


Low-Level Bi-Abduction (Artifact)

Lukáš Holík ✉ 🏠 

FIT, Brno University of Technology, Czech Republic

Petr Peringer ✉ 🏠 


FIT, Brno University of Technology, Czech Republic

Adam Rogalewicz ✉ 🏠 

FIT, Brno University of Technology, Czech Republic

Veronika Šoková ✉ 🏠 

FIT, Brno University of Technology, Czech Republic

Tomáš Vojnar ✉ 🏠 

FIT, Brno University of Technology, Czech Republic

Florian Zuleger ✉ 🏠 

Faculty of Informatics, TU Wien, Austria

Abstract

Broom is a new static analyzer for C written in OCaml. Broom primarily aims at open programs, i.e., fragments of programs, with dynamic pointer-linked data structures – in particular, various kinds of lists – that employ advanced low-level pointer operations. It is based on separation logic and the

principle of bi-abductive reasoning. The artifact is a VirtualBox image of a Linux machine with Ubuntu 20.04 operating system. It contains source code and binary of the Broom tool, benchmarks, and scripts for running our and the competing tools we compare to.

2012 ACM Subject Classification Theory of computation → Separation logic; Theory of computation → Logic and verification; Software and its engineering → Formal software verification

Keywords and phrases programs with dynamic linked data structures, programs with pointers, low-level pointer operations, static analysis, shape analysis, separation logic, bi-abduction

Digital Object Identifier 10.4230/DARTS.8.2.11

Funding The Czech authors were supported by the project 20-07487S of the Czech Science Foundation, the FIT BUT internal project FIT-S-20-6427, and L. Holík by the ERC.CZ project LL1908.

Related Article Lukáš Holík, Petr Peringer, Adam Rogalewicz, Veronika Šoková, Tomáš Vojnar, and Florian Zuleger, “Low-Level Bi-Abduction”, in 36th European Conference on Object-Oriented Programming (ECOOP 2022), LIPIcs, Vol. 222, pp. 19:1–19:30, 2022.

<https://doi.org/10.4230/LIPIcs.ECOOP.2022.19>

Related Conference 36th European Conference on Object-Oriented Programming (ECOOP 2022), June 6–10, 2022, Berlin, Germany

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2022 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

This artifact includes the Broom tool described in Section 8 of the related article. Its purpose is to allow the reproduction of the examples discussed in Section 2 and in the appendices, and the benchmarks in Section 8.4.



© Lukáš Holík, Petr Peringer, Adam Rogalewicz, Veronika Šoková, Tomáš Vojnar, and Florian Zuleger; licensed under Creative Commons License CC-BY 4.0

Dagstuhl Artifacts Series, Vol. 8, Issue 2, Artifact No. 11, pp. 11:1–11:6



DAGSTUHL
ARTIFACTS SERIES

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



11:2 Low-Level Bi-Abduction (Artifact)

2 Content

The artifact package includes:

- broom
 - code-listener/ – the Code Listener framework (a dependency of Broom)
 - doc/ – documentation generated from the source code
 - examples/ – examples presented in Figures 1, 8, 9, 10, and 11
 - scripts/ – a collection of useful scripts including broom
 - src/ – source code of the Broom tool
 - tests/ – benchmarks
 - README.md –
- broom-long-output/ – output of script `./run-long.sh`
- infer-output/ – output for Infer of script `./run-short.sh`
- gillian-output/ – output for Gillian of script `./run-short.sh`
- download-infer.sh – a script to download Infer v1.1.0
- run-long.sh – a script to run the benchmark for Table 2
- run-short.sh – a script to run the benchmark for Table 1
- show-tab1.sh – a script to show Table 1
- show-tab2.sh – a script to show Table 2
- ECOOP2022_techrep.pdf – the technical report of the related article
- README.md – a tutorial on how to use the 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). In addition, the artifact is also available at: <https://pajda.fit.vutbr.cz/rogalew/broom>.

4 Tested platforms

To get the results presented in the paper, the experiments were run on a machine with an Intel i7-4770 processor with 32 GiB of memory and the Linux operating system (Fedora 34). The current implementation of Broom uses a single core only. The tool was tested also on a machine with a 2.3 GHz Intel Core i5 processor with 8GB RAM, running macOS or Ubuntu 20.04.

This artifact contains a VirtualBox image (Ubuntu 20.04). The hardware capabilities offered by the virtual machine should fulfill the above requirements. All required software is pre-installed.

5 License

The artifact is available under the GPLv3 license.

6 MD5 sum of the artifact

101f2587490f8f755abecd453e2c01d1

7 Size of the artifact

4.9 GiB

8 Getting started

1. Extract the file `broom-ecoop22.ova`.
2. Download and install VirtualBox¹ if you do not have it already.
3. Open VirtualBox and select *File > Import Appliance...* Now, select OVA file in the import box and verify the settings in the center window. Make any changes if you need to in that center window. If you want to run Infer or Gillian, enable internet connection (optional). Click *Import* at the bottom.
4. Run the appliance. The login and password of the default user are: `osboxes / osboxes.org`.
5. Continue with the instructions in the next section (or in `${HOME}/artifact/README.md`).

9 Usage

9.1 Executing running examples from the paper

The source code of the examples is available here:

- `examples/01-circ_dll.c` for Fig. 1
- `examples/02-cond.c` for Fig. 8
- `examples/03-loop.c` for Fig. 9
- `examples/04-nondet_cond.c` for Fig. 10
- `examples/05-bitwise.c` for Fig. 11

For executing all the examples at once:

1. Go to the directory `broom`:

```
cd ${HOME}/artifact/broom
```

2. Execute the following command:

```
make examples
```

3. After a successful execution, Broom creates outputs in `_build/examples/*.txt`

One can also analyze each program `file.c` separately as follows:

```
./scripts/broom --print-cl --verbose=3 --display-stats -- file.c
```

Differences between the generated output and the description in the paper

Compared to the separation logic described in Section 5 of the related article, Broom uses a slightly different notation:

- `base(x) ≡ b(x)`
- `len(x) ≡ c(x) - x`
- `x-(c)->y` stands for $x \mapsto y$ and we have $c = \text{size}(y)$; we write c in the points-to arrow of the tool output in order to make it easier for the reader to track the size of y
- `x-(size)->T` stands for $x \mapsto \top[\text{size}]$ where size is some expression
- `x-(size)->0` stands for $x \mapsto 0[\text{size}]$ where size is some expression

¹ <https://www.virtualbox.org/>

11:4 Low-Level Bi-Abduction (Artifact)

The axioms of page 9 from the paper then become

$$\begin{aligned} \forall l. \text{len}(l) \geq 0 \wedge l \geq \text{base}(l) \wedge \text{len}(\text{base}(l)) - l = \text{len}(l) \wedge \text{base}(l) = 0 &\Leftrightarrow \text{len}(l) = 0 \\ \forall l, l'. (0 < \text{base}(l) < l' \Leftrightarrow \text{base}(l) + \text{len}(\text{base}(l)) < l' &\Leftrightarrow \text{base}(l') < l \Leftrightarrow \text{base}(l') + \text{len}(\text{base}(l')) \\ &\Rightarrow \text{base}(l) = \text{base}(l') \end{aligned}$$

Note that we can always recover $\mathfrak{c}(x)$ from this presentation by setting $\mathfrak{b}(x) = \text{base}(x)$ and $\mathfrak{c}(x) = \text{base}(x) + \text{len}(\text{base}(x))$.

For the current state formula denoted as Q in the paper (and printed as *Curr* by Broom), all variables that do not also appear in the pre-condition denoted as P in the paper (and printed as *Miss* by Broom) are assumed to be existentially quantified; however, we do not print the quantifiers in order not to clutter the presentation. For example, for $P : x = X \wedge X \mapsto Y$ and $Q : x = X \wedge X \mapsto u$, the formula Q has to be understood as $\exists u. x = X \wedge X \mapsto u$ because u does not appear in the pre-condition P .

9.2 Reproducing results for Table 1

The below table describes how benchmark names from the paper match to the corresponding source code in the `broom/tests` folder.

Shortcut	File
circ-DLL	lists/circ_dll_simple.c
circ-DLL-err	lists/circ_dll_simple-err.c
circ-DLL-embedded	lists/circ_dll_embedded_int.c
Linux-list-1	linux-list/predator-test-0156-no-include.c
Linux-list-2	linux-list/linux-list-t2.c
Linux-list-2-err	linux-list/linux-list-t2-err.c
Linux-list-all	linux-list/linux-list.c
intrusive-list	ideas/intrusive-list.c
intrusive-list-min	ideas/intrusive-list-minimal-example.c
intrusive-list-smoke	ideas/test_intrusive_single_file.c

For executing all benchmarks at once:

1. Go to the directory `artifact`:

```
cd ${HOME}/artifact
```

2. Optionally: execute the following command which will download Infer v1.1.0 into the folder `infer-linux64-v1.1.0`:

```
./download-infer.sh
```

3. Optionally: download the source code of Gillian (PLDI'20 version) from <https://github.com/GillianPlatform/Gillian/releases/tag/PLDI20>. In case of problems see <https://gillianplatform.github.io/>. For installation:

```
sudo apt install npm
npm install -g esy --prefix ~/.npm
```

```
# to .profile
```

```
# PATH="$HOME/.npm/bin:$PATH"

cd Gillian-PLDI20 # the folder containing 'esy.json'
# in 'esy.json' change "@opam/odoc": "*" to "@opam/odoc": "1.5.0"
rm -rf esy.lock
esy
esy install
```

- Execute the following command which will execute Broom, Infer (optional), and Gillian (optional) on the subset of benchmarks listed above:

```
./run-short.sh
```

- After a successful execution, Broom creates outputs in `broom-output-now`, Infer in the folder `infer-output-now` (or you can use `infer-output`), and Gillian in `gillian-output-now` (or you can use `gillian-output`).
- Execute the following command which will print the wall time correspondingly to Table 1. The number of functions for which complete contracts were produced (columns 5, 8, and 11 of Table 1) has been manually checked, i.e., we took the output of the tools (see output folders from Point 5) and manually inspected all produced contracts for completeness to get the numbers in the table.

```
./show-tab1.sh
```

9.3 Reproducing results for Table 2

The below table describes how benchmark names from the paper match to the corresponding source code in the `broom/tests` folder.

Shortcut	File
<code>intrusive-list-min</code>	<code>ideas/intrusive-list-minimal-example.c</code>
<code>intrusive-list-min-2</code>	<code>ideas/intrusive-list-minimal-example2.c</code>
<code>intrusive-list-min-3</code>	<code>ideas/intrusive-list-minimal-example3.c</code>
<code>intrusive-list-smoke</code>	<code>ideas/test_intrusive_single_file.c</code>

This experiment is rather time-consuming, so Point 2 may be skipped. For executing all experiments at once:

- Go to the directory `artifact`:

```
cd ${HOME}/artifact
```

- Optionally: execute the following command which will execute Broom on the subset of benchmarks listed above:

```
./run-long.sh
```

- After a successful execution, Broom creates outputs in `broom-long-output-now` (or you can use `broom-long-output`).
- Execute the following command which will print the wall time correspondingly to Table 2. The number of functions for which complete contracts were produced (column 5) has been manually checked, i.e., we took the output of the tools (see output folders from Point 3) and manually inspected all produced contracts for completeness to get the numbers in the table.

```
./show-tab2.sh
```

9.4 Executing Broom in general

The file `broom/README.md` provides more information about how to install the Broom tool on different operating systems and how to execute it. The file `broom/options.md` provides more information about the input parameters of Broom. One can also get this information by executing `./scripts/broom -h`. Additionally, the file `broom/doc/index.html` contains automatically generated documentation of the source code.

One can experiment with the presented files, and more C code can be found in the `broom/tests/` folder.

A smaller example than those used in Table 2 to illustrate the difference in abduction strategies is available here:

```
./scripts/broom -- tests/call-01-ok_gcc.c          # contracts incomplete
./scripts/broom --abduction-strategy=1 -- tests/call-01-ok_gcc.c
```

An example of memory leak detection after a program terminates (when Valgrind reports *still reachable*):

```
./scripts/broom -- tests/global-mem-leaks-err.c    # memory leaks from
                                                    # global variables
./scripts/broom --no-exit-leaks -- tests/global-mem-leaks-err.c
```