Goblint Validator: Correctness Witness Validation by Abstract Interpretation (Competition Contribution)

Simmo Saan[1] (✉) ⋆, Julian Erhard[2,3], Michael Schwarz[2], Stanimir Bozhilov[2], Karoliine Holter[1], Sarah Tilscher[2,3], Vesal Vojdani[1], and Helmut Seidl[2]

[1] University of Tartu, Tartu, Estonia
{simmo.saan,karoliine.holter,vesal.vojdani}@ut.ee
[2] Technische Universität München, Garching, Germany
{julian.erhard,m.schwarz,stanimir.bozhilov,
sarah.tilscher,helmut.seidl}@tum.de
[3] Ludwig-Maximilians-Universität München, Munich, Germany

Abstract. Goblint is an abstract interpretation framework for C programs with a specialty in concurrency. Using a novel approach, we turn it into a validator of YAML correctness witnesses for all SV-COMP categories. We describe its results at SV-COMP 2024 which includes the first large-scale evaluation of our validator.

1 Validation Approach

Goblint Validator is an extension of the Goblint verifier [14–16] for validation of correctness witnesses in the YAML format [1], consisting of location and loop invariants. The extension involves two related but independent components: witness invariants are checked for correctness and unassumed for speedup. We present here a high-level overview of our recently-published approach to abstract-interpretation–powered witness validation [17].

Correctness of witness invariants is determined by treating them as additional proof obligations. However, instead of inserting assert statements into the program, the validator uses the Goblint verifier as a black box to check whether its computed abstract states satisfy the witness invariants. Hence, invalid witness invariants cannot undermine soundness of the verification process via refinement.

Speedup from witness invariants is attained by incorporating novel *unassume* statements with the invariants into the program. As opposed to refining the abstract state like *assume* operations, these relax the state instead. Doing so in a controlled manner, fixpoint iteration can converge faster, i.e., in fewer iterations. In the best case, the witness invariant precisely characterizes the fixpoint, avoiding further iteration. Unassuming can also make the abstract interpreter more precise, without requiring more expressive abstract domains, by leading the solver to a more precise fixpoint, which widening would otherwise extrapolate over [17].

⋆ Jury member

Sound unassume operators must preserve all reaching concrete states, thus preserving soundness of the entire analysis. Goblint Validator implements two different unassume operators:

1. For non-relational domains (e.g., numeric intervals or points-to sets), a classic propagating algorithm for assume operators [4, 7] is adapted with minimal modifications. This admits relaxing abstract values in dynamically allocated memory through pointers.
2. For relational domains (e.g., octagons), dual-narrowing [8] is employed to retain more relations than a generic unassume operator definition [17].

2 Software Architecture

Goblint Validator builds on the Goblint verifier [14–16] which is implemented in OCaml, uses an updated fork of CIL [12] as its frontend and Apron [9] for relational domains.

Instead of altering the control-flow graphs, unassume statements are inserted implicitly as events that activated analyses can handle. In the modular architecture of Goblint [2] the unassume analysis is responsible for emitting these events after transfer functions corresponding to witness invariants. Widening tokens [10] are used to delay widening and allow the invariants to be incorporated without immediate precision loss. The solution of a side-effecting constraint system [3, 18] is post-processed to validate witness invariants and determine the verdict.

3 Strengths and Weaknesses

Overall, Goblint Validator inherits the strengths and weaknesses of Goblint, which are described in its tool papers [14–16]. Thanks to the generic validation approach, the validator works in *all* SV-COMP categories as the Goblint verifier, including those that are currently excluded from correctness witness validation, e.g., concurrency. Due to over-approximation, the verifier can only prove the absence of bugs, but not their presence. Consequently, the validator can currently only *confirm* correctness witnesses. However, it could be extended to *reject* violation witnesses in the future.

We evaluate our validator according to the same three aspects considered by Beyer et al. [6]: same-framework consistency, content-effort dependence and cross-framework validation. The first two only focus on witnesses produced by the Goblint verifier.

Regarding same-framework consistency, table 1 lists how many tasks with each property it can verify and how many of those witnesses Goblint Validator can confirm. The overall average confirmation rate of 78% is lower than the 90% Beyer et al. [6] report for CPAchecker and UAutomizer with GraphML witnesses. Reasons for unconfirmed witnesses range from excessive precision loss by unassuming to validator crashes. In some cases, the validator exceeds resource limits, likely due to large witnesses with many unhelpful invariants. A

Table 1. Number of tasks verified by Goblint and their witness validation verdicts by Goblint Validator, grouped by property.

Property	Correct tasks	Goblint verified	Goblint Validator Confirmed	Goblint Validator Unconfirmed
unreach-call	11,351	1,894	1,064 (56%)	830
no-overflow	5,562	3,932	3,416 (87%)	516
termination	1,536	619	297 (48%)	322
no-data-race	781	695	510 (73%)	185
valid-memsafety	2,796	1,963	1,801 (92%)	162
valid-memcleanup	2	0	–	–
Total	22,028	9,103	7,088 (78%)	2,015

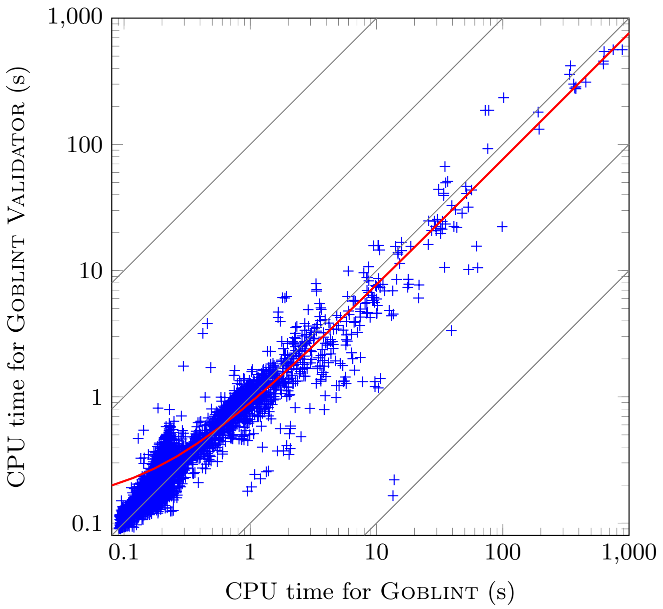

Fig. 1. CPU time scatter plot where each mark (in blue) indicates a task verified by Goblint and whose witness was confirmed by Goblint Validator. Ordinary least squares (OLS) regression (in red) follows $y = 0.76x + 0.14$ ($r^2 = 0.94$).

Table 2. Percentage of witnesses from other verifiers confirmed by Goblint Validator.

				Ultimate			
Verifier	CPAchecker	CPV	Mopsa	Automizer	GemCutter	Kojak	Taipan
Confirmed	8%	6%	78%	46%	60%	57%	51%

handful of instances indicate mismatches between witness generation and their interpretation due to implementation errors in either the verifier or the validator. Fixing such issues could improve the overall quality of the framework [6].

Regarding content-effort dependence, fig. 1 plots the corresponding verification and validation times in the 7,088 confirmed cases. While the results at the low end (< 1 s) are noisy, the results at the high end (> 5 s) show the benefit of witness validation, with up to $10\times$ improvements. Regression analysis estimates an average speedup of 24%, which matches our previous results [17], albeit with greater variance. This is unlike CPAchecker and UAutomizer for which no general performance improvement from consuming witnesses was observed [6].

Regarding cross-framework validation, table 2 presents the confirmation rate of Goblint Validator of correctness witnesses from other tools. For the Ultimate tool family, the percentage is between 46% and 60%, which is similar to what Beyer et al. [6] observed. We have a high ratio for the Mopsa abstract interpreter [11], although it only produces trivial witnesses containing no invariants, on which Goblint Validator effectively reduces to the Goblint verifier. Nevertheless, overwhelming success of Mopsa in the *SoftwareSystems* category warrants independent validation of abstract interpretation results.

4 Tool Setup and Configuration

Goblint Validator version `svcomp24-0-gc2e9465a7` took part in *all* categories except *FalsificationOverall* of SV-COMP 2024 [5, 13]. It is available in both binary (Ubuntu 22.04) and source code form at our GitHub repository.[4] Instructions for building from source can be found in the `README`.

The tool-info module for BenchExec is named `goblint` and the benchmark definition for SV-COMP is `goblint-validate-correctness-witnesses-2.0`. They correspond to running the tool as follows:

```
./goblint --conf conf/svcomp24-validate.json \
          --set witness.yaml.unassume witness.yml \
          --set witness.yaml.validate witness.yml \
          --set ana.specification property.prp input.c
```

5 Software Project and Contributors

Goblint Validator development takes place alongside Goblint on GitHub, while related publications are listed on its website.[5] It is an MIT-licensed project initiated by Technische Universität München and the University of Tartu.

Acknowledgments. This work was supported by Deutsche Forschungsgemeinschaft (DFG) – 378803395/2428 ConVeY 2. We would like to thank everyone who has contributed to the Goblint framework over the years, laying the foundation for our validator.

[4] https://github.com/goblint/analyzer/releases/tag/svcomp24
[5] https://github.com/goblint/analyzer and https://goblint.in.tum.de

Data Availability Statement. All data of SV-COMP 2024 are archived as described in the competition report [5] and available on the competition website. This includes the verification tasks, results, witnesses, scripts, and instructions for reproduction. The version of Goblint as used in the competition is archived on Zenodo [13].

Bibliography

[1] Format for correctness witnesses, version 2.0 (2023), URL https://sosy-lab.gitlab.io/benchmarking/sv-witnesses/yaml/correctness-witnesses.html

[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

[4] Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.F.: Revising hull and box consistency. In: Logic Programming, p. 230–244, The MIT Press (1999), doi: 10.7551/mitpress/4304.003.0024

[5] Beyer, D.: State of the art in software verification and witness validation: SV-COMP 2024. In: TACAS '24, Springer (2024)

[6] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M.: Correctness witnesses: exchanging verification results between verifiers. In: FSE '16, pp. 326–337, ACM (2016), doi: 10.1145/2950290.2950351

[7] Cousot, P.: The calculational design of a generic abstract interpreter. In: Calculational System Design, NATO ASI Series F. IOS Press, Amsterdam (1999), URL https://www.di.ens.fr/~cousot/COUSOTpapers/publications.www/Cousot-Marktoberdorf98.pdf.gz

[8] Cousot, P.: Abstracting induction by extrapolation and interpolation. In: VMCAI '15, pp. 19–42, Springer (2015), doi: 10.1007/978-3-662-46081-8_2

[9] Jeannet, B., Miné, A.: Apron: A library of numerical abstract domains for static analysis. In: CAV '09, pp. 661–667, Springer (2009), doi: 10.1007/978-3-642-02658-4_52

[10] Mihaila, B., Sepp, A., Simon, A.: Widening as abstract domain. In: NASA Formal Methods, pp. 170–184, Springer (2013), doi: 10.1007/978-3-642-38088-4_12

[11] Monat, R., Milanese, M., Parolini, F., Boillot, J., Ouadjaout, A., Miné, A.: Mopsa-C: Improved verification for C programs, simple validation of correctness witnesses. In: TACAS '24, Springer (2024)

[12] Necula, 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

[13] Saan, S., Erhard, J., Schwarz, M., Bozhilov, S., Holter, K., Tilscher, S., Vojdani, V., Seidl, H.: Goblint at SV-COMP 2024 (Nov 2023), doi: 10.5281/zenodo.10202867, tool artifact

[14] Saan, S., Erhard, J., Schwarz, M., Bozhilov, S., Holter, K., Tilscher, S., Vojdani, V., Seidl, H.: Goblint: Abstract interpretation for memory safety and termination (competition contribution). In: TACAS '24, Springer (2024)

[15] Saan, S., Schwarz, M., Apinis, K., Erhard, J., Seidl, H., Vogler, R., Vojdani, V.: Goblint: Thread-modular abstract interpretation using side-effecting constraints. In: TACAS '21, pp. 438–442, Springer (2021), doi: 10.1007/978-3-030-72013-1_28

[16] Saan, S., Schwarz, M., Erhard, J., Pietsch, M., Seidl, H., Tilscher, S., Vojdani, V.: Goblint: Autotuning thread-modular abstract interpretation. In: TACAS '23, vol. 2, pp. 547–552, Springer (2023), doi: 10.1007/978-3-031-30820-8_34

[17] Saan, S., Schwarz, M., Erhard, J., Seidl, H., Tilscher, S., Vojdani, V.: Correctness witness validation by abstract interpretation. In: VMCAI '24, pp. 74–97, Springer (2024), doi: 10.1007/978-3-031-50524-9_4

[18] Seidl, H., Vogler, R.: Three improvements to the top-down solver. Math. Struct. Comput. Sci. **31**(9), 1090–1134 (2021), doi: 10.1017/S0960129521000499