LLBMC: Improved Bounded Model Checking of C Programs Using LLVM*
(Competition Contribution)

Stephan Falke, Florian Merz, and Carsten Sinz

Institute for Theoretical Computer Science
Karlsruhe Institute of Technology (KIT), Germany
{stephan.falke,florian.merz,carsten.sinz}@kit.edu

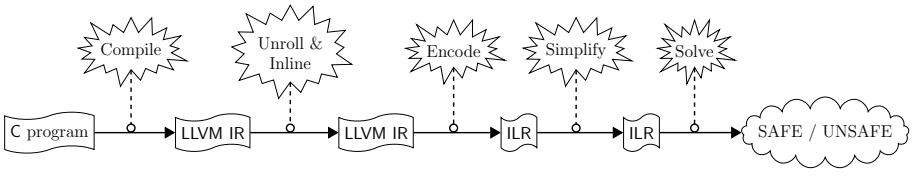
Abstract. LLBMC is a tool for detecting bugs and runtime errors in C and C++ programs. It is based on bounded model checking using an SMT solver and thus achieves bit-accurate precision. A distinguishing feature of LLBMC in contrast to other bounded model checking tools for C programs is that it operates on a compiler intermediate representation and not directly on the source code.

1 Verification Approach

Bounded model checking (BMC) of C, pioneered by Clarke, Kroening and Lerda [1], is a well-established method for detecting bugs and runtime errors. A number of mature tools for BMC of C programs already exists [1,2,6,8]. These tools only investigate finite paths in programs by bounding the number of loop iterations and the function call depth that is considered. This way, property checking becomes decidable using SMT solvers for the combined theory of bitvectors and arrays, where the latter are used to model the computer's main memory.

2 Software Architecture

Details on LLBMC's architecture and features can be found elsewhere [3,4,8,9]. The overall approach is summarized in the following figure:



First, the C program is compiled into the compiler intermediate representation LLVM IR [7]. Then, loops are unrolled, functions are inlined, and the resulting

* This work was supported in part by the "Concept for the Future" of Karlsruhe Institute of Technology within the framework of the German Excellence Initiative.

N. Piterman and S. Smolka (Eds.): TACAS 2013, LNCS 7795, pp. 623–626, 2013.

LLVM IR program is encoded into LLBMC's intermediate logic representation ILR. The ILR formula is finally simplified, lowered to an SMT formula, and solved using the SMT solver STP [5].

In comparison to the version that participated in SV-COMP 2012, this year's version of LLBMC offers the following improvements:

- Lazy, on-demand loop unrolling, function inlining, and ILR encoding.
- Uninitialized local variables are automatically set to nondeterministic values.
- SMT-based support for `memcpy`, `memset`, and `memmove` as an extension of the theory of arrays [4]. This is an alternative to providing C implementations.
- Extended support for many C library functions and `gcc` built-in functions. These library functions are provided as a module containing implementations of the functions in LLVM IR, where the module is automatically linked to the module obtained from the C program.
- Utilizes new versions of LLVM (version 3.1) and STP (revision 1668).

3 Strengths and Weaknesses of the Approach

LLBMC is tailored towards finding bugs in C programs, in particular memory-related ones. Detectable errors include common ones such as arithmetic overflow and underflow, invalid memory access operations, and invalid use of the memory allocation system (including invalid `free`s and memory leaks). Furthermore, LLBMC supports checking of user assertions and reachability of labels in the C program. In SV-COMP 2013, checking for most of these errors has been disabled and only reachability of the error label "ERROR" resp. only memory safety checks (in the category "MemorySafety") are performed.

In the competition, LLBMC is used with a maximal loop iteration bound of 10 and a maximal (recursive) function call depth of 2, where these bounds are increased iteratively based on the previous run of the tool. If no error is found within these maximal bounds, the instance is considered safe.

LLBMC did not participate in the categories "ControlFlowInteger-MemSimple" (LLBMC does not support the simplistic memory model), "Concurrency" (not supported by LLBMC), and "DeviceDrivers64" (since most of these programs implicitly also assume a simplistic memory model). In the categories where LLBMC participated, it performed very well, winning two categories ("BitVectors" and "Loops") and taking second place in four categories. LLBMC did not produce any incorrect result, but the time or memory limit was exhausted in 69 cases (out of the 1000 cases on which LLBMC was executed).

4 Tool Setup and Configuration

The version of LLBMC submitted to SV-COMP 2013 can be downloaded from

`http://llbmc.org/llbmc-sv-comp-13.zip`

LLBMC requires clang (version 3.1) in order to convert C input files to LLVM IR. The ZIP archive contains a wrapper shell script, llbmcc, to run LLBMC on individual C files that iteratively increases the loop iteration and function call depth if these bounds were reported to be insufficient by the previous run of LLBMC. In fact, the script runs LLBMC twice for each bound: In the first run it searches only for program errors, but does not check bounds. If no program error is found, a bounds check is performed in the second run.

By default, llbmcc only performs a reachability check for a basic block labelled "ERROR", but no other checks. In this case it outputs either SAFE, if the error label is unreachable (within the maximal bounds), or UNSAFE otherwise. Notice that LLBMC performs its analysis for a 32-bit machine and does not participate in the "DeviceDrivers64" category, which would require the analysis to be performed for a 64-bit machine (the script, however, supports -m64 as the first argument if the analysis for a 64-bit machine is desired). For the "MemorySafety" category, the script should be run with -mem-safety as the first argument. The script then checks for invalid frees, invalid memory dereferences, and memory leaks, but does not perform any other checks. In this case, it outputs either TRUE, if the program is memory safe (within the maximal bounds), or one of FALSE(p_valid-free), FALSE(p_valid-deref), or FALSE(p_valid-memtrack) if the corresponding memory safety property is violated in the verification task.

5 Software Project and Contributors

LLBMC is developed by Stephan Falke, Florian Merz, and Carsten Sinz at the Karlsruhe Institute of Technology (KIT) in Karlsruhe, Germany. The tool is available under either an unlimited non-commercial (academic) license or under an evaluation license that is valid for 30 days and suitable for a commercial setting. Further information on LLBMC can be found at http://llbmc.org.

References

1. Clarke, E., Kroning, D., Lerda, F.: A Tool for Checking ANSI-C Programs. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 168–176. Springer, Heidelberg (2004)
2. Cordeiro, L., Fischer, B., Marques-Silva, J.: SMT-based bounded model checking for embedded ANSI-C software. In: Proc. ASE 2009, pp. 137–148 (2009)
3. Falke, S., Merz, F., Sinz, C.: A theory of C-style memory allocation. In: Proc. SMT 2011, pp. 71–80 (2011)
4. Falke, S., Sinz, C., Merz, F.: A theory of arrays with set and copy operations (extended abstract). In: Proc. SMT 2012, pp. 97–106 (2012)
5. Ganesh, V., Dill, D.L.: A Decision Procedure for Bit-Vectors and Arrays. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 519–531. Springer, Heidelberg (2007)
6. Ivančić, F., Yang, Z., Ganai, M.K., Gupta, A., Ashar, P.: Efficient SAT-based bounded model checking for software verification. TCS 404(3), 256–274 (2008)

7. Lattner, C., Adve, V.S.: LLVM: A compilation framework for lifelong program analysis & transformation. In: Proc. CGO 2004, pp. 75–88 (2004)
8. Merz, F., Falke, S., Sinz, C.: LLBMC: Bounded Model Checking of C and C++ Programs Using a Compiler IR. In: Joshi, R., Müller, P., Podelski, A. (eds.) VSTTE 2012. LNCS, vol. 7152, pp. 146–161. Springer, Heidelberg (2012)
9. Sinz, C., Falke, S., Merz, F.: A precise memory model for low-level bounded model checking. In: Proc. SSV 2010 (2010)