FuSeBMC_IA: Interval Analysis and Methods for Test Case Generation

(Competition Contribution)

Mohannad Aldughaim[(✉)1,4], Kaled M. Alshmrany[1,5], Mikhail R. Gadelha[2], Rosiane de Freitas[3], and Lucas C. Cordeiro[1,3]

[1] University of Manchester, Manchester, UK
[2] Igalia, A Coruña, Spain
[3] Federal University of Amazonas, Manaus, Brazil
[4] King Saud University, Riyadh, Saudi Arabia
[5] Institute of Public Administration, Jeddah, Saudi Arabia
mohannad.aldughaim@manchester.ac.uk

Abstract. The cooperative verification of Bounded Model Checking and Fuzzing has proved to be one of the most effective techniques when testing C programs. FuSeBMC is a test-generation tool that employs BMC and Fuzzing to produce test cases. In Test-Comp 2023, we present an interval approach to FuSeBMC_IA, improving the test generator to use interval methods and abstract interpretation (via Frama-C) to strengthen our instrumentation and fuzzing. Here, an abstract interpretation engine instruments the program as follows. It analyzes different program branches, combines the conditions of each branch, and produces a Constraint Satisfaction Problem (CSP), which is solved using Constraint Programming (CP) by interval manipulation techniques called *Contractor Programming*. This process has a set of invariants for each branch, which are introduced back into the program as constraints. Experimental results show improvements in reducing CPU time (37%) and memory (13%), while retaining a high score.

Keywords: Automated Test-Case Generation · Bounded Model Checking · Fuzzing · Abstract Interpretation · Constraint Programming · Contractors.

1 Introduction

In Test-comp 2022 [1], cooperative verification tools showed their strength by being the best tools in each category. *FuSeBMC* [9, 10] is a test-generation tool that employs cooperative verification using fuzzing and BMC. *FuSeBMC* starts with the analysis to instrument the Program Under Test (PUT); then, based on the results from BMC/AFL, it generates the initial seeds for the fuzzer. Finally, *FuSeBMC* keeps track of the goals covered and updates the seeds, while producing test cases using BMC/Fuzzing/Selective fuzzer. This year, we introduce abstract interpretation to *FuSeBMC* to improve the test case generation. In particular, we use interval methods to help our instrumentation and fuzzing by providing intervals to help reach (instrumented) goals faster. The selective fuzzer is a crucial component of *FuSeBMC*, which generates test cases for uncovered goals based on information obtained from test cases produced by BMC/fuzzer [9]. This work is based on our previous study, where CSP/CP by contractor techniques are applied to prune the state-space search [12]. Our approach also uses Frama-C [4, 8] to

obtain variable intervals, further pruning the state space exploration. Our original contributions are: (1) improve instrumentation to allow abstract interpretation to provide information about variable intervals; (2) apply interval methods to improve the fuzzing and produce higher impact test cases by pruning the search space exploration; (3) reduce the usage of resources (incl. memory and CPU time).

2 Interval Analysis and Methods for Test Case Generation

FuSeBMC_IA improves the original *FuSeBMC* using Interval Analysis and Methods [3]. Fig. 1 illustrates the *FuSeBMC_IA*'s architecture. Our approach starts from the analysis phase of *FuSeBMC* [9, 10]. It parses statement conditions required to reach a goal, to construct a Constraint Satisfaction Problem/Constraint Programming (CSP/CP) [5] with three components: constraints (program conditions), variables (used in a condition), and domains (provided by the static analyzer Frama-C via eva plugin [7]). We instrument the PUT with Frama-C intrinsic functions to obtain the domains, which generate intervals of a given set of variables at a specific program location. Then, we apply the contractor to each goal's CSP and output the results to a file used by the selective fuzzer. Contractor Programming is a set of interval methods that estimate the solution

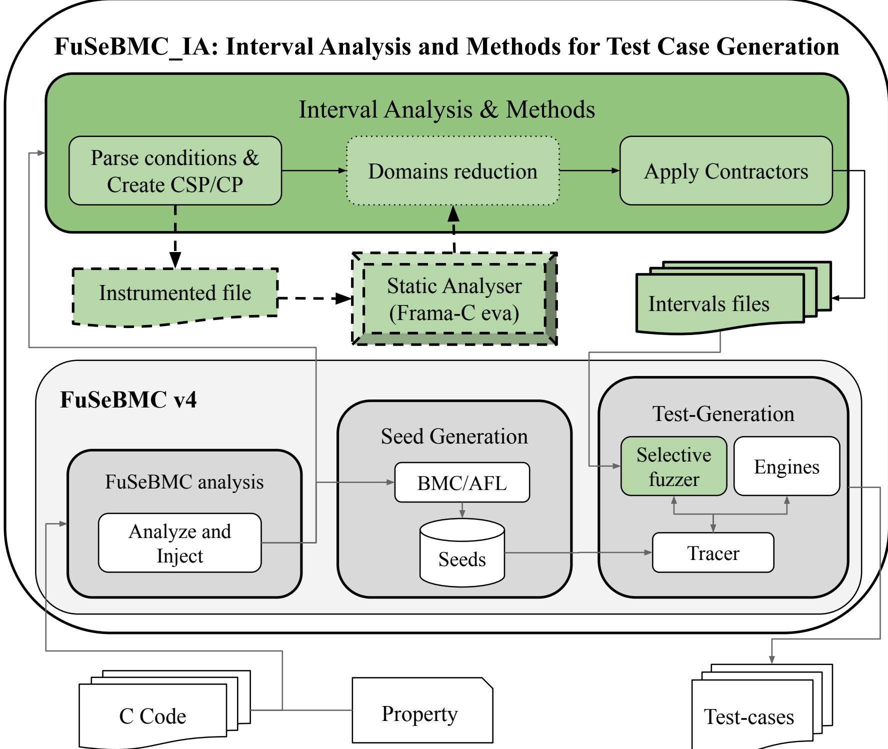

Fig. 1: *FuSeBMC_IA*'s architecture. The changes introduced in *FuSeBMC_IA* for Test-Comp 2023 are highlighted in green. The new Interval Analysis & Methods component generates intervals to be used by the selective fuzzer.

of a given CSP [5]. The used contractor technique is the Forward-Backward contractor, which is applied to a CSP/CP with a single constraint [3], which is implemented in the IBEX library [6]. IBEX is a C++ library for constraint processing over real numbers that

implement contractors. More details regarding contractors can be found in our current work-in-progress [12].

Parsing Conditions and CSP/CP creation for each goal. While traversing the PUT clang AST [2], we consider each statement's conditions that lead to an injected goal: the conditions are parsed and converted from Clang expression [2] to IBEX expression [6]. The converted expressions are used as the constraints in CSP/CP to create a contractor. After parsing the goals, we have a CSP/CP for each goal. In case of a goal does not have a CSP/CP, the intervals for the variables are left unchanged. We also create a constraint for each condition in case of multiple conditions and take the intersection/union. At the end of this phase, we have a list of each goal and its contractor. Also, a list of variables for each contractor will be used to instrument the Frama-C file in the next phase.

```
1   int main() {
2   fuSeBMC_init:;
3     int x = __VERIFIER_nondet_int();
4     int y = 0;
5     if (x <= y) {
6     GOAL_1:;
7       x++;
8     }
9     if (x >= y) {
10      if (x <= 0) {
11      GOAL_2:;
12        x = y;
13      }
14    }
15    if (x > 1 && x < -1) {
16    GOAL_3:;
17      y++;
18    }
19    return 0;
20  }
```

Instrumented file

```
1   int main() {
2     int x = __VERIFIER_nondet_int();
3     int y = 0;
4     if (x <= y) {
5       Frama_C_show_each_GOAL_1_2_(x, y);
6       x++;
7     }
8     if (x >= y)
9       if (x <= 0) {
10        Frama_C_show_each_GOAL_2_2_(x, y);
11        x = y;
12      }
13    if (x > 1 && x < -1) {
14      Frama_C_show_each_GOAL_3_1_(x);
15      y++;
16    }
17    return 0;
18  }
```

Instrumented file for Frama-C

```
1   Goal 1:
2   x
3   -2147483648.000000
4   0.000000
5   y
6   0.000000
7   0.000000
8   Goal 2:
9   x
10  0.000000
11  0.000000
12  y
13  0.000000
14  0.000000
15  Goal 3:
16  Unreachable
```

Intervals file

Fig. 2: The figure illustrates an example of files produced. We are starting from the instrumented file that shows the goals injected. Then, we instrument the file with the Frama-C intrinsic function. Finally, we produce a file with each goal and the intervals to satisfy the conditions for each goal.

Domains reduction. In this step, we attempt to reduce the domains (primarily starting from $(-\infty, \infty)$) to a smaller range. This is done via Frama-C eva plugin (evolved value analysis) [7]. First, during the instrumentation, we make an instrumented file aimed to be used by Frama-C using its intrinsic functions `Frama_c_show_each()` (cf. Fig. 2). This function allows us to add custom text to identify goals and how many variables are in each call. Second, we run Frama-C to obtain the new variable intervals. Finally, we update the domains for the corresponding CSP/CP.

Applying contractors. Contractors will help prune the domains of the variables by removing a subset of the domain that is guaranteed not to satisfy the constraints. With all the components for a CSP/CP available, we now apply the contractor for each goal and produce the output file in Figure 2. The result will be split per goal into two categories. The first category lists each variable and the possible intervals (lower bound followed by upper bound) to enter the condition given. The second category contains unreachable goals, i.e. when the contractor result is an empty vector.

Selective Fuzzer. The Selective Fuzzer parses the file produced by the analyzer, extracts all the intervals, applies these intervals to each goal, and starts fuzzing within the given interval. Thus, pruning the search space from random intervals to informed intervals. The selective fuzzer will also prioritize the goals with smaller intervals and set a low priority to goals with unreachable results.

3 Strengths and Weaknesses

Using abstract interpretation in *FuSeBMC_IA* improved the test-case generation regarding resources. The new contractors generated by the Interval Analysis and Methods component are used by our selective fuzzer: (1) the information provided helps the selective fuzzer to start from a given range of values rather than a random range (as was our strategy in the previous version); (2) the selective fuzzer uses the information about unreachable goals to set their priority low for reachability; (3) when compared to *FuSeBMC* v4, this improvement helped saving CPU time by 37% and memory by 13%, which leads to saving 40% of energy; (4) although our approach produces fewer test cases for a given category, the impact of these test cases is higher in terms of reaching instrumented goals; (5) there is potential for future work to use the information provided by Frama-C, especially regarding overflow warnings. Finally, the intervals provided may not affect the *FuSeBMC_IA*'s outcome in the worst case. i.e., the selective fuzzer performs no better than not having interval information for seed generation. The time it takes to generate the intervals is only a tiny fraction of the time it takes to produce the test cases; its impact when the information is not useful is negligible.

Our approach suffers from a significant technical limitation: *FuSeBMC_IA* cannot create complementary contractors; we can only create intervals that satisfy the constraints of a branch (i.e., outer contractors). In practice, we can only create intervals to `if`-statements and ignore its `else`-statements (the inner contractor). We also skip any `if`-statement inside `else`-statements, as this may lead to unsound intervals. This is a technical limitation rather than a theoretical one: we use run-time type information (RTTI) to identify ibex expressions. However, we link our tool with Clang, which requires compilation with no RTTI information. We are investigating approaches to address this limitation, e.g., to encapsulate all ibex expressions and manually store expression information, but currently, no proper fix has been implemented. Additionally, a bug has been found that caused *FuSeBMC_IA* to crash on some benchmarks that made *FuSeBMC_IA* scores much less than *FuSeBMC* in the coverage category.

4 Tool Setup and Configuration

When running *FuSeBMC_IA*, the user is required to set the architecture with `-a`, the property file path with `-p`, and the benchmark path, as:

```
fusebmc.py [-a {32, 64}] [-p PROPERTY_FILE]
   [-s {kinduction,falsi,incr,fixed}] [BENCHMARK_PATH]
```

For Test-Comp 2023, *FuSeBMC_IA* uses `incr` for incremental BMC, which relies on the ESBMC's symbolic execution engine [11]. The `fusebmc.py` and `FuSeBMC.xml` files are the Benchexec tool info module and the benchmark definition file respectively.

5 Software Project

FuSeBMC_IA is publicly available on GitHub[1] under the terms of MIT License. In the repository, *FuSeBMC_IA* is implemented using a combination of Python and C++. Build instructions and dependencies are all available in `README.md` file. *FuSeBMC_IA* is a fork of the main project *FuSeBMC* available on GitHub[2].

[1] https://github.com/Mohannad-Aldughaim/FuSeBMC_IA

[2] https://github.com/kaled-alshmrany/FuSeBMC

6 Data-Availability Statement

All files necessary to run the tool are available on Zenodo [13].

Acknowledgment

King Saud University, Saudi Arabia[1] supports the *FuSeBMC_IA* development. The work in this paper is also partially funded by the UKRI/IAA project entitled "Using Artificial Intelligence/Machine Learning to assess source code in Escrow".

References

1. Beyer, D. Advances in Automatic Software Testing: Test-Comp 2022. *FASE*. pp. 321-335 (2022) DOI:https://doi.org/10.1007/978-3-030-99429-7_18
2. The Clang Team, Clang documentation. (2022), https://clang.llvm.org/docs/UsersManual.html, accessed: 19-12-2022
3. Jaulin, L., Kieffer, M., Didrit, O. & Walter, E. Applied Interval Analysis. *Springer London*. pp. 11-100 (2001) DOI:https://doi.org/10.1007/978-1-4471-0249-6_2
4. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J. & Yakobowski, B. Frama-C. *International Conference On Software Engineering And Formal Methods*. pp. 233-247 (2012) DOI:https://doi.org/10.1007/978-3-642-33826-7_16
5. Mustafa, M., Stancu, A., Delanoue, N. & Codres, E. Guaranteed SLAM—An interval approach. *Robotics And Autonomous Systems*. **100** pp. 160-170 (2018) DOI:https://doi.org/10.1016/j.robot.2017.11.009
6. Chabert, G. ibex-lib.org. , http://www.ibex-lib.org/, accessed: 19-12-2022
7. Bühler, D. EVA, an evolved value analysis for Frama-C: structuring an abstract interpreter through value and state abstractions. (Rennes 1,2017) DOI:https://doi.org/10.1007/978-3-319-52234-0_7
8. Baudin, P., Bobot, F., Bühler, D., Correnson, L., Kirchner, F., Kosmatov, N., Maroneze, A., Perrelle, V., Prevosto, V., Signoles, J. & Others The dogged pursuit of bug-free C programs: the Frama-C software analysis platform. *Communications Of The ACM*. **64**, 56-68 (2021) DOI:https://doi-org.manchester.idm.oclc.org/10.1145/3470569
9. Alshmrany, K., Aldughaim, M., Bhayat, A. & Cordeiro, L. FuSeBMC: An energy-efficient test generator for finding security vulnerabilities in C programs. *International Conference On Tests And Proofs*. pp. 85-105 (2021) DOI: https://doi.org/10.1007/978-3-030-79379-1_6
10. Alshmrany, K., Aldughaim, M., Bhayat, A. & Cordeiro, L. FuSeBMC v4: Smart Seed Generation for Hybrid Fuzzing. *International Conference On Fundamental Approaches To Software Engineering*. pp. 336-340 (2022) DOI: https://doi.org/10.1007/978-3-030-99429-7_19
11. Gadelha, M., Monteiro, F., Morse, J., Cordeiro, L., Fischer, B. & Nicole, D. ESBMC 5.0: An Industrial-Strength C Model Checker. *ASE*. pp. 888-891 (2018) DOI: https://doi-org.manchester.idm.oclc.org/10.1145/3238147.3240481
12. Aldughaim, M., Alshmrany, K., Menezes, R., Stancu, A. & Cordeiro, L. Incremental Symbolic Bounded Model Checking of Software Using Interval Methods via Contractors.
13. Aldughaim, M., Alshmrany, K., Gadelha, M., Freitas, R. & Cordeiro, L. FuSeBMC v.5: Interval Analysis and Methods for Test Case Generation. DOI:https://doi.org/10.5281/zenodo.7473124(Zenodo,2022,12)

[1] https://ksu.edu.sa/en/