DepthK: A k-Induction Verifier Based on Invariant Inference for C Programs
(Competition Contribution)

Williame Rocha[1], Herbert Rocha[2(✉)], Hussama Ismail[1], Lucas Cordeiro[1,3], and Bernd Fischer[4]

[1] Electronic and Information Research Center, Federal University of Amazonas, Manaus, Brazil
[2] Department of Computer Science, Federal University of Roraima, Boa Vista, Brazil
herberthb12@gmail.com
[3] Department of Computer Science, University of Oxford, Oxford, UK
[4] Division of Computer Science, University of Stellenbosch, Stellenbosch, South Africa

Abstract. DepthK is a software verification tool that employs a proof by induction algorithm that combines k-induction with invariant inference. In order to efficiently and effectively verify and falsify safety properties in C programs, DepthK infers program invariants using polyhedral constraints. Experimental results show that our approach can handle a wide variety of safety properties in several intricate verification tasks.

1 Overview

DepthK is a software verification tool that employs bounded model checking (BMC) and k-induction based on program invariants, which are automatically generated using polyhedral constraints. DepthK uses ESBMC, a context-bounded symbolic model checker that verifies single- and multi-threaded C programs [1,2], as its main verification engine. More specifically, it uses ESBMC either to find property violations up to a given bound k or to prove correctness by using the k-induction schema [3–5]. However, in contrast to the "plain" ESBMC, DepthK first infers program invariants using polyhedral constraints. It can use the PAGAI [8] (employed in the SVCOMP'17) and PIPS tools [9,10] to infer these invariants. DepthK also integrates the witness checkers CPAchecker [6] (employed in the SVCOMP'17) and Ultimate Automizer [7] for checking verification results.

DepthK pre-processes the C program to classify (bounded and unbounded) loops by tracking variables in the loop header. Based on that categorization, DepthK verifies the C program using either plain BMC or k-induction, together with invariant inference and witness checking. The k-induction uses an iterative deepening approach and checks, for each step k up to a maximum value, three different cases, called base case, forward condition, and inductive step, respectively. Intuitively, in the base case, DepthK searches for a counterexample of

© Springer-Verlag GmbH Germany 2017
A. Legay and T. Margaria (Eds.): TACAS 2017, Part II, LNCS 10206, pp. 360–364, 2017.
DOI: 10.1007/978-3-662-54580-5_23

the safety property ϕ with up to k iterations of the loop. The forward condition checks whether loops have been fully unrolled and whether ϕ holds in all states reachable within k iterations. The inductive step verifies that if ϕ is valid for k iterations, then ϕ will also be valid for the next iteration. In order to improve the effectiveness of the k-induction algorithm, DepthK tries to infer invariants that prune the state space and strengthen the induction hypothesis.

2 Verification Approach

DepthK extends ESBMC to falsify or prove correctness of a given (safety) property for any depth without manual annotation of loops with invariants. In our preliminary experiments, the integration of the inferred program invariants, in the form of polyhedral constraints, with the k-induction algorithm allows DepthK to solve more verification tasks than plain ESBMC.

Figure 1 shows an overview of the DepthK tool, with the k-induction algorithm, invariant generation, and witness validation components. The tool's inputs are a C program P (without invariants) and a safety property ϕ. It returns *TRUE* (if there is no path that violates the safety property), *FALSE* (if there exists a path that violates the safety property), or *UNKNOWN* otherwise.

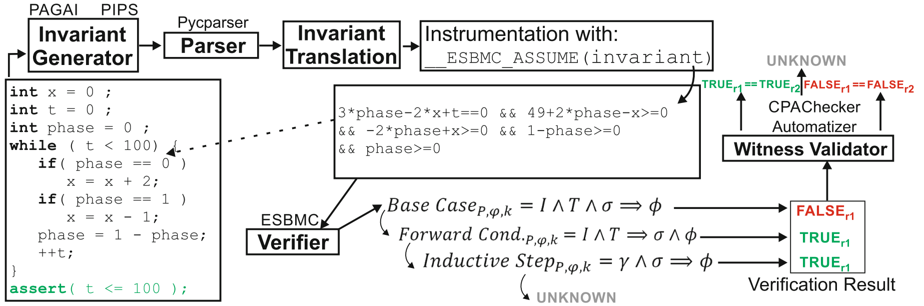

Fig. 1. Flow of the proposed method.

DepthK infers program invariants using the PAGAI and PIPS tools, which are both inter-procedural source-to-source transformation tools for C programs and rely on a polyhedral abstraction of the program behavior. PAGAI applies source code analysis to infer invariants for each control-flow point of a C program using the LLVM infrastructure (see http://llvm.org), focusing on path distinction inside the control-flow graph, while avoiding a systematic exponential path enumeration [8]. PIPS performs a two-step analysis [9]. (1) Each program instruction is associated to an affine transformer, representing its underlying transfer function. This is a bottom-up procedure, starting from elementary instructions, then working on compound statements and up to function definitions. (2) Polyhedral invariants are propagated along with instructions, using the previously computed transformers.

In DepthK, PAGAI and PIPS receive as input the program to be analyzed and generate as output C code that contains invariants written as comments around instructions. These invariants are then translated into assume statements, to constrain all possible values of those variables related to the invariants. DepthK needs to perform this step since PAGAI and PIPS generate invariants represented as mathematical expressions, which are not accepted by the syntax of C programs.

DepthK also checks the results provided by the ESBMC k-induction algorithm. In particular, DepthK checks the results related to the forward condition and inductive step using the witness validators. This re-checking procedure is needed due to the inclusion of invariants, which over-approximates the analyzed program; otherwise, the invariants could result in incorrect exploration of the states sets.

Additionally, DepthK also checks the result provided by the base case of the ESBMC k-induction algorithm, using CPAchecker (as default) or Ultimate Automizer as witness checkers via a *graphml* file. DepthK executes this step due to limitations in the memory model adopted by ESBMC [11]. We observed that the use of witness checkers has significantly improved DepthK's results, given that we are able to decrease the number of wrong proofs and false alarms by an order of magnitude.

3 Architecture, Implementation and Availability

Architecture. DepthK is implemented as a source-to-source transformation tool in Python (v2.7.1). It uses pycparser (v2.10) to parse a C program into an AST, and then identifies and tracks variables for invariant translation and loop classification. Ctags (v5.8, http://sourceforge.net/projects/ctags) identifies C language objects found in C source and header files. Clang (v3.5.0, http://clang.llvm.org) compiles a C file into LLVM bitcode that PAGAI takes as input. PAGAI (employed for SVCOMP'17, http://pagai.forge.imag.fr) generates the program invariants. It uses Uncrustify (v0.60, http://uncrustify.sourceforge.net) as a source code beautifier. ESBMC (v3.1) is employed as k-induction verifier, and CPAchecker (v1.3.10) as witness validator. In the current submission, DepthK uses Z3 (v4.0, https://z3.codeplex.com) as SMT solver in ESBMC's k-induction schema. DepthK participates in all categories of SVCOMP'17.

Availability and Installation. DepthK is freely available under the GPL license. The competition candidate DepthK v3 (for a 64-bit Linux environment) can be downloaded from https://github.com/hbgit/depthk/archive/depthk_v3.tar.gz. It must be installed as a Python script; it also requires the installation of pycparser, Uncrustify, Ctags, Clang, and open-jdk-7-jre (http://openjdk.java.net/install/). The verifiers ESBMC and CPAchecker, and the invariant generator PAGAI are included with the DepthK distribution.

User Interface. DepthK is invoked via a command-line (as in the depthk.py module for BenchExec) as follows: `./depthk-wrapper.sh -c propertyFile.prp`

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

4 Strengths and Weaknesses of the Approach

The strength of the tool lies in the combination of the proof by induction algorithm with the program invariants inference to specify pre- and post-conditions, and witness validation to check the verification results of the k-induction algorithm. DepthK uses CPAchecker as a witness validator to confirm the verification results, which leads to improvements in DepthK to avoid false alarms and wrong proof. However, DepthK is in the initial development and there are still limitations on the structure of the programs and the inference of strong program invariants to prove properties. In particular, in the preliminary experiments with SV-COMP benchmarks, we observed that PAGAI/PIPS tool could not generate strong invariants for the k-induction algorithm, either due to a weak transformer or due to invariants that are not convex. All incorrect answers produced by our tool in the competition are due to bugs in its implementation.

Results. DepthK has proven to be a noticeable improvement over "plain" ESBMC. In particular, it outperforms all ESBMC versions in the sub-categories *ReachSafety-BitVectors*, *ReachSafety-Heap*, *ReachSafety-Loops*, and *MemSafety-Arrays*. It also outperforms CPA-kInd, which implements a similar approach to DepthK, in the sub-categories *ReachSafety-Heap*, *ReachSafety-Recursive*, *Overflows-BitVectors*, as well as in the category *FalsificationOverall*. In total, DepthK produced 1091 confirmed correct true results and 1056 confirmed correct false results, with a further 467 unconfirmed results. It also produced 20 incorrect true results and 32 incorrect false results, mostly due to limitations in ESBMC's memory model.

5 Software Project and Contributors

DepthK is an open-source project, mainly developed by members of the software verification group from Federal University of Roraima and Federal University of Amazonas. The script, source code, and self-contained binaries for 64-bit Linux environments are available at https://github.com/hbgit/depthk/; versions for other operating systems are available on request. The current development of DepthK is funded by the Amazonas State Research Funding Agency (FAPEAM).

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. In: ASE, pp. 137–148 (2009)
3. Morse, J., Cordeiro, L., Nicole, D., Fischer, B.: Handling unbounded loops with ESBMC 1.20. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 619–622. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36742-7_47
4. Gadelha, M.Y.R., Ismail, H.I., Cordeiro, L.C.: Handling loops in bounded model checking of C programs via k-induction. STTT (to appear)
5. Rocha, H., Ismail, H., Cordeiro, L.C., Barreto, R.S.: Model checking embedded C software using k-induction and invariants. In: SBESC, pp. 90–95 (2015)
6. Beyer, D., Keremoglu, M.E.: CPAchecker: a tool for configurable software verification. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 184–190. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_16
7. Heizmann, M., Dietsch, D., Greitschus, M., Leike, J., Musa, B., Schätzle, C., Podelski, A.: Ultimate automizer with two-track proofs. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 950–953. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49674-9_68
8. Henry, J., Monniaux, D., Moy, M.: PAGAI: a path sensitive static analyser. Electron. Notes Theor. Comput. Sci. **289**, 15–25 (2012)
9. PIPS: Automatic parallelizer and code transformation framework (2013). http://pips4u.org
10. Maisonneuve, V., Hermant, O., Irigoin, F.: Computing invariants with transformers: experimental scalability and accuracy. In: NSAD, vol. 307, pp. 17–31 (2014)
11. Morse, J., Ramalho, M., Cordeiro, L., Nicole, D., Fischer, B.: ESBMC 1.22 - (competition contribution). In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 405–407. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54862-8_31