A. Platzer, E. M. Clarke, Computing Differential Invariants of Hybrid Systems as Fixedpoints. In: A. Gupta, S. Malik (eds.), *CAV*. 5123, Springer, 2008, 176–189.

[36] A. Platzer, E. M. Clarke, Computing Differential Invariants of Hybrid Systems as Fixedpoints. *Form. Methods Syst. Des.* **35** (2009) 1, 98–120.

[37] A. Platzer, E. M. Clarke, Formal Verification of Curved Flight Collision Avoidance Maneuvers: A Case Study. In: A. Cavalcanti, D. Dams (eds.), *FM*. 5850, Springer, 2009, 547–562.

[38] A. Platzer, J.-D. Quesel, KeYmaera: A Hybrid Theorem Prover for Hybrid Systems. In: A. Armando, P. Baumgartner, G. Dowek (eds.), *IJCAR*. 5195, Springer, 2008, 171–178.

[39] A. Platzer, J.-D. Quesel, European Train Control System: A Case Study in Formal Verification. In: K. Breitman, A. Cavalcanti (eds.), *ICFEM*. 5885, Springer, 2009, 246–265.

[40] A. Platzer, J.-D. Quesel, P. Rümmer, Real World Verification. In: R. A. Schmidt (ed.), *CADE*. 5663, Springer, 2009, 485–501.

[41] V. R. Pratt, Semantical Considerations on Floyd-Hoare Logic. IEEE, 1976, 109–121.

[42] W. Reif, G. Schellhorn, K. Stenzel, Proving System Correctness with KIV 3.0. In: W. McCune (ed.), *CADE*. 1249, Springer, 1997, 69–72.

[43] D. W. Renshaw, S. M. Loos, A. Platzer, Distributed Theorem Proving for Distributed Hybrid Systems. In: S. Qin, Z. Qiu (eds.), *ICFEM*. 6991, Springer, 2011, 356–371.

[44] K. Segerberg, A completeness theorem in the modal logic of programs. *Notices AMS* **24** (1977), 522.

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