A Preliminary Analysis on the Effect of Randomness in a CEGAR Framework

Ákos Hajdu[1,2], Zoltán Micskei[1]
[1]Budapest University of Technology and Economics, Department of Measurement and Information Systems
[2]MTA-BME Lendület Cyber-Physical Systems Research Group
Email: {hajdua, micskeiz}@mit.bme.hu

Abstract—**Formal verification techniques can check the correctness of systems in a mathematically precise way. Counterexample-Guided Abstraction Refinement (CEGAR) is an automatic algorithm that reduces the complexity of systems by constructing and refining abstractions. CEGAR is a generic approach, having many variants and strategies developed over the years. However, as the variants become more and more advanced, one may not be sure whether the performance of a strategy can be attributed to the strategy itself or to other, unintentional factors. In this paper we perform an experiment by evaluating the performance of different strategies while randomizing certain external factors such as the search strategy and variable naming. We show that randomization introduces a great variation in the output metrics, and that in several cases this might even influence whether the algorithm successfully terminates.**

I. Introduction

Formal verification techniques (such as model checking [1]) can check whether a model (formal representation) of a system meets certain requirements by exhaustively analyzing its possible states and transitions. As our reliance on computer systems grows, the importance of these techniques is also increasing. However, a typical drawback of using formal methods is their high computational complexity. The Counterexample-Guided Abstraction Refinement (CEGAR) approach [2] alleviates this problem by automatically constructing and refining abstractions that over-approximate the behavior of systems. CEGAR starts with a coarse initial abstraction (to minimize complexity) and then applies refinements based on candidate counterexamples until a sufficient precision is reached (that is fine enough for deciding whether the requirement holds).

Theta is a generic framework that includes many configurations (variants) of the CEGAR algorithm in a common environment [3]. The framework relies on first order logic (FOL): the behavior of the models is encoded in graphs annotated with FOL formulas and the algorithms use SAT/SMT solvers [4] as the underlying engine. We already performed experimental evaluations in Theta and in most cases we concluded that the configurations have a diverse performance: different configurations are more suitable for different tasks [5], [6]. However, as the underlying strategies of the configurations are also becoming more advanced, we cannot be certain whether their performance can be attributed to their intentional, algorithmic behavior. Rather, they might be unintentionally influenced by certain external factors in a way that the final outcome is a better performance in certain cases. For example, a different ordering of commutative formulas might unintentionally affect the order in which states are processed.

In this paper we investigate two such factors. A higher level, algorithmic factor is the search strategy in the abstract state space. A configuration may fail to build a suitable abstraction efficiently if a deterministic search strategy guides it in the "wrong" direction. A lower level, external factor is the naming of the variables. Most of the refinement strategies employ an SMT solver for computing over-approximations. We observed that the name of the variables affects their order in certain collections (e.g., sets), which may influence the inner heuristics of the solvers, also affecting the quality of the generated abstractions.

Our experiment shows that randomizing any of the aforementioned factors greatly increases variations in the output metrics (e.g., execution time). Furthermore, randomization often even affects whether the algorithm can successfully terminate within the given time limit. We also examine some cases where a randomized configuration can verify a model for which the deterministic ones fail. Based on this feedback, we can improve the shortcomings of the deterministic configurations and we can also introduce nondeterministic options to the configurations as a viable alternative.

II. Experiment Planning

In our experiment several *configurations* of the CEGAR algorithm of Theta were executed on various input *models* deterministically and also with randomizing the search strategy or the variable names.

A. Research Questions

The current research questions focus on a preliminary, exploratory analysis of the results.

RQ1 Are there any cases where a randomized configuration could verify a model (at least once) that its deterministic counterpart could not?

RQ2 How does randomization affect the variation of output metrics (e.g., execution time) compared to deterministic configurations? Which yields a greater variation? Randomizing search or variable names?

B. Subjects and Objects

Theta includes many parameters for the CEGAR algorithm. For this experiment we selected the two most prominent,

TABLE I
VARIABLES OF THE EXPERIMENT.

Category	Name	Type	Description
Input (model)	Category	Factor	Category of the model. Possible values: eca, hw, locks, plc, ssh (see Section II-B).
	Model	String	Unique name of the model.
Input (config.)	Domain	Factor	Domain of the abstraction. Possible values: PRED (predicate), EXPL (explicit value).
	Refinement	Factor	Refinement strategy. Possible values: BIN (binary interpolation), SEQ (sequence interpolation).
	Randomized	Factor	Factor that is randomized. Possible values: DET (deterministic, no randomization), SEARCH (random search strategy), VARS (random variable names).
Output (metrics)	Succ	Boolean	Indicates whether the algorithm successfully provided a result within the given time limit.
	TimeMs	Integer	Execution time of the algorithm (in milliseconds).
	Iterations	Integer	Number of refinement iterations until the sufficiently precise abstraction was reached.
	ArgSize	Integer	Number of nodes in the Abstract Reachability Graph (ARG), i.e., the number of explored abstract states.
	ArgDepth	Integer	Depth of the ARG.
	CexLen	Integer	Length of the counterexample, i.e., a path leading to a state of the model that does not meet the requirement.

namely the domain of the abstraction and the refinement strategy. We experimented with predicate [7] and explicit value [8] domains with binary [9] and sequence [10] interpolation-based refinements, as these strategies are also implemented in many other verification tools [11]. The third parameter is the randomized factor, i.e., the search strategy, the variable names, or nothing (deterministic). Therefore, there are a total number of $2 \cdot 2 \cdot 3 = 12$ configurations.

Due to the long execution time of the measurements, we only evaluated the configurations on 30 input models. Nevertheless, we tried to make these models relevant and diverse. Therefore, we picked 10 hardware models (hw) from different categories of the Hardware Model Checking Competition [12], 15 models from 3 categories (eca, locks, ssh) of the Competition on Software Verification [11] and 5 industrial PLC software modules (plc) from CERN [13]. Based on previous measurements, we picked models with different difficulties, including easy (verified by most configurations) and difficult instances (verified by a few or no configurations).

C. Variables

Variables of the experiment are listed in Table I, grouped into three main categories: properties of the model (input), parameters the configuration (input) and metrics of the algorithm execution (output). If the algorithm did not provide a result within the time limit, the variable Succ is false and the other output metrics are empty (NA).

D. Measurement Procedure

Measurements were executed on two 64 bit Windows 7 virtual machines with 2 cores (2.50 GHz), 8 GB RAM and JRE 8 (THETA is implemented in Java). Z3 version 4.5.0 [14] was used as an SMT solver. Each measurement was repeated 30 times with a different random seed. The time limit for each execution was 180 seconds.

E. Analysis Methods

RQ1 can be answered by summarizing heatmaps and filtering the data. Furthermore, it would be interesting to analyze each case separately in more detail using for example the logs produced by THETA. RQ2 can be examined with basic descriptive statistics and summarizing plots (e.g., box plots), yielding a good overview on the variations of the output metrics under different configurations.

F. Threats to Validity

We worked with input models from different sources, including well-known benchmarks sets such as HWMCC [12] and SV-COMP [11]. However, due to time constraints, only 30 models were picked. External validity could be improved by selecting more models both from the same sources and from additional ones. This experiment focused only on THETA, which includes many algorithms known from state-of-the-art tools [11]. However, external validity would benefit from repeating the experiment with different tools. It would also be interesting to experiment with a higher time limit (e.g., SV-COMP uses 900s) and more randomized factors (e.g., reorder commutative formulas in the models). Internal validity is increased by repeating the measurements 30 times on dedicated virtual machines. However, if more resources were available, measurements should be repeated even more times (e.g. 1000 times [15]) on dedicated physical machines.

III. ANALYSIS

This section discusses the analyses and results related to our research questions. The analyses were performed with the R software environment [16]. The raw data, the R script and a detailed report can be found on a supplementary web page.[1]

A. Terminology and Overview

A *run* is a single execution of a *configuration* on a *model*. A run is *successful* if a result (whether the model is correct or not) is provided within the given time limit. A *measurement* is the collection of all repeated runs of the same configuration on the same model. A measurement is *successful* if it includes at least one successful run. In this case we also say that the configuration *verified* the model.

[1]http://dx.doi.org/10.5281/zenodo.1117853

In this experiment 12 configurations were executed on 30 models (from 5 categories), yielding $12 \cdot 30 = 360$ measurements. Each measurement was repeated 30 times, giving $360 \cdot 30 = 10800$ runs. There are $7080/10800$ successful runs (66%) and $261/360$ successful measurements (72%). Fig. 1 summarizes the range and distribution of the output metrics.

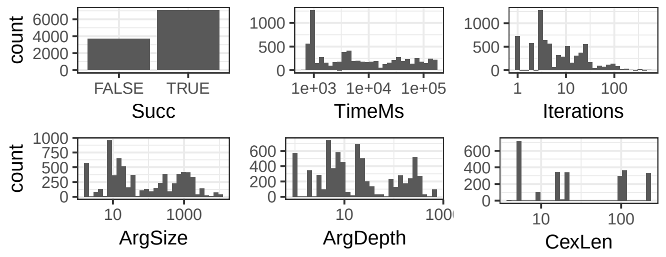

Fig. 1. Distribution of the output metrics.

Fig. 2 shows the number of successful runs for each model. It can be seen that besides the easy models in category locks, their difficulty is gradually increasing, supporting our claim on diversity.

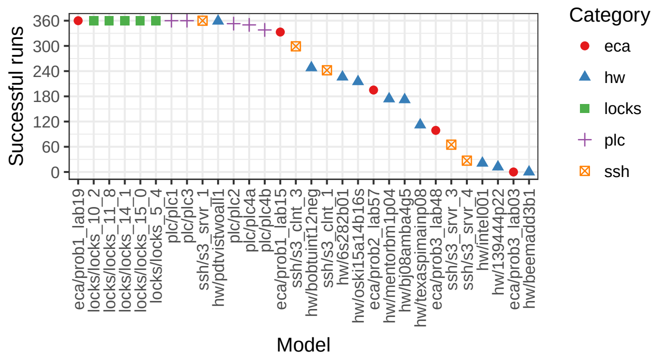

Fig. 2. Number of configurations that verified the models.

Fig. 3 presents the number of verified models and successful runs for each configuration. Configurations are abbreviated with the first letters of their parameters, e.g., PB-V stands for predicate domain, binary interpolation and variable name randomization. The difference between the best and worst configuration is $25 - 19 = 6$ regarding verified models and $685 - 518 = 167$ regarding successful runs.

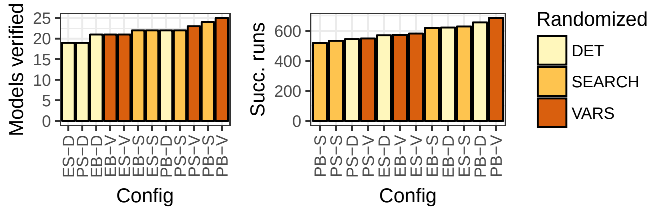

Fig. 3. Number verified models and successful runs for each configuration.

B. RQ1: Successful Verifications

Fig. 4 illustrates the number of successful runs for each measurement. White cells represent no successful runs. It can be seen that in most cases the deterministic configurations have either 0 or 30 successful runs. There are a few exceptions though, where the execution time was close to the time limit.

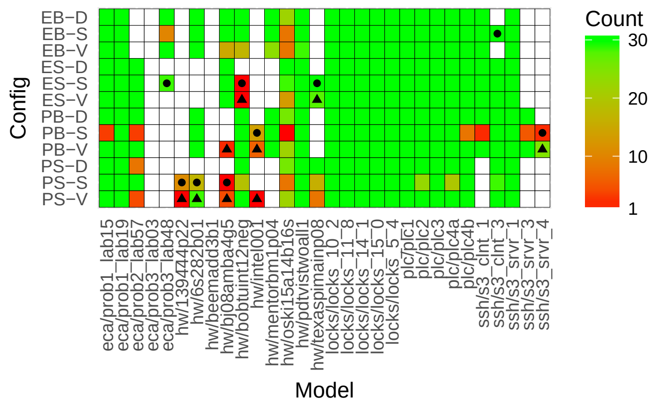

Fig. 4. Number of successful runs for each measurement.

However, there are $9 + 9$ cases where a configuration with randomized search or variable names could verify a model that its deterministic counterpart could not. These cases are marked with dots and triangles respectively in Fig. 4.

Interestingly, there are 3 models (139444p22, intel001, s3_srvr_4) where only randomized configurations were successful. It is a hard task to investigate the behavior of the algorithm in detail for these cases, as these models are very complex (sometimes consisting of thousands of variables and formulas with hundred thousands of terms and operands). Nevertheless, we examined the logs to get some insight on what is happening with and without randomization.

139444p22 is a large model, consisting of 8600 variables and formulas with a total size of 1.6×10^5 (measured in the number of terms and operands). The deterministic configuration runs out of time when checking a candidate counterexample using the SMT solver. The randomized configuration also spends roughly half of its time on checking counterexamples, but in the successful runs, it quickly finds a feasible one, terminating the algorithm. A possible explanation is that the solver can find an easy solution for feasible counterexamples, but fails to prove infeasibility due to the large formulas.

The intel001 model is not large, but the formulas refuting the feasibility of counterexamples can grow unmanageably large. In the successful runs of the randomized configuration, it manages to produce refutation formulas with a maximal size of 4.4×10^4. The deterministic configuration however, generates a refutation formula of size 4×10^6 in the 6th iteration, prohibiting the exploration of the abstract states.

Repeating the measurements for the s3_srvr_4 model (to get logs) revealed that the deterministic configuration can also verify this model, but its execution time is slightly above the limit of 180s. By examining the randomized runs as well, we observed that for this model the success of verification depends on the number of refutation formulas discovered. The deterministic configuration discovers some unnecessary formulas, making the number of abstract states higher. However, the randomized configurations can find a subset of these formulas (in some runs) that is still enough to prove the correctness of the model in less time.

Feedback learned from these cases identified various shortcomings of deterministic configurations and also gave us ideas on how to improve them.

C. RQ2: Variations

For each measurement, the 30 repeated runs form a distribution for each output metric. Variation is usually described by the standard deviation (SD). However, the output metrics have a vastly different range, making SD incomparable between them. Therefore, we calculate the *relative standard deviation* (RSD = SD / mean). Furthermore, for the Boolean variable Succ, we replace false by 0, true by 1 and calculate the SD.

The distribution of the deviations are summarized using box plots in Fig. 5, grouped by the factor that is randomized. There are 5 outlier points between 1.25 and 3.5 that were cropped so that the box plots can be depicted using the same scale.

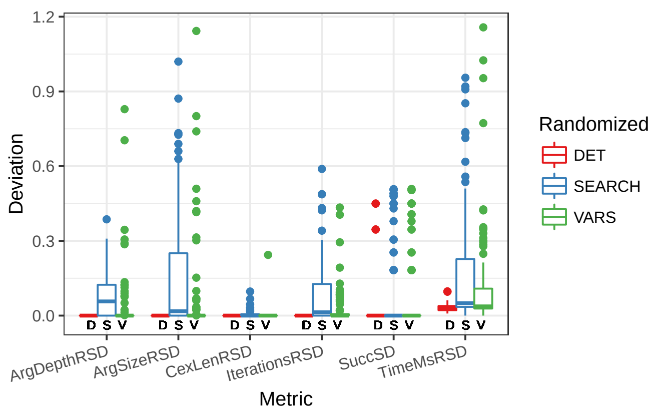

Fig. 5. Distribution of the deviations of the output metrics.

Deterministic configurations also have some small deviations for the execution time and the success indicator. As it was mentioned previously, the latter can be attributed to execution times near the time limit. All other output metrics have 0 deviation, increasing our confidence that these configurations are indeed deterministic.

It can be seen that the randomized configurations have greater deviation for all output metrics. The largest deviations appear for the execution time (TimeMs) and the number of abstract states explored (ArgSize). The length of the counterexample (CexLen) has the lowest deviations. This metric has a smaller sample size, as only 9 out of the 28 verified models were incorrect. Furthermore, counterexamples often correspond to a single concrete execution in the original model, which has a fixed length. It can also be clearly observed that in most cases randomizing the search strategy yields greater deviations than randomizing the variable names.

IV. Conclusions

In our paper we evaluated various configurations of the CEGAR algorithm in the THETA tool under randomized search strategies and variable names. Our experiment highlighted that randomizing these factors introduces a great variation in the output metrics. In several cases this also influences whether a configuration can successfully verify a model. We also examined some cases where a randomized configuration verified a model that none of the deterministic ones could. Feedback from these cases will help us to improve the current shortcomings of the algorithms. Thus, preliminary results are interesting, but to improve their external validity, a more thorough experiment is needed with more models, repetitions and randomized factors as well.

References

[1] E. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT Press, 1999.

[2] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, “Counterexample-guided abstraction refinement for symbolic model checking,” *Journal of the ACM*, vol. 50, no. 5, pp. 752–794, 2003.

[3] T. Tóth, A. Hajdu, A. Vörös, Z. Micskei, and I. Majzik, “Theta: a framework for abstraction refinement-based model checking,” in *Proc. 17th Conf. on Formal Methods in Computer-Aided Design*. FMCAD inc., 2017, pp. 176–179.

[4] A. Biere, M. Heule, and H. van Maaren, *Handbook of Satisfiability*. IOS press, 2009.

[5] A. Hajdu and Z. Micskei, “Exploratory analysis of the performance of a configurable CEGAR framework,” in *Proc. 24th PhD Mini-Symposium*. BME DMIS, 2017, pp. 34–37.

[6] G. Sallai, A. Hajdu, T. Tóth, and Z. Micskei, “Towards evaluating size reduction techniques for software model checking,” in *Proc. 5th Int. Workshop on Verification and Program Transformation*, ser. EPTCS. Open Publishing Association, 2017, vol. 253, pp. 75–91.

[7] S. Graf and H. Saidi, “Construction of abstract state graphs with PVS,” in *Computer Aided Verification*, ser. LNCS. Springer, 1997, vol. 1254, pp. 72–83.

[8] D. Beyer and S. Löwe, “Explicit-state software model checking based on CEGAR and interpolation,” in *Fundamental Approaches to Software Engineering*, ser. LNCS. Springer, 2013, vol. 7793, pp. 146–162.

[9] K. McMillan, “Applications of Craig interpolants in model checking,” in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS. Springer, 2005, vol. 3440, pp. 1–12.

[10] Y. Vizel and O. Grumberg, “Interpolation-sequence based model checking,” in *Formal Methods in Computer-Aided Design*. IEEE, 2009, pp. 1–8.

[11] D. Beyer, “Software verification with validation of results,” in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS. Springer, 2017, vol. 10206, pp. 331–349.

[12] G. Cabodi, C. Loiacono, M. Palena, P. Pasini, D. Patti, S. Quer, D. Vendraminetto, A. Biere, K. Heljanko, and J. Baumgartner, “Hardware model checking competition 2014: An analysis and comparison of solvers and benchmarks,” *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 9, pp. 135–172, 2016.

[13] B. Fernández Adiego, D. Darvas, E. Blanco Viñuela, J.-C. Tournier, S. Bliudze, J. O. Blech, and V. M. González Suárez, “Applying model checking to industrial-sized PLC programs,” *IEEE Trans. on Industrial Informatics*, vol. 11, no. 6, pp. 1400–1410, 2015.

[14] L. de Moura and N. Bjørner, “Z3: An efficient SMT solver,” in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS. Springer, 2008, vol. 4963, pp. 337–340.

[15] A. Arcuri and L. Briand, “A hitchhiker’s guide to statistical tests for assessing randomized algorithms in software engineering,” *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.

[16] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2017. [Online]. Available: https://www.R-project.org/