Project Description — FMitF: Constructing Reliable Networked Systems via an Assembly of Formal Methods

1 Introduction

Overall comment =>

Developing reliable complex systems under difficult engineering constraints, e.g., systems whose final endpoints are power-constrained, limited-bandwidth, and costly for humans to repair or even inspect, is a key challenge over the next decade; while the widely discussed "Internet of Things" may be the first thing that comes to mind given this description, the problem is also central to long-term scientific research efforts, ranging from the climate change and plant migration research platform used as a testbed in this proposal to efforts to understand the surfaces of other planets.

Formal methods have been used to verify aspects of such systems, including file systems, control elements, network protocols, Let's find some cites and add things here; however, the proofs of components are seldom, if ever, assembled into a coherent, quantified, or even fully *qualified* understanding of the extent to which the whole system has been verified. Because different components of many such systems are developed by disparate groups, sometimes collaborating internationally, even when formal methods are uniformly used, the emphasis is on the plural: formal methods, not a single formal method. The assumptions of one formal method/component are seldom formally yoked to the outputs of another formal method, with gaps in verification made clear. Finally, many components are only verified using high-confidence, automated, but non-proof-based approaches, including model checking with unsound abstractions, or automated test generation. When these kinds of "semi-formal" methods (relying on a formal model of inputs and system behavior, but not providing correctness proofs) are integrated into a design, there is almost never a characterization of the confidence thus provided. The outcome is either not used at all in other methods, or is assumed to be as solid as a full proof.

In this proposal, we aim to address this weakness by designing "glue" formal methods for assembling the results of a set of hetergenous formal and semi-formal approaches into a unified, coherent (if complex) result, able to identify both weaknesses and strengths in a system verification. E.g., if the proof of correctness of one component relies strongly on an assumption backed only by limited, manually constructed tests, this can be distinguished from the case where the assumption is backed by a proof of correctness in a different formalism, which can be distinguished from the case where it is backed by both model-based and symbolic-execution driven test generation, and a strong probabilistic, coverage-based, or mutation-based estimate of completeness is available. Moreover, the proposed approach will be able to distinguish the case where an entire verification result is tainted by a fundamental and non-proven assumption from the (common) case where different properties or behavioral paths have different degrees of reliabilty, ranging from complete independence from an uncertain assumption to confidence completely proportional to the strength of an unchecked assumption.

1.1 Problem Statement and PI Qualifications

The simplest form of problem statement in this case may be in the form of an example. Consider a networked system consisting of in-the-field power-constrained nodes with sensors and actuators, in-the-field base stations that collect data from and issue commands to these nodes, and a (possibly distributed) remote command-and-control center that handles permanent high-level data storage and analysis, coordination of activities across multiple field stations, and human oversight. The overall system might be (as in our primary case study and testbed platform) an ecological science platform performing experiments on plant migration and climate change impacts, or a NASA mission involving a swarm of robot explorers on another planet. The details of system purpose are irrelevant; what matters is that there is a complex networked system with heterogenous connectivity and computational capabilities, where failure is extremely costly, in terms of either scientific outcomes (a stuck irrigation ruins a five-year experiment on plant survival) or simply dollar value (a multi-million dollar space mission is lost).

Again, I think we should be clear about what types of systems, what are the characteristics of such systems we will consider.

When failure is sufficiently costly, verification efforts are (now and, we hope, increasingly in the future) proportionally intense. However, complex systems such as these are seldom developed by one team, in one place, with a coordinated approach to ensuring software correctness. Instead, a variety of methods may be applied, even by teams all seeking the same goal of high reliability using state-of-the-art methods.

For example, two software modules may run on the same embedded system without operating-system-level memory protection. One module may have been "verified" using high-coverage, high-quality, automatically generated tests. The other module may have been verified using bounded model checking, with a depth that can be shown to exceed the diameter of system executions under valid inputs. The tested module's testing results are valid under the assumption that the other module does not change values it accesses, and *this* assumption is guaranteed to hold, because the model-checked system is shown to never access memory in unsafe ways. The model-checked component, on the other hand, is correct under the symmetric assumption: that the tested module does not change its' memory contents; this assumption is only partially guaranteed, for a limited set of executions. This is a simple case, but even here the relationship can be used to guide engineering. If we insert runtime guards on memory accesses, these only need be applied to the tested module, not the model-checked one, to guarantee memory boundary protection for the combined system. However, if the model-checked system essentially relies on validity of outputs from the tested system for correctness, and these cannot be checked at runtime, the model-checking result itself is inherently only known to be as realiable as the tested system. The measure of the strength of testing, whether it is code coverage, expected mean-time-to-failure under a realistic operational profile of inputs, or mutation score, propagates to the "verified" module as well.

A basic aspect of the proposed work is understanding the implications of four kinds of assumption relationships:

1. First, there is the case where a formal method producing complete proofs has an assumption that is discharged by another formal method providing proofs. In this case, the core effort is providing translations between logics that allow us to understand when these assumptions are fully discharged, and to propagate any assumptions of the formal method used to guarantee the assumption to the formal method making the assumption.

2. Second, there is the case where a formal method producing complete proofs has an assumption discharged by testing or another incomplete method.

3. Third, there is the case where testing relies on an assumption from a formal method.

4. Fourth, there is the case where testing relies on an assumption validated by another testing approach.

Everything past this point is from a previous proposal, ignore it. My prior NSF may be partially valid.

1.2 Expected Outcomes

2 Related Work

Some of the related work is the general literature on software testing problems [5, 55] that is relevant, which is cited throughout this proposal, e.g. recent work on regression testing [7, 16, 21, 60, 68], seeded or parameterized test generation [39, 40, 50, 57, 63, 64, 69], flaky tests [19, 42, 48, 51, 52, 56] and compiler testing [1, 18, 32, 44, 67, 74]. This section focuses on more specific precursors to our ideas in this proposal, including our own efforts towards the idea of novel test manipulations.

The idea of algorithms that operate on tests, as such, is primarily represented in the literature by the work on delta-debugging or test reduction in general: [9, 33, 46, 47, 53, 59, 61, 70–72]. Sai proposed a very limited, ad hoc version of semantic minimization that aims to go beyond the simplifications possible with conventional delta-debugging [75]. Work on automatically producing readable tests [12, 13] is also related, in that it aims to "simplify" tests. Test case purification [65] is a kind of limited (in approach and in goal)