Exploratory Analysis of the Performance of a Configurable 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. However, their computational complexity often prevents their successful application. The counterexample-guided abstraction refinement (CEGAR) algorithm aims to overcome this problem by automatically building abstractions for the system to reduce its complexity. Previously, we developed a generic CEGAR framework, which incorporates many configurations of the algorithm. In this paper we focus on an exploratory analysis of our framework. We identify parameters of the systems and algorithm configurations, overview some possible analysis methods and present preliminary results. We show that different variants are more efficient for certain tasks and we also describe how the properties of the system and parameters of the algorithm affect the success of verification.**

I. Introduction

As safety critical systems are becoming more and more prevalent, assuring their correct operation is gaining increasing importance. Formal verification techniques (such as model checking [1]) can check whether the model (a formal representation) of a system meets certain requirements by traversing its possible states and transitions. However, a typical drawback of using formal methods is their high computational complexity. Abstraction is a general technique to reduce complexity by hiding irrelevant details. However, finding the proper precision of abstraction is a difficult task. Counterexample-guided abstraction refinement (CEGAR) is an automatic verification algorithm that starts with a coarse initial abstraction and refines it iteratively until a sufficient precision is obtained [2].

In our previous work [3] we examined different variants of the CEGAR algorithm and concluded that each of them has its advantages and shortcomings. The foundations of a modular, configurable CEGAR framework were also developed that can incorporate the different CEGAR configurations (variants) in a common environment. The framework relies on first order logic (FOL): models are described with FOL formulas and the algorithms use SAT/SMT solvers [4] as the underlying engine.

The framework is under development, but it already realizes several configurations and permits the verification of some input models. We performed an experiment by evaluating these configurations on the models of some hardware and PLC systems. In this paper we present an exploratory analysis of the results: we identify parameters and metrics of the models and configurations as input and output variables. We give an overview on possible analysis methods and present preliminary comparisons, revealing that different configurations are more suitable for certain models. We show relationships between the properties of the input model, the parameters of the algorithm (e.g., abstraction method, refinement strategy) and the success and efficiency of verification.

II. Experiment Planning

In our experiment several *configurations* of the CEGAR algorithm were executed on various input *models* and the results were analyzed [5].

A. Variables

Variables of the experiment are grouped into three main categories: parameters of the model (input), parameters of the configuration (input), metrics of the algorithm (output). Variables along with their type and description are listed in Table I. Some other parameters of the input models were also identified, but these are domain specific and not applicable to all inputs (e.g., number of gates in a hardware circuit). Therefore, these parameters were omitted in this experiment.

There are some additional constraints on the variables. UNSC refinement cannot be used in PRED domain, but besides that, all combinations of the algorithm parameters are valid, yielding a total number of 20 configurations. It is also possible that the algorithm did not terminate within a specified time. In this case all output variables are NA (not available) values. Furthermore, the length of the counterexample is NA if the model is safe.

B. Objects

As the framework is under development, its current performance and limited input format only permits the verification of smaller input models from certain domains. Nevertheless, some smaller standard benchmark instances were used from the Hardware Model Checking Competition [11]. These models encode hardware circuits with inputs, outputs, logical gates and latches. Some industrial PLC software modules from a particle accelerator were also verified. A total number of 18 models were used, consisting of 12 hardware and 6 PLCs.

C. Measurement Procedure

The framework is implemented in Java and it was deployed as an executable `jar` file. Measurements were ran on a 64 bit Windows 7 virtual machine with 2 cores (2.50 GHz), 16 GB

TABLE I
VARIABLES OF THE EXPERIMENT.

Category	Name	Type	Description
Input (model)	Type	Factor	Type of the model. Possible values: hw (hardware), plc (PLC, i.e., Programmable Logic Controller).
	Model	String	Unique name of the model.
	Vars	Integer	Number of FOL variables in the model.
	Size	Integer	Total size of the FOL formulas in the model.
Input (config.)	Domain	Factor	Domain of the abstraction. Possible values: PRED (predicate [6]), EXPL (explicit value [7]).
	Refinement	Factor	Abstraction refinement strategy. Possible values: CRAIGI (Craig interpolation [8]), SEQI (sequence interpolation [9]), UNSC (unsat core [10]).
	InitPrec	Factor	Initial precision of the abstraction. Possible values: EMPTY (empty), PROP (property-based).
	Search	Factor	Search strategy in the abstract state space. Possible values: BFS, DFS (breadth- and depth-first search).
Output (metrics)	Safe	Boolean	Result of the algorithm, indicates whether the model meets the requirement according to the algorithm.
	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.

RAM and JRE 8. No other virtual machines were running on the host during the measurements. Z3 [12] was used as the underlying SAT/SMT solver. The measurement procedure was fully automated. The configurations and models were listed in `csv` files and a script was written that loops through each configuration and model pair and runs the framework with the appropriate parameters (based on the configuration and the model). The script waits until the verification finishes or a certain time limit is reached, outputs the result (or timeout) to a `csv` file and repeats the procedure a given number of times.

In our current setting, 20 configurations were executed on 18 input models and each execution was repeated 5 times, yielding a total number of $18 \cdot 20 \cdot 5 = 1800$ measurements. The time limit for each measurement was 8 minutes. With this limit, 1120 executions terminated (successful verifications).

D. Research Questions

In our current work we focus on a preliminary, exploratory analysis of the measurement results. The following research questions are investigated.

- RQ1 What are the overall, high level properties of the data set, e.g., distribution of execution time, percentage of safe models?
- RQ2 How do individual parameters of the configuration affect certain output variables, e.g., PRED or EXPL domain yields faster execution time?
- RQ3 Which input parameters influence certain output variables the most, e.g., is the Domain or the Refinement more influential on the execution time?

E. Analysis Methods

RQ1 can be answered with basic descriptive statistics and summarizing plots (e.g., box plots, heat maps), yielding a good overview on the characteristics of the data. RQ2 can be examined with the aid of interactive, visual tools. Parallel coordinates, scatter plots and correlation diagrams are suitable for this purpose, where relationships between the different dimensions can be quickly revealed. RQ3 can be analyzed with decision trees, principle component analysis and other methods that can extract the most relevant information from a set of data. This analysis can also provide an aid to pick the most appropriate configuration for a given task.

F. Threats to Validity

External validity can be guaranteed by selecting representative input models. As mentioned in Section II-B, smaller hardware and software instances were used. As the performance and the input formats of the framework will increase, it will be possible to verify more, larger instances and other kinds of models (e.g., tasks from the Software Verification Competition [13]), which improves the external validity of the analysis. Other tools were not evaluated, because this experiment focused only on our framework. Internal validity is increased by running the measurements repeatedly on a dedicated machine. Furthermore, the framework has also been undergone unit and integration testing.

III. ANALYSIS

This section presents the analyses and results to our research questions. The results of measurements were collected to a single `csv` file, which was analyzed using the R software environment version 3.3.2 [14].

Let D denote the whole data set, $D_{\text{succ}} \subseteq D$ the successful executions (no timeout) and $D_{\text{cex}} \subseteq D_{\text{succ}}$ the successful executions where the model is not safe (i.e., a counterexample is present). The relation of the data sets along with their size is depicted in Figure 1.

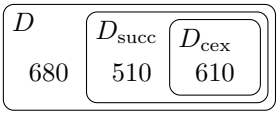

Fig. 1. Overview of the data sets with the number of measurements.

A. RQ1: High Level Overview

First we checked that (1) either all executions of a configuration on a model gives the same safety result and (2) all configurations agree on the results for each model. The lack of the previous properties would obviously mean that the algorithms are not sound, but for our data set they hold.

The histograms and box plots in Figure 2 give a high level overview of the distribution and range of output variables. It can be seen that for most of the variables, the IQR is small and there are many outliers.

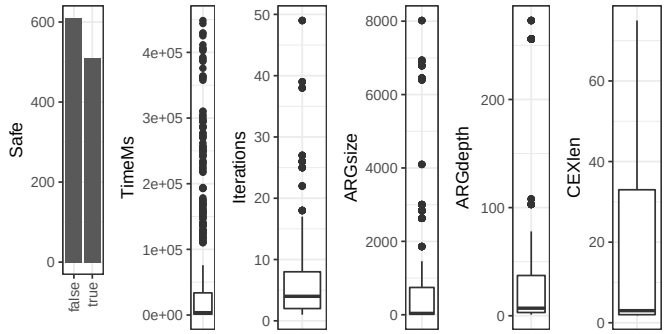

Fig. 2. Overview of individual output variables.

Figure 3 gives an overview on execution time. Each cell of the grid represents the average time of the 5 repeated executions of a configuration on a model. Configurations are abbreviated with the first letters of their parameters, e.g., PSED means PRED domain, SEQI refinement, EMPTY initial precision and DFS search. The maximal relative standard deviation (RSD) of the repeated executions is 10.5%, which is a low value. This is not surprising because our algorithms are deterministic, with the possible exception of some heuristics in external solvers. This low RSD value suggests internal validity and allows us to represent repeated measurements with their average. White cells mean that all executions timed out and colored cells correspond to a logarithmic scale in milliseconds. It can be seen that each model was verified by at least one configuration. It is interesting that plc3 was only verified by a single configuration, but in a rather short time. Also, there is no single configuration that can verify each model, but some of them can verify almost all models. Some of the models (e.g., hw9) are easy for all configurations, but some of them (e.g., plc1) expose 2–3 orders of magnitude difference between the configurations.

B. RQ2: Effect of Individual Parameters

Effect of individual parameters on certain output variables were also compared. The most interesting observation was the effect of the domain on the execution time. This analysis was done by forming pairs from the measurements, similarly to the join operation known from databases. We calculated $D \times D$ and kept rows where every input variable is the same, except the domain, which is different. This means that each row contains an execution for PRED and EXPL domains with the rest of

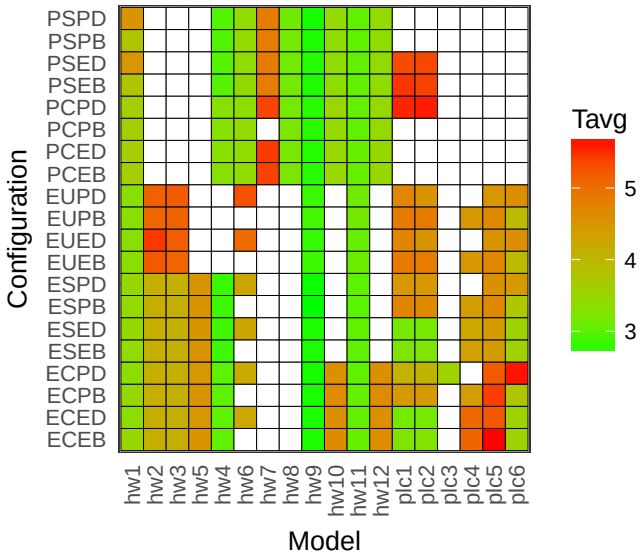

Fig. 3. Average execution time (milliseconds, logarithmic scale).

the configuration (and the model) being the same. Only those rows were kept where at least one domain was successful. Each point in Figure 4 represents a row, where the x and y coordinates correspond to the execution time of PRED and EXPL respectively. Furthermore, points have different colors based on Type. Points at the right and top edges correspond to timeouts. An important property of this kind of plot is that points above the identity line mean that PRED was faster, while points below mean the opposite. It can be observed that verification of PLC models is more efficient in the EXPL domain. Hardware models however, show some diversity, both domains have good results.

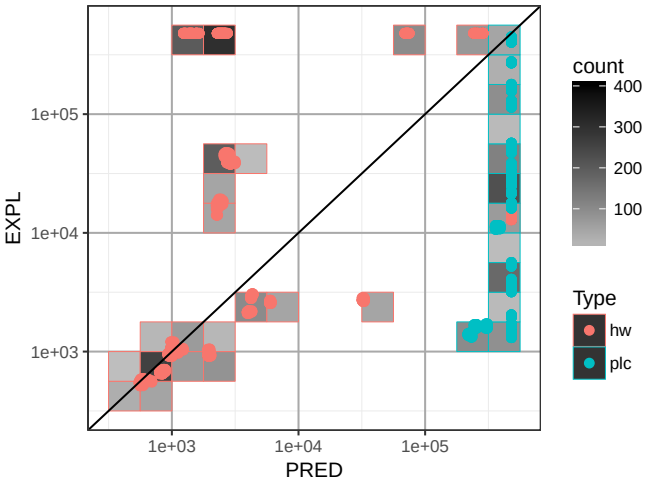

Fig. 4. Comparison of execution time for the different domains.

An other interesting result was the comparison of the number of iterations for CRAIGI and SEQI refinements. Only those rows were kept where both refinements were successful.

It can be seen in Figure 5 that SEQI yields less iterations in almost all cases. It can also be observed, that the difference between the two refinement strategies is small for hardware models, but it can be much larger for certain PLC models.

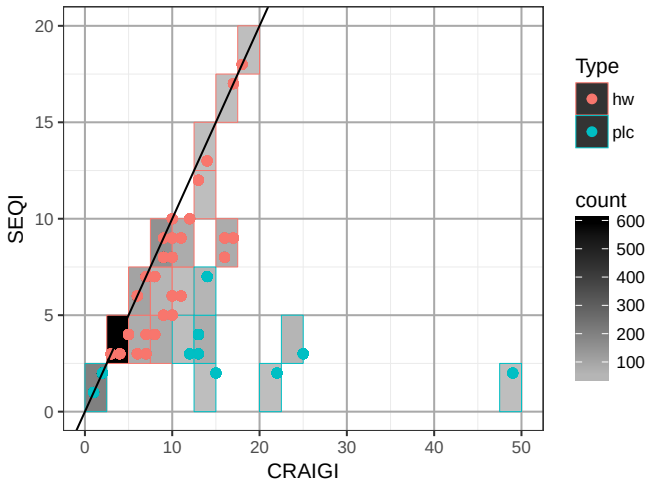

Fig. 5. Comparison of iterations for the different refinements.

C. RQ3: Influence of Input Parameters on Output Variables

The influence of input parameters on certain output variables were also examined. Amongst the observations, the most interesting was the influence of Type and the parameters of the configuration on the success of verification (i.e., a non-timeout). Figure 6 shows the decision tree. It can be seen that the most influential parameters are Domain, Type and Refinement. In the terminal nodes SUCC and TO represent success and timeout respectively. It can be observed that for example choosing PRED domain for PLCs will most likely not succeed. On the other hand, it is likely to succeed for hardware models. It can also be seen that EXPL domain with CRAIGI refinement is likely to succeed regardless of the type of the model. This fact is also confirmed by the small number of white cells in the bottom four rows of Figure 3.

IV. Conclusions

In our paper we evaluated various configurations of our CEGAR framework on different models, including hardware and PLCs. We identified properties of models and parameters of the algorithm that can serve as input and output variables. We presented some possible analysis methods with the corresponding results, including descriptive statistics, different plots and decision trees. Although the results being preliminary, we showed that different configurations are more suitable for certain tasks and we also revealed connections between the type of the model, the parameters of the algorithm and the success of verification. Based on these results we will be able to improve the framework and perform measurements with a larger number of input models and configurations, yielding a larger data set. We hope that further analysis on this data set will allow us to automatically determine the most appropriate configuration of the algorithm for a given verification task.

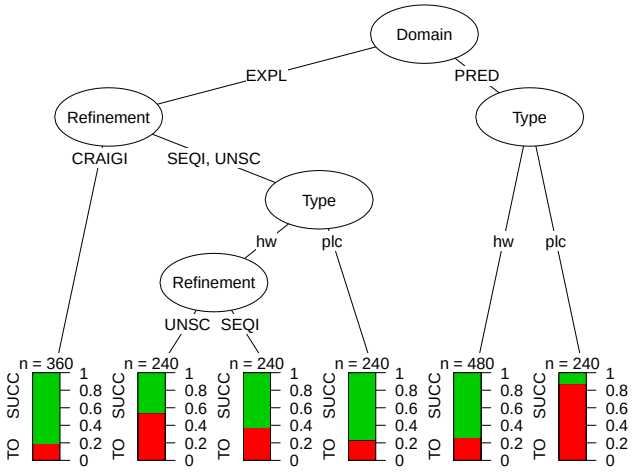

Fig. 6. Decision tree on the success of verification.

References

[1] E. M. 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] A. Hajdu, T. Tóth, A. Vörös, and I. Majzik, "A configurable CEGAR framework with interpolation-based refinements," in *Formal Techniques for Distributed Objects, Components and Systems*, ser. LNCS. Springer, 2016, vol. 9688, pp. 158–174.

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

[5] P. Antal, A. Antos, G. Horváth, G. Hullám, I. Kocsis, P. Marx, A. Millinghoffer, A. Pataricza, and A. Salánki, *Intelligens adatelemzés*. Typotex, 2014.

[6] 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.

[7] E. M. Clarke, A. Gupta, and O. Strichman, "SAT-based counterexample-guided abstraction refinement," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 7, pp. 1113–1123, 2004.

[8] 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.

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

[10] M. Leucker, G. Markin, and M. Neuhäußer, "A new refinement strategy for CEGAR-based industrial model checking," in *Hardware and Software: Verification and Testing*, ser. LNCS, vol. 9434. Springer, 2015, pp. 155–170.

[11] 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.

[12] 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.

[13] D. Beyer, "Reliable and reproducible competition results with BenchExec and witnesses (report on SV-COMP 2016)," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS. Springer, 2016, vol. 9636, pp. 887–904.

[14] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2016. [Online]. Available: https://www.R-project.org/