


# Reusing Caches and Invariants for Efficient and Sound Incremental Static Analysis (Artifact)

Mamy Razafintsialonina ✉ 

Université Paris-Saclay, CEA, List, Palaiseau, France  
Sorbonne Université, CNRS, LIP6, Paris, France

David Bühler ✉ 

Université Paris-Saclay, CEA, List, Palaiseau, France

Valentin Perrelle ✉ 

Université Paris-Saclay, CEA, List, Palaiseau, France

---

## Abstract

Static analysis by means of abstract interpretation is a tool of choice for proving absence of some classes of errors, typically undefined behaviors in code, in a sound way. However, static analysis tools are hardly integrated in CI/CD processes. One of the main reasons is that they are still time- and memory-expensive to apply after every single patch when developing a program. For solving this issue, incremental static analysis helps developers quickly obtain analysis results after making changes to a program. However, existing approaches are often not guaranteed to be sound, limited to specific analyses, or tied to specific tools. This limits their generalizability and applicability in practice, especially

for large and critical software. In this paper, we propose a generic, sound approach to incremental static analysis that is applicable to any abstract interpreter. Our approach leverages the similarity between two versions of a program to soundly reuse previously computed analysis results. We introduce novel methods for summarizing functions and reusing loop invariants. They significantly reduce the cost of reanalysis, while maintaining soundness and a high level of precision. We have formalized our approach, proved it sound, implemented it in *Eva*, the abstract interpreter of *Frama-C*, and evaluated it on a set of real-world commits of open-source programs.

**2012 ACM Subject Classification** Theory of computation → Program reasoning; Software and its engineering → Software verification and validation; Software and its engineering → Formal methods; Software and its engineering → Software notations and tools

**Keywords and phrases** Abstract Interpretation, Static Analysis, Incremental Analysis

**Digital Object Identifier** 10.4230/DARTS.11.2.15

**Funding** This work was funded by project ANR-22-PECY0005 “Secureval” managed by the French National Research Agency for France 2030.

**Acknowledgements** We thank the anonymous reviewers for their valuable feedback, which improved the final version of the artifact.

**Related Article** Mamy Razafintsialonina, David Bühler, Antoine Miné, Valentin Perrelle, and Julien Signoles, “Reusing Caches and Invariants for Efficient and Sound Incremental Static Analysis”, in 39th European Conference on Object-Oriented Programming (ECOOP 2025), LIPIcs, Vol. 333, pp. 28:1–28:29, 2025. <https://doi.org/10.4230/LIPIcs.ECOOP.2025.28>

**Related Conference** 39th European Conference on Object-Oriented Programming (ECOOP 2025), June 30–July 2, 2025, Bergen, Norway

**Evaluation Policy** The artifact has been evaluated as described in the ECOOP 2025 Call for Artifacts and the ACM Artifact Review and Badging Policy.



## 1 Scope

### Overview: What does the artifact comprise?

The artifact provides a self-contained Docker image to reproduce the experimental results presented in the related paper. It includes the implementation of the incremental static analysis approaches within the Eva plugin of Frama-C, the benchmark programs (PolarSSL, Chrony and Monocypher), and the scripts to run the analyses and generate the result figures and tables.

The artifact is intended to back the experimental claims made in the paper, regarding the performance (time efficiency, memory and disk usage, iteration count, cache loading and program differencing time) and precision (number of alarms) of the proposed approaches compared to full analysis. Specifically, it generates the data supporting Figures 8 and 9, and Table 3 of the paper.

## 2 Content

The artifact's container includes the following under `/workspace`:

- `frama-c` (directory): Source code of Frama-C with the implemented approaches (already installed).
- `polarssl|chrony|monocypher` (directories): Contain configuration files (`.frama-c`), program source code (`<program>`), and the list of analyzed commits (`applicable_commits.txt`) for each benchmark program.
- `example` (directory): Contains a small example program to try out the incremental analysis.
- `plots` (directory): Output directory for generated figures and tables (mounted to the host machine). Contains pdf and png files.
- `results` (directory): Output directory for raw benchmark data (mounted to the host machine).
- `run_short.sh` (script): Runs a quick "kick-the-tires" benchmark using 2 commits of Monocypher.
- `run_all.sh` (script): Runs the full benchmarking process for all programs and commits.
- `benchmark.py` (script): Main script orchestrating the benchmarking process (preparation, running analyses, and saving results).
- `reset.sh` (script): Cleans output directories.
- `plot.py` (script): Processes raw results and generates plots and tables.

## 3 Getting the artifact

**Download:** Link <https://doi.org/10.5281/zenodo.15516456>.

**Important Note on Commands:** To avoid potential issues with hidden characters or formatting when copying from a PDF, it is strongly recommended to **type the commands manually** into your terminal rather than using copy-paste. Alternatively, a README.md file is included within the artifact/Docker image for copying commands directly.

**Installation:** Import the Docker image using:

```
$ docker load -i frama-c-incremental.local.tar
```

**Execution:** Run commands inside an empty directory to easily access results mounted from the container.

**Kick-the-tires phase (~15 minutes)**

```
$ docker run -it --name frama-c-bench-short \
-v "$PWD/plots:/workspace/plots" \
-v "$PWD/results:/workspace/results" \
frama-c-incremental:local /bin/bash /workspace/run_short.sh
```

**Full benchmark (~20 hours)**

```
$ docker run -it --name frama-c-bench-all \
-v "$PWD/plots:/workspace/plots" \
-v "$PWD/results:/workspace/results" \
frama-c-incremental:local /bin/bash /workspace/run_all.sh
```

**Scaled-down benchmark (~N/A) (only 30 commits per program)**

```
$ docker run -it --name frama-c-bench-partial \
-v "$PWD/plots:/workspace/plots" \
-v "$PWD/results:/workspace/results" \
frama-c-incremental:local /bin/bash /workspace/run_partial.sh
```

**Cleaning up**

Output files in `plots` and `results` will be owned by root due to Docker mounting; use root privileges (`sudo rm -rf plots results`) to delete them.

## 4 Tested platforms

**Hardware requirements:** Recommend  $\geq 16$ GB RAM,  $\geq 6$  performance cores and a single thread rating  $\geq 2500$  on PassMark CPU benchmark (<https://www.cpubenchmark.net/>), and  $\geq 200$ GB disk space. Settings N1, N2, I3 require 4 threads; I1, I2, I4 are not parallelizable. Running on lower-end hardware is possible, but we recommend using the scaled-down version of the benchmarks.

**Tested systems:**

- Platform 1: Manjaro Linux (Kernel 6.6.80), Intel Core i7-12800H (14c/20t), 64GB RAM, 2TB NVMe SSD, Docker 28.0.0.
- Platform 2: Ubuntu 24.04.2 LTS (Kernel 6.11.0), Intel Core i5-12400F (6c/12t), 16GB RAM, 1TB NVMe SSD, Docker 28.0.1.

**Docker base image:** Debian 12 (Bookworm).

## 5 License

The artifact is available under the Creative Commons Attribution-ShareAlike 4.0 International license (CC BY-SA 4.0).

## 15:4 Efficient and Sound Incremental Static Analysis (Artifact)

### 6 MD5 sum of the artifact

0ced0251ab667ebd91be0ebaabfc15d2

### 7 Size of the artifact

2.0 GiB

## A Claims

We claim the **functional badge** for the artifact. All data supporting the claims (specifically Figure 8, Figure 9, and Table 3) are **automatically generated** by `plot.py` into the `plots` directory after running the benchmark scripts.

- Figure 8: `plots/<program>/box-plot.png`
- Figure 9: `plots/<program>/scatter-plot.png`
- Table 3: `plots/<program>/summary-table.png`

## B Try it out

You can experiment with incremental analysis using an example project. To start, run the following command:

```
$ docker run -it --rm \
  frama-c-incremental:local /bin/bash
```

This command launches a Docker container with Frama-C pre-installed. Inside the container, you can find an example project located at `/workspace/example`.

### Step 1: Run a Full Analysis

Inside the container, run:

```
$ cd example
$ frama-c main.c -save frama.sav -eva -eva-statistics-file stats.
  full.csv
```

This will:

- Print the analysis results to `stdout`
- Save the analysis state to `frama.sav`
- Output statistics to `stats.full.csv`

The analysis should report a **true invalid memory access** alarm.

### Step 2: Fix and Run Incremental Analysis

Apply the following patch (or edit the file yourself) to fix the alarm:

```
$ sed -i 's/i <= ARRAY_SIZE/i < ARRAY_SIZE/' main.c
$ frama-c main.c -eva -eva-load frama.sav -eva-statistics-file
  stats.incr.csv
```

This reuses the saved state and analyzes only what has changed.

## Compare Full vs. Incremental Analysis

As the example is small, the incremental analysis may not show significant performance improvements on the analysis time. That's why we focus on the statistics files to evaluate the performance gain. To evaluate the efficiency of incremental analysis, compare:

- `stats.full.csv`
- `stats.incr.csv`

Focus on these key metrics:

- **total-iterations:** Total number of fixpoint iterations
- **reused-widenings:** Number of reused widenings from the previous run
- **memexec-hits-summaries:** Number of summaries reused

These metrics help quantify the performance gain provided by incremental analysis.