ConcurrentWitness2Test: Test-Harnessing the Power of Concurrency (Competition Contribution)

Levente Bajczi [*(✉)], Zsófia Ádám, and Zoltán Micskei

Department of Measurement and Information Systems
Budapest University of Technology and Economics, Budapest, Hungary
bajczi@mit.bme.hu

Abstract. ConcurrentWitness2Test is a violation witness validator for concurrent software. Taking both nondeterminism of data and interleaving-based nondeterminism into account, the tool aims to use the metadata described in the violation witnesses to synthesize an executable test harness. While plagued by some initial challenges yet to overcome, the validation performance of ConcurrentWitness2Test corroborates the usefulness of the proposed approach.

Funding. This research was partially funded by the ÚNKP-23-{2,3}-I New National Excellence Program; and the Doctoral Excellence Fellowship Programme (funded by the NRDI Fund of Hungary and the BME University).

1 Validation Approach

There are multiple violation witness validators in the ReachSafety category of SV-COMP that are based on test harness generation [3]. However, none take part in the category for concurrent programs, presumably due to the increased complexity in orchestrating the different thread interleavings prescribed by the witness files. ConcurrentWitness2Test aims to fill this gap, by providing an enhanced test harness that takes not only the data-nondeterminism into account, but also the nondeterminism caused by concurrency. In this paper we concentrate on solving the latter, as the former is already well documented by the implementing tools [3].

The current witness format for concurrent software defines two edge data fields that we can extract information from [3]:

`createThread`: The unique ID of the new thread that results from the execution of the containing edge

`threadId`: Which thread is currently active when the containing edge is executed. Valid values have at least one `createThread` entry in the witness automaton that must be executed prior to the current edge

[*] Jury member representing ConcurrentWitness2Test at SV-COMP 2024.

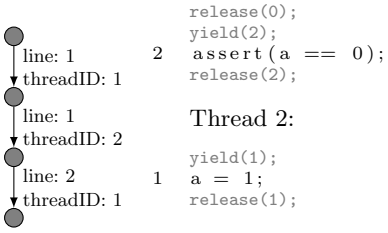

Fig. 1: Witness

```
Thread 1:

                yield(0);
1               a = 0;
                release(0);
                yield(2);
2               assert(a == 0);
                release(2);

Thread 2:

                yield(1);
1               a = 1;
                release(1);
```

Fig. 2: Source

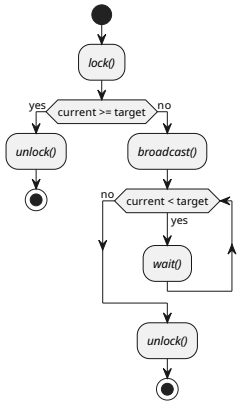

Fig. 3: `yield(target)`

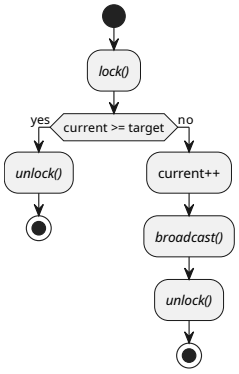

Fig. 4: `release(target)`

Using these pieces of information, we insert a `yield` and `release` call around each action (as seen in the example in Figure 2, based on the metadata from Figure 1), with the parameter `target` increasing at every encountered edge. These functions are shown in Figure 3 and Figure 4, respectively. They rely on a shared variable `current` denoting the next value where the functions need to take effect (to handle revisited locations in the source, e.g., in a loop), alongside a mutex and a condition variable. *Locking* and *unlocking* in the figures refer to operations on the mutex variable; while *broadcasting* and *waiting* refer to operations on the condition variable.

One of the main obstacles to overcome is the resolution of the `threadID` metadata. In our experience, none of the tools produce fully specified witnesses in terms of interleavings, i.e., not every action is totally ordered in the program. While this is acceptable according to the witness format [3], a certain level of nondeterminism might remain in the program after applying the witness. To overcome this problem we rely on statistics, i.e., we execute the resulting harness multiple times, and classify the results as *always observable*, *sometimes observable* and *never observable*. Observability refers to that of the error state, tested by inspecting the exit code of the program. At SV-COMP'24 we opted to only refuse witnesses with *never observable* verdicts.

2 Software Architecture

ConcurrentWitness2Test is a Python project, relying on `pycparser`[1] for parsing C files, and `networkx`[2] for parsing GraphML-based witnesses. As opposed to the harness-only solutions of other witness-to-test validators [3], Con-

[1] https://github.com/eliben/pycparser
[2] https://networkx.org/

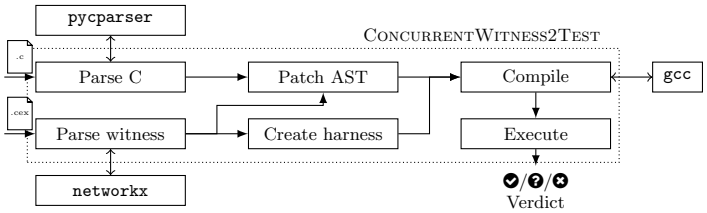

Fig. 5: Architecture of ConcurrentWitness2Test

currentWitness2Test also needs to modify the AST of the C file to insert the function calls to `yield` and `release`, therefore the intermediate output of ConcurrentWitness2Test consists of a patched C file and a separate test harness. We use `gcc`[3] to compile these resulting files to an executable. We run this executable at most 100 times, with an option for early termination if the error becomes observable. See Figure 5 for an overview of this workflow.

3 Discussion of Strengths and Weaknesses of the Approach

As seen in Table 1[4], ConcurrentWitness2Test lacks support for some tools' witnesses. Since then, this limitation has been mostly rectified, but not in time for SV-COMP. The main shortcoming of the competition version of ConcurrentWitness2Test was the handling of cases where edge attributes were given for complex syntactic elements, such as loops, and we tried to insert the function calls into the heads of loops instead of their body. This was an easy fix, and we hope to further the support for various tools even more for next year's SV-COMP.

Despite these temporary shortcomings, ConcurrentWitness2Test still correctly confirmed 1197 results[2]. In contrast, the validator was wrong only 239 times: 2 witnesses were confirmed and 237 witnesses were refused erroneously[5]. These numbers highlight the strength of our approach.

We also note that ConcurrentWitness2Test confirmed 932 results with only a *sometimes observable* verdict. This means that multiple tools produce nondeterministic witnesses, where some interleaving leads the execution to an error state, but not all. We suggest tool developers to concentrate on providing better, deterministic witnesses in order for their results to always be validated. We will aim to constrain our acceptance criteria to *always observable* in future competitions.

[3] https://gcc.gnu.org/
[4] Unofficial results, since no official results were published at the time of writing.
[5] Here, *erroneous* covers all cases when the tool could not reproduce the bug. Therefore, this might not be our tool's shortcoming, but the result of bad witnesses.

Table 1: Results per supported tool, results for wrong verdicts in parentheses

	Dartagnan	Divine	Theta	UAutomizer	UGemCutter	UTaipan
Confirmed	178	179 (2)	191	186	235	228
Refused	79	25 (1)	8	74	22	29
Error	193	111	96	168	194	170

4 Tool Setup and Configuration

The binary archive available at Zenodo [1] contains all required dependencies in the form a virtual environment except for the python 3 interpreter, which needs to be installed separately (e.g., via the `python3` package on Ubuntu 22.04).

The tool can be started either directly via the `main.py` file, or the convenience script in `start.sh`. Either way, the tool expects two inputs: an argument providing the (preprocessed) C file, and the witness file with the `--witness <file>` flag. Upon success, the tool always outputs a single line starting with the string `Verdict:`, with the verdict `SOMETIMES/ALWAYS/NEVER` directly afterward. Some handled exceptions also appear as verdicts.

Up-to-date badges on verification tool support can be seen on the main GitHub page[6]. Tool support has been significantly enhanced since the version nominated for the competition, in preparation for next year's SV-COMP, and for tools to use that may want to improve their witnesses in the meantime.

5 Software Project and Data Availability

ConcurrentWitness2Test is a validation tool maintained by the Critical Systems Research Group[7] of the Budapest University of Technology and Economics. The project is available open-source on GitHub[8] under an Apache 2.0 license. The version (`1.0.0`) used in the competition is available at [1].

References

1. Bajczi, L., Ádám, Z., Micskei, Z.: ConcurrentWitness2Test - SV-COMP'24 Validator Archive (Nov 2023). https://doi.org/10.5281/zenodo.10184336
2. Beyer, D.: State of the art in software verification and witness validation: SV-COMP 2024. In: Proc. TACAS. LNCS , Springer (2024)

[6] https://github.com/ftsrg/ConcurrentWitness2Test#tool-support
[7] https://ftsrg.mit.bme.hu/en/
[8] https://github.com/ftsrg/ConcurrentWitness2Test

3. Beyer, D., Dangl, M., Lemberger, T., Tautschnig, M.: Tests from witnesses - execution-based validation of verification results. In: Dubois, C., Wolff, B. (eds.) Tests and Proofs - 12th International Conference, TAP@STAF 2018, Toulouse, France, June 27-29, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10889, pp. 3–23. Springer (2018). https://doi.org/10.1007/978-3-319-92994-1_1