ESBMC 1.22
(Competition Contribution)

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

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

Abstract. We have implemented an improved memory model for ESBMC which better takes into account C's memory alignment rules and optimizes the generated SMT formulae. This simultaneously improves ESBMC's precision and performance.

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. ESBMC was originally branched off CBMC (v2.9) [4] and has inherited its object-based memory model. With the increasingly large SV-COMP benchmarks this is now reaching its limits. We have thus implemented an improved memory model for ESBMC; however, we opted for an incremental change and have kept the underlying object-based model in place, rather than adapting a fully byte-precise memory model as for example used by LLBMC [7]. We believe this strikes the right balance between precision and scalability.

In this paper we focus on the differences from the ESBMC version used in last year's competition (1.20) and, in particular, on the memory model; an overview of ESBMC's architecture and more details are given in our previous work [1–3, 5].

2 Differences to ESBMC 1.20

In the last year we have mostly made changes to improve ESBMC's stability, precision, and performance. In addition to the improved memory model (see below) we made a wide range of bug fixes and replaced the string-based accessor functions of the intermediate representation (which also go back to CBMC v2.9) by proper accessor functions. This change alone improves ESBMC's speed by roughly a factor of two.

3 Memory Model

The correct implementation of operations involving pointers is a significant challenge in model checking C programs. As a bounded model checker, ESBMC reduces the bounded program traces to first order logic, which requires us to eliminate pointers in the model checker. We follow CBMC's approach and use a static analysis to approximate for each pointer variable the set of data objects (i.e., memory chunks) at which it *might* point at some stage in the program execution. The data objects are numbered, and a pointer target is represented by a pair of integers identifying the data object and the offset within the object. The value of a pointer variable is then the set of (*object*, *offset*)-pairs to which the pointer may point at the current execution step. The result of a dereference is the union of the sets of values associated with each of the (*object*, *offset*)-pairs.

The performance of this approach suffers if pointer offsets cannot be statically determined, e.g., if a program reads a byte from an arbitrary offset into a structure. The resulting SMT formula is large and unwieldy, and its construction is error-prone. To avoid this, we extended the static pointer analysis to determine the weakest alignment guarantee that a particular pointer variable provides, and inserted padding in structures to make all fields align to word boundaries, as prescribed by C's semantics.

These guarantees, in combination with enforcing memory access alignment rules, allow us to significantly reduce the number of valid dereference behaviours and thus the size of the resulting formula, and to detect alignment errors which we have previously ignored. In circumstances where the underlying type of a memory allocation is unclear (e.g., dynamically allocated memory with nondeterministic size), we fall back to allocating a byte array and piecing together higher level types from the bytes.

Other models checkers (in particular LLBMC [7]) treat all memory as a single byte array, upon which all pointer accesses are decomposed into byte operations. This can lead to performance problems due to the repeated updates to the memory array that need to be reflected in the SMT formula.

4 Competition Approach

In bounded model checking, the choice of the unwinding bounds can make a huge difference. In contrast to previous years, where we only used a single experimentally determined unwinding bound, we now operate an explicit iterative deepening schema ($n = 8, 12, 16$). This replaces the iterative deepening that is implicit in the k-induction that we used last year [5]. In addition, we no longer use the partial loops option [3]. For categories other than `MemorySafety` we only check for the reachability of the error label and ignore all other built-in properties. We use a small script that implements iterative deepening and calls ESBMC with the main parameters set as follows:

```
esbmc --timeout 15m --memlimit 15g --64 --unwind <n>
 --no-unwinding-assertions --no-assertions --error-label ERROR
 --no-bounds-check --no-div-by-zero-check --no-pointer-check <f>
```

Here, `--no-unwinding-assertions` removes the unwinding assertion and thus a correctness *claim* is not a full correctness *proof*; however, this increases ESBMC's performance in the competition. The script also sets the specific parameters for the `MemorySafety` category. The run script and a self-contained binary for 64-bit Linux environments are available at `www.esbmc.org/download.html`; other versions are available on request. For the competition we used the Z3 solver (V4.0).

5 Results

With the approach described above, ESBMC correctly claims 1837 benchmarks correct and finds existing errors in 557. However, it also finds unexpected errors for 38 benchmarks and fails to find the expected errors in another 52. The failures are concentrated in the `MemorySafety` and `Recursive` categories, where we produce 36 and 15 unexpected results, respectively. In `MemorySafety`, these are caused by differences in the memory models respectively assumed by the competition, and implemented in ESBMC; in particular, in 22 cases ESBMC detects an unchecked dereference of a pointer to a freshly allocated memory chunk, which can lead to a null pointer violation and so mask the result expected by the benchmark. In `Recursive`, all unexpected results are false alarms, which are caused by bounding the programs. Additionally, ESBMC produces 259 time-outs, which mostly stem from the larger benchmarks in `ldv-consumption`, `ldv-linux-3.4-simple`, `seq-mthreaded`, and `eca`. The remaining programs fail due to parsing errors (16), conversion error (1), or different internal (mostly out-of-memory) errors during the symbolic execution (108). ESBMC produces good results for all categories but `MemorySafety` and `Recursive`; however, since we did not opt out of these, our overall result suffered substantially.

ESBMC's performance has improved greatly over last year's version (v1.20). The number of errors detected has gone up from 448 to 557, while the number of unexpected and missed errors has gone down, from 53 to 38 and from 209 to 52, respectively. The biggest improvements are in the categories `Sequentialized` and `ControlFlowInteger`.

Demonstration Section. We took part in the stateful verification, error-witness checking, and device-driver challenges tracks. In particular, we use EZProofC [6] to collect and manipulate the counterexample produced by ESBMC in order to reproduce the identified error for the first round (B1) of the error-witness checking.

Acknowledgements. The third author thanks Samsung for financial support.

References

1. Cordeiro, L., Fischer, B.: Verifying Multi-Threaded Software using SMT-based Context-Bounded Model Checking. In: ICSE, pp. 331–340 (2011)
2. Cordeiro, L., Fischer, B., Marques-Silva, J.: SMT-based bounded model checking for embedded ANSI-C software. IEEE Trans. Software Eng. 38(4), 957–974 (2012)
3. Cordeiro, L., Morse, J., Nicole, D., Fischer, B.: Context-Bounded Model Checking with ESBMC 1.17 (Competition Contribution). In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 534–537. Springer, Heidelberg (2012)
4. Kroening, D., Clarke, E., Yorav, K.: Behavioral Consistency of C and Verilog Programs Using Bounded Model Checking. In: DAC, pp. 368–371. IEEE (2003)
5. Morse, J., Cordeiro, L., Nicole, D., Fischer, B.: Handling Unbounded Loops with ESBMC 1.20 (Competition Contribution). In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 619–622. Springer, Heidelberg (2013)
6. Rocha, H., Barreto, R., Cordeiro, L., Neto, A.D.: Understanding Programming Bugs in ANSI-C Software Using Bounded Model Checking Counter-Examples. In: Derrick, J., Gnesi, S., Latella, D., Treharne, H. (eds.) IFM 2012. LNCS, vol. 7321, pp. 128–142. Springer, Heidelberg (2012)
7. Sinz, C., Falke, S., Merz, F.: A Precise Memory Model for Low-Level Bounded Model Checking. In: SSV, USENIX (2010)