

Comparing Transparent Static Analyzers with Open Verification Dashboard (Artifact)

Tom Goaldard 

ENS Rennes, Campus de Ker Lann, 11 avenue Robert Schuman, 35170 Bruz, France

Karoliine Holter 

Institute of Computer Science, University of Tartu, Estonia

Simmo Saan 

Institute of Computer Science, University of Tartu, Estonia

Vesal Vojdani 

Institute of Computer Science, University of Tartu, Estonia

Raphaël Monat 

Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRISAL, 59000 Lille, France

Abstract

This artifact accompanies the eponymous article. It enables reproduction of the experimental claims made in the article that are covered by this description.

2012 ACM Subject Classification Software and its engineering → Automated static analysis; Software and its engineering → Software testing and debugging

Keywords and phrases automated static analysis, multi-tool integration, interoperability, proof obligations, result aggregation, verification progress, selectivity metric, reproducibility, dashboard

Digital Object Identifier 10.4230/DARTS.12.1.6

Funding This work is supported by the European Union and the Estonian Research Council via projects PRG2764 and TEM-TA119, by grant agreement ANR-24-CE25-7956-01 RAISIN from the French Agence Nationale de la Recherche, and by an Amazon Research Award, Fall 2024.

Related Article Tom Goaldard, Karoliine Holter, Simmo Saan, Vesal Vojdani, and Raphaël Monat, “Comparing Transparent Static Analyzers with Open Verification Dashboard”, in 40th European Conference on Object-Oriented Programming (ECOOP 2026), LIPIcs, Vol. 372, pp. 8:1–8:27, 2026.

<https://doi.org/10.4230/LIPIcs.ECOOP.2026.8>

Related Conference 40th European Conference on Object-Oriented Programming (ECOOP 2026), June 29–July 3, 2026, Brussels, Belgium

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2026 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

The artifact allows verifying the following claims:

1. Cross-analyzer comparison of arithmetic overflow POs on the SV-COMP subset before and after alignment fixes (Table 1 of the paper, reproduced from bundled comparison JSONs and bundled analyzer outputs; the claimed reproduction uses the bundled data, while optional reruns of the analyzers are provided separately)
2. Examples of joint progress matrices illustrating optimistic combination (Table 3 of the paper)
3. Folder-level averages for RQ3 reduction comparison (Table 5 of the paper)



© Tom Goaldard, Karoliine Holter, Simmo Saan, Vesal Vojdani, and Raphaël Monat; licensed under Creative Commons License CC-BY 4.0

Dagstuhl Artifacts Series, Vol. 12, Issue 1, Artifact No. 6, pp. 6:1–6:15



Dagstuhl Artifacts Series
Schloss Dagstuhl – Leibniz-Zentrum für Informatik,
Dagstuhl Publishing, Germany



2 Content

The artifact package includes:

- Root-level entry-point scripts:
 - `quickstart-dashboard.sh` and `quickstart-all.sh`
 - `RQ1-run-dashboard-1a.sh`, `RQ1-run-dashboard-1b.sh`, and `RQ1-run-all.sh`
 - `RQ2-run-dashboard-3a.sh`, `RQ2-run-dashboard-3b.sh`, and `RQ2-run-all.sh`
 These are the main push-button workflows described in the appendix.
- `quickstart/`: a small end-to-end example.
 - `benchdefs/`: BenchExec definitions for the quickstart subset
 - `tool-outputs/`: bundled analyzer outputs for the dashboard-only quickstart
 - generated outputs go to `quickstart-results-dashboard/` and `quickstart-results-all/`
- `RQ1/`: the RQ1 benchmark setup.
 - `benchdefs/`: BenchExec definitions for the RQ1 subset
 - `tool-outputs-before/`: bundled original analyzer outputs for the pre-alignment Table 1a reproduction
 - `tool-outputs/`: bundled original analyzer outputs for the post-alignment Table 1b reproduction
 - `dashboard-outputs-before/`: bundled original comparison JSONs for the pre-alignment Table 1a reproduction
 - `dashboard-outputs/`: bundled original comparison JSONs for the post-alignment Table 1b reproduction
 - `scripts/generate-rq1-table-1.py`: post-processing for Table 1
 - `original-table/po_matrix.tex`: the paper’s original LaTeX table fragment
- `RQ2/`: the RQ2 benchmark setup.
 - `benchdefs/`: BenchExec definitions for the RQ2 subset
 - `tool-outputs/`: bundled original analyzer outputs
 - `dashboard-outputs/`: bundled original comparison JSONs
 - `scripts/generate-rq2-table-3a.py` and `scripts/generate-rq2-table-3b.py`: post-processing for Table 3a and Table 3b
- `RQ3/`: the RQ3 reduction and reporting pipeline.
 - `config.txt`: main configuration
 - filtering and reduction scripts
 - `reports/`: report-generation scripts
 - `tool-outputs/`: bundled reduction workspaces
- `shared-scripts/`: shared orchestration code.
 - `run-dashboard-comparisons.py`: runs the Open Verification Dashboard on two analyzer-output trees and produces one `*.comparison.json` file per task
- `open-verification-dashboard/`: the dashboard implementation itself.
- `sv-benchmarks-nooverflow/`: the benchmark programs used by the experiments. This is a subset of `no-overflow` property tasks derived from the original SV-COMP benchmark set at <https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks> from tag `svcomp26`.
- `tools/`: the analyzer binaries.
 - `goblint/`
 - `goblint-before.zip`: archived pre-alignment Goblint distribution containing the `goblint` binary, its runtime libraries and SV-COMP configuration files; it is not used by the claimed reproduction workflow, but is included so that Table 1a can also be reproduced by rerunning the tools in the same style as Table 1b in Appendix RQ1.3

- `mopsa-rq1/` and `mopsa-rq2/`
- `mopsa-before/`; the pre-alignment reference binary, likewise not used by the claimed reproduction workflow, but included for optional reruns of Table 1a

The reproductions in the Appendices RQ1 and RQ2 follow the same three-stage structure:

1. either rerun `Goblint` and `Mopsa` via the `BenchExec` setup above or reuse the bundled original analyzer outputs;
2. run `shared-scripts/run-dashboard-comparisons.py` to generate comparison JSON files;
3. run the RQ-specific aggregation or reporting script on those comparison JSONs.

We apply for the Available, Functional and Reusable badges. This artifact is publicly available on Zenodo, it provides steps to reproduce the experimental claims covered by this description, and the source code is provided, with ways to build it documented.

Available Artifact Claim

The available-artifact claim is based on the complete artifact package being archived on Zenodo, including benchmarks, bundled original experiment outputs, helper scripts, and the dashboard implementation needed to inspect and reproduce the covered reported results.

Functional Artifact Claim

The functional-artifact claim is based on the provided reproduction workflows in RQ1 subsections Appendices RQ1.1 to RQ1.3 (subsections 1.1–1.3) and RQ2 subsections Appendices RQ2.1 to RQ2.3 (subsections 2.1–2.3).

- The first subsection in each RQ provides the easiest reproduction mode: it starts from bundled comparison JSONs and regenerates the final paper tables directly.
- The second subsection in each RQ provides the recommended reproduction mode: it starts from bundled analyzer outputs, reruns the dashboard, and thereby reproduces both the comparison JSONs and the derived tables. This is the main functional claim of the artifact.
- The third subsection in each RQ provides examples for optional full rerun mode: it reruns the analyzers themselves with the included `BenchExec` configurations, but serves only as supplementary evidence where included.

Together, the bundled-data workflows reproduce Table 1a and Table 1b, Table 3a and Table 3b, and the corresponding comparison JSONs. The optional end-to-end reruns additionally show how to regenerate the post-alignment RQ1 results and the RQ2 Table 3b results from the benchmarks.

Reusable Software Claim

The reusable-software claim applies specifically to the `open-verification-dashboard/` component bundled in this artifact. Its source code is included, it can be rebuilt, and it can be reused to compare proof-obligation outputs from other analyzers beyond the particular experiments reported in the paper, as detailed in Appendix OVD. By contrast, the compared analyzers themselves (`Goblint` and `Mopsa`) are external tools used as inputs to the evaluation; their inclusion here is only for reproduction of the reported experiments, and we do not claim reusability for them as part of this artifact.

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS).

6:4 Comparing Transparent Static Analyzers with Open Verification Dashboard (Artifact)

Artifact DOI

The artifact is archived on Zenodo under DOI <https://doi.org/10.5281/zenodo.20139533>.

Running the Container

The artifact comes as a Docker container with Ubuntu 24.04. Start by loading it with:

```
docker load -i dashboard-artifact.tar
```

For the dashboard-only reproductions that reuse the bundled original analyzer outputs, including `quickstart-dashboard.sh` and Appendices RQ1.2 and RQ2.2, it is sufficient to run the container as:

```
docker run -it dashboard-artifact:latest
```

The BenchExec-based workflows require a privileged container. This applies to `quickstart-all.sh` and to Appendices RQ1.3 and RQ2.3. To run those workflows, start the container as:

```
docker run --privileged --cap-drop=all -it dashboard-artifact:latest
```

Without `-privileged`, only the dashboard-only reproductions can be run; the full reruns that execute Goblint and Mopsa require the privileged container command above.

Kick-the-Tires

The quickstart is available in two modes, mirroring the RQ1 and RQ2 workflows.

Dashboard-only mode (recommended)

For a lightweight check that reuses bundled analyzer outputs, start the container with:

```
docker run -it dashboard-artifact:latest
```

and then run:

```
./quickstart-dashboard.sh
```

This mode uses the bundled original analyzer outputs in `quickstart/tool-outputs/` and then runs:

1. `shared-scripts/run-dashboard-comparisons.py` on the bundled Goblint and Mopsa output trees,
2. `RQ1/scripts/generate-rq1-table-1.py` on the generated comparison JSONs.

The generated outputs are written under `quickstart-results-dashboard/`, in particular:

- comparison JSONs in `quickstart-results-dashboard/dashboard-outputs/`
- dashboard log in `quickstart-results-dashboard/dashboard-output.log`
- aggregated CSV in `quickstart-results-dashboard/table-1.csv`

Full rerun mode

To rerun the analyzers on the quickstart benchmark subset, start the container with:

```
docker run --privileged --cap-drop=all -it dashboard-artifact:latest
```

and then run:

```
./quickstart-all.sh
```

This mode follows the same three-stage workflow as the main experiments:

1. it runs Goblint and Mopsa via `quickstart/benchdefs/run.sh`,
2. it runs `shared-scripts/run-dashboard-comparisons.py` on the two resulting output trees,
3. it runs `RQ1/scripts/generate-rq1-table-1.py` on the generated comparison JSONs.

The generated outputs are written under `quickstart-results-all/`, in particular:

- analyzer outputs in `quickstart-results-all/goblint.*` and `quickstart-results-all/mopsa.*`
- comparison JSONs in `quickstart-results-all/dashboard-outputs/`
- dashboard log in `quickstart-results-all/dashboard-output.log`
- aggregated CSV in `quickstart-results-all/table-1.csv`

These modes are intended for reviewers to confirm that the bundled analyzers, dashboard, and orchestration scripts work together before attempting the larger RQ1–RQ3 reproduction workflows.

Expected result

Both modes should produce the following matrix:

Mopsa/Goblint	Unknown	Invalid	Valid	Unreach
Unknown	0	0	0	2
Invalid	0	0	0	0
Valid	2	0	12	2
Unreach	0	0	0	0

4 Tested platforms

- OS: Ubuntu 24.04.3 LTS
- Hardware used for the bundled original RQ1 and RQ2 runs: 16-core AMD EPYC host with 33.7 GB RAM (as recorded in the BenchExec result summaries in `RQ1/tool-outputs/*.txt` and `RQ2/tool-outputs/*.txt`).
- Per-task resource limits: RQ1 uses 1 CPU core, 1 GB memory, and a 90 s time limit per task; RQ2 uses 1 CPU core, 2 GB memory, and a 450 s time limit per task. The runs reported in the paper for the rerun workflows (Appendices RQ1.3 and RQ2.3) were executed with up to 15 parallel BenchExec jobs (`GOBLINT_PARALLEL=15` in the original run scripts). The bundled `benchdefs/run.sh` scripts in the artifact and Docker image default to `GOBLINT_PARALLEL=6` to fit smaller machines; this only affects runtime, not the expected outputs, for the RQ1/RQ2 rerun modes. If the message `Memory banks [0] do not have enough memory for all runs, only X bytes available. Please reduce the number of threads.` appears, reduce the value of `GOBLINT_PARALLEL` in the corresponding `benchdefs/run.sh` script for the RQ being rerun.

6:6 Comparing Transparent Static Analyzers with Open Verification Dashboard (Artifact)

- Recommended resources for full evaluation: 16 CPU cores, 32 GB RAM, and approximately 20 GB of free disk space for analyzer outputs and generated comparison files.
- Smaller modes: the dashboard-only reproductions and the quickstart workflow require less compute, but still assume a Linux environment with Python 3, Dune, and the bundled dashboard sources available.
- Known compatibility notes: the workflows were prepared for Linux paths and BenchExec-based execution; they have not been validated on macOS or Windows.

5 License

The Zenodo artifact record is published under the Creative Commons Attribution 4.0 International (CC BY 4.0) license. Internally, however, this artifact is a mixed-license aggregate, and the original licenses of all bundled third-party components continue to apply to those components. In particular, the bundled Goblint distribution is under the MIT License, the bundled Mopsa distributions are under the GNU LGPL v3.0, the bundled Open Verification Dashboard (OVD) sources are under the MIT License, and the bundled benchmark programs inherit the original per-file licenses of SV-Benchmarks. The artifact-specific scripts and glue code provided by the authors are released under the MIT License. License texts and notices for bundled components are included in the artifact where applicable.

6 MD5 sum of the artifact

c14f85bbb9dc79f6cc0d5ca8f468ac62

7 Size of the artifact

12.0 GB

RQ1 Cross-analyzer comparison of arithmetic overflow POs on the SV-COMP subset before and after alignment fixes (Table 1 of the paper)

This section explains how to reproduce RQ1 (Complementarity), that is, the cross-analyzer comparison of arithmetic overflow POs on the SV-COMP subset reported in Table 1 of the paper. The bundled workflow reproduces both the pre-alignment Table 1a comparison and the post-alignment Table 1b comparison, showing how often Goblint and Mopsa report checks for the same proof obligations and assign the same or different statuses.

This experiment can be reproduced in two modes: a recommended dashboard-only mode that reuses the bundled original analyzer outputs to reproduce the exact paper data, and a full rerun mode that starts from the benchmark programs and reruns both analyzers. The dashboard-only mode reproduces both Table 1a and Table 1b; the full rerun mode reproduces the post-alignment Table 1b only. The pre-alignment “before” tool binaries are included only so that Table 1a can also be rerun in the same style as Appendix RQ1.3 if desired, but this rerun is not the claim we want to reproduce with the artifact. The data output of the full rerun mode might not exactly match the data provided in Table 1 in the paper due to the impact of the resource limits set for the tool runs.

RQ1.1 Minimal reproduction of tables

This mode starts directly from the bundled comparison JSONs and only reruns the table aggregation script.

Reproducing Table 1a

To regenerate the bundled pre-alignment Table 1a CSV, execute:

```
mkdir -p RQ1-results-dashboard-1a
python3 RQ1/scripts/generate-rq1-table-1.py \
  --out RQ1-results-dashboard-1a/table-1a.csv \
  RQ1/dashboard-outputs-before
```

Reproducing Table 1b

To regenerate the bundled post-alignment Table 1b CSV, execute:

```
mkdir -p RQ1-results-dashboard-1b
python3 RQ1/scripts/generate-rq1-table-1.py \
  --out RQ1-results-dashboard-1b/table-1b.csv \
  RQ1/dashboard-outputs
```

Expected runtime

Each command completes in less than 1 minute.

RQ1.2 Reproduction of dashboard runs (recommended)

This reproduction mode starts from the original analyzer outputs bundled with the artifact and therefore avoids rerunning the analyzers. It offers two dashboard-only entry points, one for Table 1a and one for Table 1b.

Reproducing Table 1a

To reproduce the Table 1a example from the paper, execute:

```
./RQ1-run-dashboard-1a.sh
```

Original input data

This mode uses the bundled original analyzer outputs in `RQ1/tool-outputs-before/`.

Script description

- It directly uses the bundled original analyzer outputs in `RQ1/tool-outputs-before/`.
- It runs `shared-scripts/run-dashboard-comparisons.py` on them and produces the comparison JSON files in `RQ1-results-dashboard-1a/dashboard-outputs`.
- It runs `RQ1/scripts/generate-rq1-table-1.py` on those comparison JSONs and writes the aggregated Table 1a matrix to `RQ1-results-dashboard-1a/table-1a.csv`.

6:8 Comparing Transparent Static Analyzers with Open Verification Dashboard (Artifact)

Expected result

For the bundled original data, this procedure should reproduce the original pre-alignment Table 1a matrix exactly in `RQ1-results-dashboard-1a/table-1a.csv`. Aggregating the generated comparison JSONs yields the following matrix with total 48 985 aligned proof obligations:

	Unknown	Invalid	Valid	Unreach
Unknown	8122	0	2268	159
Invalid	1	0	0	6
Valid	222	0	11700	23754
Unreach	949	0	1804	0

Expected runtime

1 hour.

Reproducing Table 1b

To reproduce the Table 1b example from the paper, execute:

```
./RQ1-run-dashboard-1b.sh
```

Original input data

This mode uses the bundled original analyzer outputs in `RQ1/tool-outputs`. The original comparison JSONs are also included in `RQ1/dashboard-outputs`, but are not used by the script and should be identical to the ones produced. These are the raw data used for the paper.

Script description

- It directly uses the bundled original analyzer outputs in `RQ1/tool-outputs`.
- It runs `shared-scripts/run-dashboard-comparisons.py` on them and produces the comparison JSON files in `RQ1-results-dashboard-1b/dashboard-outputs`.
- It runs `RQ1/scripts/generate-rq1-table-1.py` on those comparison JSONs and writes the aggregated Table 1b matrix to `RQ1-results-dashboard-1b/table-1b.csv`.

Expected result

For the bundled original data, the result should reproduce the original post-alignment Table 1b matrix exactly. The freshly generated comparison JSONs in `RQ1-results-dashboard-1b/dashboard-outputs/` should match the bundled reference copies in `RQ1/dashboard-outputs/`. Aggregating these comparison JSONs yields the following post-alignment matrix with total 46 695 aligned proof obligations:

	Unknown	Invalid	Valid	Unreach
Unknown	8123	0	2255	146
Invalid	1	0	0	0
Valid	222	0	32226	652
Unreach	944	0	2126	0

This corresponds to the bundled post-alignment part of Table 1 in the paper.

Expected runtime

2 hours.

RQ1.3 Reproduction by rerunning the tools

This reproduction mode starts from the benchmark programs and reruns both analyzers. It is also intended as a push-button execution. To run this experiment, execute:

```
RQ1-run-all.sh
```

Disclaimer. This mode may produce results that are not exactly identical to the numbers reported in the paper. The reason is that the tool runs for some tasks are close to the resource limits, so across different executions one analyzer may timeout or succeed on a borderline case where the original run produced the opposite outcome.

Script description

1. It reruns Goblint and Mopsa on the SV-COMP no-overflow subset using the shared BenchExec setup described in the Content section, producing one output tree per analyzer.
2. It runs `shared-scripts/run-dashboard-comparisons.py` on those two output trees, producing one `*.comparison.json` file per task in `RQ1-results-all/dashboard-outputs/`.
3. It runs `RQ1/scripts/generate-rq1-table-1.py` on the comparison JSONs. This script sums the per-task `joint_progress_matrix` values and outputs the final aggregated post-alignment matrix for Table 1b both on stdout and as a CSV file in `RQ1-results-all/table-1.csv`.

Expected result

The expected output is a set of comparison JSONs for the SV-COMP no-overflow tasks and an aggregated 4×4 cross-analyzer matrix for Mopsa (rows) versus Goblint (columns), using the status order `Unknown`, `Invalid`, `Valid`, `Unreach`. Depending on marginal timeouts, the exact numbers may differ slightly from the paper. The results are saved in folder `RQ1-results-all`.

Expected runtime

5 hours.

RQ2 Examples of joint progress matrices illustrating optimistic combination (Table 3 of the paper)

This section explains how to reproduce RQ2 (Optimistic combination), that is, the examples of joint progress matrices from Table 3 of the paper. The Table 3a example can be reproduced from the RQ1 comparison dataset via a separate dashboard-only script. The bundled workflow also reproduces the comparison JSONs and the derived article-facing summary for the coreutils examples, including the Table 3b matrix for `uniq_3args_ok`.

This experiment can be reproduced in two modes: a recommended dashboard-only mode that reuses the bundled original analyzer outputs to reproduce the exact paper data, and a full rerun mode that starts from the benchmark programs and reruns both analyzers. In contrast to RQ1, the RQ2 mode that reruns the analyzers (appendix RQ2.3) also reproduces Table 3b exactly: the benchmark set is smaller, and none of these runs are close to the timeout limit.

RQ2.1 Minimal reproduction of tables

This mode starts directly from the bundled comparison JSONs and only reruns the reporting script.

Reproducing Table 3a

To regenerate the bundled Table 3a CSV, execute:

```
mkdir -p RQ2-results-dashboard-3a
python3 RQ2/scripts/generate-rq2-table-3a.py \
  --out RQ2-results-dashboard-3a/table-3a.csv \
  RQ1/dashboard-outputs
```

Reproducing Table 3b

To regenerate the bundled Table 3b CSV, execute:

```
mkdir -p RQ2-results-dashboard-3b
python3 RQ2/scripts/generate-rq2-table-3b.py \
  --out RQ2-results-dashboard-3b/table-3b.csv \
  RQ2/dashboard-outputs
```

Expected runtime

Each command completes in less than 1 minute.

RQ2.2 Reproduction of dashboard runs (recommended)

This reproduction mode starts from the original analyzer outputs bundled with the artifact and therefore avoids rerunning the analyzers. It offers two dashboard-only entry points, one for Table 3a and one for Table 3b.

Reproducing Table 3a

To reproduce the Table 3a example from Section 5.2, execute:

```
./RQ2-run-dashboard-3a.sh
```

Original input data

This mode uses the RQ1 comparison JSONs in `RQ1-results-dashboard-1b/dashboard-outputs/`. If those do not exist yet, the same script can regenerate them first from the bundled original RQ1 tool outputs in `RQ1/tool-outputs/` by running:

```
./RQ2-run-dashboard-3a.sh --rerun-rq1-dashboard
```

Script description

This script either reuses the dashboard output generated in RQ1 or, with `--rerun-rq1-dashboard`:

- first reruns the RQ1 dashboard-comparison step on the bundled RQ1 tool outputs;
- then runs `RQ2/scripts/generate-rq2-table-3a.py` and writes the Table 3a joint progress matrix for `bresenham-11_valuebound1` to `RQ2-results-dashboard-3a/table-3a.csv`.

Expected result

For the bundled original data, this procedure should reproduce the Table 3a matrix exactly in `RQ2-results-dashboard-3a/table-3a.csv`:

	Unknown	Invalid	Valid	Unreach
Unknown	2	0	3	0
Invalid	0	0	0	0
Valid	1	0	5	0
Unreach	0	0	0	0

Expected runtime

Less than 1 minute if `RQ1-results-dashboard-1b/dashboard-outputs/` already exists; otherwise 2 hours with `-rerun-rq1-dashboard`.

Reproducing Table 3b

To reproduce the Table 3b example from Section 5.2, execute:

```
./RQ2-run-dashboard-3b.sh
```

Original input data

This mode uses the bundled original analyzer outputs in `RQ2/tool-outputs`. The original comparison JSONs are also included in `RQ2/dashboard-outputs`, but are not used by the script and should be identical to the ones produced. These are the raw data used for the paper.

Script description

- It directly uses the bundled original analyzer outputs in `RQ2/tool-outputs`.
- It runs `shared-scripts/run-dashboard-comparisons.py` on them and produces the comparison JSON files in `RQ2-results-dashboard-3b/dashboard-outputs`.
- It runs `RQ2/scripts/generate-rq2-table-3b.py` on those comparison JSONs, prints the article-facing summary for the matching coreutils tasks, and writes the Table 3b joint progress matrix for `uniq_3args_ok` to `RQ2-results-dashboard-3b/table-3b.csv`.

Expected result

For the bundled original data, the freshly generated comparison JSONs in `RQ2-results-dashboard-3b/dashboard-outputs/` should match the bundled reference copies in `RQ2/dashboard-outputs/`. Aggregating these comparison JSONs should reproduce the original Table 3b joint progress matrix exactly in `RQ2-results-dashboard-3b/table-3b.csv`:

	Unknown	Invalid	Valid	Unreach
Unknown	16	0	3	0
Invalid	0	0	0	0
Valid	8	0	46	935
Unreach	2	0	14	0

The same script also prints to the terminal an article-facing summary for the three matching tasks `cut_3args_ok`, `env_3args_ok`, and `uniq_3args_ok`. Among them, only `uniq_3args_ok` is written to CSV because it is the Table 3b example.

6:12 Comparing Transparent Static Analyzers with Open Verification Dashboard (Artifact)

Expected runtime

5 minutes.

RQ2.3 Reproduction by rerunning the tools

This reproduction mode starts from the benchmark programs and reruns both analyzers. It is also intended as a push-button execution; to run it, execute:

```
./RQ2-run-all.sh
```

Script description

1. It reruns Goblint and Mopsa on the `C.no-overflow.SoftwareSystems-coreutils` subset using the shared BenchExec setup described in the Content section.
2. It runs `shared-scripts/run-dashboard-comparisons.py` on the two resulting analyzer-output trees and produces one `*.comparison.json` file per task in `RQ2-results-all/dashboard-outputs`.
3. It runs `RQ2/scripts/generate-rq2-table-3b.py` on the comparison JSONs and produces a summary for the three coreutils programs discussed in Section 5.2, phrased using the same quantities as the paper: the number of overflow POs reported by both tools, the number verified by Mopsa, the number verified by Goblint, and the number verified by their optimistic combination. In addition, it writes the detailed joint progress matrix for `uniq_3args_ok` to `RQ2-results-all/table-3b.csv`; this is the Table 3b example reproduced in this appendix. In the original data bundled with the artifact, this procedure yields three matching tasks: `cut_3args_ok`, `env_3args_ok`, and `uniq_3args_ok`.

Expected result

The expected output is a set of comparison JSONs for the selected coreutils tasks and a regenerated Table 3b joint progress matrix in `RQ2-results-all/table-3b.csv`. This rerun mode should reproduce the Table 3b matrix from the paper exactly, with the following values:

	Unknown	Invalid	Valid	Unreach
Unknown	16	0	3	0
Invalid	0	0	0	0
Valid	8	0	46	935
Unreach	2	0	14	0

The same script also prints to the terminal an article-facing summary for the three matching tasks `cut_3args_ok`, `env_3args_ok`, and `uniq_3args_ok`. Among them, only `uniq_3args_ok` is written to CSV because it is the Table 3b example.

Expected runtime

40 minutes.

RQ3 Folder-level averages for RQ3 reduction comparison (Table 5 of the paper)

This section explains how to reproduce RQ3, that is, the folder-level averages for the reduction comparison reported in Table 5 of the paper. The artifact contains both the RQ3 pipeline and a precomputed reduction-result tree. Two entry points are provided: regenerating the reports from the checked-in manifests, and rerunning the full pipeline from the benchmark inputs. See RQ3/README.md for the supported commands and argument forms.

Original input data

The artifact includes the benchmark tree under RQ3/benchmarks/ and a checked-in result tree under RQ3/tool-outputs/. As an optimization, benchmarks were selected whose verdicts differed on the official SV-COMP results. In the following workflow, these are filtered to make sure the verdicts differ in the same way when running the current configurations. Tool paths, timeouts, and output locations are pinned in RQ3/config.txt; REDUCER_JOBS defaults to 1.

Reproducing Table 5

To regenerate the CSV underlying Table 5, execute:

```
cd RQ3
python3 reports/paper_csv.py
```

This command reads the checked-in reduction manifests from RQ3/tool-outputs/ and writes RQ3/RQ3-results/paper.csv.

Additional outputs

To materialize the per-task table and the HTML view over the same data, execute:

```
cd RQ3
python3 reports/summary.py
python3 reports/html.py
```

These commands write RQ3/RQ3-results/summary.csv and RQ3/RQ3-results/html/index.html, respectively.

Full rerun

To rerun the complete RQ3 pipeline from the benchmark inputs, execute:

```
cd RQ3
./run_experiment.sh
```

Script description

The full rerun executes the following steps in order:

1. `python3 filter.py`
2. `python3 reduce_verdict.py -all`
3. `python3 reduce_dashboard.py -all`
4. `python3 reports/summary.py`
5. `python3 reports/html.py`
6. `python3 reports/paper_csv.py`

6:14 Comparing Transparent Static Analyzers with Open Verification Dashboard (Artifact)

The RQ3 scripts also support running individual task files instead of the full benchmark set. For the supported one-file commands and their expected inputs, refer to RQ3/README.md.

Filtering rule

Filtering writes `tool-outputs/filtered/` and `tool-outputs/filter/filter.csv`. It keeps Goblint tasks only when `(goblint,mopsa) = (true,false)` and `SafetyW2 > 0`, and Mopsa tasks only when `(goblint,mopsa) = (false,true)` and `SafetyW1 > 0`.

Expected result

For the checked-in result tree, report regeneration writes:

- summary CSV: `RQ3/RQ3-results/summary.csv`
- HTML report: `RQ3/RQ3-results/html/index.html`
- paper CSV: `RQ3/RQ3-results/paper.csv`

The paper CSV is the report corresponding to the folder-level averages used for Table 5 of the paper. Its expected contents are:

```
folder, files, orig_lines, orig_sw2, task_lines, task_ok_pct, task_sec, dash_lines, dash_sec
float-benchs, 6, 196.17, 1.00, 8.67, 100.00, 184.50, 8.67, 233.17
nla-digbench, 4, 42.25, 5.75, 5.00, 0.00, 104.25, 6.00, 123.00
termination-crafted, 11, 26.18, 1.55, 3.73, 18.18, 119.73, 5.91, 156.45
termination-crafted-lit, 11, 27.18, 1.73, 5.27, 45.45, 92.00, 7.00, 102.73
termination-numeric, 3, 30.00, 1.00, 4.00, 0.00, 124.00, 5.67, 190.00
termination-restricted-15, 13, 21.62, 1.00, 4.31, 7.69, 83.62, 8.77, 98.62
```

Expected runtime

Less than 1 minute for report regeneration from the checked-in manifests. A full rerun is dominated by the two reduction phases `reduce_verdict.py -all` and `reduce_dashboard.py -all` and may take several days.

OVD Rebuilding and Reusing the Open Verification Dashboard

This section elaborates the reusable-software claim stated in the Content section. The reusable component is the bundled `open-verification-dashboard/` project, whose source code is included in the artifact. This project can be rebuilt and then reused to compare proof-obligation outputs from analyzers other than the specific Goblint/Mopsa experiments described in this paper.

Source location

The `dashboard` sources are bundled under `open-verification-dashboard/`. The source code is also available as open source in the GitHub repository <https://github.com/sws-lab/open-verification-dashboard>. The project also includes its own `README.md`, `dune-project`, `dashboard.opam`, OCaml sources under `lib/` and `bin/`, and tests under `test/`.

Build from source

As specified also in `open-verification-dashboard/README.md`, from the dashboard root first install dependencies and then build with Dune:

```
cd open-verification-dashboard
opam install . --deps-only
dune build
```

After that, the CLI can be run with:

```
dune exec dashboard -- --exclude-not-found true mopsa.json goblint.json
```

Reuse scenario

The dashboard CLI compares two proof-obligation JSON files and produces a structured comparison report. The comparison scripts used elsewhere in this artifact call this CLI in batch mode, but the underlying dashboard itself is not specific to RQ1–RQ3. As long as other analyzers can produce proof obligation JSONs in the expected format described in the bundled dashboard README, the same dashboard can be rebuilt and reused to compare them. This is the basis for the reusable-software claim. By contrast, the bundled Goblint and Mopsa binaries are included only to reproduce the experiments in this artifact and are not part of the reusability claim.