Verification and Refutation using Witness Checker in ESBMC+DepthK v2.1

(Competition Contribution)

Williame Rocha[1], Herbert Rocha[2], Lucas Cordeiro[1], and Bernd Fischer[3]

[1]Electronic and Information Research Center, Federal University of Amazonas, Brazil
[2]Department of Computer Science, Federal University of Roraima, Brazil
[3]Division of Computer Science, University of Stellenbosch, South Africa

Abstract. We have improved the *k*-induction schema of ESBMC v1.20 and integrated it with CPAchecker's witness checker, in order to participate in all categories of SV-COMP. This combination is implemented in a tool called DepthK, which first analyzes the C program and then checks whether k-induction or bounded model checking is applied together with the witness checker. Combined with ongoing bug fixes and a more conservative choice of supported categories this has decreased the number of wrong proofs and false alarms by an order of magnitude.

1 Verification Approach

ESBMC is a context-bounded symbolic model checker that verifies single- and multi-threaded C programs [1,2]. ESBMC can be used either to find property violations up to a given bound k or to prove correctness by *k*-induction schema [3,4,5].

Here, we combine *k*-induction with bounded model checking (BMC) and integrated it with CPAchecker's witness checker [6]. We implement this approach in a tool called DepthK [4], which first analyzes the C program together with a given property and then determines whether *k*-induction or plain BMC is applied.

In the *k*-induction implementation, the base case tries to find a counterexample up to a maximum number of iterations *k*. In the forward condition, global correctness of the loop w.r.t. the property is shown for the case that the loop iterates at most k times; and the inductive step checks that, if the property is valid in *k* iterations, then it must be valid for the next iterations. DepthK runs (incrementally) up to 100 iterations, which produces the best scores, and only increases the value of k either if it can not prove correctness in the forward condition and inductive step or refute the property during the base case. The plain BMC implementation works similar to the base case.

In the base case, if a counterexample is found, then DepthK confirms it using CPAchecker's witness checker via a *graphml* file[1]; this step is needed due to limitations in the memory model adopted by ESBMC [7]. Additionally, in the forward condition and the inductive step, DepthK re-checks the "true" results adopting an increment of 25 to the k value; this re-checking procedure is needed due to the absence of (strong) loop invariants in ESBMC. We observed that this re-checking procedure together with the witness checker have significantly improved ESBMC's results, given that we are able to

[1] http://www.sosy-lab.org/∼dbeyer/cpa-witnesses/

decrease the number of wrong proofs and false alarms by an order of magnitude, if we compare to the last year results [8]. Differently from previous years, we now use two different SMT solvers in ESBMC: Z3[2] v4.0 for the *k*-induction schema and Boolector[3] v2.0.1 for plain BMC. DepthK v2.1 participates in all categories of SV-COMP 2016.

2 DepthK Architecture

DepthK is implemented as a source-to-source transformation tool. Fig. 1 shows the main software components of DepthK. The white boxes represent the components that we implemented in DepthK (except for C source), while the gray boxes with dashed lines represent the components that we reused from the PyCparser[4] tool. Additionally, the gray boxes with solid lines indicate the software verifiers ESBMC and CPAChecker that we call to verify the C program and validate the counterexample, respectively.

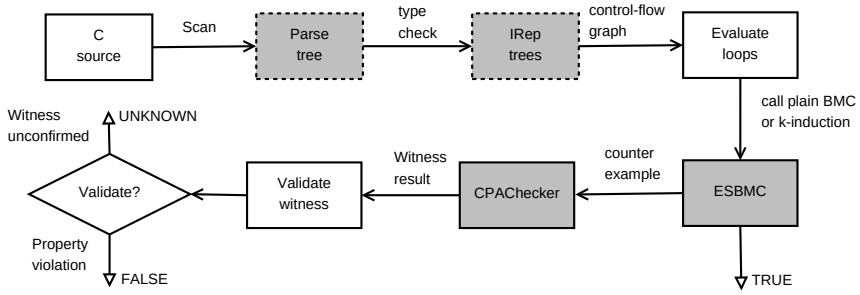

Fig. 1. Overview of the DepthK architecture.

From the control-flow graph produced by PyCparser, DepthK analyzes the (finite and infinite) loops and decides whether *k*-induction or plain BMC is applied to verify or refute the property, respectively. This step is needed since most incorrect results provided by ESBMC's *k*-induction are related to limitation in the coding of the inductive step transformations, given that it does not support `continue` and `return` (from inside a loop) statements. Additionally, we need to re-check the "true" results provided by the inductive step because we do not havoc all program variables, in order to check for the reachability of an error label, due to absence of (strong) loop invariants to constrain the havocked variables in the program.

To efficiently reduce the number of wrong alarms, DepthK calls CPAChecker to validate the counterexample in the *graphml* format, which is automatically generated by ESBMC. If CPAChecker produces a different witness result than the one produced by ESBMC, then DepthK returns *UNKNOWN* as (verification) result; otherwise, it returns *FALSE*. To further reduce the number of wrong proofs, DepthK re-checks the "true" results reported by both forward condition and inductive step, adopting an increment of the actual k value; if a property violation is found during this re-checking procedure, then the (verification) result is inconclusive and DepthK returns *UNKNOWN*; otherwise, it returns *TRUE*.

[2] https://z3.codeplex.com/
[3] http://fmv.jku.at/boolector/
[4] https://github.com/eliben/pycparser

3 Tool Setup and Configuration

Download and Installation Instructions. DepthK v2.1 for 64-bit Linux environment is available to freely download from `www.esbmc.org/download.html` under GPL license. It must be installed as a Python (v2.7.1 or higher) script and it also requires Pycparser (v2.10); Uncrustify[5] (v0.60); Ctags[6] (v5.8); and open-jdk-7-jre[7]. The verifiers ESBMC (v2.0) and CPAChecker (v1.3.10) are included into DepthK distribution.
User Interface. DepthK is invoked via a command-line as follows:

```
./depthk-wrapper -c propertyFile.prp file.i
```

DepthK accepts the property file and the verification task and provides as verification result: *TRUE, FALSE + Witness, or UNKNOWN*. For each error-path, a file that contains the violation path is generated in DepthK root-path *graphml* folder; this file has the same name of the verification task with the extension `graphml`.

4 Results

ESBMC+DepthK correctly claims 1796 benchmarks correct and finds existing errors in 1474, i.e., 3270 correct results; however, 498 errors are not validated due to the limitations in the witness generation. ESBMC+DepthK also finds unexpected errors for 11 benchmarks and fails to find the expected errors in another 11. ESBMC+DepthK achieves 3110 scores in the `overall` category. In SVCOMP 2015, ESBMC correctly claims 3898 correct results; however, it reports 122 false negatives and 318 false positives, which leads to -2161 scores in the `overall` category [8].
Limitations. Most incorrect results are related to the `concurrency` category mainly due to insufficient amount of context-switches; in particular, we use 5 context-switches only. Other incorrect results are related to the categories `Loops`, `Sequentialized`, `DeviceDrivers`, and `Heapmanipulation` due to the memory model of ESBMC [7].

References

1. L. Cordeiro and B. Fischer. Verifying Multi-Threaded Software using SMT-based Context-Bounded Model Checking. *ICSE*, pp. 331–340, 2011.
2. 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.
3. J. Morse, L. Cordeiro, D. Nicole, B. Fischer. Handling Unbounded Loops with ESBMC 1.20 - (Competition Contribution). *TACAS*, *LNCS* 7795, pp. 619–622, 2013.
4. H. Rocha, H. Ismail, L. Cordeiro, R. Barreto. Model Checking Embedded C Software using *k*-Induction and Invariants. *SBESC* (to appear), 2015.
5. M. Gadelha, H. Ismail, L. Cordeiro. Handling Loops in Bounded Model Checking of C Programs via k-Induction. *STTT* (to appear), 2015. DOI: 10.1007/s10009-015-0407-9
6. D. Beyer, M. Dangl, D. Dietsch, M. Heizmann, A. Stahlbauer: Witness validation and stepwise testification across software verifiers. *ESEC/SIGSOFT FSE*, pp. 721–733, 2015.
7. J. Morse, M. Ramalho, L. Cordeiro, D. Nicole, and B. Fischer. ESBMC 1.22 - (Competition Contribution). *TACAS*, *LNCS* 8413, pp. 405–407, 2014.
8. D. Beyer Software Verification and Verifiable Witnesses - (Report on SV-COMP 2015). *TACAS*, *LNCS* 9035, pp. 401–416, 2015.

[5] http://uncrustify.sourceforge.net
[6] http://sourceforge.net/projects/ctags/
[7] http://openjdk.java.net/install/