EmergenTheta: Verification Beyond Abstraction Refinement (Competition Contribution)

Levente Bajczi [*(✉)], Dániel Szekeres, Milán Mondok, Zsófia Ádám, Márk Somorjai, Csanád Telbisz, Mihály Dobos-Kovács, 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 conventionally based on abstraction refinement techniques. While abstraction is useful for a large number of verification problems, the over-reliance on the technique led to Theta being unable to meaningfully adapt. Identifying this problem in previous years of SV-COMP has led us to create EmergenTheta, a sandbox for the new approaches we want Theta to support. By differentiating between mature and emerging techniques, we can experiment more freely without hurting the reliability of the overall framework. In this paper we detail the development route to EmergenTheta, and its first debut on SV-COMP'24 in the ReachSafety category.

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 Software Architecture

Theta is a modular and configurable verification framework in the sense that multiple frontend subprojects are served by a vastly configurable, CEGAR-based backend ([10,6]). Frontends include *Petri-nets*, *AIGER* models, *timed automata*, and *C programs* among others (hence the modularity), and the CEGAR backend provides fine-grained access to its internal settings such as refinement and search strategy, abstract domains, and solver selection (hence the configurability). It is, however, not conventionally capable of using non-CEGAR based analyses. This behavior is engrained in the implementation in multiple ways, such as counterexamples and safety proofs requiring a partial or full abstract reachability

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

graph, and the interface of the backend containing references to *precision* [6]. Our main contribution as part of SV-COMP'24 is the removal of such dependencies on abstraction-specific classes. This enables the rapid prototyping and development of diverse verification algorithms such as this year's BMC, IMC, and k-induction algorithms [5,7,9], building the low-level core of Theta including the representation and manipulation of expressions and interfacing with several SMT-solvers.

To facilitate the implementation of these algorithms, we introduced a new `MonolithicTransitionFunction` interface to Theta, which returns a single nondeterministic action representing the whole transition system (i.e., it represents the structural information as additional variables and related guards). This is a counterpart to the previously existing TransitionFunction interface, which directly relies on the structural information for the enabledness of actions. This interface has been implemented for most of the formalisms supported by Theta.

Besides the changes detailed above, EmergenTheta still relies on Theta's Antlr-based C frontend and integrated support for SMT-solvers, as well as its existing counterexample-to-witness projection [1].

2 Verification Approach

In *bounded model checking* (BMC) [5], the transition system and the safety property are encoded as SMT [3] formulas. In each iteration of the algorithm, a path constraint is created from the formulas characterizing all execution paths of a given length k that start in an initial state and end in an error state. The satisfiability of the path constraint is checked using an SMT solver [8]. If a satisfying assignment is found, it is returned as a counterexample, else the bound k is increased until the available resources allow.

BMC is incomplete as it can only prove the absence of counterexamples up to a finite depth. *K-induction* [9] and *interpolation-based model checking* (IMC) [7] address this by adding checks that attempt to prove that the property holds for unbounded depth based on the unsatisfiability of the BMC query. K-induction does so by trying to prove the k-inductivity of the property with k being the current BMC length, while IMC derives Craig interpolants to compute an overapproximation of the set of reachable states.

Based on preliminary testing, we used a simple sequential portfolio (without algorithm selection) that executed an IMC-only verification phase first (for at most 90 seconds), then fell back to a combined BMC and k-induction-based verification phase for the rest of the time limit. EmergenTheta did not employ any of the CEGAR-based analysis methods already present it Theta, as we wanted to evaluate the newly implemented ones separately.

Tool Category	All		False		True	
	EmergenTheta	Theta	EmergenTheta	Theta	EmergenTheta	Theta
Arrays	13 (7)	13 (7)	0 (0)	5 (5)	13 (7)	8 (2)
BitVectors	16 (1)	23 (8)	7 (0)	10 (3)	9 (1)	13 (5)
Combinations	1 (0)	138 (137)	1 (0)	121 (120)	0 (0)	17 (17)
ControlFlow	7 (4)	9 (6)	1 (0)	2 (1)	6 (4)	7 (5)
ECA	1 (1)	307 (307)	0 (0)	133 (133)	1 (1)	174 (174)
Floats	25 (6)	54 (35)	2 (0)	23 (21)	23 (6)	31 (14)
Hardness	378 (269)	116 (7)	0 (0)	0 (0)	378 (269)	116 (7)
Hardware	134 (15)	194 (75)	60 (10)	89 (39)	74 (5)	105 (36)
Heap	2 (0)	2 (0)	0 (0)	0 (0)	2 (0)	2 (0)
Loops	232 (117)	161 (46)	34 (11)	40 (17)	198 (106)	121 (29)
Sequentialized	1 (0)	47 (46)	1 (0)	34 (33)	0 (0)	13 (13)
XCSP	2 (0)	45 (43)	2 (0)	44 (42)	0 (0)	1 (1)
Overall	812 (420)	1153 (761)	108 (21)	530 (443)	704 (399)	623 (318)

Table 1: Comparison of Theta and EmergenTheta for each subcategory

3 Discussion of Strengths and Weaknesses of the Approach

As our secondary goal (besides adapting Theta's architecture to a more flexible one) was to find out how the new algorithms implemented in EmergenTheta performed, we mainly compare and contrast the results of EmergenTheta (which used only the newly implemented algorithms) and Theta (which used only CEGAR). In the future, we aim to integrate the new algorithms into our mainline Theta tool, for which this evaluation is invaluable.

Table 1 compares the number of tasks correctly solved by Theta and EmergenTheta for each subcategory in ReachSafety (using official results) [4], distinguishing between true and false outputs. The numbers in parentheses show the number of correctly solved tasks that the other tool was unable to solve in time.

Looking at the overall results, we can see that Theta and EmergenTheta are suitable for different tasks: although Theta solved more tasks, EmergenTheta solved 420 tasks *that* Theta *could not solve*, which is 36% of the 1153 tasks solved by Theta. With an ideal portfolio, incorporating these algorithms could significantly increase the number of tasks solved by Theta.

Theta was much better at finding counterexamples (108 vs 530 false outputs), while EmergenTheta was slightly better at proving correctness (704 vs 623 true outputs). This goes against our intuition, as abstraction refinement is more tailored to proving correctness. This phenomenon warrants further investigation; our current hypothesis is that performing enough refinements to eliminate all spurious counterexamples had too large an overhead. More than

half of the true results for each tool were for tasks that the other one could not solve, highlighting their complementary nature.

EmergenTheta was significantly better in the Loops and the Hardness categories, while it was worse in Combinations, ECA, Sequentialized and XCSP. As for Combinations and Sequentialized, this could be attributed to Theta being generally better at finding counterexamples, as false tasks are overrepresented in these categories; but for ECA and XCSP, tasks of both types are represented nearly equally.

These relatively positive results were achieved in spite of a misconfiguration: although our preliminary measurements had shown that CVC5 and Mathsat performed best with K-IND and IMC respectively, we accidentally enrolled EmergenTheta with its default solver Z3. We consider this a failure in the design of the portfolio engine of Theta, which allowed us to submit a faulty configuration without this being evident in the logs (that no runs were using solvers other than Z3). We will prioritize improving on this aspect of Theta for next year.

4 Tool Setup and Configuration

EmergenTheta remains vastly configurable, and successfully choosing a performant configuration for a verification task at hand can be complicated. If using the competition archive [2] for software verification, we recommend using the preassembled portfolio: `theta-start.sh <input> --backend IMC_THEN_KIND`. To minimize the output verbosity and produce a witness in the working directory, the flags `--loglevel RESULT` and `--witness-only` can be added to the arguments. We also used these options at SV-COMP 2024.

5 Software Project and Contributors

EmergenTheta is integrated into the Theta verification framework maintained by the Critical Systems Research Group[1] of the Budapest University of Technology and Economics. The project is available open-source on GitHub[2] under an Apache 2.0 license. The version (`5.0.0`) used in the competition is available at [2].

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 2021. Lecture Notes in Computer Science, vol. 13244, pp. 474–478. Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_34

[1] https://ftsrg.mit.bme.hu/en/
[2] https://github.com/ftsrg/theta/releases/tag/svcomp24

2. Bajczi, L., Szekeres, D., Mondok, M., Molnár, V.: EmergenTheta - SV-COMP'24 Verifier Archive (Nov 2023). https://doi.org/10.5281/zenodo.10198872
3. Barrett, C., Tinelli, C.: Satisfiability Modulo Theories. https://doi.org/10.1007/978-3-319-10575-8_11
4. Beyer, D.: State of the art in software verification and witness validation: SV-COMP 2024. In: Proc. TACAS. LNCS , Springer (2024)
5. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic Model Checking without BDDs. In: TACAS (1999). https://doi.org/10.1007/3-540-49059-0_14
6. 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
7. McMillan, K.L.: Interpolation and SAT-Based Model Checking. In: Hunt, W.A., Somenzi, F. (eds.) Computer Aided Verification (2003). https://doi.org/10.1007/978-3-540-45069-6_1
8. 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
9. Sheeran, M., Singh, S., Stålmarck, G.: Checking Safety Properties Using Induction and a SAT-Solver. In: Formal Methods in Computer-Aided Design (2000). https://doi.org/10.1007/3-540-40922-X_8
10. Tóth', T.: Abstraction Refinement-Based Verification of Timed Automata. Ph.D. thesis, Budapest University of Technology and Economics (2021)