Handling Unbounded Loops with ESBMC 1.20
(Competition Contribution)

Jeremy Morse[1], Lucas Cordeiro[2], Denis Nicole[1], and Bernd Fischer[1,3]

[1] Electronics and Computer Science, University of Southampton, UK
[2] Electronic and Information Research Center, Federal University of Amazonas, Brazil
[3] Department of Computer Science, Stellenbosch University, South Africa
esbmc@ecs.soton.ac.uk

Abstract. We extended ESBMC to exploit the combination of context-bounded symbolic model checking and k-induction to prove safety properties in single- and multi-threaded ANSI-C programs with unbounded loops. We now first try to verify by induction that the safety property holds in the system. If that fails, we search for a bounded reachable state that constitutes a counterexample.

1 Overview

ESBMC is a context-bounded symbolic model checker that allows the verification of single- and multi-threaded C code with shared variables and locks. Previous versions of ESBMC can only be used to find property violations up to a given bound k but not to prove properties, unless we know an upper bound on the depth of the state space; however, this is generally not the case. In this paper, we sketch an extension of ESBMC to prove safety properties in bounded model checking (BMC) via mathematical induction. The details of ESBMC are described in our previous work [2–4]; here we focus only on the differences to the version used in last year's competition (1.17), and in particular, on the combination of the k-induction method with the normal BMC procedure.

2 Differences to ESBMC 1.17

Except for the loop handling described below, ESBMC 1.20 is largely a bugfixing version. The main changes concern the memory handling, the internal data structures (where we replaced CBMC's string-based accessor functions), and the Z3 encoding (where we replaced the name equivalence used in the pointer representation by the more appropriate structural equivalence). We have also changed our `pthread`-handling and added missing sequence points (most importantly at the `pthread_join`-function), which can lead to an increase in the number of interleavings to be explored. These changes lead to substantial improvements in robustness and speed, as a comparison of both versions over this year's benchmarks shows. Over the entire competition set of 2315 benchmarks and with an unwind bound of $n = 6$, ESBMC 1.20 produces 70 internal assertion violations, compared to a total of 405 for ESBMC 1.17. The new version also produces a smaller number of false results, leading to a score improvement of more than 25%, while the overall verification time is reduced by about 25%.

3 Loop Handling

One way to prove properties in model checking is by means of induction [1, 6, 8]. The k-induction method has already been successfully applied to verify hardware designs (represented as finite state machines) using a SAT solver, and first attempts to apply this technique to software have been made recently [5, 7]. We sketch the basic idea of our implementation in terms of temporal induction (i.e., the induction is carried out over the time steps of the finite state machines) [6, 7].

Our implementation uses an iterative deepening approach and checks, for each k up to a given maximum, three different cases called *base case*, *forward condition*, and *inductive step*. Intuitively, in the base case, we aim to find a counterexample with up to k loop unwindings; in the forward condition, we check that P holds in all states reachable within k unwindings; and in the inductive step, we check that whenever P holds for k unwindings, it also holds after the next unwinding of the system. We derive the verification conditions (VCs), which are denoted by $Base_k$, Fwd_k, and $Step_k$, respectively, for these program unwindings; if $Base_k$ is satisfiable, then we have found a violation of the safety property, and if Fwd_k or $Step_k$ are unsatisfiable, then the property holds.

The base case and the forward condition can be implemented with the right choice of existing command line parameters. For the base case we call ESBMC as follows:

```
esbmc --no-unwinding-assertions --unwind <i> <file>
```

This inserts an unwinding *assumption* consisting of the termination condition after each loop instead of the usual unwinding *assertion*. For the forward condition, we simply remove `--no-unwinding-assertions` from the call; note that we do not check whether paths are cycle-free as in [7]. The inductive step is more complex. In the approach by Grosse et al. [7], the state is havocked before the loop: all variables are assigned non-deterministic values. Then the loop is run $k - 1$ times, where all post-loop states are assumed to be different; in the loop body, all assertions are replaced by assumptions, which ensures that the chosen values satisfy a consequence of the (unknown) loop invariant. Lastly, the loop is run one final time, before the invariant is checked for the final state. However, as the competition benchmarks only check for reachability of the error label this schema does not have enough information to constrain the havocked variables. We thus havoc only the variables that occur in the loop's termination condition. This heuristic works well for the competition benchmarks.

4 Competition Approach

k-induction is more expensive than plain BMC because it uses iterative deepening and repeatedly unwinds the program, and because it produces more VCs (i.e., for base, forward, and step case). We thus combine k-induction and plain BMC; we first run the k-induction up to a maximum unwind bound, with an additional timeout to force early termination when its attempts fail, and then follow this by a plain BMC call:

```
esbmc --no-unwinding-assertions --partial-loops
      --unwind 6 <file>
```

`--partial-loops` removes the unwinding assumption, and thus allows paths where loops are executed only partially. We use a small script that glues together the ESBMC calls; it also sets the specific parameters for the memory safety category. The unwind bound and the distribution of the times allocated to both phases have been determined experimentally.

5 Results

With the k-induction enabled, ESBMC proves 1670 out of 1805 correct programs correct and finds errors in 424 of the 510 incorrect programs. However, it also claims errors in 17 correct programs and fails to find existing errors in 19 programs; most of these failures are in the `BitVectors` and `ControlFlowInteger` categories. ESBMC produces 115 time-outs, which are concentrated on the larger benchmarks (in particular `DeviceDrivers64` and `SystemC`). ESBMC produces good results for most categories except for `HeapManipulation`, where we opted out.

k-induction by itself is by far not as strong as plain BMC. In the training phase (which was run on similar hardware) it proved only 992 programs correct and found errors in only 98, although it also produced substantially fewer false results. The iterative deepening, with the repeated unwinding of the program, requires much more time for the symbolic execution of the program, and the higher number of VCs require more time in the SMT solver. However, in combination with plain BMC it is a useful technique, increasing the latter's raw score by about 200 marks. As expected, k-induction is particularly successful in the `Loops`-category, where it prevents 14 false results.

Availability. The script and self-contained binaries for 32-bit and 64-bit Linux environments are available at `www.esbmc.org`; versions for other operating systems are available on request. The competition version only uses the Z3 solver (V3.2).

Acknowledgements. The continued development of ESBMC is funded by the Royal Society and by INdT.

References

1. A. Bradley. SAT-Based Model Checking without Unrolling. *VMCAI*, *LNCS* 6538, pp. 70–87, 2011.
2. L. Cordeiro and B. Fischer. Verifying Multi-Threaded Software using SMT-based Context-Bounded Model Checking. *ICSE*, pp. 331–340, 2011.
3. L. Cordeiro, B. Fischer, and J. Marques-Silva. SMT-based bounded model checking for embedded ANSI-C software. *IEEE Trans. Software Eng.*, v. 38, n. 4, pp. 957–974, 2012.
4. L. Cordeiro, J. Morse, D. Nicole, and B. Fischer. Context-Bounded Model Checking with ESBMC 1.17 (Competition Contribution). *TACAS*, *LNCS* 7214, pp. 534–537, 2012.
5. A. Donaldson, D. Kroening, and P. Rümmer. Automatic analysis of scratch-pad memory code for heterogeneous multicore processors. *TACAS*, *LNCS* 6015, pp. 280–295, 2010.
6. N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. *Electr. Notes Theor. Comput. Sci.*, v. 89, n. 4, pp. 543-560, 2003.
7. D. Große, H. M. Le, R. Drechsler. Proving transaction and system-level properties of untimed SystemC TLM designs. *MEMOCODE*, pp. 113–122, 2010.
8. M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. *FMCAD*, *LNCS* 1954, pp. 108–125, 2000.