Mix Testing: Specifying and Testing ABI Compatibility of C/C++ Atomics Implementations

LUKE GEESON, University College London and Arm Ltd, United Kingdom
JAMES BROTHERSTON, University College London, United Kingdom
WILCO DIJKSTRA, Arm, United Kingdom
ALASTAIR F. DONALDSON, Imperial College London, United Kingdom
LEE SMITH, Arm, United Kingdom
TYLER SORENSEN, University of California at Santa Cruz, USA
JOHN WICKERSON, Imperial College London, United Kingdom

ACM Reference Format:
Luke Geeson, James Brotherston, Wilco Dijkstra, Alastair F. Donaldson, Lee Smith, Tyler Sorensen, and John Wickerson. 2024. Mix Testing: Specifying and Testing ABI Compatibility of C/C++ Atomics Implementations. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 287 (October 2024), 6 pages. https://doi.org/10.1145/3689727

1 Artifact

1.1 Introduction

The artifact consists of scripts to reproduce the results of running figures in the paper.

1.1.1 *Paper Claims.*

(1) Running the code in Fig. 1(a) through `herd`, using the `rc11+lb.cat` model, produces the execution shown in Example 2.2.
(2) Likewise, Running Fig. 1(d) under `aarch32.cat` produces the outcomes in Example 3.5.
(3) Running Fig. 1(b) under `aarch32.cat` produces the outcomes that match those in Example 2.2 (after mapping source locations to machine registers).
(4) Running Fig. 9(left) under `aarch64.cat` produces the same outcomes as Fig. 9(right).
(5) Running Fig. 11(left) under `rc11+lb.cat` produces the outcomes in Fig. 11(middle).
(6) Running Fig. 17(left) under `aarch64.cat` produces the outcomes in Fig. 17(right).
(7) Running Fig. 12(left) under `rc11+lb.cat` produces the outcomes in Fig. 12(middle).
(8) Running Fig. 13(left) under `aarch64.cat` produces the outcomes in Fig. 12(right).
(9) Running Fig. 14 (left) under `rc11+lb.cat` produces the outcomes in Fig. 14(middle).
(10) Running Fig. 15(left) under `aarch64.cat` produces the outcomes in Fig. 14(right).
(11) Compiling and Running Fig. 16 on GCC and LLVM produces different results.

Authors' Contact Information: Luke Geeson, University College London and Arm Ltd, London, United Kingdom, luke.geeson@cs.ucl.ac.uk; James Brotherston, University College London, London, United Kingdom, j.brotherston@ucl.ac.uk; Wilco Dijkstra, Arm, Cambridge, United Kingdom, wilco.dijkstra@arm.com; Alastair F. Donaldson, Imperial College London, London, United Kingdom, alastair.donaldson@imperial.ac.uk; Lee Smith, Arm, Cambridge, United Kingdom, lee.d.smith@acm.org; Tyler Sorensen, University of California at Santa Cruz, Santa Cruz, USA, tyler.sorensen@ucsc.edu; John Wickerson, Imperial College London, London, United Kingdom, j.wickerson@imperial.ac.uk.

ACM 2475-1421/2024/10-ART287
https://doi.org/10.1145/3689727

A number of minor claims appear in the paper, like how we implemented a 128-bit signed integer type in `herd`. To keep this document small we refer the reader to the figures that use these features. To validate the bug reports, please see our bug board[1]

1.1.2 How Delivered. The artifact is available on Zenodo and consists of a Docker container with the scripts to reproduce figures.

1.1.3 Artifact checklist.

1. **Algorithm:** mix testing (Fig. 3 of the paper).
2. **Program:** The `herd` [1] simulator.
3. **Models:** From herd toolsuite [1].
4. **Data Set:** Tests provided in `expected` directory.
5. **Test Environment/Binary:** Docker Ubuntu 20.04.
6. **Hardware:** Either x86-64 or Arm AArch64 machines.
7. **Run-time State:** not sensitive to run-time state.
8. **Metrics:** Outcomes of executing tests under models.
9. **Output:** Console and `.log` files.
10. **Experiments:** Makefile provided reproduces results.
11. **Disk-space requirements:** 5GB for Docker image.
12. **Time needed to prepare workflow**: Everything is ready.
13. **Time needed to complete experiments**: 15 minutes.
14. **Licenses:** CeCILL-B license.
15. **Workflow Frameworks:** Makefile, GNU Parallel [2].
16. **Archived(DOI):** 10.5281/zenodo.12667763
17. **Zenodo URL:** https://zenodo.org/doi/10.5281/zenodo.12667763
18. **Available:** Zenodo or Docker Hub[2].

1.2 Hardware dependencies

Either an Intel x86-64 or Arm AArch64 based machine. The artifact was tested using a MacBook Pro with a dual-core Intel i7 CPU, a Lenovo P720 with 2xIntel Xeon Gold 5120T CPUs (56 cores), a MacBook Air with an 8-core Apple M1 (Arm AArch64), a Cavium Thunder X2 with 2x28-core CPUs (Arm AArch64), and under x86-64 emulation (using the M1 machine).

1.3 Getting started guide

1.3.1 Software Dependencies. The artifact requires a machine that supports Docker running a Linux distribution such as Ubuntu. For example on Ubuntu 20.04 you can install docker using the guide[3].

1.3.2 Automatic Installation (easiest). Install docker builds for either x86-64 or AArch64:

```
> docker pull lukeg101/oopsla24-artifact:latest
```

Then run:

```
> docker run -it lukeg101/oopsla24-artifact:latest
```

1.3.3 Manual Installation.

1. Download `oopsla24-artifact-arch.tar.gz` from Zenodo (where `arch` is `arm64` or `x86`).
2. Load the Docker container:

[1]https://lukegeeson.com/blog/2023-10-17-Telechat-Bug-Board/
[2]https://hub.docker.com/r/lukeg101/oopsla24-artifact/tags
[3]https://docs.docker.com/desktop/install/ubuntu/

```
> docker load -i oopsla24-artifact-arch.tar.gz
```

(3) Run the Image:

```
> docker run -it lukeg101/oopsla24-artifact
```

This runs the Ubuntu image and mounts the current directory into the container at `artifact-output`.

1.3.4 Kick-the-tires phase. A `Makefile` drives the `herd` tool to reproduce the provided program behaviours, to run the "kick-the-tires-phase" in the container, type:

```
artifact> make figs
```

1.4 Step by step instructions

1.4.1 Evaluation and Expected Results. We assume you are running the artifact with a clean directory. To evaluate a claim, run each make command below, or run them all at once using:

```
artifact> make figs
```

and compare the expected results and timings of each (all run on an Apple M1 machine):

Claim	Command	Time Required	Expected Result (should match on stdout)
1	`make fig1a`	< 1 minute	See `~/expected/fig1a/outcomes.log`
2	`make fig1d`	< 1 minute	`~/expected/fig1d/outcomes.log`
3	`make fig1b`	< 1 minute	`~/expected/fig1b/outcomes.log`
4	`make fig9`	< 1 minute	`~/expected/fig9/outcomes.log`
5	`make fig11`	< 1 minute	`~/expected/fig11/outcomes.log`
6	`make fig17`	< 1 minute	`~/expected/fig17/outcomes.log`
7	`make fig12`	< 1 minute	`~/expected/fig12/outcomes.log`
8	`make fig13`	< 1 minute	`~/expected/fig13/outcomes.log`
9	`make fig14`	< 1 minute	`~/expected/fig14/outcomes.log`
10	`make fig15`	< 1 minute	`~/expected/fig15/outcomes.log`
11	`make fig16`	< 1 minute	stdout of running both programs will differ

1.4.2 Expected warnings or errors (see `~/artifact-output/Output/logs/errors.log`).

- *unrolling limit exceeded, legal outcomes may be missing*. This is a standard warning emitted by `herd` when simulating tests with loops, for our tests this is fine.

1.5 The Atomics Application Binary Interface Standard for the Arm 64-bit Architecture

We claim that we developed an atomics Application Binary Interface for the Arm architecture. Please see the `atomicsabi64.pdf` for the latest version of this document, and refer to the Arm software GitHub for updates to the document: https://github.com/ARM-software/abi-aa/pull/256.

1.6 Reusability guide

1.6.1 Core of the Artifact. The core of the artifact is the `herd` tool [1]. We provide the figures in the paper, and check their behaviour under models using herd. Documentation of the artifact is provided in the `README.md`.

1.6.2 Experiment Customisation. You can customise the experiments when invoking Make:

- Set source model (default `rc11+lb.cat`, options: `c11_partialSC.cat`,`c11_simp.cat`,`rc11.cat`, `rc11+lb.cat`):

  ```
  artifact> make figs CMEM=c11_simp.cat
  ```

- Set architecture model for figures featuring assembly tests (default `aarch32.cat`, options: `arm.cat`, `aarch64.cat`):

```
artifact> make figs AMEM=aarch32.cat
```

- Set simulation timeout, (default `120.0` seconds):

```
artifact> make figs TIMEOUT=1.0
```

warning: choosing an incompatible model will cause simulation to fail. This is because the model is used to choose which parser is used on the test in question, if the C Parser is used to parse an assembly test, it will fail. This is expected and a limitation of how herd works. Running each figure individually under a specific model (ie `make fig1d AMEM=arm.cat`) works in this case. If parsing succeeds (this can happen for instance with `arm.cat` and `aarch32.cat` tests which share the same parser), then the hash of the final outcomes may differ, once again causing a failure. This is also expected. The tests outcomes are the focus of our needs.

1.7 Replicating results without the artifact

It is possible to replicate the results by building `herd` from source, and running the tests in the `expected` directory directly. To build `herd` from source, please run:

```
> git clone https://github.com/herd/herdtools7
> cd herdtools7
> make
```

Note: you may need to install some dependencies.

Once you have successfully built `herd`, you can run the tests provided in the `expected` directory:

```
> ./_build/install/default/bin/herd7 \
    -model herd/libdir/rc11.cat      \
    -I herd/libdir                   \
    ~/expected/fig1a/fig1a.litmus
```

which should produce outcomes much like the artifact.

Note: the hash will differ on the output from the source build of `herd` *since we fix the build of herd in the artifact. The outcomes of executing a given test should however match the artifact.*

Note: the `rc11+lb.cat` *model is not provided by* `herd`*—you will need to get it from the artifact.*

1.8 Licenses

Atomic-Mixer consists of an External Module for use in connection with the Telechat Software.

Luke Geeson is the author of the atomic-mixer code. Atomic-mixer is separately Licensed under CeCILL-B. Please see ATOMICMIXERLICENSE.txt in the artifact.

Telechat consists of an External Module for use in connection with the herdtools Software.

Luke Geeson is the author of the Telechat code. Telechat is separately Licensed under CeCILL-B. Please see TELECHATLICENSE.txt

The herdtools suite is Licensed under CeCILL-B except as stated in LICENSE.txt, a copy of which is provided with this artefact. Jade Alglave and Luc Maranget are authors of the herdtools suite. Please see LICENSE.txt.

1.9 Frequently Asked Questions (FAQ)

"I get the clang error* `error: unknown target CPU 'armv8'`*": We have fixed this issue in v2 (or above) of the artifact.

"How do I interpret these results?": We provide expected results for each figure in, for example '~/expected/fig1b/outcomes.log'. The output of simulating each figure (on the right the output above) is compared against the expected outcomes in each outcome.log file (on the left). The outcomes in each figure should match their expected (and so the output you provide is correct). The logs show that simulation is identical to the expected outcomes, with the exception of fig19 where we are intentionally showing that LLVM and GCC produce different results for the same test (a bug we found as part of this work).

***Can you please elaborate on how to map your output with the content of the paper?*:** Consider:

```
// ~/expected/fig1b/outcomes.log
Test fig1b Allowed
States 3
0:R0=0; 1:R0=1;
0:R0=1; 1:R0=0;
0:R0=1; 1:R0=1;
No
Witnesses
Positive: 0 Negative: 3
Flag Assuming-common-inner-shareable-domain
Condition exists (0:R0=0 /\ 1:R0=0)
Observation fig1b Never 0 3
Hash=9b2b8b9d42de1ca604bdef60308a1d40
// EOF
```

Ignore everything (the herd tool we rely on is quite verbose in its operation), but the lines:

```
0:R0=0; 1:R0=1;
0:R0=1; 1:R0=0;
0:R0=1; 1:R0=1;
```

Which shows an outcome (a set of assignments on each line - think one possible test output), and the consider Example 2.2 of the camera-ready paper:

```
{ P0:R0=0; P1:R0=1; }
{ P0:R0=1; P1:R0=0; }
{ P0:R0=1; P1:R0=1; }
```

Herd uses a terse syntax for outcomes, which we clean up in the paper so it is accessible. So `0:R0` is equivalent to `P0:R0`, which means thread `P0`, local variable/register `R0`, and we add brackets around each outcome `{ P0:R0=0; P1:R0=1; }` to make it clear that each line is a set. With this in mind observe that each outcome matches one in the paper exactly.

Which folders can be deleted before running the experiment to verify the results: if you run the docker container afresh you shouldn't need to delete anything - it's all set up, however if you have run experiments already, you can delete the contents of ~`//artifact-output/Output`. This removes any output files generated during execution. All other folders are required to operate, and the provided README in the container explains the function of these folders.

1.10 Changelog

v3: Rename and merge figures in line with stream-lined camera ready paper, added an FAQ, added Licensing sections.

v2: Fixed a clang error with the struct figure, now works on both x86 and AArch64 builds.

v1: Initial artifact.

Acknowledgments

We thank the reviewers for the time they spent checking this artifact. We thank Earl Barr, Richard Grisenthwaite, Al Grant, Kyrylo Tkachov, Tomas Matheson, Sam Ellis, Nick Gasson, Arm's Compiler

Teams and Arm Architecture & Technology Group for their feedback and assistance. This work was supported by the Engineering and Physical Sciences Research Council [EP/V519625/1]. This work was also supported by EPSRC project [EP/R006865/1] The views of the authors expressed in this paper are not endorsed by Arm or any other company mentioned.

References

[1] Jade Alglave and Luc Maranget. 2021. herdtools7. https://github.com/herd/herdtools7. Accessed: 2019-10-06.

[2] O. Tange. 2011. GNU Parallel - The Command-Line Power Tool. *;login: The USENIX Magazine* 36, 1 (Feb 2011), 42–47. http://www.gnu.org/s/parallel

Received 2024-04-06; accepted 2024-08-18; revised 05 July 2024