Daniel Carton

Dr. Gregory Gay

Assignment 3

Februrary 28, 2016; Due March 3, 2016

Proving the Shalls is a paper attempting to prove that formal methods of validation could identify system requirements promptly and frugally through a Flight Guidance System (FGS), a system that strongly relies on safety-critical principles. The paper examines the research of the junction between model-based development and formal methods to present a new line of attack for validation and when to start looking at validation.

Finding faults early is crucial for the reason that "the cost of correcting a requirements error grows dramatically the later in the product life cycle it is corrected" (1). This problem stems from the mentality that investments in the test suites early on could be wasteful. This paper argues that formal methods could succeed in an economical matter when implemented at the initial phases of a project. Another opinion on profitability is that formal method tools are now mature enough to be used on industrial sized problems, which this paper demonstrates (17). Further confidence is presented by this exercise because it proves very likely that systems of all kinds can benefit from verification by model checking. By working with NASA, a company known for their genius and attitude to avoid errors, these authors ensure that their tactical decisions can be respectfully considered for real-world applications.

More important than profit is safety. Requirement errors are significant to the safety of a system. This FGS can determine a pilot's fate. The avionics industry's engineers understand this crucial need for requirement checking and have created many methodologies attempting to evade such errors (1). Because avionics systems are still using a combination that includes informal diagrams, the three groups conducting this case study use it as the perfect example for formal analysis research. Moreover by using it on a flight guidance system and trying to help the FAA, it will quickly become apparent if these approaches are advantageous to industrial software companies. The deduction that formal models will give an improvement to error checking makes this approach seems very useful. The authors even state that several methods will "scale to industrial use" (2).

This tactic falls short when there are ambiguities between models and understanding them – both for spectators and architects. For example figure 3 of the paper is not easily understood. Figure 3 is a fragment of RSML^{-e} translated to SMV. Although SMV translation has yielded promising results, the translation to human eye is very difficult without proper training or background (6). Even with my programming background it is difficult to grasp what exactly this specification is trying to convey. Following the Boolean statements is trivial, but there should be a greater breakdown of what the variables are and do. Although it might have been out of the scope of the paper, consumers would benefit from a key or legend if they did not have proper training.

I have determined that formal methods of testing would help any software-to-hardware system and most software systems with safety-critical concerns. It is conceivable that hardware of any kind would benefit from formal logic checking because of the physical gates and paths. Similarly software has digital gates with the same fundamentals. Companies like Intel that must make drivers for hardware would greatly benefit in this approach because of its robustness and scalability. A specific example would be a radiation machine that takes nurse or doctor input. The researchers address domain abstraction which could have helped save multiple lives in the 80's. Numerous occasions saw patients get drastic overdoses of radiation in the "Therac-25 incidents".

To extend this research further I believe more exploration needs to be done in the training of staff to construct proofs because of the differences in time for proof production (14). For example, some of the proofs were done by a graduate student with no avionics background. Are we to assume he is a computer science student or a biology student? If more research is done in this section, software departments can focus more on developing code while another department writes proofs, or possible train programmers in this tactic. The point is the semantics, or arrangement, of these words yield different results. Programmers have formal education in the matter, but might not have time allocated in producing or checking another person's work. During time crunches and final stretches of development where programmers are required to alter proofs, this might serve as a crucial role for the overall project to free up their time.

More information about the programming errors leading to the Therac-25 http://www.cs.umd.edu/class/spring2003/cmsc838p/Misc/therac.pdf