MAPi Doctoral Programme
Thesis Proposal

January 2014

Thematic area: Software Verification and Defect Analysis

Area in ACM Computing Classification System:

Software and its engineering / Software creation and management / Software verification and validation

- Formal software verification
- Software defect analysis

Supervisor: Jorge Sousa Pinto, HASLab-INESC TEC & University of Minho

Host institution: University of Minho

Host research unit: HASLab-INESC TEC

External member of monitoring group: Luís Miguel Pinho, CISTER-INESC TEC & ISEP, IPP

Brief Description

Safety-critical systems are those whose failure or malfunction may result in an unacceptable loss of human life or equipment, or serious environmental harm. Of course, software plays an important role in the safety of today's systems, and unfortunately minor software errors have already shown a potential to be dramatically catastrophic. Traditionally, the development of critical software is addressed using non-formal approaches, well-established but error-prone. Formal methods are however increasingly more used to guarantee software correctness in these domains, and industry standards are changing to reflect this new scenario.

These formal approaches to critical software development are still young and present a number of problems, in particular *scalability* issues. The general goal of this proposal is

to further the development of scalable formal techniques and tools that can be effectively used in an industrial settings for verification and validation of software.

The most popular currently available techniques for software verification are *deductive verification*, *static analysis by abstract interpretation*, *abstraction-based model checking*, *bounded model checking*, and *symbolic execution*. Of these, static analysis is undoubtedly the technique that scales up better, and also the one that has found application in industry, with several mature commercial tools currently available. However, it is targeted at certain very specific tasks such as value analysis, and cannot be employed in general for the detection of assertion violations. At the other extreme we find deductive-based verification, which is certainly able to deal with arbitrary functional properties, but is not designed for automation, which constitutes an important inconvenient for its usage in industrial settings.

Techniques based on model checking of software and on symbolic execution are designed to be automatic and require little user intervention. They typically target the verification of programs with respect to safety properties, such as the absence of memory violations and buffer or integer overflows. A further reason for the popularity of these techniques is the fact that they can also be used in the generation of test suites satisfying some given coverage criteria – model checkers can support this process by providing solutions for coverage constraints (achieving appropriate coverage levels is crucial for the certification of safety-critical software in accordance with doman-specific norms). This combination of automation and targeted properties has caused a surge in the development of tools based on these techniques, incorporating different proposals to approach scalability. Still, these techniques usually scale up to only a few tens of KLOCs (thousands of lines of code). The same applies to current approaches to automated coverage-directed test-case generation, whose application scope is still limited to unit testing.

The purpose of this PhD project is to study new approaches to support the effective use of formal techniques for certification of software systems in safety-critical settings. This goal is actually twofold: to achieve effective support for formal analysis of source code, and effective coverage-directed test case generation, driven by formal approaches. Our ultimate goal is to support the use of formal methods to meet the requirements imposed by industrial standards such as the DO-178C for avionics, or the CENELEC EN50128 for railways, which demand strong evidence that a software system obeys its functional and non-functional specification. Such evidence can be provided by testing, if some coverage criterion is satisfied, or by running a formal analysis on the source code.

The concrete workplan to be developed may include the development of a toolset to support automated checking of programs with respect to functional and/or non-functional (safety) properties, and automated generation of test suites to fulfill precise coverage requirements; or it may consider instead how the combination of different techniques (random testing, symbolic execution, model checking) could enable the scalable analysis of large code bases, for generating complete test-suites satisfying rigorous coverage criteria. The approach and/or toolset to be developed may be language-agnostic, or else target code of some programming language widely-used in the development of safety-critical software.