The `counterSharp` Model Counting Benchmark

Samuel Teuber and Alexander Weigl
samuel@samweb.org, weigl@kit.edu
Karlsruhe Institute of Technology

Abstract

We present the `counterSharp` benchmark consisting of 123 projected model counting instances. The instances originate from work on the reliability quantification of programs written in C. We briefly introduce the application field and describe the benchmark selection process.

1 Introduction

Proving software safety is a hard and tedious task which is not always possible in practice. If safety is out of reach, it becomes necessary to estimate the risk of using the software or, framed differently, to estimate its reliability. While safety is typically seen as a qualitative property, i.e., a system can either be safe or not, reliability is a quantitative property (e.g. the likelihood of failure).

This paper describes a benchmark for model counting, which consists out of a set of propositional formulas derived from C programs. The benchmark originates from our work [8]. We introduce the pipeline which created the instances in Section 2. The structure of the benchmark is described in Section 3. In addition to the propositional formulas given in conjunctive normal form (DIMACS format), the benchmark also contains previously measured counts and run times for the counters Ganak [3] and ApproxMC [1, 4] using a 5min timeout (details on the experimental setup can be found in [8, Section 4] as well as in the GitHub repository[1] or the evaluated artefact [7]).

The assembled benchmark is archived by Zenodo [9].

DOI 10.5281/zenodo.5984174

2 Background: Quantifying Software Reliability

In previous work we show that projected model counting can be used to quantify the reliability of C programs [8]. The presented pipeline takes C programs and a specification as input. The specification is given in form of assumption and assertion statements in the source code. The assumptions restrict the investigated span of input values. An assertion statement describes the allowed set of states when the execution reached this assertion. The fully automated pipeline contains three steps:

[1] https://github.com/samysweb/counterSharp-experiments

1. The given specification (assumptions and assertions) is transformed into global variables, in such a way that we can determine the violation (or adherence) of assumptions or assertions within the program state [8, Section 3.1]. This transformation results in an augmented C program which respects the original control and data flow.

2. Then, the augmented program is transformed into multiple propositional logic formulas using the software bounded model-checker CBMC [2].

3. Finally, the propositional logic formulas are passed to a model counter. By projecting onto the propositional variables representing the program's input, we are counting the number of input values. Further, we classify the input values, e.g., (a) violates the assumptions, (b) only violating the assertion, or (c) adhering all assumptions and assertions. Note that every model of the formulas represents a program execution in the original program modulo the introduced propositions of the Tseitin-encoding. At the end, we estimate the program's reliability by counting the input values of the program.

This analysis can be performed on deterministic or non-deterministic programs [8, Sect. 2] as well as using exact or approximate model counters [8, Sect. 3.3].

3 Structure of the Benchmark

Origin of the C Programs. For the evaluation of the approach, we performed the analysis outlined above on a number of interesting representative benchmark instances from the C-Snippets repository [6] as well from the SV Competition repository [5]. The original C programs were modified to allow for a quantitative analysis.

Generation of the DIMACS Files. The quantification pipeline produces up to five projected model counting instances per C program, encoded as a CNF in the DIMACS format. Each of the five instances originates from the same program, but counts a different category of input values. In detail, we count hits and misses of assumptions as well as assertions, and the number of inputs which need a deeper unroll than the one provided by CBMC. Note, we already decide the counted input category early (before the CNF formula is generated by CBMC) by activating a certain assert statement in the program. This allows us to exploit the pre-processing inside of CBMC. Therefore, the CNFs may originate from the same C program, but can differ in their structure. The benchmark mainly consists out of 123 CNF instances originating from 39 programs. The filename of the instances contains the name of the original program and also the input category: assertion miss (`asm`), assertion hit (`ash`), assumption hit (`amh`) and assumption miss (`amm`). The variables to project on are given in the `c ind` comment in each DIMACS file.

Other files. Additionally to the DIMACS file, we provide a CSV file `count.txt` which contains further information on the instances: known model counts and the measured run-time of Ganak and ApproxMC. The file `checksums.txt` gives the SHA256 checksum for each DIMACS file.

References

[1] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. "Algorithmic Improvements in Approximate Counting for Probabilistic Inference: From Linear to Logarithmic SAT Calls". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. Ed. by Subbarao Kambhampati. IJCAI/AAAI Press, 2016, pp. 3569–3576. URL: http://www.ijcai.org/Abstract/16/503.

[2] Edmund Clarke, Daniel Kroening, and Flavio Lerda. "A tool for checking ANSI-C programs". In: *Lecture Notes in Computer Science* 2988 (2004), pp. 168–176. ISSN: 03029743. DOI: 10.1007/978-3-540-24730-2_15.

[3] Shubham Sharma et al. "GANAK: A Scalable Probabilistic Exact Model Counter". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. 2019, pp. 1169–1176. DOI: 10.24963/ijcai.2019/163.

[4] Mate Soos and Kuldeep S. Meel. "BIRD: Engineering an Efficient CNF-XOR SAT Solver and Its Applications to Approximate Model Counting". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 1592–1599. DOI: 10.1609/aaai.v33i01.33011592. URL: https://doi.org/10.1609/aaai.v33i01.33011592.

[5] SoSy-Lab LMU. *SV-Benchmarks*. URL: https://github.com/sosy-lab/sv-benchmarks (visited on 07/14/2020).

[6] Bob Stout. *C Snippets*. 2009. URL: http://web.archive.org/web/20101204075132/http://c.snippets.org/ (visited on 07/08/2020).

[7] Samuel Teuber and Alexander Weigl. *Evaluated Artifact for "Quantifying Software Reliability via Model-Counting"*. 2021. DOI: 10.5445/IR/1000134169.

[8] Samuel Teuber and Alexander Weigl. "Quantifying Software Reliability via Model-Counting". In: *Quantitative Evaluation of Systems - 18th International Conference, QEST 2021, Paris, France, August 23-27, 2021, Proceedings*. Ed. by Alessandro Abate and Andrea Marin. Vol. 12846. Lecture Notes in Computer Science. Springer, 2021, pp. 59–79. DOI: 10.1007/978-3-030-85172-9_4.

[9] Samuel Teuber and Alexander Weigl. "The counterSharp Model Counting Benchmark". In: (Feb. 2022). DOI: 10.5281/zenodo.5984174.