GOBLINT: Thread-Modular Abstract Interpretation Using Side-Effecting Constraints (Competition Contribution)

Simmo Saan^{1(✉)*}, Michael Schwarz^{2(✉)},
Kalmer Apinis¹, Julian Erhard², Helmut Seidl²,
Ralf Vogler², and Vesal Vojdani¹

¹ University of Tartu, Tartu, Estonia

{simmo.saan, kalmer.apinis, vesal.vojdanil}@ut.ee

² Technische Universität München, Garching, Germany

{m.schwarz, julian.erhard, helmut.seidl, ralf.vogler}@tum.de

Abstract. GOBLINT is a static analysis framework for C programs specializing in data race analysis. It relies on thread-modular abstract interpretation where thread interferences are accounted for by means of flow-insensitive global invariants.

1 Verification Approach

GOBLINT is a static analyzer for C programs based on the framework of abstract interpretation [5]. It performs flow- and context-sensitive interprocedural analysis, using partial tabulation to handle procedure calls. The analysis of concurrent programs is thread-modular: analyzing each thread in isolation, as opposed to analyzing their interleavings. This scales well to larger programs with many threads. Interferences between threads happen through global variables, which are abstracted by a context- and flow-insensitive global invariant. When no other thread can interfere, copies of global variables are *privatized* within the local state. Their values may deviate from the global invariant due to local updates, thereby improving precision [11].

The analysis is specified using a side-effecting constraint system [3], in which right-hand sides of constraints can, during their evaluation, make additional contributions (*side effects*) to other constraint system variables. These side effects can be conveniently used both to express partial context-sensitivity of function calls and to add contributions to the global invariant. Such a constraint system is solved using a *local* generic solver, which yields a (post-)solution for just the reachable program points and contexts [1,8]. Solving is not strictly separated into widening and narrowing phases, but these may be intertwined instead [1]. Results of the analysis are reported only at the end based on the computed solution, as widening during the fixpoint computation might lead to spurious property violations, which later disappear due to narrowing.

* Jury member

Reachability Safety. Reachability is mainly determined using value analysis, which, for integers, employs abstract domains based on intervals and exclusion sets. The value analysis also handles pointers (computing points-to information), heap memory (using allocation-site abstraction), structs, unions and arrays. The abstraction of arrays employs partitioning by the symbolic expression that is used to index into the array. On top of that, both global variables and heap-allocated memory are partitioned into disjoint regions [9].

No Overflows. The sound interval analysis is implemented using arbitrary precision integers. If the interval for an expression lies completely in the value range of its signed integer type, no overflow can occur at this location.

No Data Race. The main goal of GOBLINT is data race detection and its analyses have been optimized for this purpose. Mutexes may be handled both path-sensitively and symbolically. Memory accesses are partitioned (e.g., by heap region [9]), while locking expressions and access expressions are correlated using address equalities (e.g., a domain of affine and Herbrand equalities [10]) in order to analyze more sophisticated locking patterns [11].

2 Software Architecture

GOBLINT is implemented in OCAML and uses an updated fork of CIL [6] as its parser frontend for the C language. Since the latter requires preprocessed code, GCC is executed for preprocessing the input, although this step should be unnecessary on the SV-COMP benchmarks. No other major libraries or external tools are required.

The architecture of GOBLINT [2] is designed to be modular. Analyses, which are defined by their abstract domains and transfer functions, can be activated via runtime configuration options. A flexible query system allows for communication between analyses. Together, the combined analyses and the control-flow graphs of the functions in the program provide the side-effecting constraint system, which is solved by some local generic solver. While a number of solvers are available, the improved top-down solver TD3 [8] was employed for SV-COMP 2021. Post-processing the solution yields results for the analysis.

3 Strengths and Weaknesses

Due to over-approximation, abstract interpretation as employed by GOBLINT can only determine whether the correctness specification *must hold* or *may be violated*, but not whether a concrete violating execution exists. Therefore, to avoid a large number of false alarms due to imprecision in SV-COMP, GOBLINT only reports results “true” and “unknown” respectively. This is a clear limitation of our approach, as all competing tools do report definite violations. The strength of our approach, on the other hand, is that it aims to be sound by design (up to out-of-scope features of the input program as, e.g., inline assembler). This is

evidenced by the fact that GOBLINT does not produce any incorrect results in the competition.

GOBLINT performs best in the *SoftwareSystems* and *ReachSafety-Product-Lines* categories that consist of larger real-world programs, for which our approach is well suited. On the downside, our verifier performs poorly in reachability safety categories that contain smaller programs with intricate correctness conditions which our abstract domains cannot express.

Even though the support for checking overflows is very new in GOBLINT, it has some success in the *NoOverflows* category. Unfortunately, the tool has no success in *SoftwareSystems-*-NoOverflows*.

Although GOBLINT specializes in concurrency, it performs quite poorly in the *ConcurrencySafety* category. We believe this is because most benchmarks in the category require rather precise analysis of thread interleavings, which is not done in our thread-modular approach.

As GOBLINT has been optimized for data race detection, it unsurprisingly performs better in the *NoDataRace* demo category. It must be noted that the majority of benchmarks in the category were submitted from our own test suite, consisting of racy and race-free programs.

While the analyses can be fine-tuned via configuration options, the parameters are static and do not currently depend on the property nor the input program. A more granular and dynamic configuration system would allow increased precision, by enabling more expensive analyses where necessary, or decreased resource usage, by disabling unnecessary analyses, e.g., concurrency analyses on single-threaded programs. Furthermore, integrating counterexample-guided abstraction refinement (CEGAR) into our framework might allow GOBLINT to also report violations, while avoiding false alarms and gaining more precision.

4 Tool Setup and Configuration

GOBLINT version `svcomp21-0-g82e03b87` participated in SV-COMP 2021 [4,7]. It is available in both binary (Ubuntu 20.04) and source code form at our GitHub repository under the `svcomp21` tag.³ The only runtime dependency is GCC. Instructions for building from source can be found in the `README`.

Both the tool-info module and the benchmark definition for SV-COMP are named `goblint`. They correspond to running the tool as follows:

```
./goblint --conf conf/svcomp21.json --sets ana.specification
property.prp input.c
```

GOBLINT participated in the following categories: *ReachSafety*, *ConcurrencySafety*, *NoOverflows*, *SoftwareSystems* (while opting-out from *SoftwareSystems*-MemSafety*) and *NoDataRace* (demo category).

³ <https://github.com/goblint/analyzer/releases/tag/svcomp21>

5 Software Project and Contributors

GOBLINT development takes place on GitHub,⁴ while related publications are listed on its website.⁵ It is an MIT-licensed joint project of the Technische Universität München (Chair of Formal Languages, Compiler Construction, Software Construction) and University of Tartu (Laboratory for Software Science).

Acknowledgements. This work was supported by Deutsche Forschungsgemeinschaft (DFG) – 378803395/2428 CONVEY and the Estonian Research Council grant PSG61. We would like to thank everyone who has contributed to GOBLINT over the years.

References

1. Amato, G., Scozzari, F., Seidl, H., Apinis, K., Vojdani, V.: Efficiently intertwining widening and narrowing. *Science of Computer Programming* **120**, 1–24 (May 2016). DOI: [10.1016/j.scico.2015.12.005](https://doi.org/10.1016/j.scico.2015.12.005)
2. Apinis, K.: Frameworks for analyzing multi-threaded C. Ph.D. thesis, Technische Universität München (2014)
3. Apinis, K., Seidl, H., Vojdani, V.: Side-Effecting Constraint Systems: A Swiss Army Knife for Program Analysis. In: *APLAS '12*. pp. 157–172. Springer (2012). DOI: [10.1007/978-3-642-35182-2_12](https://doi.org/10.1007/978-3-642-35182-2_12)
4. Beyer, D.: Software Verification: 10th Comparative Evaluation (SV-COMP 2021). In: *Proc. TACAS (2)*. LNCS 12652, Springer (2021)
5. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: *POPL '77*. pp. 238–252 (1977). DOI: [10.1145/512950.512973](https://doi.org/10.1145/512950.512973)
6. Nacula, G.C., McPeak, S., Rahul, S.P., Weimer, W.: CIL: Intermediate language and tools for analysis and transformation of C programs. In: *CC '02*. pp. 213–228. Springer (2002). DOI: [10.1007/3-540-45937-5_16](https://doi.org/10.1007/3-540-45937-5_16)
7. Saan, S., Schwarz, M., Apinis, K., Erhard, J., Seidl, H., Vogler, R., Vojdani, V.: Goblint at SV-COMP 2021 (Dec 2020). DOI: [10.5281/zenodo.4485853](https://doi.org/10.5281/zenodo.4485853)
8. Seidl, H., Vogler, R.: Three improvements to the top-down solver. In: *PPDP '18*. pp. 1–14 (2018). DOI: [10.1145/3236950.3236967](https://doi.org/10.1145/3236950.3236967)
9. Seidl, H., Vojdani, V.: Region Analysis for Race Detection. In: *SAS '09*. pp. 171–187. Springer (2009). DOI: [10.1007/978-3-642-03237-0_13](https://doi.org/10.1007/978-3-642-03237-0_13)
10. Seidl, H., Vojdani, V., Vene, V.: A Smooth Combination of Linear and Herbrand Equalities for Polynomial Time Must-Alias Analysis. In: *FM '09*. pp. 644–659. Springer (2009). DOI: [10.1007/978-3-642-05089-3_41](https://doi.org/10.1007/978-3-642-05089-3_41)
11. Vojdani, V., Apinis, K., Rõtov, V., Seidl, H., Vene, V., Vogler, R.: Static Race Detection for Device Drivers: The Goblint Approach. In: *ASE 2016*. pp. 391–402. ACM (2016). DOI: [10.1145/2970276.2970337](https://doi.org/10.1145/2970276.2970337)

⁴ <https://github.com/goblint/analyzer>

⁵ <https://goblint.in.tum.de>

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.