Practical Use of Formal Verification for Safety Critical Cyber-Physical Systems: A Case Study

Tasuku Ishigooka
Transportation, Energy and Environment Research Laboratory, Hitachi Europe GmbH, Germany
Email: tasuku.ishigoka.kc@hitachi.com

Habib Saissi, Thorsten Piper, Stefan Winter and Neeraj Suri
Department of Computer Science, TU Darmstadt, Germany
Email: {saissi, piper, sw, suri}@cs.tu-darmstadt.de

***Abstract*—Cyber-Physical Systems (CPS) linking computing to physical systems are often used to monitor and control safety-critical processes, i.e. processes that bear the potential to cause significant damage or loss in the case of failures.**

While safety-critical systems have been extensively studied in both the discrete (computing) and analog (control) domains, the developed techniques apply to either one domain or the other. As cyber-physical systems span both domains, the focus on an individual domain leaves a gap on the system level, where complex interactions between the domains can lead to failures that cannot be analyzed by considering only the physical or the digital part of the integrated CPS.

We discuss such a complex failure condition in a real-world brake control system, and demonstrate its detection using a formal verification approach specifically targeting CPS.

I. Introduction

Cyber-Physial Systems (CPS) are seeing increasing usage in transportation, power and other safety-critical systems [1]. The CPS is used to monitor and control (analog) physical phenomenon through physical actuators which are, in turn, operated by a (digital) safety critical computing system [2]. The processing needs to correctly monitor the target system behavior and to accurately operate the actuators in order to guarantee the system's safety to avoid damage to either the system or its operational environment.

Consequently, for safety critical CPS, rigorous verification technologies to ensure correct operations have been studied to consider both the discrete and continuous aspects. However, as these technologies typically only focus on an individual aspect, the complex interactions between discrete processing and continuous behavior has potential to cause complex system level malfunctions.

For example, an automotive brake control system may exhibit the difficult-to-find safety-related malfunctions caused only by a specific combination of specific driver operation and specific vehicle behavior given a specific combination of sequence and timing. While the requirement for safety verification for critical CPS is easy to state, i.e., ensuring coverage of all possible computing and control interactions, it is infeasible to achieve using conventional engineering approaches. Complicated control/computing interactions often result in timing and sequence malfunctions that are particularly hard to uncover. Additionally, as a design engineer often only has detailed knowledge of either the control or computing domain, resolving such issues is further complicated.

Differing from conventional statistical, experimental or simulation based approaches, the formal methods community has developed rigorous techniques, such as model checking, that target automated and comprehensive coverage of a system's states to discover complex malfunctions, such as timing faults. Amongst the varied model checking technologies, there exists an approach for hybrid model checking which can express both discrete and continuous aspects [3]. However, hybrid system modeling is often not easy for most engineers because they are only conversant with their own design aspects, such as software or mechanical design.

Recently researchers have started investigating formal verification methods, which construct practical hybrid system models by combining achievements from different domains to conduct hybrid system verification [4], [5], [6]. Especially, methods that combine model checking of software and simulation of plant models are being studied. However, as the verification of control systems with analog input values has to explore an enormous state space, the verification method needs huge efforts and long verification time. It is difficult to achieve the verification, taking into account effects caused by difference of analog value, such as sensor value of system input or of signal timing. Consequently, the development of a verification method, which can verify system behavior/misbehavior on exhaustive system input and timing, is challenging.

As a suitable approach for such challenging scenarios, symbolic execution based formal verification is being advocated as a high coverage testing method [7]. Symbolic execution guarantees the coverage of reachable paths and suggests relevant test cases. Verification properties are described in the form of assertion instructions. Once an unsatisfied assertion, i.e., a malfunction, is encountered, the symbolic execution engine outputs a test case with concrete input values that exhibit the detected defect during execution.

This paper proposes a practical formal verification process for safety critical CPS. We developed a formal verification framework that implements our proposed verification process. Thanks to developed framework can find system level malfunctions by checking safety relevant properties of system models, which simulate control system behaviors by combining control software and plant source code, on the basis of symbolic execution based formal verification.

Our main contributions are the following:

1) construction of a practical formal verification process for safety critical CPS based on symbolic execution,

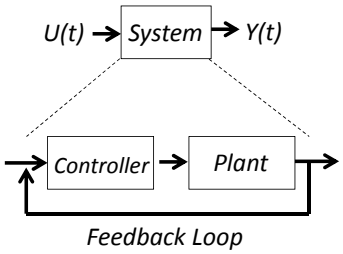

Fig. 1: Control System

2) implementation and evaluation of the proposed approach using a case study example from the automotive industry.

II. Background

An overview of a model-based development is presented in Section II-A. A general symbolic execution based formal verification is explained in Section II-B.

A. Model-based Development

Figure 1 shows a simplified model of a feedback control system, which consists of a controller model and a plant model. The controller model implements the system control logic and sends commands to the plant model which simulates the reactions of the physical component of the system. The combination of each model's sequential and iterative interactions constitutes the simulation of the system behavior.

Model-based development strongly supports the controller model design including discrete state transitions and the plant model design including differential equations. The approach has been extensively used in the industry and proven to be beneficial to the development of safety critical systems.

For example, Simulink® is an established model-based development tool, which supplies not only the controller model or plant model design supports, but also code generation functions assuring automatic translation of the controller model into a runnable program. Furthermore, it enables hardware-less system testing called hardware-in-the-loop simulation (HILS) as shown in Figure 2. The HILS technology uses micro-controllers, which implement control software involving the controller logic and basic software, and special devices, which simulate the plant model, and some wiring to physically connect these components. The plant code is generated by discretization of the continuous plant model and translation into C source code. In the discretization process, the sampling rate is chosen according to the Nyquist sampling theorem [8] such that the equivalence between the continuous plant model and the resulting discretized model is guaranteed. HILS enables the engineers to test control systems with their actual product's control software using system inputs without real hardware and is extensively used in the industry. However, as typical HILS has no synchronization mechanism between the micro-controller and the special device for HILS, it is difficult to conduct a system test on exhaustive values, timings, and sequence of system inputs.

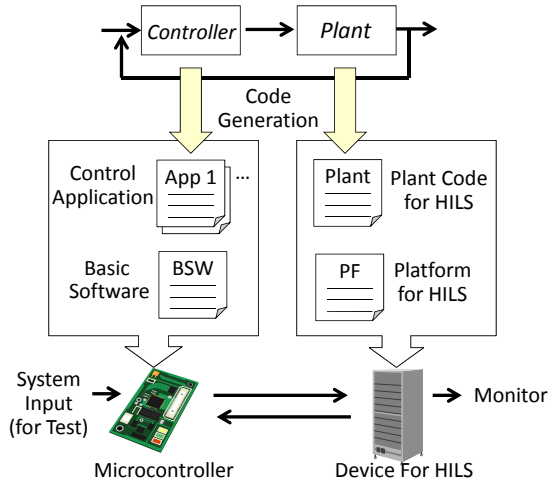

Fig. 2: Hardware-In-the-Loop Simulation Process

B. Symbolic Execution based Formal Verification

Symbolic execution based formal verification helps users to identify input values resulting in errors of target source code. The reason is that the verification can investigate all possible effects caused by changes in the values of variables, which users define as symbols. Consequently, if users define input variables of verification targets as symbols and insert assertions describing the verification property, the symbolic execution engine can find an input causing errors if the target does not satisfy the property specified in the assertions.

Figure 3 shows an example of symbolic execution based formal verification. In this example, we consider a function written in the C language (see Figure 3(a)). The target conducts a simple calculation at line 2 and an assertion code which catches the error behavior by comparing the result of the calculation to the assumed error condition at line 3, in which case the assertion is violated (see line 4). As shown in Figure 3(b), the symbolic execution based formal verification analyzes program logics of the verification target and extracts a formula describing constraints on the symbolic variables for specific paths in the code. In every step, the constraints are updated to describe the changes affecting the symbolic variables. When a branching statement is encountered, two constraint systems are created, one where the branching condition is evaluated to false and one where it's evaluated to true. Upon the execution of an assert statement the current constraint system is checked for satisfiability using a constraint solver. If it is not satisfiable, the search is resumed, otherwise an assignment of the symbolic variables is returned. The returned concrete values can be used as a test case to reproduce the found malfunction. Once all possible paths in the program are investigated, one can assume that the system satisfies the properties.

III. Our Formal Verification Approach

We propose a formal verification process in Section III-A. Our construction method of a verification target and property definition for control system verification are explained in Section III-B and Section III-C, respectively.

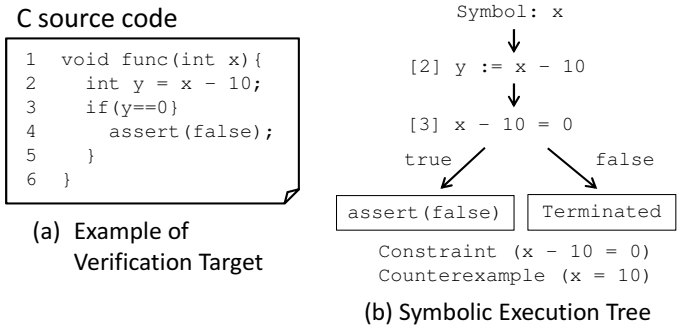

Fig. 3: Symbolic Execution based Formal Verification

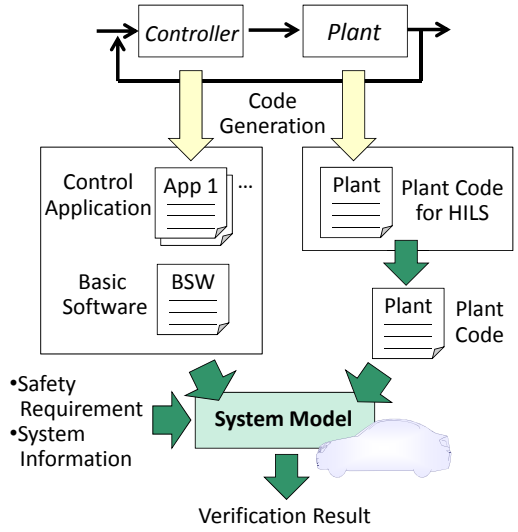

Fig. 4: Proposed Verification Process

A. Overview

It is important to establish a verification process, which is compatible with the HILS process (Figure 2), and to enable the verification by combining achievements from respective domains. We propose a verification process as shown in Figure 4. Our proposed process enables a verification of the whole control system. In the process, a system model, which simulates target control system behaviors, is built by combining the control software, plant code, safety requirements, and system information, such as system inputs, task execution periods, and plant discretization time. Next, the system model behavior is checked using symbolic execution. The system model construction is detailed in Section III-B. The plant code is extracted from its HILS counterpart, excluding code which is dependent on the HILS emulation device.

B. System Model Construction

Figure 5 shows an overview on the system model structure. Simple integration of control software and plant code cannot achieve the system model construction because there exists no synchronization mechanism for coordination in real time. Therefore, the system model includes a communication module for data synchronization and a synchronization module for time synchronization.

The data synchronization coordinates interactions between

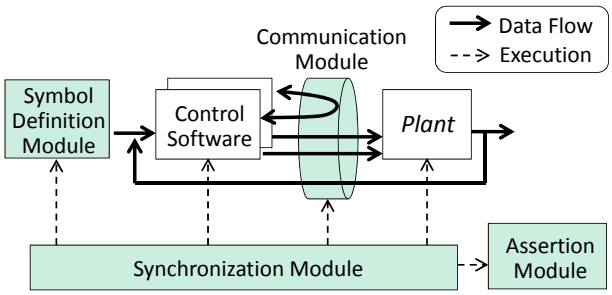

Fig. 5: System Model Structure

the control software and the plant. Specifically, the communication module forwards the current control command, which the control software calculates for the actuator control, to the plant which receives the output value from the controller model in Simulink. Also, the module forwards the current sensor values, which reflect the behavior of the plant, to the control software which receives the output value from the plant model in Simulink. The interaction between electronic control units (ECUs) is supported by the communication module as well.

The synchronization module maintains the current states of the control software and the plant by sequential and iterative invocation at appropriate timings. Additionally, the module limits the verification time in the system internal time for feasibility. The reason is that an unlimited system model shows infinite behaviors because the control software under control loops continuously maintains periodic invocations to control the plant. The user has to determine a sufficiently large bound for the verification time. For example, if users want to see the system effect caused by the combination of exhaustive timings or sequence of system inputs, the bound is determined according to appropriate time to check the combination. Consequently, the bounded time should be defined taking into account properties of the target system.

For symbolic execution based formal verification, we developed symbol definition modules and assertion modules in the system model. The symbolic definition module defines system inputs such as user operations or events from the environment of the verification target as symbols. To monitor the exhaustive effect given by the system input value change, the module needs to redefine them. However, as the redefinition creates new symbols, the verification time increases. To avoid the frequent redefinition, we limit the redefinitions to specific timing, e.g., every 1 second or after the occurrence of a specific event. The decision of optimal redefinition frequency is important for the control system verification. A decision approach is discussed using an example in Section IV-D. Additionally, as the system model updates its plant behavior at every discretization time, the decision of optimal discretization time is important as well. The discretization time should be determined on the basis of the sampling theorem. The assertion module, which is an assertion code, checks properties of the target control system by monitoring variables of the system model. The property definition method is detailed in Section III-C.

C. Property Definition for Control System Verification

The property for control system verification should carefully be defined taking into account response delays of actuator

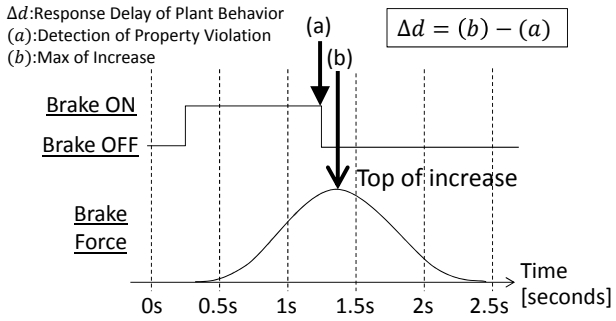

Fig. 6: Example of False Positive Caused by Response Delay of Plant Behavior

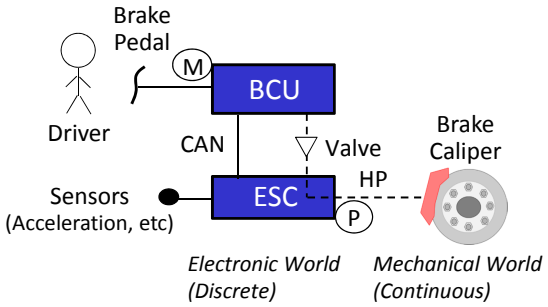

Fig. 7: Brake Control System

behaviors against system inputs or control commands from control software because the plant behavior is affected by physical phenomenon. That means the property should include waiting time to check the property. Otherwise, the verification using the property will frequently return false-positives.

Figure 6 shows an example of a false positive in safety verification of an automotive brake control system. This example indicates that the verification using the property without waiting time causes a false positive detection. As the property of this example applies that unintended brake doesn't occur, the property is defined as a condition where the brake force doesn't increase when no braking is happening. However, as the brake force still increases even though the brakes are not pushed (see (a)) because of the response delay of the plant behavior (see (b)), the verification detects false positive. Consequently, it is required to include waiting time for the response delay (Δd) in the property definition to get rid of the false positives. In Section IV-B we discuss our property definition method using an example.

IV. Experiment

A. Experiment Environment

We conducted a case study on safety verification of a simplified automotive brake control system in order to check the feasibility of our proposed formal verification approach. We attempted to find difficult-to-find malfunctions in an automotive brake control system involving the whole control system. The found malfunction results in faulty unintended braking behavior which was fortunately found during driving test of a commercial car.

For this experiment, we constructed a formal verification framework, which consists of a system model generator and a symbolic execution tool, to implement our proposed verification process. KLEE [9] is well known as a stable practical symbolic execution tool. The system model generator was implemented by ourselves from scratch, and KLEE-MultiSolver [10], which is an extension of KLEE, was applied for the symbolic execution based formal verification. KLEE-MultiSolver has mechanisms to support the use of different satisfiability modulo theories (SMT) solvers such as STP solver [11] and Z3 solver [12].

In the case study, the control software and the plant model were implemented by ourselves as a simplified real-world automotive brake control system. Specifically, we firstly abstracted a specification of the real product and implemented a simplified Simulink plant model and C language control software of two ECUs. Then, we conducted safety verification using our framework in accordance with given safety requirements. The details of the brake control system are presented in Section IV-B. The verification result is discussed in Section IV-D.

Our experiment ran on a machine with 3.60 GHz Intel® Xeon® quad-core processors and 8 GB of RAM. Operating system is Ubuntu® 12.04 64-bit. KLEE-Multisolver configuration is default. We used the STP solver which is the default solver of KLEE.

B. Example of Safety Critical CPS: Brake Control System

Figure 7 shows an overview of the simplified brake control system we used in our case study. There are two ECUs, a brake control unit (BCU), an electronic stability control unit (ESC), and a brake caliper which is a speed reducer using hydraulic pressure. BCU controls the caliper by producing hydraulic pressure using a brake assist motor in accordance with the brake pedal position. If the car sideslips, ESC takes over hydraulic pressure control of BCU and controls the caliper by producing hydraulic pressure using a pump to stabilize the car. ESC is only active during phases of sideslip. When ESC acts, the valve of the hydraulic pressure circuit is closed by ESC and its pump produces the hydraulic pressure of the ESC side by absorbing oils from the BCU side (the upstream side) of the valve. After the car becomes stable, ESC reopens the valve and returns the hydraulic pressure control to BCU. To avoid the collision of shared brake actuator control between the two ECUs, they coordinate each other using an in-vehicle network called Controller Area Network (CAN) and only one ECU can control the actuator. The interactions between ECUs, and between ECUs and hydraulic pressure control add to the complexity of the system.

Figure 8 shows an overview of the brake control system model. The model is abstracted at high level data communication. This means that the interaction between controller model and plant model is based on control commands, not low level data communication such as pulses to control the brake assist motor and so on. The control command is calculated on the basis of proportion control, which is simplified for the case study, by the control software. Each ECU executes a 1 millisecond periodic task which calculates the current control commands.

The hydraulic pressure, the pump, the brake assist motor, the valve, and the brake caliper were modeled as a plant

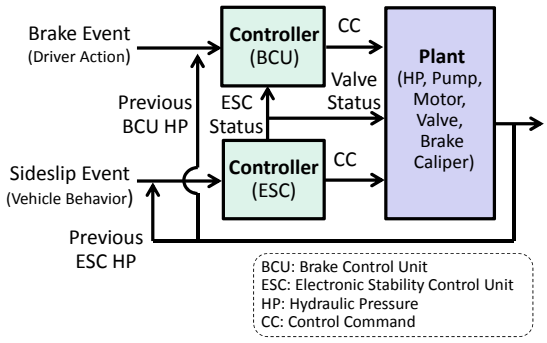

Fig. 8: Brake Control System Model

model which contains 121 blocks in Simulink. The plant code for HILS was generated by Simulink Coder™ at 100 microseconds discretization. These control software, which consist of the controller and basic software that is only a task activator of the operating system because of our abstraction, were manually implemented. We obtained the system model of the brake control system using our implemented system model generator.

C. Verification Setting

We present the property and the symbol definition for the verification target in this section.

In order to find the known malfunctions, we defined the non-existence of unintended brake behavior in the brake control system as a property. This property is the most general safety relevant property in a brake control system. We divided the property into the following conditions which should be always satisfied.

1) Driver doesn't push brake pedal
2) Sideslip of the car doesn't occur
3) Brake force doesn't increase in 500 ms under satisfactions of (1) and (2)

These conditions are implemented as an assertion code. The 500 ms of (3) indicates waiting time for the response delay to avoid false positives. While the waiting time logically may be fixed by analysis of Simulink plant models, in the case study, we fixed the waiting time by trial and error. For example, we firstly conducted safety verification of the case study by utilizing 100 ms as the waiting time because we considered that the time need to be set more than control period of the software and to be taken into account response delays of the actuator behaviour. If the verification find false positive, we added further 100 ms to the waiting time and re-verified until no false positive appears. By such process, we fixed the waiting time. We consider that the approach is better than opposite because the long time may cause false negative.

The system level malfunction of the brake control system only appears at the specific combination of the driver's specific brake and the car's specific sideslip given a specific combination of sequence and timing. The brake depends on the brake pedal stroke which means moving distance from the initial position and the slideslip depends on the car's speed. While we should define these analog system inputs as symbols, the available symbolic execution tools cannot deal with floating point data types for analog data expression in Simulink. For example, Ariadne can deal with floating point data types, but is not published yet [13]. Consequently, we transformed the analog system inputs into binary system inputs such as brake occurrence and sideslip occurrence. For example, the brake occurrence is expressed as ON or OFF. ON means to strongly push the brake pedal like sudden brake. OFF means to release the brake pedal. In the case of the sideslipe occurrence, ON means that the car sideslip at high speed and OFF means that the car is stable. This approach enables the available tools to define analog system inputs as symbols indirectly.

TABLE I: Symbol definition frequency and malfunction detection

symbol definition frequency	malfunction detection	verification time
every 100 μs	no (within one day)	-
every 100 ms	no (within one day)	-
every 1 s	yes	9 hours 9 minutes
every 2 s	yes	31 minutes

Additionally, to find the malfunctions given a specific combination of sequence and timing, we used iterative symbol definitions of respective system inputs. As the optimal redefinition frequency depend on verification targets, the frequency was clarified by trial and error. The verification time depends on the redefinition frequency because we need to define significant time to check the system behavior affected by the system input's changes. In the case study, the verification time is tentatively limited at 5 seconds taking into account the response delay of the plant behavior. Furthermore, due to efficient verification on conditions of overlapping system inputs, we inserted time offsets into the timing of the brake occurrence.

The system model generator generated the system model which consists of the control software of BCU and ESC, the plant code, the communication module, the symbol definition module, the assertion module, and the synchronization module. The system model is approximately 2000 code lines of C language source code.

D. Verification Results and Discussion

To find the complex system level malfunction, we tried to conduct the safety verification of the brake control system with different symbol definition frequency. Finally, our proposed formal verification framework detected the expected malfunction.

Table I shows relationships between the symbol definition frequency and the malfunction detection. While the verification with the symbol definition of short time duration could not finish within one day, in the case of long time duration such as a second or 2 seconds, the verification finally could find the complex malfunction. The duration satisfies practical demands because the second time scale is significant to verify the system behavior affected by changes of system inputs. The results clarified that the impact of the symbol definition frequency and showed the availability of the symbol definition process taking into account conditions causing the malfunctions which we want to detect.

To understand the details of the system behavior, we conducted a simulation using the system input pattern resulting

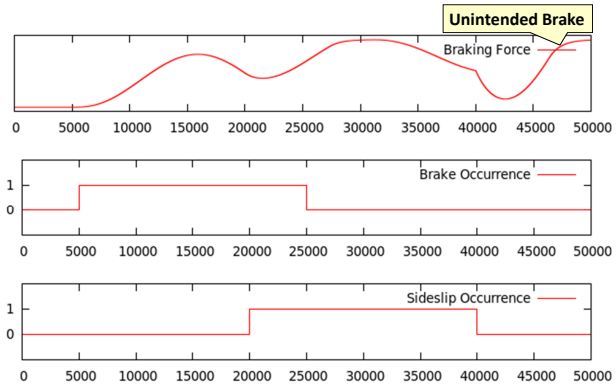

Fig. 9: Brake Force Behavior in Unsafe Case

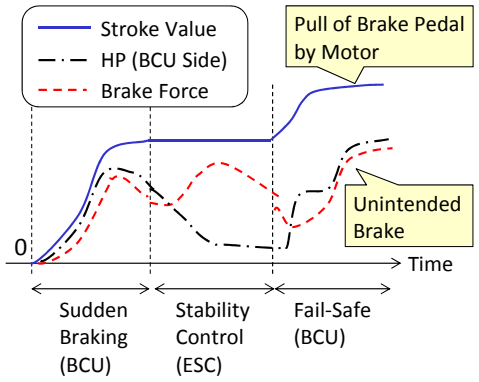

Fig. 10: Details of System Behavior in Error Case

in the property violation. Figure 9 shows the simulation result. As the graph shows, the car's sideslip occurs during sudden braking and then once the car becomes stable, unintended brake force appears. This is the complex system level malfunction given by a specific combination of sequence and timing.

Figure 10 shows the details of the system behavior in the error case. BCU involves a diagnostic program which detects oil leak on the basis of gaps between pedal stroke value and the amount of hydraulic pressure. As the graph shows, after the stability control of ESC, there exists a big gap because the amount of the oil to stabilize the high speed car was more than the prediction. Therefore, the diagnostic program detected the oil leak due to the gap which exceeds the error threshold and then BCU invoked a fail-safe program which automatically produces hydraulic pressure by making the brake assist motor pull the brake pedal in order to brake by the rest of the oil. As the result, unintended brake occurs.

The factor of the system level malfunction is related to gaps between response delays of the ECU processing (digital) and the hydraulic pressure behavior (analog). While a conventional top-down system development process of safety critical CPS bears the potential to cause the difficult-to-find system level malfunctions, through this case study, we established that our proposed formal verification approach can detect them.

V. Conclusion

In this paper we proposed a practical formal verification process for safety critical CPS and developed a formal verification framework. The framework supplies a system model generator which automatically generates system models of the verification target. The framework is capable of verifying the safety of the system model in accordance with safety relevant properties by symbolic execution based formal verification. Through our case study on safety verification of an automotive brake control system, we showed that our proposed formal verification framework can detect difficult-to-find system level malfunctions from the abstracted system model by iterative symbol definitions and property definitions taking into account the response delay of the plant behavior. Our approach has an advantage in system verification in comparison with simulation approach because the simulation approach requires test cases causing system level malfunctions even though discovery of the failure condition is difficult. Our approach produces hybrid automaton of target control system. In hybrid automaton modeling, abstraction approach to focus on verification aspects is mandatory in order to decrease verification time. However, our approach may contain irrelevant aspects. In our future work, to apply our approach to safety verification of more complex safety critical CPS such as mass production, we will develop abstraction technologies. Furthermore, we plan to develop calculation methods of response delays of plant behaviors from a Simulink plant model.

Acknowledgment

We are indebted to Hitachi Automotive System, Ltd. for their feedback and great examples.

References

[1] ACATECH (Ed.), *Cyber-Physical Systems - Driving Force for Innovation in Mobility, Health, Energy and Production*, 2011.

[2] Edward A. Lee and Sanjit A. Seshia, Introduction to Embedded Systems, A Cyber-Physical Systems Approach, http://LeeSeshia.org, ISBN 978-0-557-70857-4, 2011.

[3] R. Alur, *Formal Verification of Hybrid Systems*, EMSOFT, 2011.

[4] F. Lerda, et al., *Model Checking In-The-Loop*, in ACC, 2008.

[5] R. Kawahara, et al., *Verification of embedded system's specification using collaborative simulation of SysML and Simulink models*, in MBSE, 2009.

[6] H. Nakajima, S. Furukawa and Y. Ueda, *Co-Analysis of SysML and Simulink Models for Cyber-Physical Systems Design*, in CPSNA, 2012.

[7] C. Cadar, et al., *Symbolic Execution for Software Testing in Practice - Preliminary Assessment*, in ICSE, 2011.

[8] R. J. Marks II, *Introduction to Shannon Sampling and Interpolation Theory*, in Spinger-Verlag, 1991.

[9] C. Cadar, D. Dunbar and D. Engler, *KLEE: Unassisted and Automatic Generation High-Coverage Tests for Complex Systems Programs*, in OSDI, 2008.

[10] H. Palikareva and C. Cadar, *Multi-solver Support in Symbolic Execution*, in CAV, 2013.

[11] V. Ganesh and D. L. Dill, *A Decision Procedure for Bit-Vectors and Arrays*, in CAV, 2007.

[12] L. De Moura, et al., *Z3: An efficient SMT solver*, in TACAS, 2008.

[13] E. T. Barr, T. Vo, V. Le and Z. Su, *Automatic Detection of Floating-Point Exceptions*, in POPL, 2013.