BRICK: Path Enumeration Based Bounded Reachability Checking of C Program (Competition Contribution)*

Lei Bu(✉), Zhunyi Xie, Lecheng Lyu, Yichao Li, Xiao Guo, Jianhua Zhao, and Xuandong Li

State Key Laboratory for Novel Software Technology, Nanjing University, China
bulei@nju.edu.cn

Abstract. BRICK is a bounded reachability checker for embedded C programs. BRICK conducts a path-oriented style checking of the bounded state space of the program, that enumerates and checks all the possible paths of the program in the threshold one by one. To alleviate the path explosion problem, BRICK locates and records unsatisfiable core path segments during the checking of each path and uses them to prune the search space. Furthermore, derivative free optimization based falsification and loop induction are introduced to handle complex program features like nonlinear path conditions and loops efficiently.

1 Verification Approach

Existing bounded software checkers usually encode the bounded state space of the program into one constraint solving problem directly. However, in this manner, when the size of the program or the bound of the checking increases, the corresponding constraint solving problem explodes quickly and becomes difficult to solve by existing SAT/SMT solvers.

To solve this problem, BRICK conducts a path-oriented style checking of the bounded state space of the program, that enumerates and checks all the possible paths in the threshold one by one [1,2]. The main merit of the approach is that, in this case, the size of the problem needs to be solved by the constraint solver is well controlled and can be easily handled. The main features of BRICK's solving are reported below:

1.1 Flexible Path Enumeration

BRICK enumerates potential paths from the control flow graph (CFG) of the given program to the user-defined step bound. Two path enumeration strategies are applied in BRICK, each with its own advantages.

* This work is supported in part by the National Key Research and Development Plan (No. 2017YFA0700604), the Leading-Edge Technology Program of Jiangsu Natural Science Foundation (No. BK20202001), and the National Natural Science Foundation of China (No.62172200, No.61632015).

D. Fisman and G. Rosu (Eds.): TACAS 2022, LNCS 13244, pp. 408–412, 2022.
https://doi.org/10.1007/978-3-030-99527-0_22

First, we can simply conduct classical Depth-first-search (DFS) to enumerate program paths. The benefit of this approach is that, if the DFS stops without touching the given bound, we can get a result that the target state is not reachable in general, not only in the bounded state space.

We have also implemented a special method to encode the jump-to relation between different code blocks into an SAT formula and obtain the potential path by SAT solving. The benefit is that if the potential path is confirmed to be infeasible by following path condition solving, the infeasible path segment in the path can be located and encoded back to the SAT formula to prune all the future paths containing such infeasible segment.

1.2 Infeasible Path Segment Pool Guided State Space Pruning

BRICK conducts the lazy solving of the path by encoding the path condition of the potential path into a feasibility problem. BRICK asks a constraint solver, i.e. SMT solver (Z3 [6]), interval analysis (dReal [4]), and derivative-free optimization-based solving (Section 1.3), to solve the problem. If the path is decided to be infeasible by the solver, BRICK tries to extract the unsatisfiable core (UC) of the feasibility problem of this path, and maps the UC constraints to a infeasible path segment in the path, which will be added to infeasible paths pool. After that, all the paths that contain any infeasible path in the infeasible path pool will be reported as unreachable directly in the following path enumeration.

1.3 Derivative-free Optimization Based Constraint Falsification

We can see that constraint solving plays an important role in BRICK. However, complex path conditions, like nonlinear constraints, which widely appear in programs, are hard to be handled efficiently by the existing solvers. In BRICK, a classification model-based derivative-free-optimization (DFO) approach is used to alleviate this difficult situation by conducting a sample-feedback-learn style DFO solving [8].

More specifically, the underlying solver guesses sample solution for the feasibility problem. Then, we evaluate whether the sampled solution can satisfy the path constraint or not, and calculate the distance between the sampled solution and the correct one if the sampled one does not satisfy the path constraint. Such distance will be used as the metric of feedback in the classification-based DFO learning, to guide the solver to converge to the value that fits the path constraint. In practice, this approach works very well in nonlinear problem solving. However, this DFO-based approach can not tell the target is not reachable, if it fails to find a solution.

1.4 Induction-based Loop Handling

If the target program contains loops, the number of potential paths may explode. To alleviate this problem, we conduct an induction-based proof to try to handle the loop before we start to do the BMC.

First of all, we collect the constraints from the assertions and generate the weakest precondition respectively. Then, we conduct the normal induction-based proof to see whether such constraints are satisfied in any iteration. If no counterexamples are returned, we know that the assertions won't be violated in the loop. Furthermore, we are also working on the integration of loop invariant generation to further refine the CFG under checking.

2 Software Architecture

The architecture of BRICK is shown in Fig.1. It consists of a loop processing module, a path enumerating module, and a constraint solving module, all implemented in C++ language.

In the loop processing module, if the program contains assertion-related loop, BRICK conducts loop induction-based verification firstly. If the induction works, BRICK reports unreachable; otherwise, it builds the program CFG, and performs the following path enumeration based checking.

In the path enumerating module, BRICK employs SAT-based and DFS-based path enumerating methods to extract the program path and its corresponding path condition. The constraint solving module accepts the path condition and performs constraint solving accordingly. All the techniques used has been mentioned in Section 1 respectively. The solvers used in BRICK including SAT solver MiniSAT [3], SMT solver Z3 [6], interval analysis solver dReal [4], and our implementation of the DFO method RACOS [9].

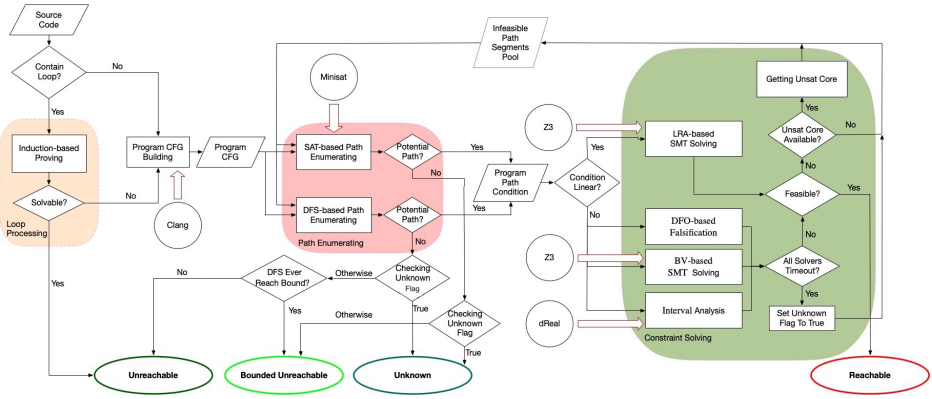

Fig. 1. Architecture of BRICK

3 Strengths and Weaknesses

Most of the bounded reachability checkers, i.e. CBMC [5], encode the bounded state space to a huge SMT formula consisting of both conjunction and disjunc-

tion of different kinds of formulas, which are difficult for the existing solvers to handle and may cause memory explosion easily. Instead, BRICK conducts the verification in a path-oriented way:

- BRICK enumerates and checks all the potential paths one by one. In this manner, the computation complexity is well controlled.
- Meanwhile, as only the ongoing path is saved in the memory and the corresponding path constraints of the path will be disjunction-free, the solving problem is much easier to handle.
- For the sake of processing capability, UC-guided backtracking and path pruning is proposed to prune the search space substantially, and DFO-based solving is conducted to handle complex nonlinear constraints efficiently.

BRICK had participated in the ReachSafety/Floats category of SV-COMP 2022 [10]. BRICK has successfully verified 439 of all the 469 tasks, ranked 1st in this sub-category. Furthermore, we can see that for these 439 solved cases, BRICK only used 1000 seconds in total. On the other hand, CoveriTeam and VeriAbs [7] which won the 2nd and 3rd place in this category spent 9300 and 18000 seconds respectively, which are 9 and 18 times higher than BRICK.

For the weakness, like all the other bounded checkers, BRICK may not be able to give proofs of correctness of a program, if it can not finish the search in the given step bound. In this case, BRICK can only report bounded true. For example, on the cases of SV-COMP 2022, besides the 439 cases which are proved by BRICK, there are also several programs that BRICK can only give a bounded result or just timeout. Therefore, for the future work, we are implementing techniques including loop summary, k-induction and so on to try to abstract the loops and give a proof of the correctness in certain cases.

4 Tool Setup and Configuration

The binary file of BRICK for Ubuntu 20.04 is available at https://github.com/brick-tool-dev/BRICK-2.0. To install the tool, please clone this repository and follow the instruction in README.md. A tailored version of BRICK took part in the ReachSafety/Floats category in SV-COMP 2022 [10]. The version [11] supports the checking of reachability of Error Function. The BenchExec wrapper script for the tool is *brick.py* and *brick.xml* is the benchmark description file.

5 Software Project and Contributors

BRICK is available under MIT License. The team of BRICK is from Software Engineering Group, Nanjing University. We would like to thank Sicun Gao for his kindly help with the usage of dReal.

Data Availability Statement. All data of SV-COMP 2022 are archived as described in the competition report [10] and available on the competition web site. This includes the verification tasks, results, witnesses, scripts, and instructions for reproduction. The version of our verifier as used in the competition is archived together with other participating tools [11].

References

1. L. Bu, *et al.*. BACH: Bounded Reachability Checker for Linear Hybrid Automata. In FMCAD'08, pp. 65-68.
2. D. Xie, *et al.*. SAT-LP-IIS Joint-Directed Path-Oriented Bounded Reachability Analysis of Linear Hybrid Automata. In FMSD. 45(1): 42-62, 2014.
3. N. Eénand N. Sörensson. An extensible SAT-solver. In SAT'03, 502-518.
4. S. Gao, *et al.*. dReal: An SMT solver for nonlinear theories over the reals. In CADE'13, 208-214.
5. D. Kroening, *et al.*. CBMC - C Bounded Model Checker. In TACAS'14: 389-391.
6. L. De Moura and N. Bjørner. Z3: An Efficient SMT Solver. In TACAS'08: 337-340.
7. P. Darke, *et al.*. VeriAbs: Verification by Abstraction and Test Generation - (Competition Contribution). In TACAS'18: 457-462.
8. L. Bu, *et al.*. Machine learning steered symbolic execution framework for complex software code. In Formal Aspects of Computing, 33:3, 301-323, 2021.
9. Y. Yu, *et al.*. Derivative-free optimization via classification. In AAAI'16, 2286-2292.
10. D. Beyer, Progress on Software Verification: SV-COMP 2022. In TACAS'22
11. D. Beyer, Verifiers and Validators of the 11th Intl. Competition on Software Verification (SV-COMP 2022). Zenodo, DOI:10.5281/zenodo.5959149, 2022