CPA-BAM-Slicing: Block-Abstraction Memoization and Slicing with Region-Based Dependency Analysis
(Competition Contribution)

Pavel Andrianov, Vadim Mutilin, Mikhail Mandrykin(✉), and Anton Vasilyev

Ivannikov Institute for System Programming of the Russian Academy of Sciences, Moscow, Russia
{andrianov,mutilin,mandrykin,vasilyev}@ispras.ru

Abstract. Our submission to SV-COMP'18 is a composite tool based on software verification framework CPAchecker and static analysis platform Frama-C. The base verifier uses a combination of predicate and explicit value analysis with block-abstraction memoization as the CPA-BAM-BnB tool presented at SV-COMP'17. In this submission we augment the verifier on reachability verification tasks with a slicer that is able to remove those statements that are irrelevant to the reachability of error locations in the analysed program. The slicer is based on context-sensitive flow-insensitive separation analysis with typed polymorphic regions and simple dependency analysis with transitive closures. The resulting analysis preserves reachability modulo possible non-termination while removing enough irrelevant code to achieve considerable speedup of the main analysis. The slicer is implemented as a Frama-C plugin.

1 Verification Approach

The submission presents a composite setting comprised of a mature static verification tool CPAchecker [1] and an experimental reachability slicer (a Frama-C [2] plugin) intended to speed up verification by pruning the verification scope prior the application of the main analysis. By verification scope we understand the code to be analyzed rather than the search space explored by the main analysis since the slicer doesn't prune the search space as it is, but rather removes statements (including function calls) that can be proved to not influence the verification outcome. The slicer included in this submission is currently only applicable to reachability verification tasks, though the underline algorithm is not generally limited to reachability of a small number of error locations and so can be potentially extended to support e.g. memory safety properties.

The slicer is based on a relatively simple mark-and-sweep algorithm, where the relevant statements are first identified by computing transitive closure of

M. Mandrykin—Jury member.

The research was supported by RFBR grant 18-01-00426.

D. Beyer and M. Huisman (Eds.): TACAS 2018, LNCS 10806, pp. 427–431, 2018.
https://doi.org/10.1007/978-3-319-89963-3_26

the dependency relation, then marked, and finally the remaining statements are removed to produce a sliced verification task. The mark-and-sweep slicing is performed on top of preliminary region analysis, which allows to handle abstract memory locations ascribed to the corresponding disjoint memory regions essentially similar to usual unaliased program variables.

The region analysis implemented in the current submission is a conservative over-approximation of context-sensitive flow-insensitive separation analysis with polymorphic regions for deductive verification. It was first described in [3] and later substantially extended in [4]. The conservative approximation is needed because the original analysis generally requires user annotations. The over-approximation is expressed in the form of additional dependencies introduced on the marking stage rather than in the region analysis itself. The dependencies allow to approximate reinterpretations of memory regions (corresponding to the use of unions and arbitrary pointer type casts), but not some corner cases of pointer arithmetic (mostly arithmetic dependent on a particular layout of structure fields), so the resulting analysis remains unsound in the general case. However, the results of analysis benchmarking using CPAchecker as reachability verifier on the tasks in SV-COMP `SoftwareSystems` category showed no cases of unsoundness caused by the region analysis. This may be explained by the fact that most of the cases where the analysis is unsound with respect to a low-level C memory model are also regarded as undefined behavior by the C standard, so are probably quite rarely used in practice.

2 Software Architecture

The main CPAchecker verification framework is included in the submission without any considerable changes. The combined tool is implemented as a wrapper script that encapsulates the main verifier invocation and does the following:

- extracts the property specification and verification task from the arguments;
- runs the slicer with timeout of 400 s (the sliced program is written to an intermediate C file);
- runs CPAchecker configuration `ldv-bam-svcomp` on the sliced program;
- post-processes the witness produced by CPAchecker.

The slicer (named Crude_slicer) is implemented as a plugin to Frama-C [2], an extensible platform for source-code analysis of C software. The plugin implementation does not interact with other Frama-C plugins and only makes use of the Frama-C kernel. The plugin also uses OCamlgraph [5] library. Both the Frama-C platform and the Crude_slicer plugin are implemented in OCaml.

The witness post-processing stage currently simply removes the character offsets from the resulting witness (the line numbers are preserved using line directives supported by CPAchecker) and substitutes checksum of the original program source.

Since the `SoftwareSystems` category of the competition also contains memory safety (and overflow) verification tasks, the submission also includes memory safety configuration `smg-ldv` based on shape analysis presented in [6].

3 Evaluation of the Approach

The slicer is currently able to handle only reachability verification tasks. It was evaluated on 2734 tasks from the `Systems_DeviceDriversLinux64_ReachSafety` subcategory of the SV-COMP'18 benchmarks on Intel Xeon E3-1230 v5 (3.4 GHz) machines in the competition setting. The submitted configuration with slicing was compared to baseline CPA-BAM-BnB [7,8] configuration (`-ldv-bam-svcomp`) without slicing that was also submitted to this year's competition. The results are presented in the following table:

`TRUE` verdicts			`FALSE` verdicts			Speedup		
New (+)	Lost (−)	Total	New (+)	Lost (−)	Total	Min	Max	Average
151	10	2252	97	11	267	0.03 ×	18.59 ×	1.17 ×

The table presents the results for correct verdicts only and does not take witness checking into account.

There are two significant limitations of the approach. First, the slicing is performed under assumption that all possible execution paths in the verified program are finite. This does not lead to unsoundness, since reachability (as a safety property) can be assumed to be violated only on finite paths. However, there is 3 wrong `FALSE` verdicts reported on the benchmarks where an error location is spuriously reached after passing through an infinite loop removed by the slicer. Another limitation is that the resulting tool can not produce precise witnesses both due to imprecision in source code locations and (more importantly) due to unavailability of either invariants or error paths in the sliced out parts of the code. The caused 1090 `TRUE` verdicts and all `FALSE` verdicts to fail to be confirmed by the witness checkers on the competition.

The time required for slicing varies from 0.08 to 1905.47 s with an average of 14.82 s. So in the submission the slicer is run with a timeout of 400 s and the remaining tasks (17 out of 2734 in the evaluation) are passed to the main verifier without slicing.

4 Tool Setup and Configuration

The submission is available for download as a ZIP archive named `cpa-bam-slicing.zip` from the SV-COMP repository by following URL: https://gitlab.com/sosy-lab/sv-comp/archives/tree/master/2018. The submission includes CPAchecker version 1.6.1 and a statically linked version of Frama-C `Sulfur-20171101-beta` with Crude_slicer plugin. The version of the plugin corresponds to commit `fcd3b927`. CPAchecker requires Java 8 runtime environment. The invocation of the slicer is embedded in the CPAchecker wrapper script, so the whole tool has to be executed with the following command line:

```
scripts/cpa.sh -ldv-bam-svcomp -disable-java-assertions
               -heap 10000m -spec prop.prp program.c
```

The tool participates in `SoftwareSystems` category, the corresponding benchmark definition is `cpa-bam-slicing.xml`.

Acknowledgements. The CPAchecker project is open-source and developed by an international research group from Ludwig-Maximilian University of Munich, University of Passau, Ivannikov Institute for System Programming of the Russian Academy of Sciences and several other universities and institutions. More information about the project can be accessed at https://cpachecker.sosy-lab.org. The slicer is developed as part of the Linux Driver Verification project [9] (http://linuxtesting.org/ldv), the slicer project page is https://forge.ispras.ru/projects/crude_slicer. Both the CPAchecker tool and the Crude_slicer plugin are distributed under the terms of the Apache License, Version 2.0. The Frama-C platform (http://frama-c.com/) is co-developed at two French public institutions: CEA LIST and INRIA Saclay – Île-de-France, and licensed under GNU LGPL v2. We thank all contributors of the projects for their work.

References

1. 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
2. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-C: a software analysis perspective. In: Eleftherakis, G., Hinchey, M., Holcombe, M. (eds.) SEFM 2012. LNCS, vol. 7504, pp. 233–247. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33826-7_16
3. Hubert, T., Marché, C.: Separation analysis for deductive verification. In: Heap Analysis and Verification (HAV 2007), Braga, Portugal, pp. 81–93, March 2007
4. Mandrykin, M.U., Khoroshilov, A.V.: Region analysis for deductive verification of C programs. Program. Comput. Softw. **42**(5), 257–278 (2016)
5. Conchon, S., Filliâtre, J.C., Signoles, J.: Designing a generic graph library using ML functors. In: Morazán, M.T. (ed.) Trends in Functional Programming, vol. 8, Selected Papers of the 8th International Symposium on Trends in Functional Programming (TFP 2007), New York, USA. Intellect (2008)
6. Muller, P., Vojnar, T.: CPAlien: shape analyzer for CPAChecker. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 395–397. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_28
7. Andrianov, P., Friedberger, K., Mandrykin, M., Mutilin, V., Volkov, A.: CPA-BAM-BnB: block-abstraction memoization and region-based memory models for predicate abstractions. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 355–359. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54580-5_22
8. Volkov, A., Mandrykin, M.: Predicate abstractions memory modeling method with separation into disjoint regions. Proc. Inst. Syst. Program. **29**, 203–216 (2017)
9. Zakharov, I.S., Mandrykin, M.U., Mutilin, V.S., Novikov, E.M., Petrenko, A.K., Khoroshilov, A.V.: Configurable toolset for static verification of operating systems kernel modules. Program. Comput. Softw. **41**(1), 49–64 (2015)