Theta: Abstraction Based Techniques for Verifying Concurrency (Competition Contribution)

Levente Bajczi [*(✉)], Csanád Telbisz, Márk Somorjai, Zsófia Ádám, Mihály Dobos-Kovács, Dániel Szekeres, Milán Mondok, and Vince Molnár

Department of Measurement and Information Systems
Budapest University of Technology and Economics, Budapest, Hungary
bajczi@mit.bme.hu

Abstract. Theta is a model checking framework, with a strong emphasis on effectively handling concurrency in software using abstraction refinement algorithms. In SV-COMP 2024, we use 1) an abstraction-aware partial order reduction; 2) a dynamic statement reduction technique; and 3) enhanced support for call stacks to handle recursive programs. We integrate these techniques in an improved architecture with inherent support for portfolio-based verification using dynamic algorithm selection, with a diverse selection of supported SMT solvers as well. In this paper we detail the advances of Theta regarding concurrent and recursive software support.

Funding. This research was partially funded by the ÚNKP-23-{2,3}-I New National Excellence Program; Project no. 2019-1.3.1-KK-2019-00004 (implemented with the support provided from the NRDI Fund of Hungary under the 2019-1.3.1-KK funding scheme); and the Doctoral Excellence Fellowship Programme (funded by the NRDI Fund of Hungary and the BME University).

1 Verification Approach

Theta [15,8] first competed at SV-COMP as a standalone tool in 2022, with initial support for some multi-threaded tasks using a crude version of a partial order reduction (POR) algorithm [2], and no practical support for recursion.

This year, we implemented a novel *abstraction-based partial order reduction* algorithm [13] that enables Theta to solve significantly more tasks compared to previous SV-COMPs, especially in the ReachSafety category. Our algorithm considers two program statements independent even if they use the same shared variable when the current abstraction has no information about this variable. For example, the statements $y = x$ and $x = 1$ are classically considered dependent

[*] Jury member representing Theta at SV-COMP 2024.

due to x. However, if the current abstraction has no information about x (e.g., we only track the predicates $y > 0$ and $z = y$), we consider these statements independent as they are commutative in the abstract state space. We extend a static source-set based POR algorithm [1] with our abstraction-based technique.

A novel statement reduction algorithm has also been developed for the verification of concurrent programs [14]. Our algorithm is similar to program slicing and cone-of-influence techniques in the sense that it detects and removes statements that do not affect the verified property [5,9]. However, our approach analyzes the current local states of concurrent threads and data-flow between threads to dynamically detect irrelevant statements that do not affect the verified property in the current thread interleaving. The evaluation of such statements is skipped which considerably reduces the time cost of successor state calculation during state space exploration. Our technique is especially useful for concurrent tasks where the reducing capability of existing slicing and cone-of-influence techniques is limited due to the many possible interleavings of threads: our algorithm can skip (sub-)statements in certain contexts even if these statements cannot be removed generally (that is, statements that may be important in other thread interleavings). Our algorithm is different from dynamic program slicing [9] since those techniques do not consider the current interleaving of threads for slicing.

Theta has been extended with enhanced interprocedural analysis [12]. Previously, all procedures have been inlined at all of their calls before verification. Procedure support was implemented last year, which handles procedures dynamically during verification, using a stack to keep track of calling locations. This year, procedure support is further improved by applying abstraction to location stacks. If an abstract state overapproximates another with the bottom of their stacks abstracted away, then all abstract paths going out from the covered state are present at the covering state until the current procedure returns. Therefore, the top location of the covered state is popped and exploration continues from the outer procedure, eliding unnecessary exploration [12].

The main advantage of handling procedures dynamically is that it allows Theta to verify recursive programs, which was not possible with inlining. Applying abstraction to stacks also enables the verification of some infinitely recursive programs. Additionally, it reduces the size of the abstract state-space and improves Theta's verification performance with predicate abstraction.

2 Software Architecture

Since last year, we opted to keep our initial portfolio-based approach [2], but used a separate process for each configuration, which can easily be killed using signals, as opposed to the thread-based approach of Theta at SV-COMP'22. Furthermore, we created a generic interface that allows easy co-development of portfolios without having to recompile Theta. The architecture of Theta can be seen in Figure 1: Theta parses and transforms the input program into an eXtended CFA, then, based on the configuration in the portfolio, spawns one or more worker Theta processes that perform the verification. The portfolio en-

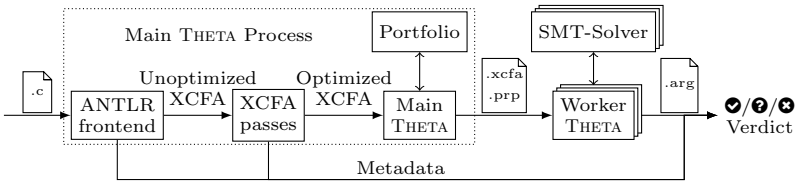

Fig. 1: The architecture of THETA for software verification

gine has been re-written this year to better support pre-compiled configurations written in kotlin instead of kotlin scripts, due to uncovering the dire performance implications of using the script execution engine, which often takes multiple tens of seconds to initialize and start. Dynamic algorithm selection is used to select a suitable configuration for each input task, with several ways of recovering, should the first algorithm take too long or encounter an exception.

THETA uses Z3 [10] versions 4.12.2 and 4.5.0 (the latter is integrated natively via the Java API, while the former is used via SMT-LIB), MathSAT [7] version 5.6.10, CVC5 [4] version 1.0.8 and Princess [11] version 2023-06-19 as SMT solvers under the hood. Compared to previous years, THETA utilizes the new interpolation API of Z3 to support interpolation-dependent refinement strategies with the new solver (removed previously in 4.8.0).

THETA has seen several major updates in its C-frontend for the new tasks introduced to the benchmark repository since SV-COMP'23. The most notable improvements were made around its ANTLR-based grammar for lexing and parsing C files, and some further tweaks in the transformation step from the AST to CFA to avoid some wrong verdicts that plagued THETA in earlier SV-COMPs.

3 Strengths and Weaknesses of the Approach

In ReachSafety, THETA achieved a score of 2119 [6]. Although THETA still has known limitations regarding some C elements (e.g., structs), recent technical improvements of the frontend resulted in THETA not giving any wrong results in any categories, except for 3 wrong results in ConcurrencySafety-NoOverflows. Furthermore, THETA achieved a score of 2354 in ConcurrencySafety. To show the negative influence of frontend limitations, we recalculated the score for the participating tools on those ConcurrencySafety tasks that did not end in a frontend failure for THETA. In this alternative scoring THETA would move from the 7th to the 3rd place, highlighting the serious need for further frontend development.

It is worth looking at THETA's performance in the reachability category over the years. As seen in Figure 2, THETA has dipped in performance for last year's installment of SV-COMP (the figure shows only those tasks that have been the same for the last 3 years) from that of SV-COMP'22 [2]. This year we managed to bring the performance back to even outperform THETA'22, especially in the ConcurrencySafety, Sequentialized and Combinations subcategory. However, we did lose a significant number of tasks in some other subcategories, such as Loops.

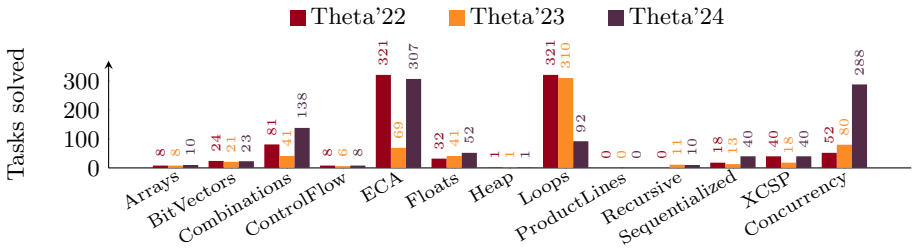

Fig. 2: Overview of successful tasks for Theta per year on common tasks

This can either be a result of a suboptimal portfolio for such tasks, or the result of some tweaks we had to make in order to achieve this year's outstanding 0 incorrect tasks, a feat perfomed only by 3 other tools. We plan to prioritize the analysis of these cases for future development. We also plan to support categories such as ProductLines and Heap, where we have almost no successful results. This entails supporting structs, function pointers, and heap manipulation.

The novel algorithms implemented in Theta especially helped recursive and multithreaded programs. Theta gained support for recursive programs by implementing the aforementioned stack-based approach, and support for reachability queries in multithreaded programs grew more than 3.5-fold since last year, as seen in Figure 2. In particular, our internal evaluation shows that the size of the state space reduced by the abstraction-based partial order reduction algorithm is 15% smaller on average compared to the case when we use traditional partial order reduction. Our dynamic statement reduction technique can eliminate 22% of statements reducing the time of successor state calculation by up to 60% and the overall verification time by 15% on average depending on the configuration.

4 Tool Setup and Configuration

Theta is vastly configurable [8], and successfully choosing a performant configuration for a verification task at hand can be complicated. For software verification, we recommend using the portfolio (`complex`) in the competition archive [3]: `./theta-start.sh <input> --portfolio COMPLEX`. To minimize the output verbosity and produce a witness, `--loglevel RESULT` and `--witness-only` can be added to the arguments. We also used these options at SV-COMP 2024.

5 Software Project and Data Availability

Theta is a verification framework maintained by the Critical Systems Research Group of the Budapest University of Technology and Economics. The project is available open-source on GitHub[1] under an Apache 2.0 license. The version (`5.0.0`) used in the competition is available at [3].

[1] https://github.com/ftsrg/theta/releases/tag/svcomp24

References

1. Abdulla, P.A., Aronis, S., Jonsson, B., Sagonas, K.: Comparing source sets and persistent sets for partial order reduction. Lecture Notes in Computer Science, vol. 10460, pp. 516–536. Springer (2017). https://doi.org/10.1007/978-3-319-63121-9_26
2. Ádám, Z., Bajczi, L., Dobos-Kovács, M., Hajdu, Á., Molnár, V.: Theta: portfolio of CEGAR-based analyses with dynamic algorithm selection (Competition Contribution). In: Fisman, D., Rosu, G. (eds.) TACAS 2021. Lecture Notes in Computer Science, vol. 13244, pp. 474–478. Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_34
3. Bajczi, L., Telbisz, C., Somorjai, M., Ádám, Z., Dobos-Kovács, M., Szekeres, D., Molnár, V.: Theta - SV-COMP'24 Verifier Archive (Nov 2023). https://doi.org/10.5281/zenodo.10202679
4. Barbosa, H., et al.: cvc5: A Versatile and Industrial-Strength SMT Solver. In: Fisman, D., Rosu, G. (eds.) TACAS 2022. pp. 415–442. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_24
5. Berezin, S., Campos, S.V.A., Clarke, E.M.: Compositional Reasoning in Model Checking. Lecture Notes in Computer Science, vol. 1536, pp. 81–102. Springer (1997). https://doi.org/10.1007/3-540-49213-5_4
6. Beyer, D.: State of the art in software verification and witness validation: SV-COMP 2024. In: Proc. TACAS. LNCS , Springer (2024)
7. 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
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. Harman, M., Hierons, R.M.: An overview of program slicing. Softw. Focus **2**(3), 85–92 (2001). https://doi.org/10.1002/swf.41
10. 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
11. Rümmer, P.: A constraint sequent calculus for first-order logic with linear integer arithmetic. LNCS, vol. 5330, pp. 274–289. Springer (2008). https://doi.org/10.1007/978-3-540-89439-1_20
12. Somorjai, M.: Abstraction-Based Interprocedural Software Verification. Students' scientific association (tdk) submission, Budapest University of Technology and Economics (2023), https://tdk.bme.hu/VIK/DownloadPaper/Absztrakcioalapu-interproceduralis
13. Telbisz, C.: Partial Order Reduction for Abstraction-Based Verification of Concurrent Software in the Theta Framework. Bachelor's thesis, Budapest University of Technology and Economics (2022), https://tdk.bme.hu/VIK/DownloadPaper/Reszleges-rendezes-redukcio-tobbszalu
14. Telbisz, C.: Abstract Data-Flow-Based Statement Reduction for Model Checking Concurrent Software. Students' scientific association (tdk) submission, Budapest University of Technology and Economics (2023), https://tdk.bme.hu/VIK/DownloadPaper/Absztrakt-adatfolyamalapu-utasitasredukcio
15. 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