Project Summary

Overview

The core problem we aim to address in this proposal is that use of formal modeling, advanced static analysis, and advanced dynamic analysis for verification and validation, especially on critical embedded systems, is prohibitively difficult and lacks sufficient synergy for cost-effective application. Making it possible to follow one critical path to combine methods is feasible given current technologies in the subdomains (formal modeling, static analysis, and dynamic analysis) and a set of specific advances in bridging the gap between the technologies. By focusing on adding annotations to implementation code, and exploiting those annotations to enable a set of potentially bug-finding or correctness-proving analyses, we promise to always give embedded systems engineers a reasonable payoff, often in the form of tests showing real bugs. This project therefore proposes to make it possible to introduce specifications into implementation code that can be directly checked using sophisticated automated test generation strategies, including symbolic execution, advanced fuzzing, explicit-state model checking, and SAT/SMT-based bounded model checking. We will also advance these technologies to support extensions to advanced specification annotations in code. Furthermore, these specifications may be directly exported to form the basis for formal models using, e.g., timed automata. In the long run, to benefit those developers who are more open to formal methods already, we hope that these annotations can be imported from a timed automata representation, but we begin where most embedded systems developers are, now, not where we hope they may be, someday.

Intellectual Merit

Open research questions here are numerous, and include: (1) how to extend existing specification languages to support timing, interrupts, and uncertainty; (2) how to assign the same meaning to a specification construct in various contexts, ranging from fuzzing to symbolic execution to explicit-state model checking to bounded model checking in a "dynamic" context, as well as a static context and a modeling context for timed automata; (3) how to minimize annotation burden while allowing developers to include information that can be exploited by those methods: e.g., to make intelligent use of preconditions in fuzzing, to automatically derive loop bounds in bounded model checking, and to restrict branching factors and store state in explicit-state model checking; (4) how to handle intra-program parallelism; (5) how to ensure that the methods are sufficiently automatic and behave in ways engineers (not formal modeling, static, or dynamic analysis experts) will expect. Our focus will be on practical solutions, guided by the embedded domain experts, rather than on purely theoretical approaches that do not scale to real systems. Practical solutions here require fundamental contributions to system and specification design and dynamic analysis and model checking technologies.

Broader Impacts

We expect that successful completion of this project will lead to methods and tools that will improve the reliability and security of embedded and networked systems, including Internet-of-Things-related systems. We are already in discussion with NASA's Jet Propulsion Laboratory to incorporate our approach into the methods and tools for use in NASA's open source flight software architecture and framework, FPrime. A second broader impact, therefore, is likely to be the improvement of systems specifically used in critical scientific research; one such system, the Distributed Sensing and Computing Over Sparse Environments (DISCOVER) testbed is a core case study for the project; FPrime represents a more extreme case, that of scientific systems operating outside Earth's atmosphere and thus inherently difficult to troubleshoot and debug. We additionally propose to use our effort to further outreach to under-represented groups in CS, with active learning experiences for undergraduates and K7-K9 students.

Keywords: embedded systems, automated test generation, model checking, specification languages