MITL Specification Debugging for Monitoring of Cyber-Physical Systems

Adel Dokhanchi, Bardh Hoxha, and Georgios Fainekos
School of Computing, Informatics and Decision Systems
Arizona State University, Tempe, AZ, U.S.A.
Email: {adokhanc,bhoxha,fainekos}@asu.edu

***Abstract*—A framework for the debugging of formal specifications for Cyber-Physical Systems is presented. Two debugging algorithms are presented. The first checks for erroneous or incomplete temporal logic specifications without considering the system. The second can be utilized for the analysis of reactive requirements with respect to system test traces. The specification debugging framework is applied on a number of formal specifications collected through a user study. The user study establishes that requirement errors are common and that the debugging framework can resolve many insidious specification errors.**

I. Introduction

In formal verification of Cyber-Physical Systems (CPS), a system is verified with respect to formal specifications. However, developing formal specifications using logics is a challenging and error prone task even for experts who have formal mathematical training. To assist in the elicitation of formal specifications, in [6], we presented a graphical formalism and tool VISPEC that can be utilized by both expert and non-expert users. Namely, a user-developed graphical input is translated to a Metric Interval Temporal Logic (MITL) formula [2]. The formal specifications in MITL can be used for testing and verification with tools such as S-TALIRO [3].

In [6], the tool was evaluated through a usability study which showed that both expert and non-expert users were able to use the tool to elicit formal specifications. The usability study results also indicated that in many cases the developed specifications were incorrect. In our on-line survey[1], we tested how well formal method experts can translate natural requirements to MITL. The preliminary results indicate that even experts can make errors in their specifications. For example, for the natural language specification *"At some time in the first 30 seconds, the vehicle speed (v) will go over 100 and stay above 100 for 20 seconds"*, the specification $\varphi = \Diamond_{[0,30]}((v > 100) \Rightarrow \Box_{[0,20]}(v > 100))$ was provided as an answer by an expert user. Here, $\Diamond_{[0,30]}$ stands for "*eventually within 30 time units*" and $\Box_{[0,20]}$ for "*always from 0 to 20 time units*". However, the specification φ is a *tautology*, i.e. it evaluates to true no matter what the system behavior is and, thus, the requirement φ is invalid.

This indicates that the specification correctness is a major issue in testing and verification since effort can be wasted in checking incorrect requirements, or even worse, the system can pass the incorrect requirements. Clearly, this can lead to a false sense of system correctness. In this work, we have developed a specification analysis framework that would enable the debugging of specifications. The specification debugging algorithm identifies invalid and wrong specifications.

II. Specification debugging

We first provide a framework that helps the users to detect the specification errors, where the requirement issues can be corrected before any test and verification process is initiated. However, some specification issues cannot be detected unless we consider the system, and test the system behaviors with respect to the specification. Thus, we provide algorithms to detect specification issues with respect to signals in order to help the CPS developers find more in depth specification issues during system testing.

Problem 1 (System Independent MITL Analysis): Given an MITL formula φ, find whether φ has any of the following logical issues:

1) Validity: the specification is unsatisfiable or a tautology.
2) Redundancy: the formula has redundant conjuncts.
3) Vacuity: some subformulas do not contribute to the satisfiability of the formula.

In particular, issues 2 and 3 usually indicate some misunderstanding in the requirements. The overview of our proposed solution to Problem 1 is provided in Fig. 1. We present a specification debugging algorithm for a fragment of MITL specifications. This framework appeared in [5], and it can solve the specification correctness problem for VISPEC [6] requirements. The user of VISPEC can benefit from our feed-back and fix any reported issues.

[1]The on-line survey is available through: http://goo.gl/forms/YW0reiDtgi

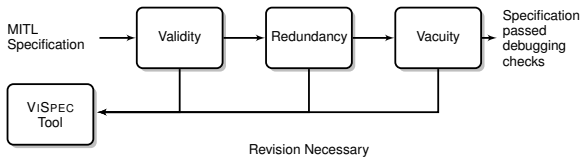

Fig. 1. Specification Debugging Framework [5]

In order to find the logical issues in MITL specification, we need to apply the MITL satisfiability solver [4]. For checking the validity issue we can easily use the MITL satisfiability solver. To find the redundancy and vacuity issues, we need to apply syntactic manipulations on the MITL formulas. In [5], we proved that for the fragments of MITL where only eventually ($\Diamond$) and always ($\Box$) operators are used, we can apply the syntactic manipulations for detecting the redundancy and vacuity of specifications (Problem 1). Therefore, we can find the redundancy and vacuity issues using the MITL satisfiability checking on the manipulated MITL formulas [5]. As a result, we implemented the whole framework of Fig. 1, using the MITL satisfiability solver [4]. We checked the correctness of MITL specifications provided in the usability study of [6]. The experimental results of applying Fig. 1's framework is reported in [5]. Application of our specification debugging tool on user derived requirements shows that all three types of the issues are common [5].

Now, Consider the MITL specification $\varphi = \Box_{[0,5]}(req \Rightarrow \Diamond_{[0,10]} ack)$. This formula will pass the MITL Specification Debugging method provided in Fig. 1. However, any signal μ that does not satisfy req at any point in time during the test will vacuously satisfy φ. We refer to signals that do not satisfy the antecedent (precondition) of the subformula as vacuous signals.

Problem 2 (System Dependent Vacuity Checking): Given an MITL formula φ, and signal μ, check whether μ satisfies the antecedent failure mutation of φ.

An antecedent failure mutation is a new formula that is created with the assertion that the precondition (for example req in case of $\varphi = \Box_{[0,5]}(req \Rightarrow \Diamond_{[0,10]} ack)$) never happens. To detect the antecedent failure of signals, we extract all the antecedent subformulas of φ and create a new MITL formula φ' (for each antecedent) that asserts that the antecedent never happens. As a result, we run the monitoring algorithm in S-TALIRO [3] to check whether signal s satisfies φ'. If s satisfies φ', then s is a vacuous signal and it is reported to the user.

In order to use vacuity detection (Problem 2) in CPS testing, we provide our framework in Fig. 2. The input generator creates initial conditions and inputs to the system under test. An example of a test generation technology that implements the architecture in Fig. 2 is presented in [1]. The system executes or simulates to generate an output trace. Then, a monitor checks the trace with respect to the specification and reports to the user whether the system trace satisfies or falsifies

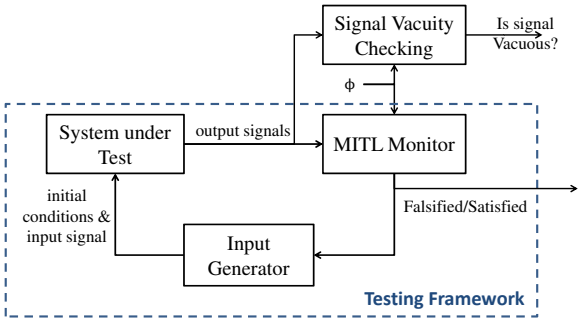

Fig. 2. Using signal vacuity checking to improve the confidence of an automatic test generation framework.

the specification. Signal vacuity checking is conducted, and vacuous traces (signals) are reported to the user for more inspection (See Fig. 2). Our experimental results showed that for CPS applications, antecedent failure is a serious problem and it needs to be addressed in the testing and monitoring of CPS.

III. Conclusion and Future Work

We provide a framework that helps the developers to correct their specifications and avoid wasted effort in checking incorrect requirements. Currently, we are working on using vacuous signals to improve the counter example generation process and system debugging using signal vacuity.

Acknowledgment

This research was partially funded by NSF awards CNS 1350420, CNS 1319560, IIP-1361926 and the NSF I/UCRC Center for Embedded Systems.

References

[1] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 12(2s):95:1–95:30, May 2013.

[2] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. In *Symposium on Principles of Distributed Computing*, pages 139–152, 1991.

[3] Y. S. R. Annapureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Tools and algorithms for the construction and analysis of systems*, volume 6605 of *LNCS*, pages 254–257. Springer, 2011.

[4] M. Bersani, M. Rossi, and P. San Pietro. A tool for deciding the satisfiability of continuous-time metric temporal logic. *Acta Informatica*, pages 1–36, 2015.

[5] A. Dokhanchi, B. Hoxha, and G. E. Fainekos. Metric interval temporal logic specification elicitation and debugging. In *13. ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2015, Austin, TX, USA, September 21-23, 2015*, pages 70–79, 2015.

[6] B. Hoxha, N. Mavridis, and G. Fainekos. VISPEC: a graphical tool for easy elicitation of MTL requirements. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Hamburg, Germany*, September 2015.