Artifact Evaluation for Paper “Software Model Checking via Summary-Guided Search”

1 Introduction

Context of this artifact. Our paper introduces a new software-model-checking algorithm called GPS (Guided by Path Summaries). GPS treats the task of model checking a program as a directed search of the program states, guided by over-approximate summaries about program paths produced by a compositional static analysis. Summary information is used both to prune away infeasible paths and to drive test generation to reach new, unexplored program states. This artifact is a Docker image that accompanies our paper, and includes:

- An implementation of the GPS algorithm (along with its ablated variants studied in the paper) with compositional recurrence analysis as the underlying static analysis, using the Duet program-analysis framework; the implementation is written in OCaml.
- State-of-the-art software model checkers (CPAChecker, VeriAbs, Symbiotic) that we opted to compare with GPS in the paper.
- The benchexec tool used to run benchmarks and report statistics[1].
- A suite of benchmarks used in our paper, including both benchmarks from the software-verification competition (SV-COMP), as well as benchmarks introduced in our paper, based on prior literature and synthetic examples.

In the artifact-evaluation process, you will attempt to verify the results presented in the evaluation section of our paper: this means re-running GPS, its ablated variants, and other software model checkers on the benchmark sets used in our paper, and verifying that GPS indeed has better performance compared to the other tools.

Structure of this document.

- Section 2 covers the scope of reproduction for artifact evaluation.
- Section 3 covers the hardware / system requirements required to perform the artifact evaluation.
- Section 4 includes a getting-started guide designed to familiarize you with running each tool and the VM environment.
- For detailed instructions about reproducing experiment results, refer to Section 5.
- Section 6 is a “peeking-under-the-hood” guide that covers how to run and use our software model checker, and explains details about other content included in this artifact.
- Section 7 is a FAQ section that lists some common questions you might encounter.
- The last section states the license of this artifact.

Note about time consumption. Our experiments involve running many software model-checking tools (and variants of the GPS algorithm) on a large suite of programs. As a

[1] For more on BenchExec, please visit https://github.com/sosy-lab/benchexec

result, the experiments may take several days to complete end-to-end. We provide instructions for running less time-consuming versions of our experiment (where the time-out threshold for each tool is reduced; see Sections 4 and 5 for details) should you be under time pressure, but these less time-consuming benchmarks may hinder full reproducibility.

2 Scope of Evaluation

The scope of this artifact evaluation is:

- Table 1 on page 19 of the attached manuscript (comparison between GPS, GPSLite, GPS-nogas, GPS-nogas-nosum, Impact, and CRA);

- Table 2 on page 19 of the attached manuscript (comparison between GPS, VeriAbs, Symbiotic, CPAChecker (Portfolio)).

Figures 10 on page 20, 11 on page 27 (in the appendix), figures 12 and 13 on page 28 -29 (in the appendix) are figures derived from the experiments performed for producing Tables 1 and 2, so they are not included in the scope of this artifact evaluation.

Due to hardware performance differences (and because our experiments are performed in a bare-metal environment, whereas this artifact is packaged as a Docker container), you might produce different statistics for Tables 1 and 2. Hence, your key goal is to verify that GPS (with both gas-instrumentation and CRA-generated summaries) has the best performance (in terms of number of benchmarks solved) overall, relative to the other tools as well as GPS variants:

- In Table 1, GPS verifies the most number of benchmarks in each suite (Column 3), with GPS-nogas being in second place and GPSLite being in third place, for the total number of benchmarks solved. The other variants (as well as the CPAChecker implementation of Impact) should trail the top performers both in terms of number of benchmarks solved and total running time.

- In Table 2, GPS verifies the most number of benchmarks in each suite (Column 3) with VeriAbs being in the second place (in terms of total benchmarks solved), followed by CPAChecker (Portfolio) and Symbiotic.

3 System Requirements

OS requirements. Due to requirements of the benchexec tool, the Docker container must be run on a Linux host machine with x86-64 architecture and Cgroups v2 support (https://kubernetes.io/docs/concepts/architecture/cgroups/). To test whether your machine supports Cgroups v2, you can type in the following command:

```
sudo stat -fc %T /sys/fs/cgroup/
```

Cgroups v2 is enabled if the output is *not* “`tmpfs`.” Most up-to-date Linux distributions (e.g. Ubuntu 24.04) should support Cgroups v2 by default.

Hardware requirements. For reference, the (bare-metal) benchmark environment used in our paper is a Dell XPS 15 laptop with Intel i7-13700H CPU, 6 high-performance cores, and 16 GB of RAM. It is recommended that your hardware is at least as powerful as the bare-metal environment used in our paper.

4 Getting Started Guide

Step 1: Clone image from Docker.

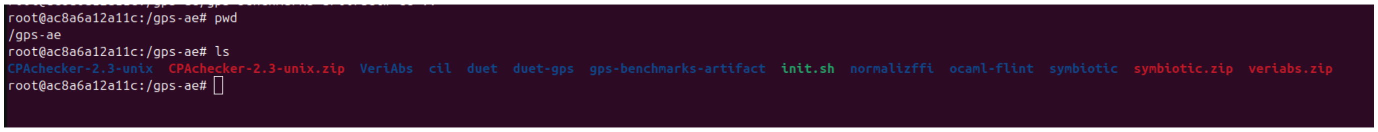
```
docker pull ruijiefang/gps-oopsla25-ae
```

Step 2: Run the container using the following command (may require root privileges):

```
docker run --privileged --cap-drop=all -it ruijiefang/gps-oopsla25-ae
```

Step 3: Verify the docker container image.

Step 3a) Execute `` `cd /gps-ae` `` to go to the `/gps-ae/` folder in the command line of the container. The directory should contain the following files:



```
root@ac8a6a12a11c:/gps-ae# pwd
/gps-ae
root@ac8a6a12a11c:/gps-ae# ls
CPAchecker-2.3-unix  CPAchecker-2.3-unix.zip  VeriAbs  cil  duet  duet-gps  gps-benchmarks-artifact  init.sh  normalizffi  ocaml-flint  symbiotic  symbiotic.zip  veriabs.zip
root@ac8a6a12a11c:/gps-ae# 
```

Step 3b) Execute the following command:

```
  python3 -m benchexec.check_cgroups
```

No output is expected from this command. If the script outputs an error message, there is an issue with the Cgroups v2 configuration of your host operating system. We recommend checking the documentation for benchexec[2] for additional information, and contacting us if you could not resolve this problem.

An alternative solution is to set up a virtual machine running Ubuntu 24.04 (or another environment with Cgroups v2 support built-in) and run our Docker image inside the VM. We do not recommend this approach as it induces a lot of overhead and may affect the reproduction results.

[2] https://github.com/sosy-lab/benchexec/blob/main/doc/benchexec-in-container.md

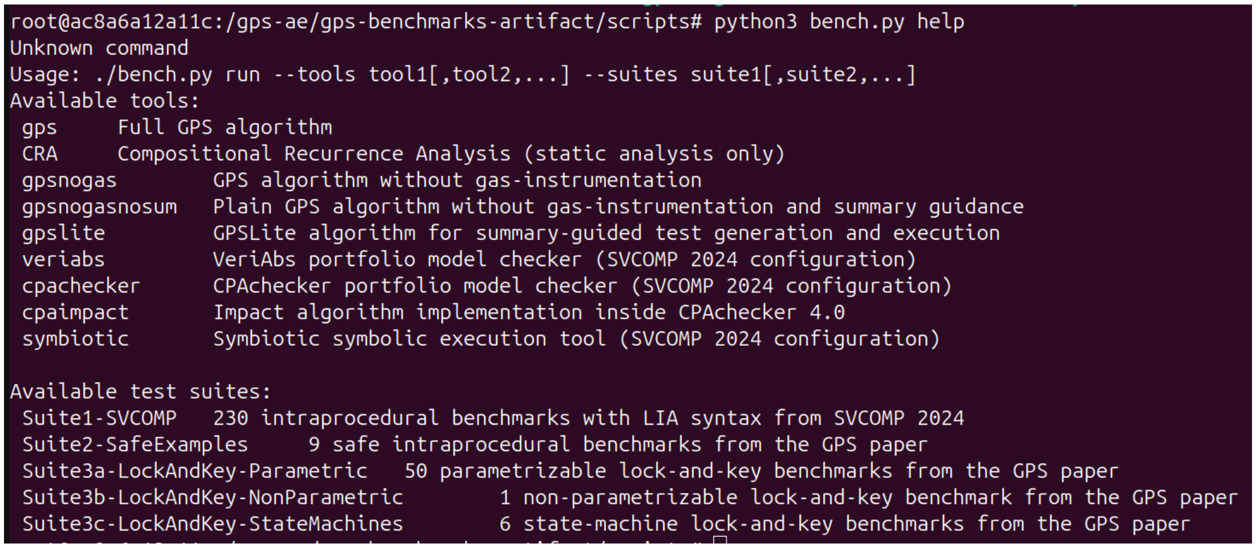
Step 4: Initialize the environment by executing the `./init.sh` script in the /gps-ae/ folder.
The last line of the output should be

```
      "Complete. Please go to ./gps-benchmarks-artifact/scripts/ to
start running benchmarks."
```

After these steps, you should be able to proceed to the next step, where you will start evaluating our artifact.

Step 5: High-level overview of benchmark scripts.
Go to the `/gps-ae/gps-benchmarks-artifact/scripts/` folder. The folder contains the following script:

- `bench.py`, a benchexec wrapper script used to run each tool in this evaluation on benchmark suites used in our paper. Executing "`python3 benchexec.py help`" should yield the following output:

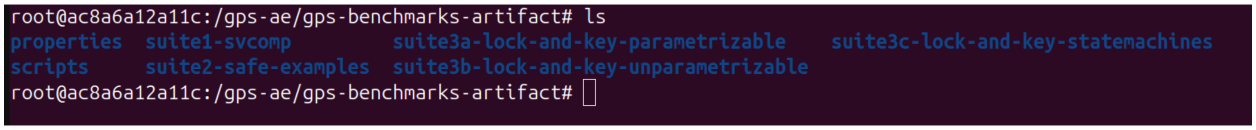


```
root@ac8a6a12a11c:/gps-ae/gps-benchmarks-artifact/scripts# python3 bench.py help
Unknown command
Usage: ./bench.py run --tools tool1[,tool2,...] --suites suite1[,suite2,...]
Available tools:
 gps     Full GPS algorithm
 CRA     Compositional Recurrence Analysis (static analysis only)
 gpsnogas        GPS algorithm without gas-instrumentation
 gpsnogasnosum   Plain GPS algorithm without gas-instrumentation and summary guidance
 gpslite         GPSLite algorithm for summary-guided test generation and execution
 veriabs         VeriAbs portfolio model checker (SVCOMP 2024 configuration)
 cpachecker      CPAchecker portfolio model checker (SVCOMP 2024 configuration)
 cpaimpact       Impact algorithm implementation inside CPAchecker 4.0
 symbiotic       Symbiotic symbolic execution tool (SVCOMP 2024 configuration)

Available test suites:
 Suite1-SVCOMP   230 intraprocedural benchmarks with LIA syntax from SVCOMP 2024
 Suite2-SafeExamples     9 safe intraprocedural benchmarks from the GPS paper
 Suite3a-LockAndKey-Parametric   50 parametrizable lock-and-key benchmarks from the GPS paper
 Suite3b-LockAndKey-NonParametric        1 non-parametrizable lock-and-key benchmark from the GPS paper
 Suite3c-LockAndKey-StateMachines        6 state-machine lock-and-key benchmarks from the GPS paper
```

In the above picture, the available tools cover all tools involved in our paper's benchmarks section. In particular, "gps" and "gps*" tools are the GPS algorithm and its ablated variants. The available test suites correspond to test suites described in page 18-19 of our paper:

- **`Suite1-SVCOMP`** stands for the suite "Suite #1: 230 intraprocedural benchmarks from SV-COMP" discussed on line 845 of page 18 in our paper.
- Suite2-SafeExamples stands for the suite "Suite #2: 9 Safety-verification examples from this paper" discussed on line 847 of our paper.
- **`Suite3a-LockAndKey-Parametric`** stands for item (2) of "Suite #3: 57 Lock&Key-Style Benchmarks" on line 859 of our paper
- **`Suite3b-LockAndKey-NonParametric`** stands for the only non-parametrizable L&K example (discussed on lines 854 and 859 of our paper)

- **`Suite3c-LockAndKey-StateMachines`** stands for the string-processing state machine examples covered in item (3) of “Suite #3” discussed on line 863 of our paper.
- The C program source for each benchmark suite may be found in the `suite1-svcomp`, `suite2-safe-examples`, and `suite3*` folders inside the `/gps-ae/gps-benchmarks-artifact` directory, correspondingly:



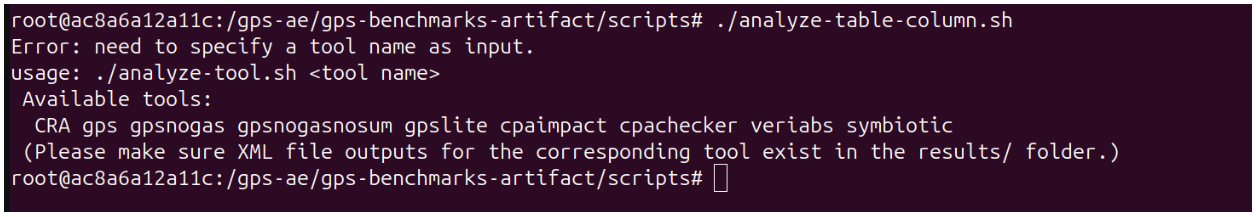
```
root@ac8a6a12a11c:/gps-ae/gps-benchmarks-artifact# ls
properties  suite1-svcomp         suite3a-lock-and-key-parametrizable    suite3c-lock-and-key-statemachines
scripts     suite2-safe-examples  suite3b-lock-and-key-unparametrizable
root@ac8a6a12a11c:/gps-ae/gps-benchmarks-artifact# 
```

To verify each suite of benchmarks contain the correct number of programs (as listed in the evaluations section on page 18 of our paper), you could execute the following commands:

1. **`tree /gps-ae/gps-benchmarks-artifact/suite1-svcomp | grep "\.yml" | wc -l`** `# should return 230`
2. **`tree /gps-ae/gps-benchmarks-artifact/suite2-safe-examples | grep "\.yml" | wc -l`** `# should return 9`
3. **`tree /gps-ae/gps-benchmarks-artifact/suite3a-lock-and-key-parametrizable | grep "\.yml" | wc -l`** `# should return 50`
4. **`tree /gps-ae/gps-benchmarks-artifact/suite3b-lock-and-key-unparametrizable | grep "\.yml" | wc -l`** `# should return 1`
5. **`tree /gps-ae/gps-benchmarks-artifact/suite3c-lock-and-key-statemachines | grep "\.yml" | wc -l`** `# should return 6`

In addition to the bench.py script, the `/gps-ae/gps-benchmarks-artifact/scripts/` folder also contains the following scripts, which you will use to actually reproduce our experiment results in Tables 1 & 2 (we explain how to use the in the section below):

- `table1-*.sh` scripts for reproducing each column of Table 1 in our paper. These scripts simply call bench.py with a given tool and benchmark suite supplied as arguments.
- `table2-*.sh` scripts for reproducing each column of Table 2 in our paper, similar to the above.
- Each of the `table1*.sh` and `table2*.sh` scripts will produce benchmark results (including detailed running time, output, etc. of a tool for each benchmarked program) in XML format, in the `results/` sub-folder. You can then invoke `analyze-table-column.sh <tool name>` to reproduce statistics for each column in our paper (commands given in sections below). For now, executing “`./analyze-table-column.sh`” should give the following output:



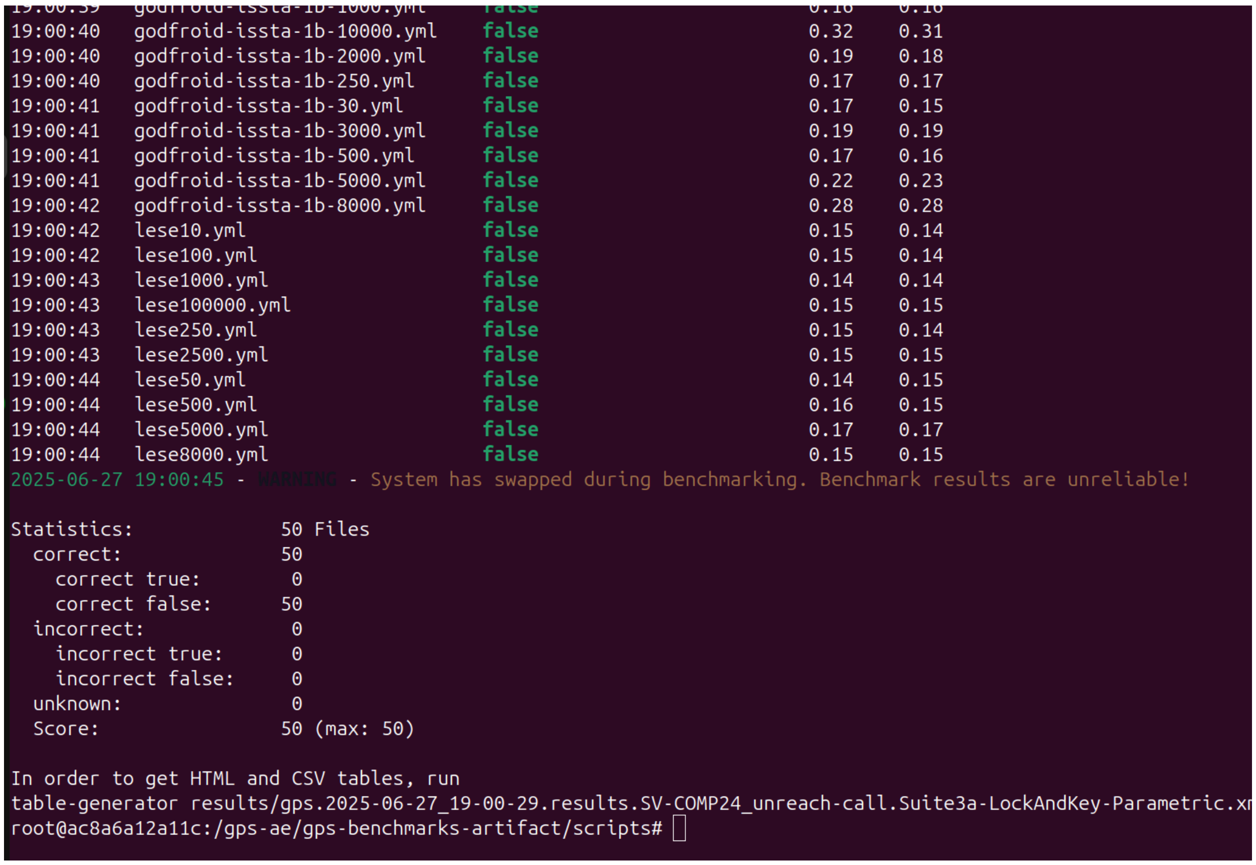
```
root@ac8a6a12a11c:/gps-ae/gps-benchmarks-artifact/scripts# ./analyze-table-column.sh
Error: need to specify a tool name as input.
usage: ./analyze-tool.sh <tool name>
 Available tools:
  CRA gps gpsnogas gpsnogasnosum gpslite cpaimpact cpachecker veriabs symbiotic
 (Please make sure XML file outputs for the corresponding tool exist in the results/ folder.)
root@ac8a6a12a11c:/gps-ae/gps-benchmarks-artifact/scripts# 
```

Step 6: Trial run.

Now, let's do a trial run of GPS using the `bench.py` script. The command below will execute the GPS algorithm with a 60-second timeout on `Suite3a-LockAndKey-Parametric` benchmark suite, which contains 50 parameterizable Lock & Key-style benchmarks (discussed on line 851, page 18 of our paper):

```
python3 bench.py run --timeout 60 --tools gps --suites
Suite3a-LockAndKey-Parametric
```

This command should finish reasonably quickly (within 2-3 minutes). The last few lines of the output should look like the following:

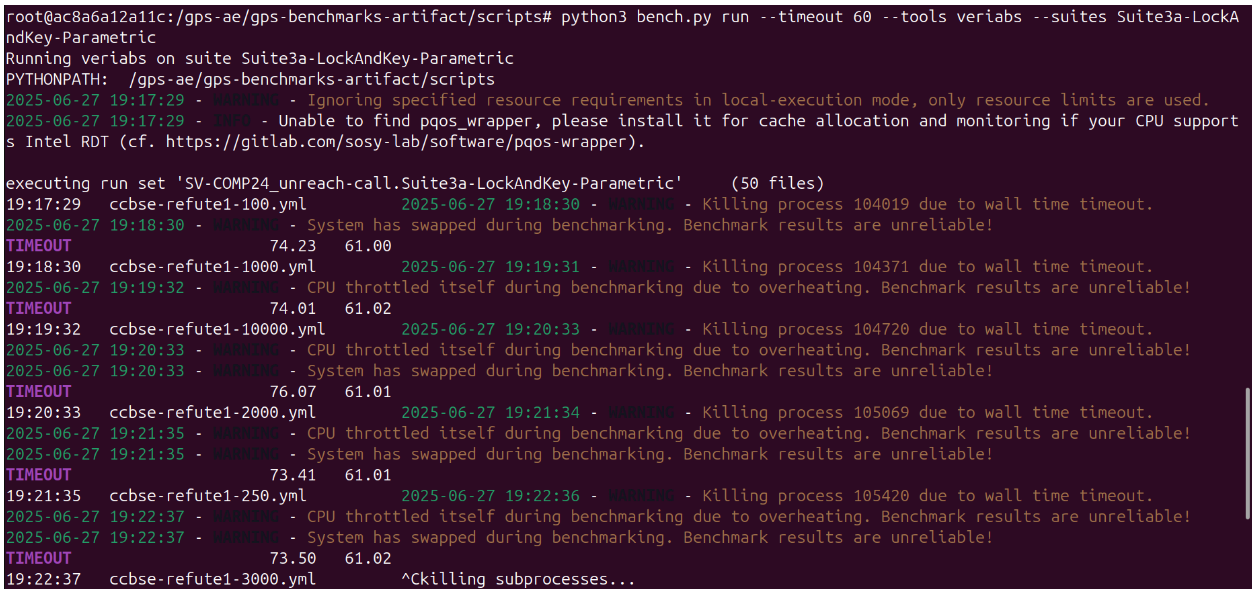


```
19:00:39   godfroid-issta-1b-1000.yml     false                           0.16    0.16
19:00:40   godfroid-issta-1b-10000.yml    false                           0.32    0.31
19:00:40   godfroid-issta-1b-2000.yml     false                           0.19    0.18
19:00:40   godfroid-issta-1b-250.yml      false                           0.17    0.17
19:00:41   godfroid-issta-1b-30.yml       false                           0.17    0.15
19:00:41   godfroid-issta-1b-3000.yml     false                           0.19    0.19
19:00:41   godfroid-issta-1b-500.yml      false                           0.17    0.16
19:00:41   godfroid-issta-1b-5000.yml     false                           0.22    0.23
19:00:42   godfroid-issta-1b-8000.yml     false                           0.28    0.28
19:00:42   lese10.yml                     false                           0.15    0.14
19:00:42   lese100.yml                    false                           0.15    0.14
19:00:43   lese1000.yml                   false                           0.14    0.14
19:00:43   lese100000.yml                 false                           0.15    0.15
19:00:43   lese250.yml                    false                           0.15    0.14
19:00:43   lese2500.yml                   false                           0.15    0.15
19:00:44   lese50.yml                     false                           0.14    0.15
19:00:44   lese500.yml                    false                           0.16    0.15
19:00:44   lese5000.yml                   false                           0.17    0.17
19:00:44   lese8000.yml                   false                           0.15    0.15
2025-06-27 19:00:45 - WARNING - System has swapped during benchmarking. Benchmark results are unreliable!

Statistics:             50 Files
  correct:              50
    correct true:        0
    correct false:      50
  incorrect:             0
    incorrect true:      0
    incorrect false:     0
  unknown:               0
  Score:                50 (max: 50)

In order to get HTML and CSV tables, run
table-generator results/gps.2025-06-27_19-00-29.results.SV-COMP24_unreach-call.Suite3a-LockAndKey-Parametric.x
root@ac8a6a12a11c:/gps-ae/gps-benchmarks-artifact/scripts# 
```

By contrast, if we run the VeriAbs tool on the same test suite, the running time would be longer:

```
python3 bench.py run --timeout 60 --tools veriabs --suites
Suite3a-LockAndKey-Parametric
```

The output would contain more time-outs and out-of-memory results. For instance, on our machine, VeriAbs produced timeout results on the first few benchmarks:



```
root@ac8a6a12a11c:/gps-ae/gps-benchmarks-artifact/scripts# python3 bench.py run --timeout 60 --tools veriabs --suites Suite3a-LockA
ndKey-Parametric
Running veriabs on suite Suite3a-LockAndKey-Parametric
PYTHONPATH:  /gps-ae/gps-benchmarks-artifact/scripts
2025-06-27 19:17:29 - WARNING - Ignoring specified resource requirements in local-execution mode, only resource limits are used.
2025-06-27 19:17:29 - INFO - Unable to find pqos_wrapper, please install it for cache allocation and monitoring if your CPU support
s Intel RDT (cf. https://gitlab.com/sosy-lab/software/pqos-wrapper).

executing run set 'SV-COMP24_unreach-call.Suite3a-LockAndKey-Parametric'     (50 files)
19:17:29   ccbse-refute1-100.yml         2025-06-27 19:18:30 - WARNING - Killing process 104019 due to wall time timeout.
2025-06-27 19:18:30 - WARNING - System has swapped during benchmarking. Benchmark results are unreliable!
TIMEOUT                       74.23   61.00
19:18:30   ccbse-refute1-1000.yml        2025-06-27 19:19:31 - WARNING - Killing process 104371 due to wall time timeout.
2025-06-27 19:19:32 - WARNING - CPU throttled itself during benchmarking due to overheating. Benchmark results are unreliable!
TIMEOUT                       74.01   61.02
19:19:32   ccbse-refute1-10000.yml       2025-06-27 19:20:33 - WARNING - Killing process 104720 due to wall time timeout.
2025-06-27 19:20:33 - WARNING - CPU throttled itself during benchmarking due to overheating. Benchmark results are unreliable!
2025-06-27 19:20:33 - WARNING - System has swapped during benchmarking. Benchmark results are unreliable!
TIMEOUT                       76.07   61.01
19:20:33   ccbse-refute1-2000.yml        2025-06-27 19:21:34 - WARNING - Killing process 105069 due to wall time timeout.
2025-06-27 19:21:35 - WARNING - CPU throttled itself during benchmarking due to overheating. Benchmark results are unreliable!
2025-06-27 19:21:35 - WARNING - System has swapped during benchmarking. Benchmark results are unreliable!
TIMEOUT                       73.41   61.01
19:21:35   ccbse-refute1-250.yml         2025-06-27 19:22:36 - WARNING - Killing process 105420 due to wall time timeout.
2025-06-27 19:22:37 - WARNING - CPU throttled itself during benchmarking due to overheating. Benchmark results are unreliable!
2025-06-27 19:22:37 - WARNING - System has swapped during benchmarking. Benchmark results are unreliable!
TIMEOUT                       73.50   61.02
19:22:37   ccbse-refute1-3000.yml        ^Ckilling subprocesses...
```

This should help you sanity-check our claim that GPS is indeed of good performance on our benchmark suite, even when compared with state-of-the-art tools. In what follows, you will find step-by-step instructions for running scripts to reproduce our experiments.

Step 7: Clean-up before proceeding to the next section.

Execute the following command, which cleans up the results/ folder containing results from previous trial runs:

```
rm -rf /gps-ae/gps-benchmarks-artifact/scripts/results/*
```

Note that it is important (for the "`analyze-table-column.sh`" script that analyzes the results of each benchmarking run) for the results/ sub-folder to be empty between benchmarking runs (For more details on this, see our explanation in the FAQ section at the end of this document). You can save the detailed output in another location, should you wish.

5 Step-by-Step Instructions

Note: The full set of benchmarks would likely take upwards to 5-6 days to reproduce, due to the time-out threshold of 600 seconds (which is the threshold used in our paper's evaluation section). Unless you run multiple instances of the Docker container in parallel, we *do not* recommend running the scripts concurrently.

If you are under time constraint, we provide a "quick" version of each script (simply add "-quick" to the name of the script, for instance, `table1-column2-gps.sh` ⟶ `table1-column2-gps`**`-quick.`**`sh`), where the timeout threshold is set to 200 seconds. However, the results produced might differ from our benchmark results (i.e., each tool would encounter more time-outs). Feel free to modify our scripts and insert a custom time-out threshold, but shorter time-out thresholds may result in your benchmark results deviating further from those reported in our paper.

Step 5a: Reproducing Table 1 in the paper

Go to the `/gps-ae/gps-benchmarks-artifact/scripts/` folder in the Docker image. Ensure the "`results/`" sub-folder is empty (If not, please remove the content inside).

Execute the following scripts in the folder, each of which is used to produce the statistics of a corresponding column in the table. The script will invoke benchexec and display per-benchmark statistics as it executes the corresponding tool on each benchmark. After the benchmarks are done, the script will output a table which allows you to compare statistics with the corresponding column in Table 1 of our paper.

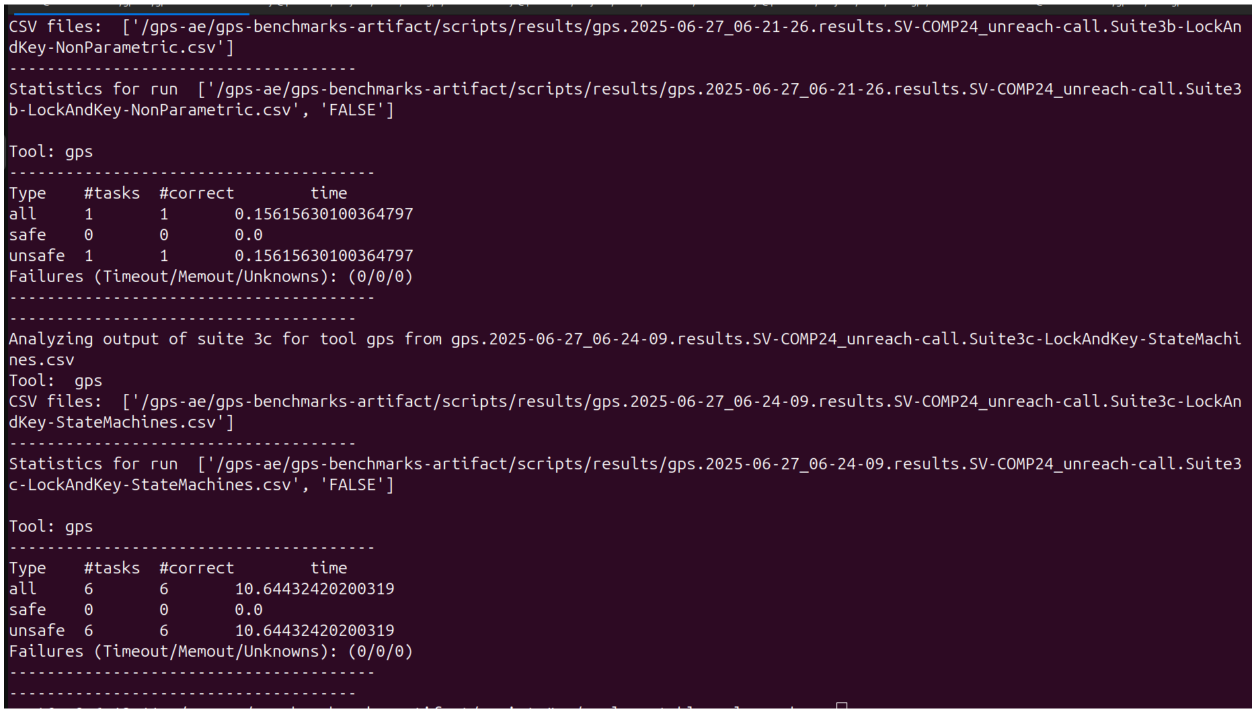
Note: Please go to the /gps-ae/gps-benchmarks-artifact/scripts/ directory first before executing any command below.

Reproduction of Column 2
First, execute the following command(s):

```
cd /gps-ae/gps-benchmarks-artifact/scripts && ./table1-column2-gps.sh
# estimated time: around 3 hours
```

# after the above command finishes, use this command to reproduce statistics
```
./analyze-table-column.sh gps
```

The sample output should look like:



```
CSV files:  ['/gps-ae/gps-benchmarks-artifact/scripts/results/gps.2025-06-27_06-21-26.results.SV-COMP24_unreach-call.Suite3b-LockAn
dKey-NonParametric.csv']
------------------------------------
Statistics for run  ['/gps-ae/gps-benchmarks-artifact/scripts/results/gps.2025-06-27_06-21-26.results.SV-COMP24_unreach-call.Suite3
b-LockAndKey-NonParametric.csv', 'FALSE']

Tool: gps
--------------------------------------
Type    #tasks  #correct        time
all     1       1       0.15615630100364797
safe    0       0       0.0
unsafe  1       1       0.15615630100364797
Failures (Timeout/Memout/Unknowns): (0/0/0)
--------------------------------------
------------------------------------
Analyzing output of suite 3c for tool gps from gps.2025-06-27_06-24-09.results.SV-COMP24_unreach-call.Suite3c-LockAndKey-StateMachi
nes.csv
Tool:  gps
CSV files:  ['/gps-ae/gps-benchmarks-artifact/scripts/results/gps.2025-06-27_06-24-09.results.SV-COMP24_unreach-call.Suite3c-LockAn
dKey-StateMachines.csv']
------------------------------------
Statistics for run  ['/gps-ae/gps-benchmarks-artifact/scripts/results/gps.2025-06-27_06-24-09.results.SV-COMP24_unreach-call.Suite3
c-LockAndKey-StateMachines.csv', 'FALSE']

Tool: gps
--------------------------------------
Type    #tasks  #correct        time
all     6       6       10.64432420200319
safe    0       0       0.0
unsafe  6       6       10.64432420200319
Failures (Timeout/Memout/Unknowns): (0/0/0)
--------------------------------------
------------------------------------
```

There should be a total of 5 tables generated. The “Time” column represents the total wall-time consumed for each benchmark category. The “safe” row denotes statistics summed across all safe benchmarks, and the “unsafe” row denotes statistics summed across all unsafe benchmarks.

- Table for Suite 1 corresponds to Suite #1 in our paper;
- Table for Suite 2 corresponds to Suite #2 in our paper;
- Table for Suite 3a corresponds to the parametrized L&K benchmarks of Suite #3 in our paper;
- Table for Suite 3b corresponds to the un-parametrized L&K benchmarks of Suite #3 in our paper;
- Table for Suite 3c corresponds to the state machines L&K benchmarks of Suite #3 in our paper.

Lastly, execute the following command:

```
rm ./results/* # clean up results folder before next run
```

Reproduction of Column 3

```
./table1-column3-gpslite.sh     # estimated time: under 3 hours
# after the above command finishes, use this command to reproduce
statistics
./analyze-table-column.sh gpslite # output similar to the above
rm -rf ./results/*
```

Reproduction of Column 4

```
./table1-column4-gps-nogas.sh   # estimated time: under 4 hours
# after the above command finishes, use this command to reproduce
statistics
./analyze-table-column.sh gps-nogas # output similar to the above
rm -rf ./results/*
```

Reproduction of Column 5 (Warning: Time-consuming Benchmark)

```
./table1-column5-gps-nogas-nosum.sh  # estimated time: 1-2 days
# after the above command finishes, use this command to reproduce
statistics
./analyze-table-column.sh gps-nogas-nosum # output similar to the
above
rm -rf ./results/*
```

Reproduction of Column 6 (Warning: Time-consuming Benchmark)

```
./table1-column6-impact.sh           # estimated time: 1-2 days
./analyze-table-column.sh cpaimpact # output similar to the above
rm -rf ./results/*
```

Reproduction of Column 7

```
./table1-column7-cra.sh              # estimated time: 1 hour
./analyze-table-column.sh CRA # output similar to the above
rm -rf ./results/*
```

Step 5b: Reproducing Table 2 in the paper.

In the same directory, the following scripts reproduce each column of Table 2. They work in similar fashion as the scripts for reproducing Table 1. Notice column 2 of Table 2 is identical to column 2 of Table 1, so you don’t have to re-run the benchmarks for this column again.

Reproduction of Column 3

```
./table2-column3-veriabs.sh     # estimated time: 3-4 hours
./analyze-table-column.sh veriabs
rm -rf ./results/*
```

Reproduction of Column 4 (Warning: Time-consuming benchmark)

```
./table2-column4-symbiotic.sh        # estimated time: 1-2 days
./analyze-table-column.sh symbiotic
rm -rf ./results/*
```

Reproduction of Column 5 (Warning: Time-consuming benchmark)

```
./table2-column5-cpachecker.sh  # estimated time: 1-2 days
./analyze-table-column.sh cpachecker
rm -rf results/*
```

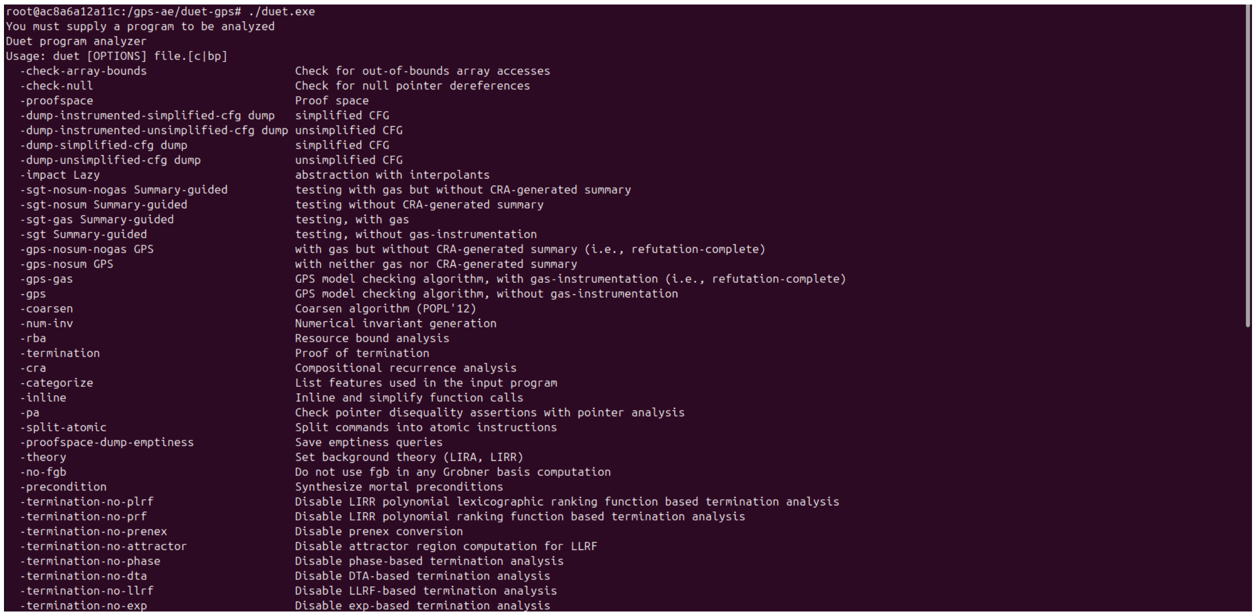
After executing the scripts above, you would have completed the artifact evaluation.

6 Peeking Under the Hood

This section describes the source code and scripts inside the artifact in more detail, and provides some guidance for the curious experimenter who wants to tinker more with this artifact.

6a) Using our GPS implementation in standalone fashion.

GPS is implemented atop the Duet program analysis tool[3]. An executable named “duet.exe” under /gps-ae/duet-gps/ is the main entry point to our implementation of GPS. Directly running this executable should result in the following output:



```
root@ac8a6a12a11c:/gps-ae/duet-gps# ./duet.exe
You must supply a program to be analyzed
Duet program analyzer
Usage: duet [OPTIONS] file.[c|bp]
  -check-array-bounds                    Check for out-of-bounds array accesses
  -check-null                            Check for null pointer dereferences
  -proofspace                            Proof space
  -dump-instrumented-simplified-cfg dump   simplified CFG
  -dump-instrumented-unsimplified-cfg dump unsimplified CFG
  -dump-simplified-cfg dump              simplified CFG
  -dump-unsimplified-cfg dump            unsimplified CFG
  -impact Lazy                           abstraction with interpolants
  -sgt-nosum-nogas Summary-guided        testing with gas but without CRA-generated summary
  -sgt-nosum Summary-guided              testing without CRA-generated summary
  -sgt-gas Summary-guided                testing, with gas
  -sgt Summary-guided                    testing, without gas-instrumentation
  -gps-nosum-nogas GPS                   with gas but without CRA-generated summary (i.e., refutation-complete)
  -gps-nosum GPS                         with neither gas nor CRA-generated summary
  -gps-gas                               GPS model checking algorithm, with gas-instrumentation (i.e., refutation-complete)
  -gps                                   GPS model checking algorithm, without gas-instrumentation
  -coarsen                               Coarsen algorithm (POPL'12)
  -num-inv                               Numerical invariant generation
  -rba                                   Resource bound analysis
  -termination                           Proof of termination
  -cra                                   Compositional recurrence analysis
  -categorize                            List features used in the input program
  -inline                                Inline and simplify function calls
  -pa                                    Check pointer disequality assertions with pointer analysis
  -split-atomic                          Split commands into atomic instructions
  -proofspace-dump-emptiness             Save emptiness queries
  -theory                                Set background theory (LIRA, LIRR)
  -no-fgb                                Do not use fgb in any Grobner basis computation
  -precondition                          Synthesize mortal preconditions
  -termination-no-plrf                   Disable LIRR polynomial lexicographic ranking function based termination analysis
  -termination-no-prf                    Disable LIRR polynomial ranking function based termination analysis
  -termination-no-prenex                 Disable prenex conversion
  -termination-no-attractor              Disable attractor region computation for LLRF
  -termination-no-phase                  Disable phase-based termination analysis
  -termination-no-dta                    Disable DTA-based termination analysis
  -termination-no-llrf                   Disable LLRF-based termination analysis
  -termination-no-exp                    Disable exp-based termination analysis
```

The table below lists commands for running the various Duet analyses relevant to this artifact (an un-modified version of Duet, which contains compositional recurrence analysis by default, is located under `/gps-ae/duet/` and is used for running vanilla CRA):

[3] https://github.com/zkincaid/duet

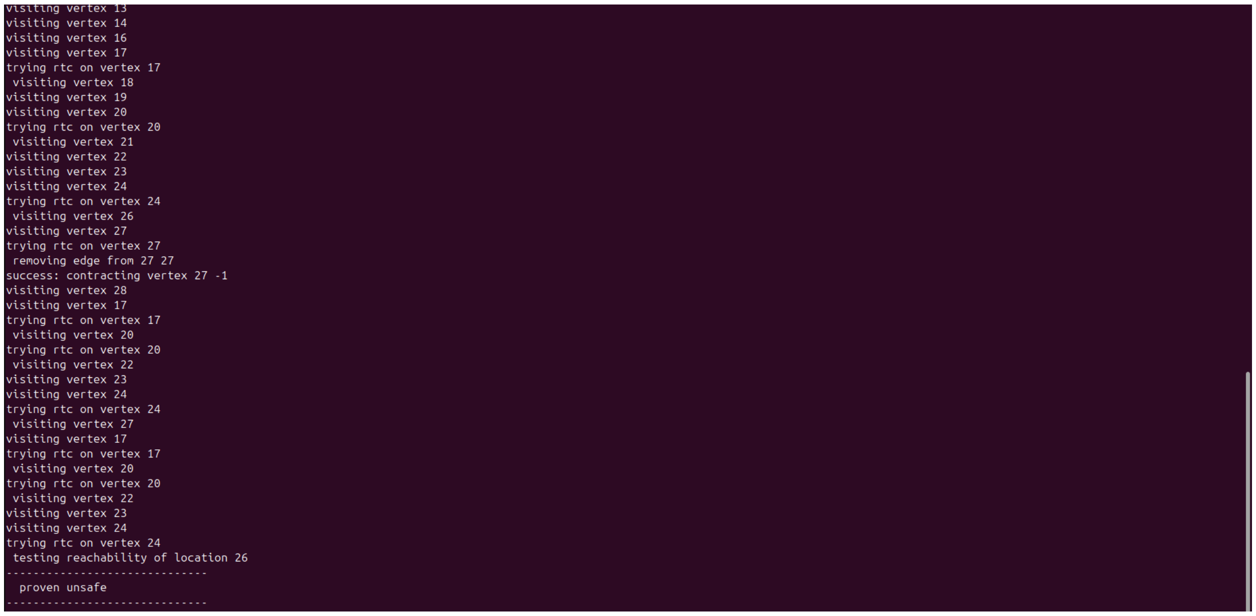
/gps-ae/duet/duet.exe -monotone -cra-split-loops -cra <file.c>	Runs compositional recurrence analysis in monotone mode
/gps-ae/duet-gps/duet.exe -monotone -cra-split-loops -gps-gas <file.c>	Runs GPS with summaries generated by CRA in monotone mode
/gps-ae/duet-gps/duet.exe -monotone -cra-split-loops -gps <file.c>	Runs GPS with CRA-generated summaries but without gas-instrumentation
/gps-ae/duet-gps/duet.exe -cra-split-loops -gps-nosum-nogas <file.c>	Runs GPS without CRA-generated summaries and without gas-instrumentation
/gps-ae/duet-gps/duet.exe -monotone -cra-split-loops -sgt-gas <file.c>	Runs the GPSLite algorithm with CRA-generated summaries

You can run Duet with any configurations above with a file in our benchmark suite to examine its outputs.

Example 1. Go to the /gps-ae/duet-gps/ folder. Run the following command:

```
./duet.exe -monotone -cra-split-loops -gps-gas
/gps-ae/gps-benchmarks-artifact/suite1-svcomp/loop-acceleration/simpl
e_3-1.c
```

The above command gives the output “proven unsafe,” shown as follows:



```
visiting vertex 13
visiting vertex 14
visiting vertex 16
visiting vertex 17
trying rtc on vertex 17
 visiting vertex 18
visiting vertex 19
visiting vertex 20
trying rtc on vertex 20
 visiting vertex 21
visiting vertex 22
visiting vertex 23
visiting vertex 24
trying rtc on vertex 24
 visiting vertex 26
visiting vertex 27
trying rtc on vertex 27
 removing edge from 27 27
success: contracting vertex 27 -1
visiting vertex 28
visiting vertex 17
trying rtc on vertex 17
 visiting vertex 20
trying rtc on vertex 20
 visiting vertex 22
visiting vertex 23
visiting vertex 24
trying rtc on vertex 24
 visiting vertex 27
visiting vertex 17
trying rtc on vertex 17
 visiting vertex 20
trying rtc on vertex 20
 visiting vertex 22
visiting vertex 23
visiting vertex 24
trying rtc on vertex 24
 testing reachability of location 26
------------------------------
  proven unsafe
------------------------------
```

This means the “simple_3-1.c” file contains an assertion violation. By contrast, you may verify that running only CRA on the same file yields the output “1 errors total” as a result, which corresponds to CRA reporting an unknown (instead of an actual bug, as CRA could only be used to verify safety):

```
./duet.exe -monotone -cra-split-loops -cra
/gps-ae/gps-benchmarks-artifact/suite1-svcomp/loop-acceleration/simpl
e_3-1.c
```

Example 2. Go to the /gps-ae/duet-gps/ folder. Run the following command (which corresponds to example EX-3 in our paper):

```
./duet.exe -monotone -cra-split-loops -gps-gas
/gps-ae/gps-benchmarks-artifact/suite2-safe-examples/ex3-a.c
```

The last few lines of the output should show "proven safe." This means that GPS is able to verify that the file "ex3-a.c" contains a safe assertion (that cannot be violated). By contrast, vanilla CRA could not prove safety of this file; running the command

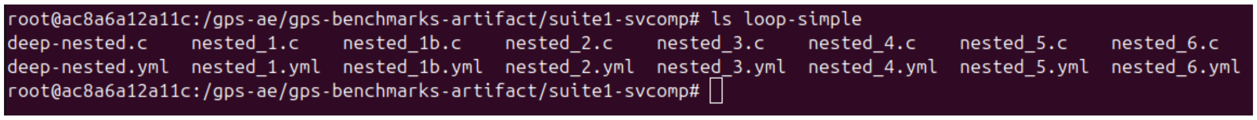
```
./duet.exe -monotone -cra-split-loops -cra
/gps-ae/gps-benchmarks-artifact/suite2-safe-examples/ex3-a.c
```

results in the output "1 errors total," meaning CRA failed to prove safety of the given file. Similarly, the GPSLite algorithm (which only uses summary-directed testing without refinement) would time-out when trying to verify this example:

```
timeout 10 ./duet.exe -monotone -cra-split-loops -sgt-gas
/gps-ae/gps-benchmarks-artifact/suite2-safe-examples/ex3-a.c # feel
free to try a greater time-out threshold
```

Additional Examples. You are welcome to try executing our tool on more examples inside the /gps-ae/gps-benchmarks-artifact/suite* folders. The layout of each folder is explained below:

- **`/gps-ae/gps-benchmarks-artifact/suite1-svcomp`**: Contains several sub-folders prefixed with "loop," each sub-folder contains a subset of files from a corresponding sub-folder in the SV-COMP 2024 benchmarks[4]. For instance, the "loop-simple/" sub-folder should contain the following files:



```
root@ac8a6a12a11c:/gps-ae/gps-benchmarks-artifact/suite1-svcomp# ls loop-simple
deep-nested.c    nested_1.c    nested_1b.c    nested_2.c    nested_3.c    nested_4.c    nested_5.c    nested_6.c
deep-nested.yml  nested_1.yml  nested_1b.yml  nested_2.yml  nested_3.yml  nested_4.yml  nested_5.yml  nested_6.yml
root@ac8a6a12a11c:/gps-ae/gps-benchmarks-artifact/suite1-svcomp# 
```

The ".c" files correspond to the C programs included in the SV-COMP benchmark suite. The ".yml" files are used for the benchexec benchmark environment; they contain the ground-truth verification result. For instance, "nested_2.yml" contains the following lines:

[4] https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/tree/main/c?ref_type=heads

```
  - property_file: ../properties/unreach-call.prp
    expected_verdict: true
```

This means the file “nested_2.c” contains a safe assertion. If a file contains a buggy assertion, the “expected_verdict” attribute under the “unreach-call” property should be marked as false.

- **`/gps-ae/gps-benchmarks-artifact/suite2-safe-examples`**: Contains 9 C program files (and corresponding YML files for benchexec) used as synthetic examples in our paper. All programs in this folder are safe.

- **`/gps-ae/gps-benchmarks-artifact/suite3a-lock-and-key-parametrizable:`** Contains 50 parametrizable lock-and-key benchmark programs in our paper. The number in the suffix of each program name denotes the parameter value (e.g., “ccbse-refute2-8000.c” means the parameter value is set to 8000). All programs in this folder are unsafe.

- **`/gps-ae/gps-benchmarks-artifact/suite3b-lock-and-key-unparametrizable`**: Contains a single un-parametrizable lock-and-key program in our paper. All programs in this folder are unsafe.

- **`/gps-ae/gps-benchmarks-artifact/suite3c-lock-and-key-statemachines`**: Contains 6 state machine programs from the lock-and-key benchmarks in our paper. All programs in this folder are unsafe.

6b) Folder structure of our artifact

As mentioned above, all tools used inside this artifact are under the `/gps-ae/` folder. Inside this folder, you will find:

- duet-gps, an implementation of GPS inside the Duet program analysis tool inside the `/gps-ae/duet-gps/` folder
    - The main GPS algorithm is implemented inside the `duet/gps.ml` file inside `/gps-ae/duet-gps/` folder.
- Various dependencies for building duet (ocaml-flint, normlizffi, etc)
- CPAchecker, VeriAbs, and symbiotic distributions used to compare with GPS and its variants in our benchmarks.
- The `/gps-ae/gps-benchmarks-artifact` folder contains benchmark suites and scripts used to run our evaluations.

6c) Running other tools involved in our benchmarks

This section provides some instructions should you want to run one of {VeriAbs, CPAChecker (Portfolio), CPAChecker (Impact), Symbiotic} on an individual benchmark.

Running VeriAbs on an individual benchmark. The VeriAbs tool reads in a YML file associated with each benchmark. As explained in Section 6a, each YML file contains a path to the benchmark program, path to the property file (with suffix ".prp"), and the ground-truth value for the result of the benchmark task. To run VeriAbs with a benchmark, run the following command:

```
/gps-ae/VeriAbs/scripts/veriabs --sv22 --property-file <.prp file
referenced in the YML file> -32 <path to the YML file>
```

For instance, the following command runs VeriAbs on a parametrized Lock-and-Key benchmark:

```
/gps-ae/VeriAbs/scripts/veriabs --sv22 --property-file
/gps-ae/gps-benchmarks-artifact/properties/unreach-call.prp -32
../suite3a-lock-and-key-parametrizable/ccbse-refute1-1000.c
```

Running Symbiotic on an individual benchmark. Similar to VeriAbs, Symbiotic also requires a property file (the path to the property file can be found in the YML file of the corresponding benchmark), and takes in a C program as input:

/gps-ae/symbiotic/bin/symbiotic --witness witness.graphml --sv-comp --prp=<.prp file> --32 <.c file>

For instance, the following command runs Symbiotic on a benchmark program from suite #1 (SV-COMP):

/gps-ae/symbiotic/bin/symbiotic --witness witness.graphml --sv-comp
--prp=/gps-ae/gps-benchmarks-artifact/suite1-svcomp/properties/unreach-call.prp --32
../suite1-svcomp/loop-acceleration/const_1-1.c

Running CPAChecker (Portfolio mode, SVCOMP 2024) on an individual benchmark. We use CPAChecker 2.3 in portfolio mode; this is the CPAChecker release configured for SVCOMP 2024. CPAChecker again requires a property file (the path to the property file for each benchmark may be found in the YML file for the corresponding benchmark) as well as the input C program:

```
/gps-ae/CPAchecker-2.3-unix/scripts/cpa.sh -svcomp24 -heap 10000M
-benchmark -timelimit '200 s' -stats -spec <path to .prp file> -32
<path to C program>
```

For instance, the following command runs CPAChecker (Portfolio mode) on a benchmark program from suite #1 (SV-COMP) with a time-out threshold of 200 seconds:

```
/gps-ae/CPAchecker-2.3-unix/scripts/cpa.sh -svcomp24 -heap 10000M
-benchmark -timelimit '200 s' -stats -spec
/gps-ae/gps-benchmarks-artifact/suite1-svcomp/properties/unreach-call
.prp -32
/gps-ae/gps-benchmarks-artifact/suite1-svcomp/loop-acceleration/const
_1-1.c
```

Running CPAChecker (Impact mode, latest version) on an individual benchmark. We use the latest version of CPAChecker for its implementation of the Impact algorithm. To run CPAchecker with the Impact algorithm, execute the following command:

```
/gps-ae/CPAchecker-4.0-unix/bin/cpachecker
--predicateAnalysis-ImpactRefiner-SBE -heap 10000M --benchmark
--timelimit '200 s' --stats --spec <path to .prp file> --32 <path to
C program>
```

For instance, the following command runs CPAChecker (Impact mode) on a benchmark program from suite #1 (SV-COMP) with a time-out threshold of 200 seconds:

```
/gps-ae/CPAchecker-4.0-unix/bin/cpachecker
--predicateAnalysis-ImpactRefiner-SBE -heap 10000M --benchmark
--timelimit '200 s' --stats --spec
/gps-ae/gps-benchmarks-artifact/suite1-svcomp/properties/unreach-call
.prp --32
/gps-ae/gps-benchmarks-artifact/suite1-svcomp/loop-acceleration/const
_1-1.c
```

7 Q&A

This section lists some questions you might encounter during artifact evaluation.

Q1. What does the "`/gps-ae/init.sh`" script do? I encountered a problem when running the script.

The initialization script does two things: (1) it pulls the latest version of benchmark scripts and data from a GitHub repository; (2) The benchexec tool requires all CPU cores used during benchmark execution to be identical (i.e., no mixing of high-performance cores and energy-efficient cores in modern CPUs that have asymmetric cores)[5]. Our script uses the

[5] For more details, see https://github.com/sosy-lab/benchexec/issues/850

following commands to find the (likely identical) cores that share the highest frequency on your host machine, and then dumps the core numbers to a text file in the /gps-ae/tmpfiles/ folder:

```
FREQ=`lscpu -e | awk '//{print $7}' | sort | uniq -c | sort -n | tail
-1 | awk '//{print $2}'`
CPUS=`lscpu -e | awk -v freq="$FREQ" '//{ if($7==freq) print $1 }'`
echo $CPUS | tr ' ' ',' > /gps-ae/tmpfiles/cpus
```

You can verify via the "`lscpu`" command whether the CPU cores listed in the text file are indeed identical. If they are not, the bench.py script used to run experiments will fail, with an error indicating that this is the problem.

If the above command fails due to your system environment, you can manually specify the identical CPU core numbers in the `/gps-ae/tmpfiles/cpus` file. On our machine, this text file looks like the following (one single line, with no new lines or blank spaces trailing):

```
0,1,2,3,8,9,10,11
```

Our benchmarking script requires the above file to be present in order to work.

Q2. The `bench.py` script complains that it "could not deduce the hardware CPU requirements."

Please verify if the `/gps-ae/tmpfiles/cpus` file exists; if not, please refer to our answer to Q1 above. In particular, you should either run the `/gps-ae/init.sh` script (which would try to deduce the CPU cores and create the missing file), or manually run the given commands and manually edit the `/gps-ae/tmpfiles/cpus` file to include the appropriate CPU core numbers.

Q3. The benchexec output for a given run contains various warnings (e.g. about pqos_wrapper, or swap space, or CPU throttling)

Feel free to ignore these warnings; in our experience, they generally do not impact reproducibility of our results.

Q4. The "`analyze-table-column.sh`" script reports "Bad: expected exactly one CSV file for…"

This script must parse the corresponding benchmark output XML file (for a given tool and test suite) into a CSV file, and then parse the CSV file to print out overall statistics about each test suite. Thus, it is important that only one XML file from one benchmarking run for each tool and test suite is saved (otherwise, the script is unable to infer which XML file to parse).

To rectify this error, make sure the results/ folder does not contain results from previous benchmarking runs of the same tool on the same test suite.

Q5. How to verify the actual arguments used to run each tool in your benchmarks?

Our tool configurations are specified in a manner consistent with benchexec and SV-COMP requirements; in particular, the tool configurations are given to benchexec as XML files under the `/gps-ae/gps-benchmarks-artifact/scripts/benchmark-defs/` folder. In this folder, you can find the precise arguments of each tool passed to benchexec, specified in the XML file for each corresponding tool.

Q6. I have additional questions. Where should I direct them / who should I contact?

We maintain a file https://github.com/ruijiefang/gps-oopsla25-ae-faq online to document any additional questions, either from artifact reviewers or discovered ourselves. You can take a look at this document first should you encounter any questions not covered in this document.

Should you encounter any trouble not covered by this document, you are most welcome to message us on HotCrp.

8 License