INVITED: Safety Guards: Runtime Enforcement for Safety-Critical Cyber-Physical Systems

Meng Wu
Virginia Tech, USA

Haibo Zeng
Virginia Tech, USA

Chao Wang
University of Southern California, USA

Huafeng Yu
Boeing Research & Technology, USA

ABSTRACT

Due to their safety-critical nature, cyber-physical systems (CPS) must tolerate faults and security attacks to remain fail-operational. However, conventional techniques for improving safety, such as testing and validation, do not meet this requirement, as shown by many of the real-world system failures in recent years, often with major economic and public-safety implications. We aim to improve the safety of critical CPS through synthesis of runtime enforcers, named *safety guards*, which are *reactive components* attached to the original systems to protect them against catastrophic failures. That is, even if the system occasionally malfunctions due to unknown defects, transient errors, or malicious attacks, the guard always reacts instantaneously to ensure that the combined system satisfies a predefined set of safety properties, and the deviation from the original system is kept at minimum. We illustrate the main ideas of this approach with examples, discuss the advantages compared to existing approaches, and point out some research challenges.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Software and its engineering** → **Formal methods**; *Embedded software*;

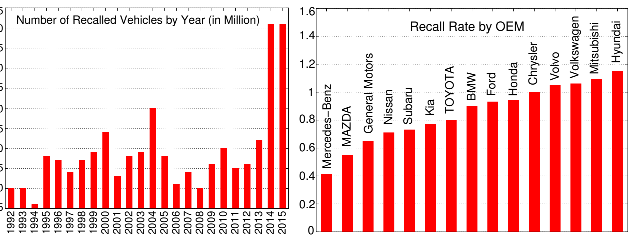
KEYWORDS

Safety and security, reactive synthesis, runtime enforcement, software synthesis, real-time schedulability

ACM Reference format:
Meng Wu, Haibo Zeng, Chao Wang, and Huafeng Yu. 2017. INVITED: Safety Guards: Runtime Enforcement for Safety-Critical Cyber-Physical Systems. In *Proceedings of DAC '17, Austin, TX, USA, June 18-22, 2017,* 6 pages.
DOI: http://dx.doi.org/10.1145/3061639.3072957

1 INTRODUCTION

The industry is facing significant challenges in managing risks for various cyber-physical systems (CPS). In the automotive industry, for example, there is a large number of safety recalls in recent years (Fig. 1) that cost billions of dollars and affected all major original equipment manufacturers (OEMs), with the recall rate generally ≥ 0.5 and sometimes ≥ 1.0 [25, 47]. In 2014 and 2015, for instance, the number of recalled vehicles in the U.S. reached record highs, affecting 51 million vehicles each year [39, 45]. A closer look at the root causes of these safety recalls shows that the situation is deteriorating and the trend will likely continue, for two reasons.



(a) Number of recalled vehicles in each year (1992-2015)

(b) Recalled vehicles (1985–2014) per unit sold (1980–2013)

Figure 1: Recalled vehicles in the U.S. [19, 20, 31, 39, 45].

Verification feasibility: Automotive OEMs are hard pressed to deliver products at a faster pace with lower cost [4]. Vehicles are also increasingly more complex in terms of both the embedded software that routinely exceed 100 million lines of code and the supporting hardware architecture that consists of 50-100 electronic control units (ECUs) and dozens of in-vehicle communication buses [4]. Furthermore, since the supply chain values IP protection, parts are often provided as *blackboxes* [46], even for autonomous driving features that involve multiple parts across the entire vehicle [15]. The combined high system complexity, short time-to-market, and lack of accessibility to vital source code make it *unrealistic or even impossible* to formally verify the system.

Security: When vehicles are under malicious attacks, ensuring safety becomes even harder. Unfortunately, vehicle accessibility is rapidly increasing due to ubiquitous cellular connectivity, Internet access, remote diagnostic/service entry points, and the soon-to-be-available vehicle-to-vehicle and vehicle-to-infrastructure communications. It opens doors for attacks [28] while few of today's systems are designed with security concerns in mind [11]. Thus, one compromised ECU may subject the entire network of ECUs to security risks. For example, in 2015, millions of cars from Chrysler and Tesla were in danger as hackers demonstrated the possibility of remotely controlling a running vehicle [25]. Therefore, security vulnerabilities are becoming a major threat to vehicle safety [5, 18].

Related Work. Broadly speaking, existing methods for improving the safety of critical CPS fall into four categories. Methods in the first category seek to reduce the failure rate using fault-tolerance techniques such as N-modular redundancy [26], N-version programming [1], re-execution [38], and error-correcting codes [17]. They are effective in dealing with faults induced by the environment *but not design defects*. Methods in the second category, such as testing and verification [6, 16], focus on eliminating design defects but suffer from *scalability problems* and thus cannot eliminate all defects. Methods in the third category perform runtime monitoring of critical requirements (i.e., properties) [7, 23]. However, they only check but *do not enforce* these requirements. Methods in the fourth category can enforce properties [12, 40], but they are *fail-stop* as opposed to *fail-operational*, and thus are ill-suited for *reactive systems* such as CPS. For example, the safety kernel in [40] simply halts the system in case of property violations, while the monitor in [12] buffers output strings until they are proved safe and thus causes significant delay in the response time.

In summary, none of the aforementioned methods can correct safety violations *instantaneously*, i.e., in the same control step when the violation occurs, while assuming no knowledge on the implementation details of the system. Furthermore, none of these methods addresses failures caused by security attacks, despite that security risks in CPS are rapidly increasing in recent years [5, 13, 18, 22, 29, 41–43].

Our Approach. We propose to leverage recent advances in *reactive synthesis*, such as Bloem et al. [3, 48], to improve CPS safety. Specifically, we want to ensure that a predefined set of safety-critical properties are always satisfied by the system even in the presence of unknown failures or security attacks. Toward this end, we take the safety properties as input and treat the original system as a blackbox, and automatically synthesize a reactive component called the *safety guard*. The guard behaves as a *runtime enforcer*: at runtime it monitors the input-output behaviors of the (blackbox) system and corrects any property violation instantaneously.

Consider the vehicle system in Fig. 2 (a), where various control subsystems such as braking, steering, engine control, and suspension are implemented on a distributed network of ECUs. Assume that the original system occasionally violates the safety specification. Given the safety specification φ (but *not the original system*), our method automatically generates the guard $\mathcal{S}$ shown in Fig. 2 (b) as output. The guard is a component that regulates the input-output behaviors of the system to avoid safety violations. *At run time*, it takes the input I and output O of the original system as its input, and produces the new output O' such that the combined system always satisfies the safety specification. That is, $\varphi(I, O')$ holds even if $\varphi(I, O)$ is violated occasionally by the original system.

Specifically, our **contributions** in this position paper are:

- We advocate the use of *safety guards* and related synthesis algorithms [3, 48] to enforce the safety of critical CPS.
- We argue that, since the proposed approach requires no knowledge on the implementation details of the system, it is well suited for a wide variety of CPS applications.
- We demonstrate the feasibility of this approach using examples from related industries and point out its advantages over existing approaches.
- Finally, we outline the research challenges, discuss their potential solutions, and suggest some ways of integrating the proposed technique into industrial design practice.

The remainder of the paper is organized as follows. We illustrate the concept of safety guards with examples in Section 2 before formally defining the guard synthesis problem in Section 3. We demonstrate the feasibility and effectiveness of existing synthesis algorithms in Section 4. We discuss the advantages and assumptions of this approach in Section 5. Finally, we present some open research problems and then conclude in Section 6.

2 THE CONCEPT OF SAFETY GUARD

Instead of matching known fault patterns or attack patterns and then taking a predefined set of remedial measures, we choose to enforce critical requirements (*properties*) by automatically synthesizing a safety guard from a formal specification of these properties.

2.1 An Illustrative Example

We use the brake and engine control units in Fig. 2 to illustrate how safety guards can be used to protect against an erroneous behavior similar to the unintended acceleration in the 2009-2011 Toyota recalls [47]. The original system takes as input the positions of brake pedal (`bp`) and accelerator pedal (`ap`) as well as the throttle demand from cruise control (`cc`), and returns as output the control commands for throttle (`tc`) and braking (`bc`). For simplicity, we assume all signals are *Boolean*. Below are four example properties:

- **Req 1:** If the brake pedal is pressed (`bp=1`), the brake should be applied (`bc=1`) and the throttle should be closed (`tc=0`). That is, `bp=1` $\rightarrow$ `bc=1` $\wedge$ `tc=0`;
- **Req 2:** `bp=0` $\wedge$ `ap=1` $\rightarrow$ `bc=0` $\wedge$ `tc=1`;
- **Req 3:** `bp=0` $\wedge$ `ap=0` $\wedge$ `cc=0` $\rightarrow$ `bc=0` $\wedge$ `tc=0`;
- **Req 4:** `bp=0` $\wedge$ `ap=0` $\wedge$ `cc=1` $\rightarrow$ `bc=0` $\wedge$ `tc=1`.

Requirements 1–3 are safety-critical as any violation can lead to fatal consequences. In particular, Req 1 satisfies the basic safety needs for a brake override system to prevent unintended acceleration, as suggested by the U.S. National Highway Traffic Safety Administration (NHTSA) [33]. In contrast, Req 4 is not necessarily safety-critical: when violated, it is unpleasant but can be regarded as a mere non-function of the cruise control. In practice, there can be many additional properties like Req 4, which are not safety-critical and therefore do not need to be enforced by the guard.

First, consider enforcing Requirements 1–3, whose safety guard is the combinational logic shown in Fig. 3 (b)–(c). Here, `tc` and `bc` are output signals of the original system, whereas `tc'` and `bc'` are new output signals of the guard. If there are no safety violations, the guard satisfies that `tc'` = `tc` and `bc'` = `bc`. However, the system, complicated by many other requirements like Req 4, may violate the critical properties. For example, when the accelerator pedal is trapped (`ap` = 1) and the driver presses the brake pedal (`bp` = 1), the system may disrespect Req 1, as indicated by the first three rows of the truth table in Fig. 3 (a). In such cases, the guard instantaneously corrects the output to `bc'` = 1 and `tc'` = 0.

While the guard for Requirements 1–3 can be realized solely in combinational logic, it is not always the case for more complex requirements. For example, Req 1 alone may be replaced by the following set of refined requirements:

- **Req 1.1:** If the brake pedal is pressed before the accelerator, both should be applied (`bc=1` $\wedge$ `tc=1`) to avoid interfering

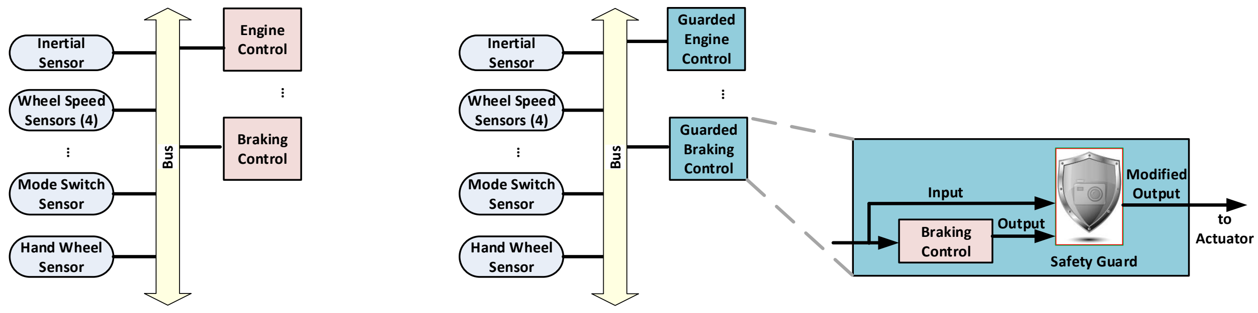


Figure 2: (a) Original vehicle system; (b) New vehicle system protected by automatically synthesized safety guards.

bp	ap	cc	bc	tc	bc'	tc'
1	1	x	1	1	1	0
1	1	x	0	1	1	0
1	1	x	0	0	1	0
1	1	x	1	0	1	0
1	0	x	1	0	1	0
..	..	..	..	..	..	..

bp: brake pedal
ap: accelerator pedal
cc: cruise control
bc: brake command
tc: throttle command
bc': guarded brake command
tc': guarded throttle command

(a)

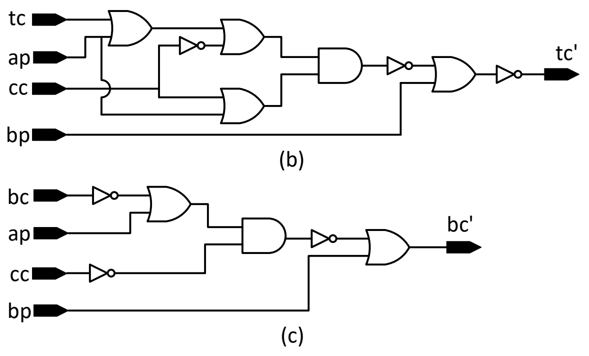


Figure 3: Safety guards for Reqs 1–3: (a) the truth table; (b) guard for the engine control; (c) guard for brake control.

with the driver's true intention in situations such as trailer positioning or starting on a steep slope [33];

- **Req 1.2:** If the brake pedal is pressed together with or after the accelerator, the brake should be applied but the throttle should be closed ($bc=1 \wedge tc=0$);
- **Req 1.3:** If only the brake pedal is pressed ($bp=1 \wedge ap=0$), the brake should be applied but the throttle should be closed ($bc=1 \wedge tc=0$).

Requirements 1.1 and 1.2 involve temporal operators [9], resulting in safety guards generally represented as finite automata. Fig. 4 shows the automatically synthesized guard to enforce Req 1.2. Although in practice the guards for brake control and engine control may be implemented separately in a distributed network of ECUs, here, we represent them in one automaton for ease of comprehension. In the automaton, s0, s1, s2 are safe states where the guard does not change the output of the design (hence $eq = 1$); $s2'$ is a state where the guard needs to correct the erroneous output.

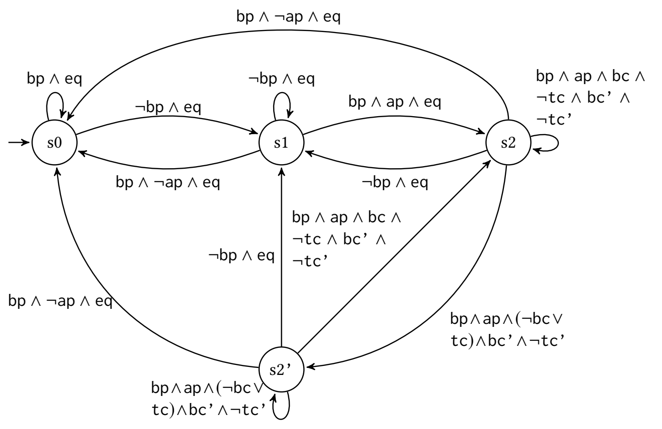


Figure 4: Safety guard for enforcing Req 1.2, where the signal "eq" is defined as $\neg(tc \oplus tc') \wedge \neg(bc \oplus bc')$.

Table 1: Simulation trace for the guard in Fig. 4.

Step	0	1	2	3	4	5	6	7	8	9	10	11
State in guard (Fig. 4)	S0	S1	S1	S2	S2	S1	S1	S2'	S2'	S0	S1	...
Input (bp, ap)	10	00	01	11	11	00	01	11	11	10	00	...
System Output (bc, tc)	10	00	01	10	10	00	01	11↯	11↯	10	00	...
Guard Output (bc', tc')	10	00	01	10	10	00	01	10	10	10	00	...

To understand how the guard corrects the erroneous output, refer to the simulation trace in Table 1. When the system behavior satisfies the specification (Steps 0–6), the guard's output remains the same as the design's output. However, in Steps 7 and 8 where the brake pedal is pressed after the accelerator, the throttle control output is unexpectedly open ($tc = 1$), the guard reacts instantaneously and corrects it. Furthermore, the guard stops interfering as soon as the design is back to normal—the guard's output remains the same as the design's output starting from Step 9.

2.2 Preventing Costly Recalls

We argue that the use of safety guards in vehicle control subsystems can help avoid safety recalls due to problems similar to the Toyota unintended acceleration. Table 2 shows a partial list of similar recalls in recent years, together with descriptions of the violated

Table 2: Partial List of Recent Automotive Recalls and Violated Critical Properties (Extended from [7]).

Year	OEM	# Recalled Vehicles	Violated Critical Property	Source
2010	Toyota	7.5M	See Section 2.1	[33, 47]
2011	Honda	2.5M	Cruise control shall be turned off if brake pedal or cancel button is pressed	[32]
2011	Jaguar	18K	The engine speed threshold allowed for executing transmission shifts shall be sufficiently low	[24]
2013	Honda	350K	The Vehicle Safety Assist System shall not apply the brakes without driver's input	[34]
2014	Ford	595K	The airbags must deploy within 10ms of sudden acceleration along any axis	[35]
2015	Toyota	625K	While moving, the hybrid system can only be shut down through the ignition switch	[37]
2015	Ford	432K	The engine shall shut off after the car key is removed	[36]
2015	Chrysler	1.4M	Only messages in scope of the target ECU should be processed	[25]
2016	GM	4.3M	The software controlling the airbags shall always deploy them during a crash	[14]

safety properties. Our preliminary investigation shows that all these critical properties can be specified in ways similar to our example for the engine and braking system in Section 2.1. Hence, with automatically synthesized safety guards, such recalls may be prevented.

3 SYNTHESIS OF SAFETY GUARDS

In this section, we formally define the guard synthesis problem. Recall that the guard must be *reactive*, i.e., capable of correcting the violation in the same control step. Furthermore, it must be constructed solely from the safety properties denoted φ, regardless of the implementation details of the design $\mathcal{D}$ (the system may be a *blackbox*). This ensures the simplicity of the guard as well as the scalability of the guard synthesizer. However, to satisfy φ, the guard must not generate the output O' arbitrarily; it must *guess the moves* of the design while correcting the erroneous output O, to minimize the deviation between O' and O.

This may be achieved by solving a two-player safety game as shown in [3, 48], where one player represents the design $\mathcal{D}$ that controls the values of O and the other player represents the guard $\mathcal{S}$ that controls the value of O'. The goal of the guard is making sure that (a) $\varphi(I, O')$ is always satisfied, and (b) the deviation from O to O' is minimized, which means the guard must keep the difference between O and O' as small as possible. Intuitively, $\mathcal{S}$ must start interfering only when it has to, and stop interfering as soon as $\varphi(I, O)$ is satisfied by $\mathcal{D}$ again.

Design and Guard. The design is a reactive system represented by a Mealy machine $\mathcal{D} = (Q, q_0, \Sigma_I, \Sigma_O, \delta, \lambda)$, where Q is a set of states, $q_0 \in Q$ is the initial state, Σ_I and Σ_O are the input/output symbols, $\delta : Q \times \Sigma_I \rightarrow Q$ is the transition function, and $\lambda : Q \times \Sigma_I \rightarrow \Sigma_O$ is the output function. The guard is also a reactive system $\mathcal{S} = (Q', q'_0, \Sigma, \Sigma_O, \delta', \lambda')$, where $\Sigma = \Sigma_I \cup \Sigma_O$ is the set of new input symbols. The composition $\mathcal{D} \times \mathcal{S}$ is $(\hat{Q}, \hat{q_0}, \Sigma_I, \Sigma_O, \hat{\delta}, \hat{\lambda})$, where $\hat{Q} = Q \times Q'$, $\hat{q_0} = (q_0, q'_0)$, $\hat{\delta}((q, q'), \sigma_I) = (\delta(q, \sigma_I), \delta'(q', (\sigma_I, \lambda(q, \sigma_I))))$ is the transition function, and $\hat{\lambda}((q, q'), \sigma_I) = \lambda'(q', (\sigma_I, \lambda(q, \sigma_I)))$ is the output function.

We assume that both δ and λ of the design $\mathcal{D}$ are unknown, and our goal is to synthesize δ' and λ' of the guard $\mathcal{S}$ directly from the safety specification φ.

Specification. The *specification* φ is a set of safety properties that can be represented by an automaton $\varphi = (Q, q_0, \Sigma, \delta, F)$, where $\Sigma = \Sigma_I \cup \Sigma_O$, $\delta : Q \times \Sigma \rightarrow Q$, and $F \subseteq Q$ is the set of safe states. A trace $\overline{\sigma} = \sigma_0\sigma_1 \ldots \in \Sigma^\omega$ satisfies φ if the induced state sequence $\overline{q} = q_0 q_1 \ldots$, where $q_{i+1} = \delta(q_i, \sigma_i)$, visits the safe states only. The language $L(\varphi)$ is the set of all traces satisfying φ.

Synthesizing the Safety Guard. Let $L(\mathcal{D})$ be the set of all traces generated by $\mathcal{D}$. We want to verify that $L(\mathcal{D}) \subseteq L(\varphi)$. However, this can be difficult or impossible if $\mathcal{D}$ is large and complex, despite that φ is often small. Instead, we choose to synthesize the guard $\mathcal{S}$ that ensures $L(\mathcal{D} \times \mathcal{S}) \subseteq L(\varphi)$ even if $L(\mathcal{D}) \nsubseteq L(\varphi)$.

4 FEASIBILITY EVALUATION

There are two existing methods that can be leveraged to synthesize guards from safety specifications.

k-Stabilizing Safety Guard [3]. The guard synthesized by this method can enforce safety properties and minimize the deviation of the guard from the design, under the notation of k-stabilization. That is, when a property violation by $\mathcal{D}$ becomes unavoidable, the output of the guard $\mathcal{S}$ is allowed to deviate from $\mathcal{D}$'s output for at most k consecutive control steps. Only after these k steps, the next violation is tolerated. If a second violation occurs within the k-step recovery period, the guard enters a *fail-safe* mode, where it still enforces φ but stops minimizing the deviation.

Handling Burst Error [48]. *Burst error* is a contiguous sequence of errors, e.g., due to radiation interference or RAM failure. Unfortunately, the method in [3] cannot handle burst error: For any k-stabilizing guard where $k > 1$, it may stop minimizing the deviation under burst error. The method in [48] avoids this problem by insisting that the guard never enters the fail-safe mode. That is, if a second violation occurs within the k-step recovery period, the guard will continue minimizing the difference between O' and O. This makes it more suitable for automotive applications.

In the remainder of this section, we use the synthesis algorithm in [48] to construct guards for the following safety specifications:

- Toyota powertrain control verification benchmark [21];
- Properties for vehicle engine and brake controls [33];
- Traffic light controller from the VIS model checker [10];
- Properties for an ARM AMBA bus arbiter [2].

Properties from the Toyota powertrain control verification benchmark are based on the model of a fuel control system [21], specifying the requirements in various operational modes. Originally, they were represented in signal temporal logic (STL). We translated them to linear temporal logic (LTL) by replacing the numerical predicates with Boolean variables. For example, $x + y > 3.53$ is replaced by $P_1 = 1$ and $z^2 \leq 0.06$ by $P_2 = 1$, where x, y, z are real-valued

Table 3: Evaluation results for Benchmarks.

Property φ	Spec. $\|\varphi\|$	Guard $\|\mathcal{S}\|$	Time (s)
Toyota powertrain [21]	23	38	0.3
Engine and brake ctrl [33]	5	7	0.1
Traffic light [10]	4	7	0.2
AMBA G1+2+3 [2]	12	22	0.1
AMBA G1+2+4 [2]	8	78	2.2
AMBA G1+3+4 [2]	15	640	97.6
AMBA G1+2+3+5 [2]	18	1405	61.8
AMBA G1+2+4+5 [2]	12	253	472.9
AMBA G5+6 [2]	16	63	0.2
AMBA G5+7 [2]	16	362	43.4

variables, and P_1 and P_2 are Boolean variables. Properties from the engine and brake controls include Req 1.1–Req 3 in our illustrative example. Properties from the traffic light control [10] are for the safety of a crossroads traffic light. Properties from AMBA [2] are combinations of various safety properties that must be satisfied by the ARM bus arbiter.

Table 3 shows the evaluation results for these benchmarks, including the name, specification size $|\varphi|$ (in terms of the number of states in φ), guard size $|\mathcal{S}|$ (in terms of the number of states in $\mathcal{S}$), and the synthesis time. All experiments are performed on a computer with 3.1GHz CPU and 4GB RAM. The results show that the guard synthesizer scales well for frequently encountered properties in critical control subsystems: it took only a few minutes for all benchmarks.

The examples and results presented so far demonstrate the applicability of safety guards to CPS, as well as the generality of the synthesis algorithm. In the remainder of this paper, we focus on discussing the benefits and limitations of the proposed approach.

5 ADVANTAGES AND ASSUMPTIONS

The advantages of using safety guards in the design and implementation of safety-critical CPS are as follows:

(1) The safety guard is *correct-by-construction* since it is automatically synthesized from the specification of safety properties.

(2) The synthesis procedure is not tied to any particular system or architecture and thus is more *generally applicable* than custom-designed solutions.

(3) The synthesis procedure is *scalable* compared to existing testing/verification techniques, since the guard has to regulate only a small set of properties (whose violations cause catastrophe) as opposed to the system's functional specification.

(4) Due to the same reason as in (3), the safety guard is significantly *smaller* than the design and thus can be hardened for fault-tolerance and security protection at a lower cost.

(5) The use of safety guards may *simplify the development*. Separating safety enforcement from the design allows developers to focus on crafting the most efficient design without worrying about safety violations.

(6) Safety guards are well suited for autonomous and semi-autonomous systems, which are heavily dependent on AI/ML techniques that continuously evolve with new data and thus are hard to analyze and verify.

(7) Safety guards are well suited for improving potentially buggy *legacy* components, which may not be maintainable due to lack of documentation or change of development teams.

(8) The use of safety guards may *simplify safety certification*. Instead of certifying the design, we can certify the guard, which is significantly smaller and changes less often, thus reducing the certification cost.

Indeed, it can help solving two challenges in current certification practice [50]. One is the lack of access to source code, which generates potential conflicts between OEMs and suppliers. The safety guard addresses this issue as it does not require any knowledge on the design. The other is the lack of compositional certification: the system needs to be re-certified from scratch with any changes. The safety guard addresses this issue as it will remain the same (and hence no recertification is needed) as long as the input-output relationship of the system does not change.

(9) With simplified certification, we hope to expedite the advancement of safety regulation in certain industries. For example, right now, ISO 26262 is an automotive industry standard without a regulator. Whereas in other safety-critical industries, the regulator's role is essential in enforcing safety standards and nurturing a "safety-first" culture. Examples include the U.S. Food and Drug Administration (FDA) for medical devices, Nuclear Regulatory Commission (NRC) for nuclear power plants, and Federal Aviation Administration (FAA) for aircrafts. By lowering the certification cost, we hope to remove the cost concerns and allow the certification authority to emerge quickly [50].

However, we are mindful that there is no panacea, and the proposed approach is likely only part of the solution for ensuring CPS safety. Furthermore, to obtain the full benefits of safety guards, we need to make sure that **(a)** the guards are correctly implemented in hardware or software, and **(b)** the input signals of the guards are trustworthy. Nevertheless, both of these two assumptions are easily satisfied by current industry practice.

Specifically, the *first assumption* can be satisfied in two steps. The assumption of a correct implementation fits well with the current trend of adopting rigorous model-based design process in industry. The automatically synthesized safety guard is represented as a finite automaton, which is similar to the ones modeled in Stateflow from MathWorks: implementations can be automatically generated to match these automata (hence avoiding implementation bugs). Then, the implementation may be *hardened* to protect against soft errors as well as security attacks. The *second assumption* can be satisfied by the typical distributed control systems upon which the guards are deployed [23]. In these systems, critical sources of input data may be protected using various fault-tolerance and security-enhancement mechanisms.

6 SUMMARY AND RESEARCH OPPORTUNITIES

We have proposed the use of safety guards for protecting critical CPS against unknown faults, and demonstrated their feasibility and effectiveness. Still, there are many open problems.

First, in practice, there is a lack of formal specification for safety properties, although we expect that the recently proposed safety standard, ISO26262, will ease the concern over time. Furthermore, automated software tools and techniques for mining temporal logic specifications [49] from legacy designs will help.

Second, the scalability of the guard synthesis algorithms shall be improved. For example, it would be helpful if the safety guard can be synthesized by exploiting compositionality. That is, instead of synthesizing a monolithic guard $\mathcal{S}$ to protect against all safety violations, we synthesize a series of smaller safety guards $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \ldots$, whose composition is functionally equivalent to $\mathcal{S}$.

Third, the guard synthesis algorithms need to be extended to handle a more diverse set of properties as well as deviation metrics. Furthermore, existing algorithms rely on solving two-player safety games [3] where the guard $\mathcal{S}$ acts against the system $\mathcal{D}$. However, in reality, $\mathcal{S}$ and $\mathcal{D}$ are mostly cooperating and occasionally opposing, since errors in $\mathcal{D}$ (whether caused by reliability issues or security attacks) are rare. Exploiting this unique characteristic may help improving the synthesis algorithms.

Finally, there need to be efficient implementations of the guards. In vehicle subsystems, for example, a single-task implementation of the guard, as in code generators for Simlunk/Stateflow, may be inefficient [30, 52]. Although a multi-task implementation is possible, there are some challenges. One challenge is to balance the accuracy and efficiency of the related schedulability analysis, since exact analysis of the static priority scheduling (used in the AUTOSAR/OSEK OS standard) is *strongly NP-hard* [44, 51]. Another challenge is to scale up the implementation, since the safety guard may have hundreds or even thousands of states and need to share the ECU with many other periodic tasks [8, 27].

7 ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation (NSF) under grants CCF-1435281 and CNS-1617203 and Office of Naval Research (ONR) under the grant N00014-13-1-0527.

REFERENCES

[1] Algirdas A Avizienis. 1995. A Methodology of N-Version Programming. *Software fault tolerance* 3 (1995), 23–46.
[2] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. 2012. Synthesis of Reactive(1) designs. *J. Comput. Syst. Sci.* 78, 3 (2012), 911–938.
[3] R. Bloem, B. Könighofer, R. Könighofer, and C. Wang. 2015. Shield Synthesis: Runtime Enforcement for Reactive Systems. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems.*
[4] S. Chakraborty. 2012. Keynote Talk: Challenges in Automotive Cyber-physical Systems Design. In *International Conference on VLSI Design.*
[5] Ruchir Chauhan. 2014. *A Platform for False Data Injection in Frequency Modulated Continuous Wave Radar.* Ph.D. Dissertation. Utah State University.
[6] E. Clarke, O. Grumberg, and D. Peled. 1999. *Model checking.* MIT press.
[7] P. Daian, S. Shiraishi, A. Iwai, B. Manja, and G. Rosu. 2016. RV-ECU: Maximum Assurance In-Vehicle Safety Monitoring. In *SAE World Congress.*
[8] M. Di Natale, L. Guo, H. Zeng, and A. Sangiovanni-Vincentelli. 2010. Synthesis of Multitask Implementations of Simulink Models With Minimum Delays. *IEEE Transactions on Industrial Informatics* 6, 4 (Nov 2010), 637–651.
[9] M. Dwyer, G. Avrunin, and J. Corbett. 1999. Patterns in Property Specifications for Finite-State Verification. In *International Conf. Software Engineering.*
[10] R. K. Brayton et al. 1996. VIS: A System for Verification and Synthesis. In *International Conference on Computer Aided Verification.*
[11] F. Sagstetter et al. 2013. Security Challenges in Automotive Hardware/Software Architecture Design. In *Design, Automation and Test in Europe.*
[12] Y. Falcone, J.-C. Fernandez, and L. Mounier. 2012. What can you verify and enforce at runtime? *STTT* 14, 3 (2012), 349–382.
[13] A. Francillon, B. Danev, S. Capkun, S. Capkun, and S. Capkun. 2011. Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars.. In *Network and Distributed System Security Symposium.*
[14] Greg Gardner. 2016. *GM recalls 4.3M vehicles to fix air bag software.* USA Today.
[15] C. Grote. 2014. IoT on the move: The ultimate driving machine as the ultimate mobile thing. In *IEEE Int. Conf. Pervasive Computing and Communications.*
[16] Gerard Holzmann. 2003. *The Spin Model Checker: Primer and Reference Manual* (first ed.). Addison-Wesley Professional.
[17] W Cary Huffman and Vera Pless. 2003. *Fundamentals of error-correcting codes.* Cambridge university press.
[18] I. Rouf et al. 2010. Security and Privacy Vulnerabilities of In-car Wireless Networks: A Tire Pressure Monitoring System Case Study. In *19th USENIX Conference on Security.*
[19] C. Jensen. *Safety Agency Says 22 Million Vehicles Recalled in 2013.* The New York Times.
[20] C. Jensen. 2015. *A Record Year of Recalls: Nearly 64 Million Vehicles.* The New York Times.
[21] X. Jin, J. Deshmukh, J. Kapinski, K. Ueda, and K. Butts. 2014. Powertrain Control Verification Benchmark. In *Int. Conf. Hybrid Systems: Computation and Control.*
[22] K. Koscher et al. 2010. Experimental Security Analysis of a Modern Automobile. In *IEEE Symposium on Security and Privacy.*
[23] Aaron Kane. 2015. *Runtime Monitoring for Safety-Critical Embedded Systems.* Ph.D. Dissertation. Carnegie Mellon University.
[24] Leo King. 2011. *Jaguar recalls 18,000 cars over cruise control software fault.* Computer World UK.
[25] V. Kljaic. 2015. *New Security Breach: Hacked Cars Force Recalls.* Auto VR World.
[26] R. Lyons and W. Vanderkulk. 1962. The use of triple-modular redundancy to improve computer reliability. *IBM J. Research and Development* 6, 2 (1962), 200–209.
[27] M. Panić et al. 2014. RunPar: An Allocation Algorithm for Automotive Applications Exploiting Runnable Parallelism in Multicores. In *International Conf. Hardware/Software Codesign and System Synthesis.*
[28] C. McCarthy. 2014. *National Institute of Standards and Technology cybersecurity risk management framework applied to modern vehicles.* Technical Report DOT HS 812-073. NHTSA.
[29] Charlie Miller and Chris Valasek. 2013. Adventures in Automotive Networks and Control Units. In *DEFCON 21.*
[30] M. Di Natale and H. Zeng. 2012. Task implementation of synchronous finite state machines. In *Design, Automation & Test in Europe.*
[31] NHTSA. *NHTSA Announces More Than 17.8 Million Products Recalled in 2012.* NHTSA.
[32] NHTSA 2011. *Honda: RECALL Subject: Automatic Transmission Control Module Software.* NHTSA.
[33] NHTSA. 2012. *49 CFR Part 571: Federal Motor Vehicle Safety Standards; Accelerator Control Systems.* Department of Transportation.
[34] NHTSA 2013. *Honda: RECALL Subject: Brakes may Unexpectedly Apply.* NHTSA.
[35] NHTSA 2014. *Ford: RECALL Subject: Side-Curtain Rollover Air Bag Deployment Delay.* NHTSA.
[36] NHTSA 2015. *Ford: RECALL Subject: Engine may Continue to Run/FMVSS 114.* NHTSA.
[37] NHTSA 2015. *Toyota: RECALL Subject: Inverter Failure may cause Hybrid Vehicle to Stall.* NHTSA.
[38] F. Rashid, K. Saluja, and P. Ramanathan. 2000. Fault tolerance through re-execution in multiscalar architecture. In *Conf. Dependable Systems and Networks.*
[39] M. Rosekind. 2016. *Remarks: Washington Auto Show keynote address.*
[40] John Rushby. 1989. Kernels for Safety? In *Safe and Secure Computing Systems*, T. Anderson (Ed.). Blackwell Scientific Publications, Chapter 13, 210–220.
[41] S. Checkoway et al. 2011. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *20th USENIX Conference on Security.*
[42] Y. et al. Shoukry. 2013. Non-invasive Spoofing Attacks for Anti-lock Braking Systems. In *Int. Conf. Cryptographic Hardware and Embedded Systems.*
[43] Jason Staggs. 2013. How to Hack Your Mini Cooper: Reverse Engineering CAN Messages on Passenger Automobiles. In *DEFCON 21.*
[44] Martin Stigge and Wang Yi. 2012. Hardness Results for Static Priority Real-Time Scheduling. In *24th Euromicro Conference on Real-Time Systems.*
[45] Jacqui Trotta. *Mercedes has the Lowest "Recall Rate" and BMW is the Most Timely in Making Recalls.* iSeeCars.
[46] A. Wasicek. 2014. Protection of Intellectual Property Rights in Automotive Control Units. *Society of Automotive Engineers World Congress* (2014).
[47] Wikipedia. *2009–11 Toyota vehicle recalls.* Wikipedia.
[48] M. Wu, H. Zeng, and C. Wang. 2016. Synthesizing Runtime Enforcer of Safety Properties Under Burst Error. In *8th Int. Symposium on NASA Formal Methods.*
[49] X. Jin et al. 2015. Mining Requirements From Closed-Loop Control Models. *IEEE Trans. CAD of Integrated Circuits and Systems* 34, 11 (Nov 2015), 1704–1717.
[50] H. Yu, C.-W. Lin, and B. Kim. 2016. Automotive Software Certification: Current Status and Challenges. *SAE Int. J. Passenger Cars-Electronic and Electrical Systems* 9 (2016), 74–80.
[51] H. Zeng and M. Di Natale. 2012. Schedulability Analysis of Periodic Tasks Implementing Synchronous Finite State Machines. In *Euromicro Conference on Real-Time Systems.*
[52] Q. Zhu, P. Deng, M. Di Natale, and H. Zeng. 2013. Robust and extensible task implementations of synchronous finite state machines. In *Design, Automation and Test in Europe.*