Theta: portfolio of CEGAR-based analyses with dynamic algorithm selection (Competition Contribution)

Zsófia Ádám[1], Levente Bajczi[1], Mihály Dobos-Kovács[1], Ákos Hajdu[2], and Vince Molnár[1] ⋆(✉)

[1] Department of Measurement and Information Systems
Budapest University of Technology and Economics, Budapest, Hungary
molnarv@mit.bme.hu
[2] Meta Platforms Inc., London, United Kingdom

Abstract. Theta is a model checking framework based on abstraction refinement algorithms. In SV-COMP 2022, we introduce: 1) reasoning at the source-level via a direct translation from C programs; 2) support for concurrent programs with interleaving semantics; 3) mitigation for non-progressing refinement loops; 4) support for SMT-LIB-compliant solvers. We combine all of the aforementioned techniques into a portfolio with dynamic algorithm selection.

1 Verification Approach and Software Architecture

Theta [10] is a generic and configurable model checking framework written in Java 11. A simplified version of the architecture (focusing on software verification aspects) can be seen in Figure 1.

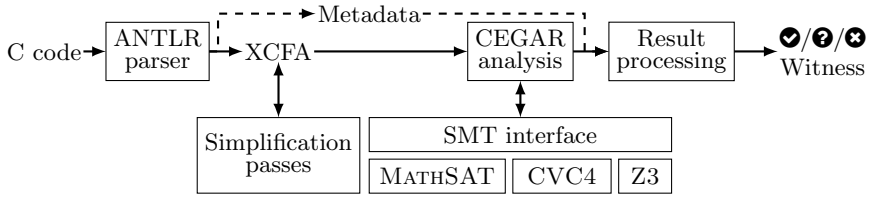

Fig. 1. Architecture of Theta.

The input is a C program that is first translated to extended control-flow automata (XCFA). Previously, Theta used LLVM [3], which had various advantages, but its static single assignment (SSA) form proved overall disadvantageous for abstraction-based algorithms. This year we use a new, direct translation (no

⋆ Jury member representing Theta at SV-COMP 2022.

intermediate language and SSA form) via an ANTLR parser. Furthermore, the CFA being "extended" refers to the fact that since this year we support concurrent programs by an analysis with interleaving semantics. After parsing we apply various passes to the XCFA (e.g., large-block encoding or partial order reduction). The core of Theta is a CEGAR-based analysis framework, targeting *reachability properties* via predicate and explicit analyses [8], along with interpolation and Newton-based refinements [7]. This year, Theta added generic support for SMT solvers (including interpolation) via the SMT-LIB interface. At SV-COMP'22 we use CVC4 [4], MathSAT [6], and Z3 [9], where the latter is used via the Java API from before. Finally, a verdict (safe, unsafe, unknown) and a witness is produced corresponding to the C program (using metadata from the translation).

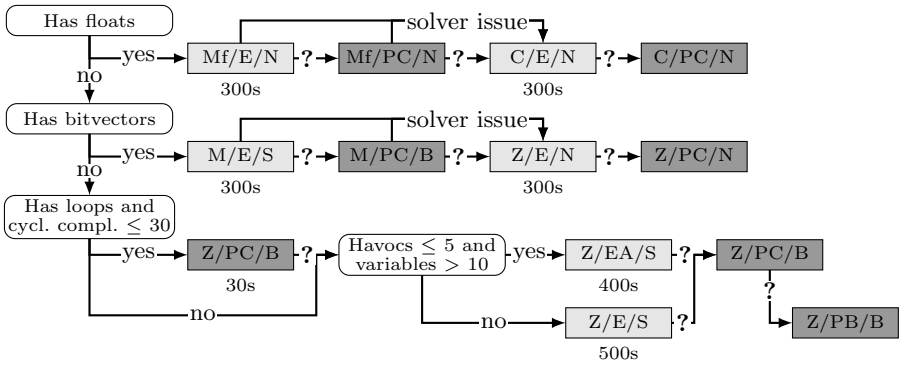

Fig. 2. Overview of the dynamic portfolio of Theta.

Verification portfolio. Based on preliminary experiments and domain knowledge, we manually constructed a dynamic algorithm selection portfolio [1] for SV-COMP'22, illustrated by Figure 2. Rounded white boxes correspond to decision points. We start by branching on the arithmetic (floats, bitvectors, integers). Under integers, there are further decision points based on the cyclomatic complexity and the number of havocs and variables. Grey boxes represent configurations, defining the *solver*/*domain*/*refinement* in this order. Lighter and darker grey represents explicit and predicate domains respectively. Internal timeouts are written below the boxes. An unspecified timeout means that the configuration can use all the remaining time. The solver can be CVC4 (C) [4], MathSAT (M), MathSAT with floats (Mf) [6] or Z3 (Z) [9]. Abstract domains are explicit values (E), explicit values with all variables tracked (EA), Cartesian predicate abstraction (PC) or Boolean predicate abstraction (PB) [8]. Finally, refinement can be Newton with weakest preconditions (N) [7], sequence interpolation (S) or backward binary interpolation (B) [8]. Arrows marked with a question mark (?) indicate an inconclusive result, that can happen due to timeouts or unknown re-

sults. Furthermore, this year's portfolio also includes a novel dynamic (run-time) check for refinement progress between iterations that can shut down potential infinite loops (by treating them as unknown result) [1]. Note also that for solver issues (e.g., exceptions from the solver) we have different paths in some cases.

2 Strengths and Weaknesses

THETA currently targets *ReachSafety* and *ConcurrencySafety* with limited support for structs, arrays and pointers, and no support for dynamic memory allocation, mutexes and recursion. Due to this, THETA fails for most tasks in *ProductLines*, *Recursive*, *Heap* and *Arrays*. Out of the 6163 tasks, roughly 2/3 can be translated and there are 888 confirmed correct (541 safe, 347 unsafe), 116 unconfirmed correct, and only 15 incorrect (11 false positive, 4 false negative) results [5]. Note that almost all unsupported cases are detected and reported as an error, and we only have a few incorrect results due to subtle issues.

The main strength of the tool is the combination of algorithm selection (pick algorithm based on input) and portfolios (try multiple algorithms until one succeeds). Out of the 1004 correct results, 315 could not be solved by the first configuration that the portfolio tries: dynamic checks intervened for 181 internal timeouts, 72 solver issues (e.g. wrong models), 19 non-progressing refinements, and 74 other (unknown) faults before the eventual success.

Having a diverse portfolio also paid off. Bitvector and float arithmetic tasks were either solved by explicit analyses (with a mixture of interpolation- and Newton-based refinements) before even trying predicate configurations, or if explicit analyses failed, predicate configurations were unsuccessful too. The integer arithmetic required a more diverse configuration set: Predicate abstraction solved roughly 48% of the tasks (45% Cartesian, 3% Boolean) and explicit analysis solved 52% (33% with empty precision, 19% with all variables tracked).

The SMT-LIB support provided a great improvement: previously we only had Z3, which still dominates the integer cases. However, all of the bitvector tasks were solved by MATHSAT, making Z3 an unused backup. With floats, roughly half of the tasks were solved by MATHSAT, while the other half needed CVC4 as backup. Since floats are reduced to bitvectors, we did not rely on Z3 based on poor performance in our preliminary experiments.

The most successful subcategories are *BitVectors*, *ControlFlow*, *Loops*, *XCSP* (38-45% correct), mostly because they use features of C that our frontend supports well. We plan to mitigate the high number of timeouts in the future with approximations (e.g. mixing integers and bitvectors), and further analyses (e.g., inferring loop invariants). We also have a significant amount of unconfirmed results: we believe this can be improved by generating more compact witnesses.

This year THETA added support for sequential concurrency via a preprocessing step: it yields an encoding where exploring all interleavings preserve inter-thread behaviors. The analyses treat consecutive non-global memory accesses as one atomic block, reducing the exploration of unnecessary total orders. A drawback of using preprocessing for partial order reduction instead of an on-line

algorithm is the superfluous exploration of *certain* total orders, e.g., all interleavings of independent global memory accesses will also be explored. This is because such accesses *might* overlap with non-independent memory accesses at other times, and the preprocessing step is not aware of such details.

Using a wrapper, Theta integrates concurrency seamlessly with the existing framework (abstract domains, refinements), except the error location-based search [8] (used for non-concurrent cases) because the required distance metric is not well defined in concurrent programs. Instead, we opted to use a breadth-first search, which had outperformed depth-first strategies in preliminary tests. We theorize that this is due to bugs being reachable within the first few instructions most of the time, but only via a specific total order. The performance for concurrent programs is still limited though, and we plan to integrate a declarative approach in the future, which could be used for weakly-ordered programs as well.

3 Tool Setup and Configuration

The competition contribution is based on Theta 3.0.0-svcomp22-v1.[3] Additionally, Theta uses CVC4 v1.9, MathSAT v5.6.6 and Z3 v4.5.0. The project's repository contains build instructions, but an archive can be found at the SV-COMP repository[4] and Zenodo [2]. with pre-built binaries for Ubuntu 20.04 (LTS). The toolchain requires packages `openjdk-11-jre-headless`, `libgomp1` and `libmpfr-dev` to be installed. The entry point of the toolchain is the script `theta/theta-start.sh`, which takes the verification task (C program) as its only mandatory input and runs the portfolio. As additional arguments we use `--portfolio COMPLEX --witness-only --loglevel RESULT`. Further arguments are described in the readme included with the binaries.

4 Software Project

Theta is maintained by the Critical Systems Research Group[5] of the Budapest University of Technology and Economics with various contributors. The project is available open-source on GitHub[3] under an Apache 2.0 license.

Data Availability. The version of Theta used in this paper is available at [2].

Acknowledgment and Funding. The authors would like to thank Tamás Tóth, Milán Mondok, István Majzik, Zoltán Micskei and András Vörös for their contributions to the project; and the competition organizers, especially Dirk Beyer for their help during the preparation for SV-COMP. The research contributions of the authors from the Budapest Univ. of Tech. and Econ. were funded by the EC and NKFIH through the Arrowhead Tools project (EU grant No. 826452, NKFIH grant 2019-2.1.3-NEMZ ECSEL-2019-00003), and by the UNKP-21-2 New National Excellence Program of ITM from the NRDI Fund.

[3] https://github.com/ftsrg/theta/releases/tag/svcomp22-v1
[4] https://gitlab.com/sosy-lab/sv-comp/archives-2022/-/blob/main/2022/theta.zip
[5] https://ftsrg.mit.bme.hu

References

1. Ádám, Zs.: Efficient techniques for formal verification of C programs. Bachelor's thesis, Budapest University of Technology and Economics (2021)
2. Ádam, Z., Levente, B., Dobos-Kovács, M., Hajdu, A., Molnár, V.: Theta: portfolio of CEGAR-based analyses with dynamic algorithm selection (competition contribution): Tool archive (data set) (2022). https://doi.org/10.5281/zenodo.5956737
3. Ádám, Zs., Sallai, Gy., Hajdu, Á.: Gazer-Theta: LLVM-based verifier portfolio with BMC/CEGAR (competition contribution). In: TACAS 2021, LNCS, vol. 12652, pp. 435–439. Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_27
4. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: CAV 2011, LNCS, vol. 6806, pp. 171–177. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_14
5. Beyer, D.: Progress on software verification: SV-COMP 2022. In: Proc. TACAS. Springer (2022)
6. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The MathSAT5 SMT solver. In: TACAS 2013, LNCS, vol. 7795, pp. 93–107. Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_7
7. Dobos-Kovács, M., Hajdu, Á., Vörös, A.: Bitvector support in the Theta formal verification framework. In: Proceedings of the 2nd Workshop on Validation and Verification of Future Cyber-Physical Systems (2021), in press.
8. Hajdu, Á., Micskei, Z.: Efficient strategies for CEGAR-based model checking. Journal of Automated Reasoning **64**(6), 1051–1091 (2020). https://doi.org/10.1007/s10817-019-09535-x
9. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS 2008, LNCS, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24
10. Tóth, T., Hajdu, Á., Vörös, A., Micskei, Z., Majzik, I.: Theta: a framework for abstraction refinement-based model checking. In: FMCAD 2017. pp. 176–179 (2017). https://doi.org/10.23919/FMCAD.2017.8102257

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.