Towards Foundational Verification of Cyber-physical Systems

(Invited Paper)

Gregory Malecha Daniel Ricketts Mario M. Alvarez Sorin Lerner
University of California, San Diego
La Jolla, California 92037
Email: {gmalecha,daricket,mmalvarez,lerner}@cs.ucsd.edu

Abstract—**The safety-critical aspects of cyber-physical systems motivate the need for rigorous analysis of these systems. In the literature this work is often done using idealized models of systems where the analysis can be carried out using high-level reasoning techniques such as Lyapunov functions and model checking. In this paper we present VERIDRONE, a foundational framework for reasoning about cyber-physical systems at all levels from high-level models to C code that implements the system. VERIDRONE is a library within the Coq proof assistant enabling us to build on its foundational implementation, its interactive development environments, and its wealth of libraries capturing interesting theories ranging from real numbers and differential equations to verified compilers and floating point numbers. These features make proof assistants in general, and Coq in particular, a powerful platform for unifying foundational results about safety-critical systems and ensuring interesting properties at all levels of the stack.**

I. INTRODUCTION

Errors in cyber-physical systems can lead to disastrous consequences. Classic examples date back to the Therac-25 radiation incidents in 1987 [1] and the Ariane 5 rocket crash in 1996 [2]. More recently, Toyota's unintended acceleration bug was caused by software errors [3], and certain cars were found vulnerable to attacks that can take over key parts of the control software, even allowing attackers to remotely disable the brakes [4], [5]. Pacemakers were also found vulnerable to attacks that can cause deadly consequences for the patient [6].

The importance of safety for cyber-physical systems has motivated a lot of work to improve their reliability. Techniques include model checking [7]–[10], hybrid automata [11], [12], interactive theorem proving [13]–[17], control theory [18]–[22], and domain specific languages [23]. While these techniques are a marked improvement, previous work in these areas fall short of a complete solution for at least one of four reasons. First, many prior tools and techniques are customized to prove particular classes of properties, for example stability or safety properties, but cannot provide whole-system verification results. Second, most tools and techniques are developed in isolation and the lack of a formal connection between the individual tools and formalisms makes it difficult to combine their results in a rigorous way. Third, many prior approaches work on a *model* of the cyber-physical system, rather than the *code* that implements the model. This leaves open the possibility of bugs in the implementation. In cyber-physical systems this discrepancy can be particularly problematic due both to timing characteristics of real-time systems and discrete approximations, for example computing with floating point (in the code) rather than real numbers (in the model). Fourth, the guarantees provided by prior work often come in the form of proofs produced by *unverified* decision procedures such as SMT solvers, complex math solvers, or custom-built analyzers, thus still leaving the possibility of bugs in the verification tool themselves.

In this paper, we present a brief overview of our VERIDRONE project, which aims to address the above limitations by building a *foundational* verification infrastructure for cyber-physical systems in the Coq proof assistant [24]. Coq provides a trustworthy substrate in which to state and prove interesting properties in a rigorous and foundational way. We believe that this substrate is the key to solving the above problems. Using Coq's rich logic we can express and verify a wide range of properties across many levels of abstraction. We can also connect the abstractions used by different tools making it possible to compose properties established by different systems. Further, by leveraging existing Coq libraries for programming languages and compilers, we can formally connect high-level models of cyber-physical systems to the concrete code that implements these models. Finally, the trustworthiness of all of our results can be traced back to Coq's small trusted core providing high assurance of even the most complex properties.

Outline. We begin with an example showing our use of VERIDRONE to foundationally verify a simple runtime monitor in the context of quadcopters (Section II). Using this example as a springboard, we then describe four broad directions that can be explored in the context of foundational verification of cyber-physical systems:

- *Properties*: Section IV describes some interesting properties that we can formalize in VERIDRONE.
- *Proofs*: Section V describes proof techniques for establishing these properties in a foundational setting.
- *Compilation*: Section VI describes the challenges associated with compiling cyber-physical controllers down to a computational platform in a reliable way.
- *Uncertainty*: Section VII describes the challenges of dealing with uncertainty due to sensors, actuators, and models of the physical world.

All of our work is available on the VERIDRONE website:

http://veridrone.ucsd.edu/

II. Case Study: Runtime Monitors

We begin by considering building runtime monitors to enforce safety properties. In cyber-physical systems, runtime monitors are a lightweight way to gain formal guarantees of the behavior of unverified control software. These runtime monitors sit between the control software and the actuators and ensure that the outputs from the control software are "safe" with respect to some property, for example staying within a geo-fence. If the outputs from the unverified control software might cause the system to violate the property, the monitor intervenes and adjusts the outputs accordingly.

In VeriDrone, we specify and verify cyber-physical systems in the style of Lamport [25] using discrete-time linear temporal logic. Within this formalism we have defined an abstraction (called $\mathtt{Sys}$) to specify the transitions of timed hybrid systems. For example, consider a monitor that enforces an upper bound on velocity ($\texttt{MAX\_VEL}$). Letting $\mathtt{v}$ be the velocity, $\mathtt{a}$ the acceleration, and $\mathtt{a}_p$ the acceleration proposed by the existing control software, this velocity monitor can be specified as follows:

$\texttt{Def}\ \mathtt{D} := (\mathtt{v} + \mathtt{a}_p * \Delta \le \texttt{MAX\_VEL} \wedge \mathtt{a}' = \mathtt{a}_p) \vee \mathtt{a}' = 0.$
$\texttt{Def}\ \mathcal{W} := \dot{v} = \mathtt{a} \wedge \dot{a} = 0.$
$\texttt{Def}\ \texttt{VelMon} := \texttt{Sys}_\Delta\ \mathtt{D}\ \mathcal{W}.$

Informally, $\mathtt{D}$ represents the discrete part of the system (including the monitor), $\mathcal{W}$ represents the differential dynamics of the world, and Δ captures an upper bound on the worst-case execution time of $\mathtt{D}$. The logic of the discrete transition states that if the proposed acceleration $\mathtt{a}_p$ will keep the system safe until the next time the monitor runs ($\mathtt{v} + \mathtt{a}_p * \Delta \le \texttt{MAX\_VEL}$) it is safe ($\mathtt{a}' = \mathtt{a}_p$). Otherwise, the monitor issues the safe acceleration ($\mathtt{a}' = 0$). Here, we use prime to denote the value of a variable after a transition. Finally, $\mathcal{W}$ expresses the dynamics of velocity in a single dimension. We use Coq to prove that this is an abstraction of the full dynamics.

It is important to note that the value of a_p (the proposed acceleration) is unconstrained, which represents that we impose no assumptions on the existing, unverified control software. This means that the monitor ensures safety regardless of what the existing control software does.

Using temporal logic we can state and prove properties of systems such as the velocity monitor. For example, the correctness criterion for the velocity monitor states that if the system starts in a region (described by $\mathtt{I}$) and evolves according to the specification of the velocity monitor, then the system will never exceed the bound on velocity. In VeriDrone, we state this property using the following formula (the $\Box$ represents "always"):

$\mathtt{I} \wedge \Box\ \texttt{VelMon} \vdash \Box\ (\mathtt{v} \le \texttt{MAX\_VEL})$

Describing both the system and the property within temporal logic makes it easy to reason about both. Temporal induction lies at the heart of verifying properties such as this one, and indeed most properties. The following Sys-Ind rule specializes the principle of temporal induction for our $\mathtt{Sys}$ abstraction, allowing us to avoid repeatedly proving some of the administrative obligations that arise from timing.

$$\frac{\begin{array}{c}\mathtt{P} \ \wedge\ 0 \le \tau \le \Delta \ \wedge\ \mathtt{D} \ \wedge\ 0 \le \tau' \le \Delta \ \vdash\ \mathtt{P}' \\ \mathtt{P} \ \wedge\ \tau \le \Delta \ \wedge\ \texttt{Cont}(\mathcal{W} \wedge \dot{\tau} = -1 \wedge \tau \ge 0) \ \vdash\ \mathtt{P}'\end{array}}{\mathtt{P} \ \wedge\ \Box\ (\texttt{Sys}_\Delta\ \mathtt{D}\ \mathcal{W}) \ \vdash\ \Box\ \mathtt{P}}\ \textsc{Sys-Ind}$$

In this definition, τ represents the timer tracking the amount of time remaining before the discrete component must run, and $\texttt{Cont}\ \mathcal{W}$ specifies the continuous transition. We use $\mathtt{P}'$ to denote the formula $\mathtt{P}$ with all variables replaced with their primed version.

By working in Coq's expressive logic, we are able to prove this (and other) specialized reasoning principles foundationally. Crucially, we obtain the benefits of customizable domain-specific reasoning without jeopardizing soundness.

III. Foundational Proof Assistants

Coq is an interactive proof assistant that allows users to write specifications, programs, and proofs in a rich logic based on type theory. While the use of type theory is not essential, the richness of the logic is crucial because it makes it natural to express interesting properties, e.g. stability, without relying on complex encodings. Whereas other systems might define abstractions such as temporal logic or hybrid systems axiomatically (i.e.: by stating axioms that are assumed to hold), in VeriDrone we define these abstractions from first principles in Coq, and then prove theorems about them (rather than assuming they hold). Doing the verification from first principles in this way provides two benefits. First, it gives our results a solid foundation in Coq's own logic ensuring that our reasoning is sound. This comes for free when working in Coq since all results are justified by independently checkable proof witnesses. Second, building within Coq allows us to use theories developed in it, for example, the theory of real arithmetic or the semantics of C programs.

To see what we mean, let us dive into the definition of the $\mathtt{Sys}$ abstraction that our velocity monitor uses.

$\texttt{Def}\ \texttt{Sys}_\Delta\ \mathtt{D}\ \mathcal{W} :=$
$\quad(\mathtt{D} \wedge 0 \le \tau' \le \Delta) \vee \texttt{Cont}\ (\mathcal{W} \wedge \dot{\tau} = -1 \wedge \tau \ge 0).$

In this (stylized) Coq definition, we state explicitly that a $\texttt{Sys}_\Delta$ transition is either a transition of the discrete action (D) or a transition of the continuous dynamics ($\mathcal{W}$). The formula $\mathcal{W}$ is a predicate over the variables of a system and their derivative, allowing one to specify constraints on the evolution that mention the time derivatives of the state of the system. The extra τ variable encodes a stop-watch that is re-initialized when the discrete program runs and counts down in lockstep with time during the continuous transition. The final assertion in the $\texttt{Cont}$ clause ensures that the value of the stop-watch never goes negative ensuring that the discrete transition runs at least once every Δ time.

Explicitly defining abstractions, such as $\mathtt{Sys}$, allows us to prove the rules about them as Coq theorems. Our formalism, however, does not bottom out at $\mathtt{Sys}$. We defined $\texttt{Cont}$ explicitly in Coq leveraging Coq's theory of analysis. Doing this allowed us to foundationally prove, rather than axiomatize, Platzer's differential induction proof technique [26] for reasoning about continuous transitions. To be concrete, the less-than case of the property is the following:

$$\frac{\mathcal{W} \ \vdash\ \dot{a} \le \dot{b}}{\mathtt{a} < \mathtt{b} \ \wedge\ \texttt{Cont}(\mathcal{W}) \ \vdash\ (\mathtt{a} < \mathtt{b})'}\ \textsc{Diff-Ind-}<$$

This formalizes the idea that $\mathtt{a} < \mathtt{b}$ is preserved by a continuous transition $\mathcal{W}$ if $\mathcal{W}$ ensures that the derivative of $\mathtt{a}$ is bounded by the derivative of $\mathtt{b}$.

While this proof had previously been done on paper, ours is the first fully foundational version of this proof that we are aware of and it *connects directly to our definitions*; there is no gap where errors could have crept in. This is important because several of our early adaptations of differential induction were unsound for subtle reasons and it was only while completing the proof that we discovered (and fixed) the problems.

IV. Properties

The richness of our formalism makes it possible to apply VeriDrone to reason about a range of interesting properties.

Monitor Composition. Beyond the individual monitors that we presented in Section II we also developed a theory of monitor composition [27] by building on Coq's higher-order logic. We developed (and proved) theorems that allow us to perform spatial transformations, e.g. enforcing a lower-bound on velocity instead of an upper bound, and to combine individual monitors in both a conjunctive and disjunctive style.

Using spatial transformation and conjunctive composition allowed us to build a monitor that guarantees that the quadcopter stays within a cube by (mostly) composing proofs of our simple monitors. Carrying this proof all the way down to the physical dynamics of the quadcopter (based on roll, pitch, yaw, and total thrust) required that we reason about highly non-linear equations including trigonometric functions. Our reasoning even includes enforcing a "small angle constraint" ensuring that the quadcopter remains roughly vertical where the dynamics are more predictable. Doing this sort of reasoning within completely automated tools would likely be difficult since automated solvers struggle with non-linear dynamics. Using an interactive proof assistant makes it tractable.

On top of the development of the cube, our composition theorems make it trivial to enforce any "pixelized" shape. For example, we can ensure that the quadcopter does not come within 10 feet of the pilot despite being able to fly over and around the pilot.

Our composition theorems crucially rely on Coq's higher-order logic. While the base monitors took longer to develop than they might in another system, the ability to foundationally compose these individual components turns them into building blocks for larger verification efforts.

Stability & Robustness. We have also explored verification of stability [28] and robustness properties [29] in VeriDrone. The expressiveness of our temporal logic makes it relatively straightforward to state both properties. For example, we can state stability with the following definition:

```
Def LyapunovStable x :=
  ∃ x₀ α, α > 0 ∧ x = x₀ ∧ □ (|x| <= |x₀| ∗ α).
```

Stability is a trace predicate, describing the boundedness of the value x over time. While proving stability manually can be quite painful, Coq's rich logic makes it possible to formalize (and verify) the theory of Lyapunov functions and use them to foundationally verify stability of our models. This technique drastically simplifies proving stability for our systems.

Robustness is an interesting property that essentially allows us to "blame violations of properties on disturbances." For example, it allows us to formally prove that any violation of the boundary can be directly attributed to random forces such as wind. Tackling this problem presented interesting challenges because it required that our models represent non-deterministic forces that we did not model in earlier developments.

Our work on robustness up until this point has just scratched the surface, and we believe that a promising future direction is to understand how robustness can allow us to vertically compose systems. For example, can we use robustness to compose a safety monitor that outputs attitudes with an attitude controller?

High-level properties. Beyond classic control theory properties, there are higher-level properties that one might be interested in proving, such as: collision avoidance; following a path; and always having enough battery to go back home. Even more aggressively, one might also want to look at proving correctness properties about application-specific controllers, for example controllers that deliver packages, survey geographic areas, film sporting events, or help with rescue operations. Solving these problems requires both developing formal definitions of correctness and building techniques for reasoning about them. Crucially, working in Coq's expressive logic will not limit the types of properties we can express or the reasoning techniques we employ.

V. Proof Techniques

One of the perceived drawbacks of working in such an expressive logic is that verification often requires manual intervention. This may initially give researchers (and practitioners) pause in adopting proof assistants given the wealth of more automated techniques such as hybrid model checking. However, even in this comparison, proof assistants provide some very significant benefits. For example, the richness of Coq's logic makes it possible to implement automated tools and verify them. Previous work has shown how to develop and verify model checkers within proof assistants [30]. While these formalisms rely on automata-based formalisms, we can formalize the connection to our logic-based formalism foundationally in Coq. This connection would allow us to take formal results (justified by automated tools) and compose them with results that are beyond the scope of fully automated tools in a completely foundational way.

Because of the interaction with the physics of the real world, reasoning about cyber-physical systems often boils down to real-valued mathematics. While Coq provides some automation for dealing with real arithmetic in a fully foundational way [31], Coq's procedures can sometimes be slow or incomplete. However, when complete rigor is not necessary, we can still benefit from state-of-the-are solvers that are more efficient and complete by integrating them into Coq, albeit in an unverified way. In particular, we have developed a Coq plugin [32] that allows us to interface with state-of-the-art SMT solvers including Z3 [33], CVC [34], and Polya [35]. Although properties verified by the plugin are not foundational – thus leaving a gap in the otherwise fully foundational verification – current efforts are underway by other groups to build foundational proofs automatically [36], [37]. In the meantime relying on these tools is not uncommon, for example KeYmaera [38] and many other tools [39], [40] trust the results of Z3.

VI. Verified Compilation

By integrating with language formalisms, the VeriDrone framework can be used to verify properties on the actual code that runs on the system. In practice, it is common to rely on unverified translations or "manual inspection" to relate code and its model. However, this kind of translation is exactly the source of problems such as the Patriot Missile floating point bug where 28 American soldiers died due the software's incorrect accounting of floating point inaccuracies [41].

To close this gap we have begun developing tools to connect our models, defined using real numbers and temporal logic, to more traditional programs. We have developed an embedding operator that allows us write imperative code directly within our temporal logic and prove properties of the resulting system.

$$\begin{array}{l}\texttt{Sys}_{\Delta}\ (\texttt{EmbedC}\left\{\begin{array}{l}\texttt{if(v + a}_{p}\star\Delta\texttt{<MAX\_VEL)}\\ \texttt{ a = a}_{p}\texttt{;}\\ \texttt{else a = 0;}\end{array}\right\})\mathcal{W}\\ \vdash \Box(\texttt{v} \leq \texttt{MAX\_VEL})\end{array}$$

Here, EmbedC embeds a C-like program into our temporal logic. The definition of EmbedC ensures that the only actions allowed are the ones that correspond to valid executions of the C program. While we have so far only explored C, our embedding process is language-agnostic, allowing us to embed a variety of languages into our hybrid system formalism.

Crucially, the EmbedC construct allows us to embed the C code from the implementation into our logic, and thus reason about it formally to show that the low-level C code is equivalent property-wise to the high-level model over which our previous properties were proved. To reason about the C-level code, we can leverage a range of verification frameworks such as the Verified Software Toolchain [42] to assist with language-level abstractions. Connecting the model to the code in this way forces us not only to reason about the connection between real and floating point numbers, but also to prove that the C programs are crash-free.

Integrating with the Rest of the Stack. Once we have used the above embedding mechanism to connect our properties from the high-level model to the C-level implementation, we can then push these properties all the way down to machine code by leveraging verification results such as the CompCert verified C compiler [43]. Finally, we can connect our verified code to verified systems software such as the CertiKOS [44] verified hypervisor in a completely foundational way.

Compiling Down to Hardware. Even more aggressively, one might also consider compiling the high-level model directly to hardware in a fully verified way. Compiling down to Verilog has several benefits. First, it would allow us to more accurately reason about timing of the controller, which is an important consideration when interacting with the physical world. For example, we would be able to prove strong bounds on the timing of hardware-generated low-level controllers, while still allowing a higher-level controller to run complex and time-consuming algorithms, such as object recognition, on a separate general purpose processor. Second, compiling controllers all the way down to hardware would make the controllers more resilient to malicious attacks. For example, certain attacks against quadcopters use vulnerabilities in the software stack, such as the communication software, to take control of the software that runs on the quadcopter. By generating specialized hardware, we would make such attacks much harder to mount, which in certain high-assurance scenarios might be beneficial.

VII. Sensors & Uncertainty

All control software running on a cyber-physical system relies on knowledge of the system's state (e.g. current position, attitude, etc). Up until now, we have ignored the uncertainty inherent in this knowledge. Moving to an uncertain world raises two interesting directions for investigation.

First, we would like to formally characterize how our existing systems behave in the presence of uncertainty. For example, building a probabilistic analogue of monitor safety will allow us to formally quantify how the degree of state uncertainty affects the properties that our systems guarantee.

Second, modern cyber-physical systems rely on sophisticated sensor fusion algorithms to combine the knowledge from multiple sensors to estimate the current state. For example, a quadcopter may combine GPS input with data from an accelerometer to tolerate variations in sensed values that might occur due to GPS inaccuracy. We would like to formally reason about the uncertainty in the estimates produced by sensor fusion. Combining these results with the previous results would give us a probabilistic guarantee about the entire system, consisting of both sensor fusion and control software.

Addressing these questions will require adapting our LTL model to support probability. Coq's expressive logic provides a powerful framework for exploring the tradeoffs between existing probabilistic temporal logics [45] and gives us the flexibility to develop new ones if necessary.

VIII. Conclusions

VeriDrone is a foundational framework for research in and verification of cyber-physical systems. Throughout this paper we demonstrated its core principles. Foundational proofs in a general-purpose formalism give a high level of assurance but also make it possible to extend the model to support verification ranging from straightforward safety properties in simple models to robustness and probabilistic properties in more nuanced models with disturbances. Connecting with previous developments such as CompCert and Flocq allows us to connect our high-level models down to the machine code that runs on the quadcopter and leverage purely digital results such as cryptography [46] and verified systems software to gain deeper guarantees. In the future, we believe that foundational frameworks such as VeriDrone will provide a unifying foundation for establishing deep end-to-end properties of the cyber-physicals systems that we bet our lives on.

References

[1] N. G. Leveson and C. S. Turner, "An investigation of the therac-25 accidents," *Computer*, vol. 26, no. 7, pp. 18–41, 1993.

[2] J.-L. Lions *et al.*, "Ariane 5 flight 501 failure," 1996.

[3] S. Kane, E. Liberman, T. DiViesti, and F. Click, "Toyota sudden unintended acceleration," *Safety Research & Strategies*, 2010.

[4] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *Proceedings of IEEE Security and Privacy ("Oakland") 2010.* IEEE Computer Society, May 2010, pp. 447–62.

[5] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of USENIX Security 2011*. USENIX, Aug. 2011, pp. 77–92.

[6] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel, "Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses," in *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. IEEE, 2008, pp. 129–142.

[7] T. A. Henzinger, P. Ho, and H. Wong-Toi, "HYTECH: A model checker for hybrid systems," in *CAV '97*, 1997. [Online]. Available: http://dx.doi.org/10.1007/3-540-63166-6_48

[8] G. Frehse, "PHAVer: algorithmic verification of hybrid systems past HyTech," *STTT*, vol. 10, no. 3, pp. 263–279, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10009-007-0062-x

[9] E. M. Clarke, A. Fehnker, Z. Han, B. H. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald, "Abstraction and counterexample-guided refinement in model checking of hybrid systems," *Int. J. Found. Comput. Sci.*, vol. 14, no. 4, pp. 583–604, 2003. [Online]. Available: http://dx.doi.org/10.1142/S012905410300190X

[10] R. Alur, T. Dang, and F. Ivancic, "Counter-example guided predicate abstraction of hybrid systems," in *TACAS '03*, 2003, pp. 208–223. [Online]. Available: http://dx.doi.org/10.1007/3-540-36577-X_15

[11] T. A. Henzinger, "The theory of hybrid automata," in *LICS '96*, 1996, pp. 278–292. [Online]. Available: http://dx.doi.org/10.1109/LICS.1996.561342

[12] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" *J. Comput. Syst. Sci.*, vol. 57, no. 1, pp. 94–124, 1998. [Online]. Available: http://dx.doi.org/10.1006/jcss.1998.1581

[13] P. Collins, M. Niqui, and N. Revol, "A taylor function calculus for hybrid system analysis: Validation in coq (extended abstract)," 2010.

[14] H. Geuvers, A. Koprowski, D. Synek, and E. van der Weegen, "Automated machine-checked hybrid system safety proofs," in *ITP '10*, 2010, pp. 259–274. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14052-5_19

[15] O. Tveretina, "Towards the Safety Verification of Real-Time Systems with the Coq Proof Assistant," *JAMRIS*, vol. 3, pp. 30–32, 2009.

[16] E. Abraham-Mumm, U. Hannemann, and M. Steffen, "Verification of hybrid systems: formalization and proof rules in pvs," in *ECCS '01*, 2001, pp. 48–57.

[17] N. Vlker, "Towards a HOL Framework for the Deductive Analysis of Hybrid Control Systems," 2000.

[18] D. Liberzon, *Switching in systems and control*. Springer Science & Business Media, 2003.

[19] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *Hybrid Systems: Computation and Control*. Springer, 2004, pp. 477–492.

[20] J. P. Hespanha, "Tutorial on supervisory control," in *Lecture Notes for the workshop Control using Logic and Switching for the 40th Conf. on Decision and Contr., Orlando, Florida*, 2001.

[21] H. Lin and P. J. Antsaklis, "Stability and stabilizability of switched linear systems: a survey of recent results," *Automatic control, IEEE Transactions on*, vol. 54, no. 2, pp. 308–322, 2009.

[22] C. J. Tomlin, J. Lygeros, and S. S. Sastry, "A game theoretic approach to controller design for hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 949–970, 2000.

[23] P. C. Hickey, L. Pike, T. Elliott, J. Bielman, and J. Launchbury, "Building Embedded Systems with Embedded DSLs," in *ICFP'14*. New York, NY, USA: ACM, 2014, pp. 3–9. [Online]. Available: http://doi.acm.org/10.1145/2628136.2628146

[24] Coq Development Team, "The Coq proof assistant reference manual, version 8.5," 2016. [Online]. Available: http://coq.inria.fr/distrib/V8.5/refman/

[25] L. Lamport, "Hybrid systems in TLA+," in *Hybrid Systems*, ser. Lecture Notes in Computer Science, R. Grossman, A. Nerode, A. Ravn, and H. Rischel, Eds. Springer Berlin Heidelberg, 1993, vol. 736, pp. 77–102. [Online]. Available: http://dx.doi.org/10.1007/3-540-57318-6_25

[26] A. Platzer, *Logical analysis of hybrid systems: proving theorems for complex dynamics*. Springer Publishing Company, Incorporated, 2010.

[27] Daniel Ricketts and Gregory Malecha and Sorin Lerner, "Modular Reasoning about Cyber-physical Systems," 2016.

[28] Matthew Chan and Daniel Ricketts and Sorin Lerner and Gregory Malecha, "Formal Verification of Stability Properties of Cyber-physical Systems," 2016. [Online]. Available: http://veridrone.ucsd.edu/papers/coqpl2016.pdf

[29] Daniel Ricketts and Gregory Malecha and Sorin Lerner, "Verifying Robustness of Cyber-Physical Systems," 2016.

[30] H. Amjad, "Combining model checking and theorem proving," Tech. Rep. 601, September 2004.

[31] F. Besson, "Fast reflexive arithmetic tactics the linear case and beyond," in *TYPES*, ser. LNCS. Springer Berlin Heidelberg, 2007, vol. 4502, pp. 48–62. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74464-1_4

[32] G. Malecha, "Coq SMT Check," 2016. [Online]. Available: https://github.com/gmalecha/coq-smt-check

[33] L. De Moura and N. Bjørner, "Z3: An Efficient SMT Solver," ser. TACAS'08/ETAPS'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340. [Online]. Available: http://dl.acm.org/citation.cfm?id=1792734.1792766

[34] C. Barrett and C. Tinelli, "CVC3," in *CAV*. Springer Berlin / Heidelberg, 2007, vol. 4590, pp. 298–302. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-73368-3_34

[35] Jeremy Avigad and Robert Y. Lewis and Cody Roux, "A Heuristic Prover for Real Inequalities," 2016. [Online]. Available: http://www.andrew.cmu.edu/user/avigad/Papers/polya.pdf

[36] F. Besson, P.-E. Cornilleau, and D. Pichardie, *CPP'11*. Springer Berlin Heidelberg, 2011, ch. Modular SMT Proofs for Fast Reflexive Checking Inside Coq, pp. 151–166. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25379-9_13

[37] M. Armand, G. Faure, B. Grégoire, C. Keller, L. Théry, and B. Werner, *CPP'11*. Springer Berlin Heidelberg, 2011, ch. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses, pp. 135–150. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25379-9_12

[38] Andre Platzer, "KeYmaera: A Hybrid Theorem Prover for Hybrid Systems." [Online]. Available: http://symbolaris.com/info/KeYmaera.html

[39] M. Barnett, B.-Y. Chang, R. DeLine, B. Jacobs, and K. Leino, "Boogie: A modular reusable verifier for object-oriented programs," in *Formal Methods for Components and Objects*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, vol. 4111, pp. 364–387. [Online]. Available: http://dx.doi.org/10.1007/11804192_17

[40] K. Leino and P. Rmmer, "A polymorphic intermediate verification language: Design and logical encoding," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, vol. 6015, pp. 312–327. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12002-2_26

[41] "Patriot Missile Software Problem," http://sydney.edu.au/engineering/it/∼alum/patriot_bug.html, accessed: 2015-04-30.

[42] Andrew W. Appel and Robert Dockins and Aquinas Hobor and Josiah Dodds and Xavier Leroy and Sandrine Blazy and Gordon Stewart and Lennart Beringer, *Program Logics for Certified Compilers*, April 2014.

[43] X. Leroy, "Formal verification of a realistic compiler," *Communications of the ACM*, vol. 52, no. 7, pp. 107–115, 2009. [Online]. Available: http://gallium.inria.fr/∼xleroy/publi/compcert-CACM.pdf

[44] L. Gu, A. Vaynberg, B. Ford, Z. Shao, and D. Costanzo, "Certikos: A certified kernel for secure cloud computing," in *Proceedings of the Second Asia-Pacific Workshop on Systems*, ser. APSys '11. New York, NY, USA: ACM, 2011, pp. 3:1–3:5. [Online]. Available: http://doi.acm.org/10.1145/2103799.2103803

[45] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994. [Online]. Available: http://dx.doi.org/10.1007/BF01211866

[46] A. W. Appel, "Verification of a cryptographic primitive: Sha-256," *ACM Trans. Program. Lang. Syst.*, vol. 37, no. 2, pp. 7:1–7:31, Apr. 2015. [Online]. Available: http://doi.acm.org/10.1145/2701415