Formal Artifacts as Explanations for System Correctness in Cyber-Physical Systems

Stefan Mitsch
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
Email: smitsch@cs.cmu.edu

Abstract—**Autonomous systems, such as self-driving cars and robots, are increasingly often entrusted with operating in safety-critical ways, which makes certification an important tool to ensure that systems are operating as intended. This paper argues that formal methods are a useful basis for certification not only in terms of their rigor, but also as a way of explaining evidence of system correctness at the right level of detail.**

Index Terms—**formal verification, theorem proving, hybrid systems, refinement, runtime verification, explanations**

I. Introduction

Semi-autonomous and fully autonomous systems—such as driver assistance systems, self-driving cars, aircraft, and robots—are increasingly often entrusted with operating in safety-critical ways. This makes certification an important tool to scrutinize systems before they are deployed and ensure that they are operating as intended. Obtaining evidence of system correctness, however, is not straightforward: assurance cases in combination with testing the system itself is understood well and produces well-understood statistical estimates of the risks of safety violation, but in autonomous systems it requires unrealistically large amounts of test runs to achieve the required confidence [1]. Formal verification of symbolic models, in contrast, provides guarantees for infinitely many situations, but only some methods provably guarantee that the analyzed models are sufficiently accurate [2]. Reliably predicting the behavior of a system and understanding its assurance arguments becomes even more challenging when machine learning components are operating parts of the system [3]. Recently, using formal methods to explain the behavior of learned agents has sparked research interest, e.g., [4], [5].

Even though formal verification is a suggested technique in some certification standards (e.g., the formal methods supplement to DO-178C in avionics), it is still an open question how to best present and explain formal artifacts. In this paper, we argue that *(i)* deductive verification and proofs provide various aspects of direct evidence about system correctness, and that *(ii)* formal techniques are a useful basis for explaining such evidence to a diverse audience with their different situational needs in understanding systems at the right level of detail [5].

II. Models as Explanations

Computing systems that reach out into the real world with actuators to affect physical processes can be understood mathematically as hybrid systems and hybrid games [6], which describe computations plus their physical effects in a uniform modeling language. Deductive verification tools like KeYmaera X [7] implement differential dynamic (game) logic [6] to rigorously analyze the behavior of hybrid systems and games models for safety and liveness properties, for invariants that are preserved throughout system execution, as well as for winning strategies in game settings.

With their unambiguous semantics, formal models can themselves be useful sources for explanation in the following ways: *(i)* the structure of modular formal hybrid systems models with clearly identified assumptions and guarantees may help in understanding the dynamic interaction between the components of a system; *(ii)* formal hybrid systems models can be expressed at varying levels of abstraction, navigating these levels may help in gaining an understanding of system behavior at an abstract level before understanding the details of the full system; and *(iii)* the predictive power of formal hybrid systems models may help in understanding the link between model and true system. We discuss details for each of these three aspects in the subsections below.

A. Decomposition

Component-based verification approaches [8], [9] help manage proof complexity by allowing users to decompose a system into separate components with local responsibilities. In that process, we identify contracts in terms of assumptions that components make about their environment and verified guarantees about their outputs under these assumptions. In hybrid systems models, the physical dynamics of components is expressed with differential equations, and for contracts additionally summarized using first-order real-arithmetic ($\text{FOL}_{\mathbb{R}}$) abstractions. We construct $\text{FOL}_{\mathbb{R}}$ abstractions from the hybrid systems models when conducting a formal proof, rather than purely modeling them as requirements. Still, when creating architecture models, there is significant freedom in how responsibility is shared between components: the proofs linking differential equations with their $\text{FOL}_{\mathbb{R}}$ abstractions can serve as explanations for archictural choices.

Additionally, the interaction between components in a cyber-physical system takes on a variety of different forms, such as direct communication, sensing, or physical manipulation, all with their own assumptions and guarantees (e.g., about sensor uncertainty). When composing component proofs to a full system proof, the composition arguments about the compatibility between assumptions and guarantees may serve as explanations about the dynamic interaction between the components of a system.

For example, in [10] an ego-agent controller safely navigates among mobile environment agents. The discrete abstraction of the environment agent motion describes a region of positions that is reachable from the current position in some bounded time. Using such an abstraction, we separate the local responsibilities of the ego-agent controller and the mobile environment agents with contracts. The components become connected through sensors that obtain position measurements.

B. Model Refinement and Instantiation

Mathematical arguments about cyber-physical systems are often more crisply formulated at a higher level of abstraction and in formal models with fully symbolic parameters. This makes the resulting proofs applicable to a wide range of actual systems, but may obfuscate the assurance argument for each particular one of them. Formal refinement arguments (e.g., in Hybrid Event-B [11], differential refinement logic [12], [13]) or formal instantiation/substitution arguments [14] create a hierarchy of models at various levels of abstraction, which can be navigated to gain an increasingly detailed understanding of the system behavior while preserving provable correctness along the refinement/instantiation hierarchy.

Crucially, however, in cyber-physical systems the final refinement step (from model to true system) is not solvable through refinement or code synthesis, because a vital part of the system is formed by non-engineered real-world dynamics. Next, we discuss how we can obtain explanations for this final refinement step even in cyber-physical systems.

C. Model Validation

For a comprehensive assurance argument, we want evidence about the correctness of the true system (not just about models of it). Neither testing alone nor formal verification alone can provide a comprehensive explanation of system correctness. Testing can only cover a finite amount of the infinitely many possible scenarios in a cyber-physical system. Formal methods, in contrast, provide strong correctness guarantees about all of the possible scenarios in a *formal model* of the system, e.g., in the form of proofs. This poses an inherent limitation for refinement and instantiation arguments: *there always exists a gap to the true system behavior, no matter how detailed we express formal models*. The remaining question, therefore, is whether the analyzed formal model accurately reflects the modeled system, a question tackled by *model validation*. Only few methods, however, combine offline proofs and model validation in a provably correct way: ModelPlex [2] transforms *by proof* hybrid systems models into monitoring conditions. The monitoring conditions inherit the predictive power of models to describe expected behavior, which makes the monitors capable of flagging discrepancies between models and true system execution. The monitoring conditions serve as a provably correct link between formal verification and model validation/testing and can be useful as explanations in the following ways. ModelPlex monitoring conditions can provide

- validation evidence about how accurately a model reflects the modeled system (are there discrepancies between collected data and the formal model?);
- validation evidence about model parameters (how robustly does the system fit the model, how robustly does it satisfy the inherited safety properties?);
- coverage evidence about a test suite (which aspects of the formal model are covered by test scenarios?);
- validation evidence about the safety-relevant similarity between a simulation environment and a true environment; may help in understanding the challenges of sim-to-real transfer in machine learning (how do the training scenarios in simulation compare to the encountered situations in true system execution?);
- dynamic evidence about system degradation or environment changes (how does safety robustness compare over a history of system executions?).

Model validation is particularly challenging in cyber-physical systems, because judgments about model accuracy must be made from imperfect sensors or based on partially available information [15]. In the presence of sensor uncertainty and partial observability, model validation, therefore, checks for existence of model executions that explain the values observed in the true system [15].

In order to align a symbolic monitor obtained from a symbolic model with the dynamic realities of the system, parametrization is required prior to model validation. Uniform substitution [14] is a technique to prove parameterized models from the symbolic model proofs without additional effort.

III. Proofs as Explanations

Hybrid systems proofs combine reasoning principles from a multitude of fields. In order to be useful as explanations, the challenge is to present proofs at an appropriate level of abstraction, without jeopardizing the soundness of their conclusions. High-level reasoning principles can be useful to gain an overall understanding of the safety argument, while foundational axioms can provide justification for each of the high-level reasoning steps if needed. In [16], different prover designs are compared for their tradeoffs in terms of achieving such levels of abstraction: traditional prover designs favor raw reasoning speed but each new rule increases the trusted code base, while a small-core design emphasizes a small set of foundational axioms and a universal axiom application algorithm, e.g., uniform substitution [14]. In either design, proofs can be explained using invariants and winning strategies as major proof insights, but only the small-core design with executable derived rules (e.g., in a tactic framework) allows us to explore a proof at varying degrees of abstraction without increasing the trusted code base.

A. System Invariants and Winning Strategies

Hybrid systems and hybrid games proofs, just like other proofs of sufficient complexity, are typically structured into supporting lemmas that identify intermediate proof obligations. In hybrid systems, those intermediate proof obligations identify important safety-related characteristics of the system; their role as proof insight in derived proof rules concisely summarizes high-level reasoning concepts—justified from foundational axioms—and might serve as explanations of safety-related properties of the system behavior as follows.

a) Inductive invariants: are guaranteed to be maintained by the combined discrete and continuous system behavior. The derived rule "loop" [16] below expresses that property P is true after all possible ways of executing the repeated program α^* (conclusion below the horizontal bar) when we show three properties of a loop invariant J (premises above the horizontal bar): *(i)* J must hold under the assumptions (left premise, J or alternatives Δ follow from the assumptions Γ), *(ii)* be maintained by the program (middle premise, assuming J, all runs of the loop body α maintain J, expressed by $[\alpha]J$), and *(iii)* imply safety (right premise, P is true assuming J).

$$\text{(loop)}\ \frac{\Gamma \vdash J, \Delta \quad J \vdash [\alpha]J \quad J \vdash P}{\Gamma \vdash [\alpha^*]P, \Delta}$$

Conducting proofs with parametric loop invariants [17] allows us to explore different options for loop invariants in the same proof. That way we can explore modifications of provided loop invariants when scrutinizing a proof for certification to gain a better understanding of the ways in which a safety argument could be strengthened.

b) Intermediate conditions: identify program contracts, e.g., what a controller must guarantee for the continuous dynamics to stay within a safe region. The derived rule MRp below identifies condition Q to explain how reasoning about a sequential program $\alpha; \beta$ is separated into local program proofs $[\alpha]Q$ from assumptions Γ and $[\beta]P$ from assumptions Q:

$$\text{(MRp)}\ \frac{\Gamma \vdash [\alpha]Q, \Delta \quad Q \vdash [\beta]P}{\Gamma \vdash [\alpha; \beta]P, \Delta}$$

c) Proof rules for continuous dynamics: differential cuts [18] describe continuous dynamics with increasingly fine-grained regions, which explain the safety-relevant boundaries of continuous dynamics (e.g., dynamics never exceeds some threshold). A useful technique to prove and explain safety properties of continuous dynamics are energy conservation arguments, which typically require constructing a "ghost" differential equation [18] that balances energy with the original differential equation. Once constructed, we then characterize the continuous dynamics with differential invariants [14], Barrier certificates [19], or Lyapunov functions that are preserved throughout the continuous dynamics. Proof rules for such reasoning concepts about continuous dynamics can be derived [18] in dL. These derived proof rules not only summarize the more fine-grained reasoning steps of the prover core, but also can check the results of numeric invariant/Lyapunov generation methods in order to symbolically explain and justify their correctness.

d) Stability proofs: explain absence of subtle instability behavior [20] known to occur in switched systems of combined discrete switching behavior and continuous system behavior. In [21] several proof rules for different switching mechanisms are derived within dL. Stability arguments based on these rules explain how a system remains stable under small perturbations (stability) and eventually dissipates the energy of these perturbations (attractivity).

e) Winning strategies: provide witnesses for the choices (existential) of an ego-agent in order to counteract adversarial (universal) environment behavior in hybrid games proofs. For example, [22] identifies climb rate strategies for vertical aircraft collision avoidance (e.g., when intruder aircraft is in a certain region, climb with at least some minimum upwards acceleration; in another region, descend). Together with discrete and continuous invariants, winning strategies summarize and explain the main proof arguments.

In terms of explanations, invariant regions and other conditions are amenable for graphical presentation, while winning strategies represent step-by-step instructions of how to react to certain events and conditions. Their justification using proof rules derived from core axioms opens up the possibility to understand the proof in a hierarchical sense, as discussed next.

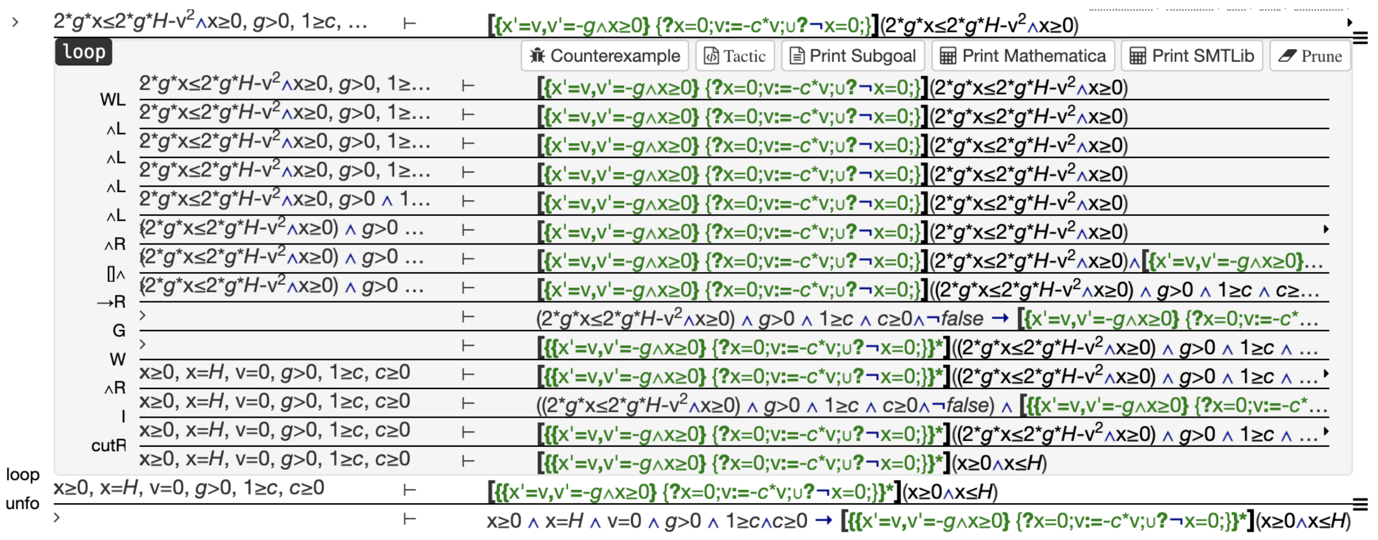
B. Proofs at Different Levels of Abstraction

Theorem provers range from fully interactive to fully automatic, with most systems implementing a combination of interaction and automation to tackle complex systems. For example, the hybrid systems prover KeYmaera X [7] provides automated proof heuristics built on top of a tactics framework [23], which steers the soundness-critical prover core to generate proof terms [24]. Proof terms provide the most detailed view of a proof in the form of the soundness-critical operations of the core, such as builtin rules and axioms, while tactics provide an intermediate explanation in terms of the main proof insights, and the automated proof heuristics document existence of a proof. Even though proof terms justify correctness from just a small set of core logical concepts and, hence, reduce trusting the proof to trusting just these core concepts (instead of trusting the much larger prover software), they are difficult to decipher for their sheer size and lack of presentation structure. Proof terms are therefore mainly useful for proof checking. On the other end of the spectrum, fully automated proofs (as produced, e.g., by SAT/SMT solvers) or automated checks produced by reachability analysis tools, require trust in the full codebase of the tool.

A small-core theorem prover design, such as followed by KeYmaera X, provides opportunity to implement high-level reasoning steps as derived rules in a way that does not extend the soundness-critical core of the theorem prover [16]. Such a design allows us to provide proof explanations by unwinding proof details for all or some steps in a proof to produce a hierarchy of proof details, similar to abstraction in models. In this way, we can navigate a proof at varying levels of detail as our understanding of the proof increases. Fig. 1 shows a screenshot of the KeYmaera X UI with the single-step derived rule "loop" for induction proofs, and its expanded justification in terms of others tactics, core axioms, and proof rules.

IV. Discussion and Open Challenges

Early successes in formal methods for certifying cyber-physical systems (e.g., the Clearsy safety platform [25]) focus on providing a pre-certified platform, so that subsequent development using the platform can fast-track certification. Others emphasize the development process for certification, rather than the produced artifacts. In this paper, we argue for a complementary approach that considers aspects of formal models and proofs as explanations for certification. We take a holistic view that emphasizes the importance of providing explanations for certification authorities to scrutinize both the



```
›  2*g*x≤2*g*H-v^2∧x≥0, g>0, 1≥c, …          ⊢  [{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}](2*g*x≤2*g*H-v^2∧x≥0)
       2*g*x≤2*g*H-v^2∧x≥0, g>0, 1≥…      ⊢  [{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}](2*g*x≤2*g*H-v^2∧x≥0)
  WL
       2*g*x≤2*g*H-v^2∧x≥0, g>0, 1≥…      ⊢  [{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}](2*g*x≤2*g*H-v^2∧x≥0)
  ∧L
       2*g*x≤2*g*H-v^2∧x≥0, g>0, 1≥…      ⊢  [{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}](2*g*x≤2*g*H-v^2∧x≥0)
  ∧L
       2*g*x≤2*g*H-v^2∧x≥0, g>0, 1≥…      ⊢  [{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}](2*g*x≤2*g*H-v^2∧x≥0)
  ∧L
       2*g*x≤2*g*H-v^2∧x≥0, g>0 ∧ 1…      ⊢  [{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}](2*g*x≤2*g*H-v^2∧x≥0)
  ∧L
       (2*g*x≤2*g*H-v^2∧x≥0) ∧ g>0 …      ⊢  [{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}](2*g*x≤2*g*H-v^2∧x≥0)
  ∧R
       (2*g*x≤2*g*H-v^2∧x≥0) ∧ g>0 …      ⊢  [{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}](2*g*x≤2*g*H-v^2∧x≥0)∧[{x'=v,v'=-g∧x≥0}…
  []∧
       (2*g*x≤2*g*H-v^2∧x≥0) ∧ g>0 …      ⊢  [{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}]((2*g*x≤2*g*H-v^2∧x≥0) ∧ g>0 ∧ 1≥c ∧ c≥…
  →R
       ›                                  ⊢  (2*g*x≤2*g*H-v^2∧x≥0) ∧ g>0 ∧ 1≥c ∧ c≥0∧¬false → [{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*…
  G
       ›                                  ⊢  [{{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}}*]((2*g*x≤2*g*H-v^2∧x≥0) ∧ g>0 ∧ 1≥c ∧ …
  W
       x≥0, x=H, v=0, g>0, 1≥c, c≥0       ⊢  [{{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}}*]((2*g*x≤2*g*H-v^2∧x≥0) ∧ g>0 ∧ 1≥c ∧ …
  ∧R
       x≥0, x=H, v=0, g>0, 1≥c, c≥0       ⊢  ((2*g*x≤2*g*H-v^2∧x≥0) ∧ g>0 ∧ 1≥c ∧ c≥0∧¬false) ∧ [{{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*…
  I
       x≥0, x=H, v=0, g>0, 1≥c, c≥0       ⊢  [{{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}}*]((2*g*x≤2*g*H-v^2∧x≥0) ∧ g>0 ∧ 1≥c ∧ …
  cutR
       x≥0, x=H, v=0, g>0, 1≥c, c≥0       ⊢  [{{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}}*](x≥0∧x≤H)
loop
   x≥0, x=H, v=0, g>0, 1≥c, c≥0           ⊢  [{{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}}*](x≥0∧x≤H)
unfo
   ›                                      ⊢  x≥0 ∧ x=H ∧ v=0 ∧ g>0 ∧ 1≥c∧c≥0 → [{{x'=v,v'=-g∧x≥0} {?x=0;v:=-c*v;∪?¬x=0;}}*](x≥0∧x≤H)
```

Fig. 1. Screenshot of the KeYmaera X UI: the derived rule "loop" offers a one-step explanation in terms of an induction proof; the icon ≡ to the right of this step is activated and shows a detailed justification of "loop" in terms of core axioms and other proof rules. The internal steps of tactic "unfold" are hidden.

models (how accurately do models reflect reality?) and the proofs (what are the major proof arguments?).

Even though the proof structure in terms of lemmas and derived rules are a possible way of achieving proof explanations at varying degree of detail, proof trustworthiness and presentation is challenging since computer-checked proofs may have millions of proof steps. Another aspect of proofs that is not yet explored here is vacuity: proofs may succeed for the wrong reasons (e.g., because models exclude important behavior), and we need logic tools to demonstrate that our models and proofs are not vacuously true [26].

Acknowledgments

This material is based upon work supported by the US Air Force and DARPA under Contract No. FA8750-18-C-0092.

References

[1] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" RAND Corporation, Tech. Rep. RR-1478-RC, 2016.

[2] S. Mitsch and A. Platzer, "ModelPlex: Verified runtime validation of verified cyber-physical system models," *Form. Methods Syst. Des.*, vol. 49, no. 1-2, pp. 33–74, 2016, selected papers from RV'14.

[3] H. Torfah, S. Junges, D. J. Fremont, and S. A. Seshia, "Formal analysis of ai-based autonomy: From modeling to runtime assurance," in *RV*, 2021, pp. 311–330.

[4] S. Jha, T. Sahai, V. Raman, A. Pinto, and M. Francis, "Explaining AI decisions using efficient methods for learning sparse boolean formulae," *J. Autom. Reason.*, vol. 63, no. 4, pp. 1055–1075, 2019.

[5] D. Bayani and S. Mitsch, "Fanoos: Multi-resolution, multi-strength, interactive explanations for learned systems," in *VMCAI*, 2022, pp. 43–68.

[6] A. Platzer, "Differential game logic," *ACM Trans. Comput. Log.*, vol. 17, no. 1, pp. 1:1–1:51, 2015.

[7] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, "KeYmaera X: An axiomatic tactical theorem prover for hybrid systems," in *CADE*, 2015, pp. 527–538.

[8] A. Müller, S. Mitsch, W. Retschitzegger, W. Schwinger, and A. Platzer, "Tactical contract composition for hybrid system component verification," *STTT*, vol. 20, no. 6, pp. 615–643, 2018.

[9] S. Lunel, S. Mitsch, B. Boyer, and J. Talpin, "Parallel composition and modular verification of computer controlled systems in differential dynamic logic," in *FM*, 2019, pp. 354–370.

[10] A. Müller, S. Mitsch, W. Schwinger, and A. Platzer, "A component-based hybrid systems verification and implementation tool in KeYmaera X (tool demonstration)," in *CyPhy*, 2018, pp. 91–110, selected papers.

[11] R. Banach, M. J. Butler, S. Qin, N. Verma, and H. Zhu, "Core hybrid Event-B I: single hybrid Event-B machines," *Sci. Comput. Program.*, vol. 105, pp. 92–123, 2015.

[12] S. M. Loos and A. Platzer, "Differential refinement logic," in *LICS*, 2016, pp. 505–514.

[13] S. Mitsch, J.-D. Quesel, and A. Platzer, "Refactoring, refinement, and reasoning: A logical characterization for hybrid systems," in *FM*, 2014, pp. 481–496.

[14] A. Platzer, "A complete uniform substitution calculus for differential dynamic logic," *J. Autom. Reas.*, vol. 59, no. 2, pp. 219–265, 2017.

[15] S. Mitsch and A. Platzer, "Verified runtime validation for partially observable hybrid systems," *CoRR*, vol. abs/1811.06502, 2018.

[16] ——, "A retrospective on developing hybrid systems provers in the KeYmaera family - A tale of three provers," in *Deductive Software Verification: Future Perspectives*, ser. LNCS, W. Ahrendt, B. Beckert, R. Bubel, R. Hähnle, and M. Ulbrich, Eds. Springer, 2020, vol. 12345, pp. 21–64.

[17] S. Mitsch, "Implicit and explicit proof management in KeYmaera X," in *Proceedings of the 6th Workshop on Formal Integrated Development Environment, F-IDE@NFM 2021, 24-25th May 2021*, 2021, pp. 53–67.

[18] A. Platzer and Y. K. Tan, "Differential equation invariance axiomatization," *J. ACM*, vol. 67, no. 1, pp. 6:1–6:66, 2020.

[19] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *HSCC*, 2004, pp. 477–492.

[20] D. Liberzon, *Switching in Systems and Control*, ser. Systems & Control: Foundations & Applications. Birkhäuser, 2003.

[21] Y. K. Tan, S. Mitsch, and A. Platzer, "Verifying switched system stability with logic," in *HSCC*. ACM, 2022, pp. 1–11.

[22] R. Cleaveland, S. Mitsch, and A. Platzer, "Formally verified next-generation airborne collision avoidance games in ACAS X," *Transactions on Embedded Computing Systems*, 2022.

[23] N. Fulton, S. Mitsch, R. Bohrer, and A. Platzer, "Bellerophon: Tactical theorem proving for hybrid systems," in *ITP*, 2017, pp. 207–224.

[24] N. Fulton and A. Platzer, "A logic of proofs for differential dynamic logic: Toward independently checkable proof certificates for dynamic logics," in *CPP*, 2016, pp. 110–121.

[25] T. Lecomte, D. Déharbe, P. Fournier, and M. Oliveira, "The CLEARSY safety platform: 5 years of research, development and deployment," *Sci. Comput. Program.*, vol. 199, p. 102524, 2020.

[26] Y. Selvaraj, J. Krook, W. Ahrendt, and M. Fabian, "On how to not prove faulty controllers safe in differential dynamic logic," *CoRR*, vol. abs/2207.05854, 2022.