Ownership Refinement Types for Pointer Arithmetic and Nested Arrays (Artifact)

Yusuke Fujiwara
Kyoto University, Japan

Yusuke Matsushita
Kyoto University, Japan

Kohei Suenaga
Kyoto University, Japan

Atsushi Igarashi
Kyoto University, Japan

Abstract

This artifact accompanies the paper "Ownership Refinement Types for Pointer Arithmetic and Nested Arrays." It provides `nested_array_ConSORT`, a tool for automated ownership type inference and refinement type checking for imperative programs with pointer arithmetic and nested arrays. The artifact includes the tool implementation, benchmark programs, evaluation scripts to reproduce the experimental results (Tables 1, 3, 4, and 5) from the paper, and a comparison baseline (Extended_ConSORT[1] by Tanaka et al. [3]). The artifact is packaged as a Docker image with all dependencies pre-installed, supporting both x86_64 and ARM64 platforms.

2012 ACM Subject Classification Theory of computation → Program verification

Keywords and phrases aliasing, fractional ownership, program verification, refinement types, type systems

Digital Object Identifier 10.4230/DARTS.12.1.23

Funding This work is partially supported by JSPS KAKENHI Grant Number 20H05703, Japan.
Yusuke Matsushita: His research was supported in part also by the Hakubi Project at Kyoto University and JSPS KAKENHI Grant Number JP24KJ0133.
Kohei Suenaga: His research was supported in part also by JST CREST Grant Number JPMJCR2012, Japan, and JSPS KAKENHI Grant Number 25H01113, Japan.

Acknowledgements We thank the reviewers for their constructive comments. We are also grateful to Takashi Suwa for helping us test the artifact. The second author is currently hosted by MPI-SWS as a JSPS Overseas Research Fellow.

Related Article Yusuke Fujiwara, Yusuke Matsushita, Kohei Suenaga, and Atsushi Igarashi, "Ownership Refinement Types for Pointer Arithmetic and Nested Arrays", in 40th European Conference on Object-Oriented Programming (ECOOP 2026), LIPIcs, Vol. 372, pp. 6:1–6:31, 2026.
`https://doi.org/10.4230/LIPIcs.ECOOP.2026.6`

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] `https://github.com/mamizu-git/Extended_ConSORT`

1 Scope

The artifact supports the experimental evaluation presented in the paper. Specifically, it enables reproduction of:

- **Table 1**: Verification timing and alias statement counts for 19 nested array benchmarks.
- **Table 3**: Performance comparison with the tool by Tanaka et al. [3] on 8 integer array benchmarks.
- **Table 4**: Z3 [2] version comparison (4.14.1 vs. 4.11.2) across all 35 benchmarks. The paper reports mean times over 10 runs; the artifact performs a single run per benchmark to keep evaluation time manageable.
- **Table 5**: Summary of Z3 version comparison results (derived from Table 4).

Table 2 (inferred ownership terms) is not included in the automated evaluation. The ownership terms are output as SMT2 formulas in `experiment/out_sat_ans.smt2` after each verification run, but interpreting them as the ownership terms presented in the paper requires manual inspection.

The tool accepts programs written in a custom `.imp` language and automatically performs ownership type inference (using Z3 [2]) followed by refinement type checking (using HoICE [1]). Both positive examples (expected to verify) and negative examples (expected to fail) are included.

2 Content

The artifact consists of:

- Pre-built Docker images for x86_64 and ARM64 hosts (`nested-array-consort:ecoop26`) containing:
  - The `nested_array_ConSORT` tool (OCaml, built with dune)
  - Z3 SMT solver [2] versions 4.11.2 and 4.14.1
  - HoICE CHC solver [1] v1.10.0
  - Extended_ConSORT [3] for baseline comparison
  - All benchmark `.imp` programs
  - Evaluation scripts for reproducing Tables 1, 3, 4, and 5
- Source code of the tool (with `Dockerfile` for building from source)
- Artifact documentation (`ARTIFACT.md`) and license file (`LICENSE`)

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).

The artifact is archived on Zenodo at `https://doi.org/10.5281/zenodo.19521221`. The Zenodo record was published on 2026-04-12 and contains Docker images for x86_64 and ARM64, a source archive, artifact documentation, and the license file. The source code repository is available at `https://github.com/SoftwareFoundationGroupAtKyotoU/nested_array_ConSORT` (branch `ecoop26-artifact-eval`).

4 Tested platforms

- **Docker** (tested with Docker 20.x and later).
- Approximately **7 GB** of disk space for one compressed Docker image download, plus additional space after loading it into Docker.
- **x86_64** or **ARM64** (Apple Silicon) Linux or macOS with Docker support.
- No internet connection is required after loading the image.

5 License

The software artifact includes a `LICENSE` file for the Apache License 2.0. This artifact description is released under the Creative Commons Attribution 4.0 International license (CC BY 4.0), as required by DARTS.

6 MD5 sum of the artifact

43b0b114ef7b0e475d98337274426571

The MD5 sum above is for the `nested-array-consort-ecoop26.arm64.tar.gz` which is available at `https://doi.org/10.4230/DARTS.12.1.23`.

The Zenodo artifact payload files have the following MD5 sums:

- `nested-array-consort-ecoop26.x86.tar.gz`: `04d2d06f981e876e44a9f4815015f86e`
- `nested-array-consort-ecoop26.arm64.tar.gz`: `43b0b114ef7b0e475d98337274426571`
- `nested-array-consort-source.zip`: `dfc7f63fa49aa9921bff7603b4394147`
- `ARTIFACT.md`: `3a5898add4067a10925b993dc586a333`
- `LICENSE`: `817c302b311b2996450510af7cb0990a`

7 Size of the artifact

- 6.7 GB for the artifact available at `https://doi.org/10.4230/DARTS.12.1.23`
- 13,129,518,292 bytes in total for the Zenodo artifact payload files listed above (approximately 12.23 GiB).

A Getting Started Guide

A.1 Loading the Docker Image

If you have the pre-built image for an x86_64 host:

```
docker load -i nested-array-consort-ecoop26.x86.tar.gz
docker run -it nested-array-consort:ecoop26
```

If you have the pre-built image for an ARM64 host:

```
docker load -i nested-array-consort-ecoop26.arm64.tar.gz
docker run -it nested-array-consort:ecoop26
```

Alternatively, you can build the image from source:

```
git clone https://github.com/SoftwareFoundationGroupAtKyotoU/
    nested_array_ConSORT.git
cd nested_array_ConSORT
docker build -t nested-array-consort:ecoop26 .
```

Both x86_64 and ARM64 platforms are supported. Building from source requires an internet connection (to download Z3, HoICE, and Extended_ConSORT) and takes 10–15 minutes on x86_64. On ARM64, Z3 4.11.2 is compiled from source via opam because no pre-built ARM64 Linux binary exists; this adds approximately 10 minutes to the build and requires at least 8 GB of memory allocated to Docker.

This drops you into a shell at `/home/opam/app/myproject/`.

A.2 Quick Verification (10 minutes)

Run the kick-the-tires script:

```
bash eval/run_short.sh
```

This runs a small subset of benchmarks from Tables 1 and 3 and completes within 10 minutes. You should see a few nested array benchmarks with timing output (ownership time, refinement time, total time), one integer array benchmark from Table 3 comparing our tool with Extended_ConSORT, and a final summary indicating which paper tables correspond to each output.

A.3 Running a Single Benchmark

To verify a single program manually:

```
dune exec myproject -- ./example/nested_arrays/swap.imp
```

Expected output includes `ownership: sat` and `refinement: sat` (or `unsat` for programs in `negative_example/`).

B Evaluation Instructions

B.1 Full Evaluation

To reproduce all tables from the paper:

```
bash eval/run_all.sh
```

This takes approximately 2 hours (Tables 1 and 3 take about 30 minutes; Table 4 takes 1–2 hours as it runs all 35 benchmarks with two Z3 versions). Results are saved as Markdown and CSV files in `eval/results/`.

B.2 Individual Tables

Each table can be reproduced independently:

Paper	Script	Output	Time
Table 1	`eval/table1.sh`	`eval/results/table1.md`	∼10 min
Table 3	`eval/table3.sh`	`eval/results/table3.md`	∼10 min
Table 4	`eval/table4.sh`	`eval/results/table4.md`	∼60 min
Table 5	`eval/table5.sh`	`eval/results/table5.md`	<1 min

CSV files with raw data are also produced in `eval/results/` for further analysis.

B.3 Claims and Evidence Mapping

Paper	Output File	What to Check
Table 1	`table1.md`	Timing and alias counts for 19 nested array benchmarks
Table 3	`table3.md`	Timing comparison with Tanaka et al. on 8 integer benchmarks
Table 4	`table4.md`	Z3 4.14.1 vs. 4.11.2 timing for all 35 benchmarks
Table 5	`table5.md`	Summary: how many benchmarks each Z3 version is faster

B.4 Expected Variation

- **Absolute timing** will differ from the paper due to hardware differences. The paper's experiments were conducted on a specific machine; the Docker container will have different CPU characteristics. Table 4 uses a single run (vs. the paper's 10-run mean), so individual timing values may show more variance, but the relative trends between Z3 versions should be consistent.
- **Nondeterminism**: Due to the nondeterminism of the solution search procedure in our tool, some benchmarks may experience timeouts even though all benchmarks succeed in the paper.
- **Relative ordering** of benchmarks by time should be approximately consistent with the paper.
- **Sat/unsat results** must match the paper exactly. Every benchmark in `positive_example/` and `nested_arrays/` should report `sat`; every benchmark in `negative_example/` should report `unsat`.

C Directory Structure

The project is organised as follows (relevant subset):

```
nested_array_ConSORT/
  myproject/
    bin/main.ml       -- Entry point (CLI argument parsing)
    lib/              -- Core library modules
                         (syntax, parser, simple typing,
                          ownership & CHC constraint generation,
                          workflow orchestration)
    example/
      nested_arrays/  -- 19 nested array benchmarks (Table 1)
      int_arrays/     -- 8 integer array benchmarks (Table 3)
      positive_example/
      negative_example/
    eval/
      run_all.sh      -- Master evaluation script
      run_short.sh    -- Kick-the-tires evaluation
      table1.sh table3.sh table4.sh table5.sh
      lib/common.sh   -- Shared helper functions
      z3-versions/    -- Z3 binaries (4.11.2 and 4.14.1)
      Extended_ConSORT/ -- Tanaka et al. tool (Table 3)
      results/        -- Generated result files
    experiment/       -- Intermediate solver files (runtime)
```

D Troubleshooting

"busy" error when running `dune`

If `dune` reports a "busy" error, use an alternative build directory:

```
dune exec --build-dir="_tmp" myproject -- \
    ./example/nested_arrays/indexed_value.imp
```

Z3 or HoICE not found

Both solvers must be on `$PATH`. Inside the Docker container, they are already installed. Outside Docker, check with:

```
which z3
which hoice
```

Container runs out of memory

The evaluation requires at least 4 GB of RAM (8 GB when building on ARM64). If benchmarks are killed, increase Docker's memory allocation in Docker Desktop settings.

E Notes

The tool invokes Z3 [2] and HoICE [1] as external processes. The evaluation scripts switch between Z3 versions by manipulating the `PATH` environment variable. All intermediate solver files are written to `experiment/` within the project directory. Due to this shared state, benchmarks must be run sequentially (not in parallel).

References

1 Adrien Champion, Tomoya Chiba, Naoki Kobayashi, and Ryosuke Sato. Ice-based refinement type discovery for higher-order functional programs. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I*, Lecture Notes in Computer Science, pages 365–384. Springer, 2018. doi:10.1007/978-3-319-89960-2_20.

2 Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

3 Izumi Tanaka, Ken Sakayori, and Naoki Kobayashi. Ownership types for verification of programs with pointer arithmetic. In *Proceedings of the 2024 ACM SIGPLAN International Workshop on Partial Evaluation and Program Manipulation*, PEPM 2024, pages 94–106, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3635800.3636965.