PeSCo: Predicting Sequential Combinations of Verifiers (Competition Contribution)

Cedric Richter(✉) and Heike Wehrheim

Department of Computer Science, Paderborn University, Paderborn, Germany
cedricr@mail.upb.de, wehrheim@upb.de

Abstract. PeSCo is a tool for predicting a (likely best) sequential combination of verifiers on a given verification task and then running it. The approach is based on machine learning, more precisely on learning *rankings* of verifiers on verification tasks (where the ordering of verifiers is based on the SV-COMP scoring schema). The learning part employs Support Vector Machines; as base verifiers we use CPAchecker in 6 different configurations.

1 Verification Approach

Composing verification techniques in sequence has in the past been a promising approach in the annual software verification competition SV-COMP. Especially in 2018[1], the software verification framework CPAchecker [3], using a composition of analyses, was able to outperform competitors in category `ReachSafety`. However, the analysis sequence is often predefined and fixed. In other words, a problem instance might pass through a sequence of unsuccessful verification configurations until it is processed by the right technique or exceeds a time limit.

Our competition contribution utilizes the sequential setting of CPAchecker (more precisely, of CPA-Seq), but *predicts* the order of verification tools viz. configurations. For this, we applied an extension of our rank prediction approach introduced in [7]. Basically, for a given verification task we predict an ordering of CPAchecker configurations, and then sequentially run these configurations. Configurations are ordered with respect to their (likely) performance on the verification task.

The prediction employs machine learning. For the learning, we extract *features* of verification tasks via an encoding of programs as graphs combining concepts of control-flow and program dependence graphs with abstract syntax trees. Features represent certain graph substructures of programs, where the depth of substructures considered is configurable.

[1] sv-comp.sosy-lab.org/2018/results/results-verified/.

C. Richter—Jury member.

© The Author(s) 2019
D. Beyer et al. (Eds.): TACAS 2019, Part III, LNCS 11429, pp. 229–233, 2019.
https://doi.org/10.1007/978-3-030-17502-3_19

To obtain the execution order for a new problem instance, the *Ranking by pairwise comparison* (RPC) [9] framework is employed utilizing *kernelized Support Vector Machines* (SVM) [11] as base learners. By employing SVMs, we are able to choose a kernel function[2] (similar to Weisfeiler-Lehman kernels [12]) that is specifically designed for graph substructures. However, the function proposed in [7] needed to be computed between the input instance X (the graph of a verification task) and *every* training sample Y, which can be quite costly in practice. As a consequence, we have re-implemented this approach and now compute Weisfeiler-Lehman-based features of *single* graphs. This significantly improves the performance of prediction.

2 Software Architecture

Our tool contribution PeSCo embeds a `Planning` step in the `restart` algorithm employed in the verification framework CPAchecker [3]. The `restart` algorithm [10] is used in a sequential combination of verifiers to let the next verifier start on already computed (partial) results of previous verifiers, in particular when the previous verifier could not solve the verification problem. However, instead of executing a fixed list of verification techniques, our algorithm plans an execution order dependent on the verification task to be solved. Our approach consists of the following steps.

Training. To train our rank predictor, we employ rankings obtained by executing 5 CPAchecker configurations on the verification tasks of SV-COMP 2018. Similar to CPA-Seq [10] from 2018, we use *Value Analysis* [5], *Value Analysis + CEGAR* [5], *Predicate Analysis* [4], *k-Induction* [1] and *Bounded Model Checking* [6]. In addition, we introduced and carried out training with a special `UNKNOWN` configuration. This extension will allow our prediction procedure to cut off an analysis when it will most probably fail.

Planning. As can be seen in Fig. 1, we utilize the preprocessor and control flow automaton (CFA) construction implemented in CPAchecker. Instead of passing the CFA directly to an analysis, we first query our rank prediction process. The prediction process starts by building an intermediate graph representation. This is followed by a feature extraction and the final ranking procedure (details in [7]). If a prediction is not achievable in a certain time frame, we fall back to the standard CPA-Seq.

Execution. After planning a sequential composition, we can apply the analyses in the given order. If an analysis fails or exceeds its time limit, we switch to the next configuration. On reaching the `UNKNOWN` configuration, we exit the verification sequence. Instead of leaving the overall process, specialized techniques will be applied in the following situations: For recursive programs we facilitate *Block Abstraction Memoization* (BAM) [8,13] and *Binary Decision Diagrams* (BDD) [2] are used for concurrent programs. Witnesses are written as generated by the verifiers.

[2] Kernels are similar to a similarity functions between feature vectors.

Despite the fact that our implementation is only dependent on Java 8, we need MathSAT 5[3] to run individual configurations. Furthermore, parser frontend for C programs are used according to CPAchecker.

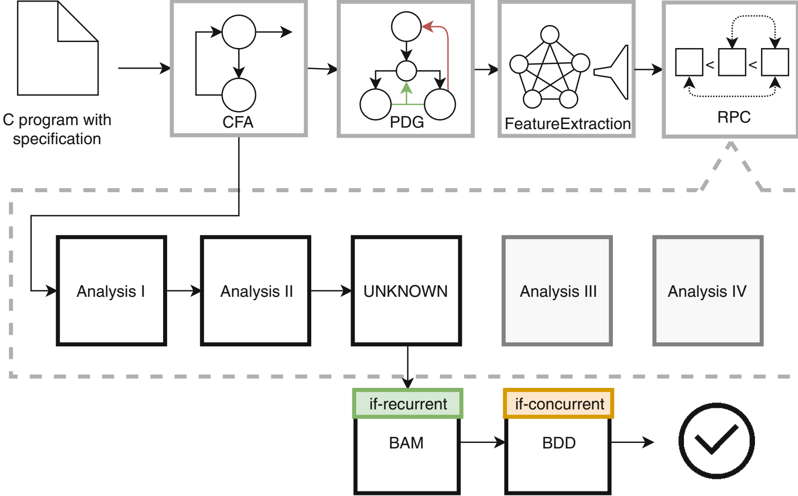

Fig. 1. Architecture of PeSCo implemented within CPAchecker. The dotted box represent the Restart Algorithm enhanced by our rank prediction. Hence, Analysis II receives partial results of Analysis I for a restart of the verification. The rank prediction utilizes the control flow automaton extended by data and control dependencies (PDG).

3 Strengths and Weaknesses

In contrast to traditional compositional approaches, PeSCo adapts to the given tasks. As a result, our tool is able to decrease the runtime by skipping techniques that do not fit to the given verification task. More importantly, the adaptation allows us to omit analyses which introduce failures. Consequently, PeSCo improves the number of correct results in a given time frame.

Nevertheless, learning the optimal ranking requires time and introduces uncertainty to the verification process. Experiments on 1148 tasks in *ReachSafety-ECA* show that optimal rankings on a large number of *similar* programs with *different* requirements are difficult to predict. Still, the results of SV-COMP 2019 show that PeSCo can effectively verify a number of C programs in that category.

Due to the prediction process, PeSCo is furthermore limited to the configurations that occur during training. Since we trained our predictor with the version of CPAchecker employed in SV-COMP 2018, we perform slightly worse than the improved 2019 version of CPA-Seq.

[3] mathsat.fbk.eu.

4 Tool Setup and Configuration

PeSCo is fully integrated in the official source code of CPAchecker. Thus, it can be downloaded as a fork: https://github.com/cedricrupb/cpachecker. We use Revision `b8d6131` for the competition. To compile the tool, `ant` should be executed on the checkout folder. After this step, our tool requires Java 8 and MathSAT 5 as external tools. To verify a test program, CPAchecker is executed with the following command line:

```
$ scripts/cpa.sh -svcomp19-pesco -benchmark -heap 10000M -stack
   ↪ 2048k -timelimit 900s -spec prop.spc program.c
```

For programs expecting a 64 Bit model, add the parameter `-64`. PeSCo participates in category `ReachSafety`, `Falsification` and `Overall`. The corresponding specification can be found in the checkout folder under `config/specification/sv-comp-reachability.spc`.

5 Software Project and Contributors

Being an extension of the CPAchecker project, PeSCo is developed as an open-source project by a research group from Paderborn University. Contributors were so far Mike Czech, Marie-Christine Jakobs, Cedric Richter and Heike Wehrheim. We would furthermore like to thank Eyke Hüllermeier for machine learning expertise and his contribution to the prediction process. We aso thank the CPAchecker team for allowing us to use their tool.

References

1. Beyer, D., Dangl, M., Wendler, P.: Boosting k-induction with continuously-refined invariants. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 622–640. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_42
2. Beyer, D., Friedberger, K.: A light-weight approach for verifying multi-threaded programs with CPAchecker. In: Electronic Proceedings in Theoretical Computer Science, no. 233, pp. 61–71 (2016)
3. Beyer, D., Keremoglu, M.E.: CPAchecker: a tool for configurable software verification. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 184–190. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_16
4. Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Proceedings of the 2010 Conference on Formal Methods in Computer-Aided Design, pp. 189–198. FMCAD Inc. (2010)
5. Beyer, D., Löwe, S.: Explicit-state software model checking based on CEGAR and interpolation. In: Cortellessa, V., Varró, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 146–162. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37057-1_11
6. Clarke, E., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 168–176. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24730-2_15

7. Czech, M., Hüllermeier, E., Jakobs, M., Wehrheim, H.: Predicting rankings of software verification tools. In: Baysal, O., Menzies, T. (eds.) Proceedings of the 3rd ACM SIGSOFT International Workshop on Software Analytics, SWAN@ESEC/SIGSOFT FSE 2017, pp. 23–26. ACM (2017). https://doi.org/10.1145/3121257.3121262
8. Dangl, M., Löwe, S., Wendler, P.: CPAchecker with support for recursive programs and floating-point arithmetic. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 423–425. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_34
9. Fürnkranz, J., Hüllermeier, E.: Preference learning and ranking by pairwise comparison. In: Preference Learning, pp. 65–82 (2010). https://doi.org/10.1007/978-3-642-14125-6-4
10. Löwe, S., Mandrykin, M., Wendler, P.: CPAchecker with sequential combination of explicit-value analyses and predicate analyses. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 392–394. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_27
11. Schölkopf, B., Smola, A.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2001)
12. Weisfeiler, B., Lehman, A.: A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno Technicheskaya Informatsia **2**(9), 12–19 (1968)
13. Wonisch, D., Wehrheim, H.: Predicate analysis with block-abstraction memoization. In: Aoki, T., Taguchi, K. (eds.) ICFEM 2012. LNCS, vol. 7635, pp. 332–347. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34281-3_24

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.