CSE 231—Advanced Operating Systems "Bugs as Deviant Behavior"

Andrew Quinn

Summary. This paper introduces the idea of looking for deviations from the norm to automatically find bugs in large scale software systems. Their tools find dozens of bugs in large-scale open sources operating systems using this approach.

Discussion. Although it is never explicitly stated, this paper is really about solving the oracle problem in software testing. The oracle problem is about determining whether some possible behavior in an application should be considered correct. For example, one possible oracle is to build a simple but correct version of your program (e.g., one that performs poorly) and ensure that the full implementation has the same output as the correct version (note, you cannot guarantee this for all inputs, but you can do it for many inputs).

Bugs as deviant behavior shows that we can essentially infer an oracle for a programs behavior by looking for "normal" behavior and deviations from it. For this to work well, the approach needs for most of the program to be correct[1]. As such, the approach is probably best as an addition to software testing rather than as a replacement for software testing: ideally, testing will identify egregious bugs, while deviant behavior will generalize the bugs identified by testing[2]

My biggest question after reading the paper is how well their system actually realizes there goals. First, many of their bugs were not deviant behavior, but essentially violations that are guaranteed to be failures (e.g., `null` pointer dereferences). Are these really using their approach? Second,

[1] many of you noted this limitation in reviews and/or during discussion

[2] This might actually be a nice idea for a bug finding tool...

their approach relies on these templates, which limits the space of behaviors to inspect, but also detracts from their inference claims.

The final paper related thing to discuss is the use of static bug detection. Many systems today are not using a compiler-based approach, since it seems difficult to have sufficient static knowledge to find bugs. Systems that strive for the greater code coverage that comes with a static tool are using techniques that allow them to simulate an executing program. Such tools include symbolic execution (e.g., KLEE [**?**]) and concollic execution (e.g., Dart [**?**]).

The ideas from this paper eventually became a fully blown system and startup called Coverity. Those involved in Coverity eventually wrote a retrospective, "A few billion lines of code later" where they describe their experiences [1]. There are many anecdotes and lessons that they present, but my two favorites are (1) practical bug finders need to be deterministic, otherwise developers cannot easily know if they are making progress on removing bugs from their software and (2) the biggest challenges in testing arbitrary code is dealing with peculiarities of parsing and compiling code in a non-standard build system.

References

[1] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. A few billion lines of code later: Using static analysis to find bugs in the real world. *Commun. ACM*, 53(2):66–75, feb 2010.