Developing your own SMT-based Model Checker

Prof. Natasha Sharygina and Grigory Fedyukovich

Model Checking [1] is a well-known scientific approach to check safety of a program. It is a fully automatic approach to decide whether a program is safe with respect to a given assertion or to provide a witness of the bug. Assertion is a logical expression over program variables in a particular program location. The set of well-known model checkers for software [2,3,4,5,6] includes FunFrog [5] and eVolCheck [6], developed at Formal Verification and Security group at USI.

In a nutshell, FunFrog delegates the problem of checking safety to an external decision procedure (more concretely, a SAT solver) called PeRIPLO [7]. SAT solving allows precise reasoning over the program variables and arithmetic operations, but requires an expensive encoding the program into a bit-blasted form. As a result, the SAT formulas produced even from the simplest C programs become large and incomprehensible. Alternatively, there exist SAT modulo theories (SMT) solvers that can deal with reals and integers without bit-blasting. One of the most known SMT solvers, Z3 [8], provides an understandable interactive framework available online.

We propose to extend FunFrog to support SMT encoding. The student will be given an access to the sources of the model checker, and will be taught the algorithms used in the SAT/SMT encoding. Then the student will be asked to implement new algorithms and evaluate the resulting tool. We believe, the support of SMT solving will drastically improve the performance of FunFrog for certain benchmarks. In such a case, the results of the student's work will be used as a basis for a publication in a highly ranked scientific conference in Model Checking. Thus, if the student is planning to start a research career, the experience with SAT/SMT techniques will be useful.

While this project does not require any theoretical background in Formal Methods, we will appreciate if the student already has some background knowledge in Logics and Theory of Computation. Initial experience in C++ is required.

References:

[1] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999

[2] E. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In *Tools and Alg. for Con. and Anal. of Sys. (TACAS '04)*, LNCS, pages 168–176, 2004.
[3] E. Clarke, D. Kroening, N. Sharygina, and K. Yorav. SATABS: SAT-based Predicate Abstraction for ANSI-C. In *Tools and Alg. for Con. and Anal. of Sys. (TACAS '05)*, LNCS, pages 570–574, 2005.
[4] D. Beyer and M. E. Keremoglu. CPAchecker: A Tool for Configurable Software Verification. In *Computer Aided Verification (CAV '11)*, LNCS, pages 184–190, 2011.
[5] O. Sery, G. Fedyukovich, and N. Sharygina. FunFrog: Bounded Model Checking with Interpolation-based Function Summarization, In *ATVA'12*
[6] G. Fedyukovich, O. Sery, and N. Sharygina. eVolCheck: Incremental Upgrade Checker for C. In *TACAS'13*
[7] http://verify.inf.unisi.ch/periplo.html
[8] http://rise4fun.com/z3