Theta: Various Approaches for Concurrent Program Verification (Competition Contribution)

Csanád Telbisz, Levente Bajczi(✉), Dániel Szekeres, and András Vörös

Department of Artificial Intelligence and Systems Engineering
Budapest University of Technology and Economics, Budapest, Hungary
`csanadtelbisz@edu.bme.hu, {bajczi,szekeres,vori}@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 2025, we complement our existing approach (abstraction-aware partial order reduction) for multi-threaded programs with a happens before propagator-based BMC check, expecting a significant increase in performance. We again utilize our portfolio with dynamic algorithm selection from last year, with improvements regarding solver choice and configuration ordering. In this paper, we detail our algorithmic improvements in Theta regarding the verification of concurrent software.

1 Verification Approach

Theta [17,11] has been a participant in SV-COMP as a standalone tool since 2022. Earlier versions of Theta exclusively applied abstraction-refinement-based model checking algorithms [4,1] mainly focusing on multi-threaded tasks. The focus remained on concurrency; however, different verification approaches have also been implemented in Theta.

The main contribution for this year's SV-COMP version of Theta is a symbolic *bounded model checking* algorithm for the verification of concurrent programs. Compared to abstraction-based analyses, this BMC approach has the advantage of being faster, while it has the disadvantage of providing a bounded correctness proof in many cases. Since Theta is not branded as a bounded model checker, only complete safe results are accepted. If the BMC algorithm produces an incomplete safe result, the tool falls back on an abstraction-based analysis (see more details on our algorithm selection portfolio in Section 2).

The applied BMC algorithm is based on reasoning about the happens-before relation of concurrent program instructions. Our algorithm builds on the concepts of several partial order-based verification algorithms [2,15,18]. The program and the property are symbolically encoded into a Satisfiability Modulo Theories (SMT) formula along with some scheduling constraints based on possible

L. Bajczi—Jury member representing Theta at SV-COMP 2025.

A. Gurfinkel and M. Heule (Eds.): TACAS 2025, LNCS 15698, pp. 260–265, 2025.
https://doi.org/10.1007/978-3-031-90660-2_22

happens-before relations. The (partial) models provided by the SMT solver are used to analyze possible partial orders of concurrent instructions and to prevent scheduling inconsistencies that may arise. An inconsistency corresponds to a cycle in the happens-before relation, since a valid execution of concurrent threads must have a linearization of instructions. A conflict clause is generated and given to the SMT solver when such a happens-before cycle is found to exclude the invalid program execution. The cycle detection and conflict generation algorithm is integrated into Z3 as a custom user propagator [9] via JavaSMT [3].

The first stage of the algorithm is based on the happens-before propagator algorithm of *Sun et al.* [15]. However, their algorithm is insufficient for verification under the sequential consistency memory model [6]: the axioms of [15] can be directly mapped into the rules that define the *weak sequential consistency* memory model [18] that are shown to be insufficient for sequential consistency. Therefore, we extend our algorithm with a second stage when the first stage finds a candidate program execution (i.e., a valid program execution under weak sequential consistency) that violates the safety property. To check whether the happens-before relation representing the program execution is also valid under sequential consistency as well, we explicitly encode the total store order (order of *write* instructions) to guarantee that every write instruction is ordered (a requirement of sequential consistency [18]).

We also implemented a novel optimization for the happens-before propagator [16]. The search space of the SMT solver is reduced by extending the encoding formula. We achieve this by analyzing the program structure and possible partial orders of program instructions, and collect possible scheduling inconsistencies (cycles that may arise in the happens-before relation during verification) before starting the verification decision procedure. Our algorithm searches for potential happens-before cycles of bounded size. Conflict clauses formulated from these potential cycles are appended to the encoding formula. This enhancement greatly reduces the solver search space and improves the verification performance.

The mainline verification method of Theta for proving (unbounded) safety is still a configurable *Counterexample-Guided Abstraction Refinement* (CEGAR) algorithm. For the verification of multi-threaded programs, our approach uses an abstraction-based partial order reduction algorithm and a cone-of-influence reduction technique tailored specifically to concurrent programs. For recursive tasks, an interprocedural analysis is applied with call stack abstraction that can handle infinitely recursive programs. However, this year, only minor improvements have been implemented for our abstraction-based analyses. Detailed descriptions of these analyses in Theta have been presented previously in [4,11,17].

2 Software Architecture

We use portfolio-based algorithm selection as in previous years of SV-COMP [4]. Each configuration is executed in a separate process. A generic interface allows the easy development of complex portfolios defined by finite-state machines. Dynamic algorithm selection is used to select a performant configuration for each input task, with several ways of recovering when a selected algorithm takes too long or encounters an exception. For reachability properties in concurrent tasks, the introduced BMC algorithm is applied first to discover unsafe tasks and prove safety for loop-free programs (or programs where complete loop unrolling can be performed). For programs where complete loop unrolling is not possible (because the number of loop iterations cannot be determined), each loop is unrolled twice for the BMC algorithm. However, in this case, only unsafe results are accepted: we switch to an abstraction-based analysis otherwise. Therefore, Theta always provides complete proofs of safety. The architecture of Theta has not changed considerably since last year's SV-COMP: Theta parses and transforms the input program into a CFA (supporting multi-threading), then, based on the configuration in the portfolio, spawns one or more worker Theta processes that perform the verification. We refer the interested reader to our previous tool paper for a more detailed description of the architecture of Theta [4].

Theta is implemented in Java and Kotlin, and uses Z3 [13] versions 4.12.2, 4.13.0 and 4.5.0 (the latter two versions are integrated natively via the Java API, while the former one is used via SMT-LIB), MathSAT5 [10] version 5.6.10, CVC5 [7] version 1.0.8 and Princess [14] version 2023-06-19 as SMT solvers under the hood. There were major C-frontend updates in Theta, this year, mainly concerning memory handling language elements, and new C language features introduced in benchmarks since last year's SV-COMP.

3 Strengths and Weaknesses of the Approach

The main scope of development for Theta has been reachability properties and data-race freedom. In these categories, Theta only gives 2 wrong verdicts, which is a notable precision among other verifiers at SV-COMP (actually, the wrong verdicts are the results of the hurried development to adapt to last-minute changes in the language features used in the SV-COMP benchmarks, e.g., the use of atomic types). Theta produces several wrong verdicts for other properties (such as memory safety or termination), however, these properties are only experimental in Theta. We plan to properly support these verification properties in future versions.

In Figure 1, we include a comparison of the results with the performance of Theta in SV-COMP, last year. The figure includes both confirmed and unconfirmed correct results (since result validation and the possibility of an unconfirmed verdict status have only been introduced this year for data race tasks). The figure shows the categories affected by recent development: reachability tasks with memory operations or multi-threading, and data-race detection tasks. The figure highlights the performance increase achieved by the new version of Theta.

We also performed an internal evaluation of our algorithms separately (not in a portfolio). The BMC method is able to provide a (bounded) verdict for 520 tasks for the reachability property in the concurrency category out of the 544 programs whose language features are supported by this analysis. Without the optimization described in Section 1, Theta could solve only 514 tasks. The

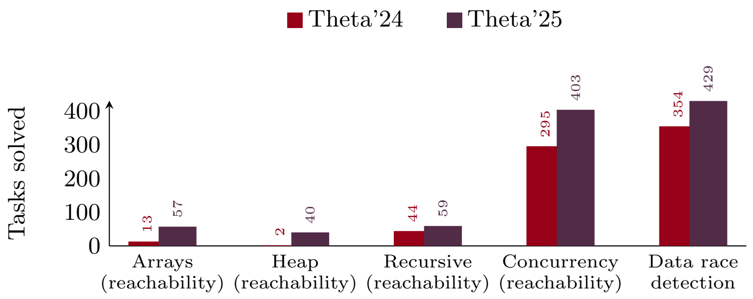

Fig. 1: Comparison of successful tasks for THETA on common tasks

optimization achieves a more significant improvement in CPU time: the verification time is reduced by more than 30% on average. While many of these results are only obtained by applying a bounded loop unrolling (and thus these verdicts are not reported by THETA as final results), other BMC tools among SV-COMP competitors apply a similar strategy. Therefore, we executed `Deagle` (a BMC tool with a bounded loop unrolling strategy [12], winner of SV-COMP concurrency category [8]) on our infrastructure for a fair comparison. `Deagle` achieved the same result: 520 solved tasks on the same set of programs. While THETA is disadvantaged on the full benchmark set due to its frontend limitations, this comparison on a major portion of tasks clearly shows the potential of our algorithm. It also underlines the need for frontend improvements in THETA to support even more language features.

Our internal experiments also reveal that our CEGAR analysis is capable of verifying 365 programs for reachability in the concurrency category. Unfortunately, we slightly misconfigured the algorithm-selection portfolio for concurrent tasks for the competition. Therefore, the best-performing CEGAR configuration using predicate abstraction was not selected for concurrent programs. We calculated that around 2% more tasks could be solved by a proper configuration.

4 Tool Setup and Configuration

THETA is highly configurable [11], and choosing a suitable configuration for a verification task can be challenging. For software verification, we recommend using our complex portfolio in the competition archive [5]: `./theta-start.sh <input> --svcomp --portfolio STABLE`. To minimize the output verbosity, `--loglevel RESULT` can be added. We used these options at SV-COMP 2025.

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 (`6.8.6`) used in the competition is available at [5]. Theta participated in the reachability, memory-safety, concurrency, overflow detection, and termination categories of SV-COMP 2025.

Funding. This research was partially funded by the EKÖP-24-{2,3} New National Excellence Program under project numbers EKÖP-24-2-BME-118, EKÖP-24-3-BME-213 and EKÖP-24-3-BME-159, and the Doctoral Excellence Fellowship Programme under project numbers 400434/2023 and 400443/2023; funded by the NRDI Fund of Hungary.

References

1. 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 2022. Lecture Notes in Computer Science, vol. 13244, pp. 474–478. Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_34
2. Alglave, J., Kroening, D., Tautschnig, M.: Partial Orders for Efficient Bounded Model Checking of Concurrent Software. In: Sharygina, N., Veith, H. (eds.) CAV 2013. Lecture Notes in Computer Science, vol. 8044, pp. 141–157. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_9
3. Baier, D., Beyer, D., Friedberger, K.: Javasmt 3: Interacting with SMT solvers in java. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. Lecture Notes in Computer Science, vol. 12760, pp. 195–208. Springer (2021). https://doi.org/10.1007/978-3-030-81688-9_9
4. Bajczi, L., Telbisz, C., Somorjai, M., Ádám, Z., Dobos-Kovács, M., Szekeres, D., Mondok, M., Molnár, V.: Theta: Abstraction Based Techniques for Verifying Concurrency (Competition Contribution). In: TACAS 2024. Lecture Notes in Computer Science, vol. 14572, pp. 412–417. Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_30, https://doi.org/10.1007/978-3-031-57256-2_30
5. Bajczi, L., Telbisz, C., Somorjai, M., Ádám, Z., Dobos-Kovács, M., Szekeres, D., Molnár, V.: Theta - SV-COMP'25 Verifier Archive (Nov 2024). https://doi.org/10.5281/zenodo.14194483
6. Bajczi, L., Telbisz, C., Szekeres, D., Vörös, A.: On Stability in a Happens-Before Propagator for Concurrent Programs (Reproducibility Study). In: TACAS 2025. LNCS , Springer (2025), https://ftsrg.mit.bme.hu/paper-tacas25-ocfix/paper.pdf
7. 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
8. Beyer, D., Strejček, J.: Improvements in software verification and witness validation: SV-COMP 2025. In: Proc. TACAS. LNCS, Springer (2025)
9. Bjørner, N.S., Eisenhofer, C., Kovács, L.: Satisfiability Modulo Custom Theories in Z3. In: Dragoi, C., Emmi, M., Wang, J. (eds.) VMCAI 2023. Lecture Notes in Computer Science, vol. 13881, pp. 91–105. Springer (2023). https://doi.org/10.1007/978-3-031-24950-1_5

[1] https://github.com/ftsrg/theta/releases/tag/svcomp25

10. 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
11. 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
12. He, F., Sun, Z., Fan, H.: Deagle: An SMT-based Verifier for Multi-threaded Programs (Competition Contribution). In: Fisman, D., Rosu, G. (eds.) TACAS 2022. Lecture Notes in Computer Science, vol. 13244, pp. 424–428. Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_25
13. 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
14. 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
15. Sun, Z., Fan, H., He, F.: Consistency-preserving propagation for SMT solving of concurrent program verification. Proc. ACM Program. Lang. **6**(OOPSLA2), 929–956 (2022). https://doi.org/10.1145/3563321
16. Telbisz, C.: Efficient automatic verification of concurrent programs. Master's thesis, Budapest University of Technology and Economics (2024), https://theta.mit.bme.hu/publications/telbiszcsMsc2024.pdf
17. 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
18. Zennou, R., Atig, M.F., Biswas, R., Bouajjani, A., Enea, C., Erradi, M.: Boosting Sequential Consistency Checking Using Saturation. In: Hung, D.V., Sokolsky, O. (eds.) ATVA 2020. Lecture Notes in Computer Science, vol. 12302, pp. 360–376. Springer (2020). https://doi.org/10.1007/978-3-030-59152-6_20