Faster Automatic Test Case Generation

Ott Tinn
University of Edinburgh

This talk presents a new tool that extends an existing symbolic execution tool for C programs to partially support C++ and makes it faster at finding inputs that could crash the analysed programs. The speedup is achieved by adding a new transformation phase that aims to make bug finding faster without optimizing any relevant bugs away by trying to computes just the one bit per input that shows whether the program crashes on that input.

The transformation phase transforms the programs such that regular observable behaviour is removed if it is not needed (for example it is not necessary to produce output if it is not evaluated). Then it adds validation calls to make sure the compiler can not optimize away potentially dangerous operations such as memory accesses. After that optimizing transforms such as the following are applied:

- Validation removal: if a validation call is always preceded by another one that would catch at least the same set of bugs then it could be removed. The same can be done for calls that can never fail.
- Validation hoisting: move each validation call to the earliest point in the control flow graph (CFG) where it could be executed without changing the CFG or the set of inputs on which the program crashes.
- Loop removal: if a loop has no side effects and a known iteration count then it could be removed by replacing the induction variable with a symbolic variable that takes the same range. This makes the loop get executed with any of the values instead of all, but that is enough to find any potential crashes.

The sample was an extensive set of programs from a programming competition archive. The evaluation showed that on that class of C++ programs an input that triggers a crash can be found in less than five minutes for a significant number of cases and that the new system takes about half as much time as the base system if the time limit is between a minute and an hour per program. A bug was found in at most one hour in roughly half the cases where a program was known to crash[1] on some input.

In general similar bug finding systems should be usable for finding bugs in small programs with clearly defined input spaces and not too complicated logic. Thus these systems should be usable for finding bugs in students' assignments, other small programs, and parts of programs (e.g. a few classes).

The new transformation stage should also be applicable to other similar bug finding systems. Without the loop transformations it could even be used in systems without symbolic reasoning abilities, such as fuzzers.

[1]Some of the known crashes were undetectable by the systems so it is not clear how many detectable crashes were not found.