A. Artifact Appendix

A.1 Artifact check-list (meta-information)

- **Program:** The code repository for our framework along with the test suite. Note that this is already setup in the docker image.
- **Compilation:** The Lean4 toolchain, downloaded via `elan`. Note that this is already setup in the docker image.
- **Run-time environment:** Any operating system that supports Docker.
- **Hardware:** Any x86-64 machine.
- **Output:** Key theorems of the paper will be built and shown to have no unsound axioms.
- **How much disk space required (approximately)?:** 10GB
- **How much time is needed to prepare workflow (approximately)?:** 1hr
- **How much time is needed to complete experiments (approximately)?:** 1hr
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** MIT
- **Archived (provide DOI)?:** 10.5281/zenodo.11519739

A.2 Description

A.2.1 Software dependencies

Docker is necessary to run our artifact. The Docker image has all dependencies needed to compile our framework with Lean4.

A.3 Experiment workflow

Access the docker image `opencompl-ssa` from `https://zenodo.org/records/11519739`.

```
$ docker load -i opencompl-ssa.tar
$ docker run -it siddudruid/opencompl-ssa
# | This clears the build cache,
# | fetches the maths library from the build cache,
# | and builds our framework.
$ cd /code/ssa && lake clean && lake exe cache get && lake build
# | This allows to check that the key theorems of our framework are
# |guarded, and that they do not contain `sorry`s.
$ rg -g "**/*.lean" "#guard_msgs in #print axioms" -C2 | grep "sorry"
$ rg -g "**/*.lean" "#guard_msgs in #print axioms" -C2
```

A.4 Evaluation and expected results

On running `lake build`, the build succeeds with no errors.

Next, run:

```
$ rg -g "**/*.lean" "guard_msgs in #print axioms" -C2 | grep "sorry"
```

Grep returns no matches, which checks that all guarded theorems do not use the sorry axiom. To manually inspect this, run:

```
$ rg -g "**/*.lean" "#guard_msgs in #print axioms" -C2
```

In the output, observe the following lines, which tells us that, for example, the theorem `denote_rewritePeepholeAt` depends only on the axioms `propext`, `Classical.choice`, and `Quot.sound`. That is, this theorem *does not depend* on axioms such as `sorryAx` that compromise the correctness of Lean's verification.

Now, since the Lean build was successful in the previous step, and since we use standard Lean axioms, we are justified in our claims that we have mechanized the key theorem statements from the paper.

Below, we list the key theorems that we claim to have mechanized, and their guarded verification below. Please inspect the output of (`$ rg -g "**/*.lean" "#guard_msgs in #print axioms" -C2 | grep "sorry"`) and confirm that these guarded statements occur, with the guard containing only the axioms `propext`, `Classical.choice`, and `Quot.sound`.

A.4.1 Core Framework Theorems

The core correctness claim of the peephole rewriter is guarded below:

```
SSA/Core/Framework.lean
2422-
2423-/-- info: 'denote_rewritePeepholeAt' depends on axioms: [propext, Classical.choice, Quot.sound] -/
2424:#guard_msgs in #print axioms denote_rewritePeepholeAt
2425-
2426-/- repeatedly apply peephole on program. -/
--
2458-
2459-/-- info: 'denote_rewritePeephole' depends on axioms: [propext, Classical.choice, Quot.sound] -/
2460:#guard_msgs in #print axioms denote_rewritePeephole
2461-
2462-end SimpPeepholeApplier
```

A.4.2 Five Hardest Alive Examples

The correctness claim of the hardest Alive rewrites that time out on an SMT solver, which we verify for arbitrary bitwidth:

```
SSA/Projects/InstCombine/AliveHandwrittenLargeExamples.lean
53-/--info: 'AliveHandwritten.DivRemOfSelect.alive_DivRemOfSelect' depends on
54-axioms: [propext, Classical.choice, Quot.sound] -/
55:#guard_msgs in #print axioms alive_DivRemOfSelect
56-
57-end DivRemOfSelect
--
215-/--info: 'AliveHandwritten.MulDivRem.alive_simplifyMulDivRem805' depends on axioms:
216-[propext, Classical.choice, Quot.sound] -/
217:#guard_msgs in #print axioms alive_simplifyMulDivRem805
218-
219-open Std (BitVec) in
--
302-/--info: 'AliveHandwritten.MulDivRem.alive_simplifyMulDivRem805'' depends on axioms:
303-[propext, Classical.choice, Quot.sound] -/
304:#guard_msgs in #print axioms alive_simplifyMulDivRem805'
305-
306-/-
--
357-/-- info: 'AliveHandwritten.MulDivRem.alive_simplifyMulDivRem290' depends on
358-axioms: [propext, Classical.choice, Quot.sound]-/
359:#guard_msgs in #print axioms alive_simplifyMulDivRem290
360-
361-end MulDivRem
--
416-/-- info: 'AliveHandwritten.AndOrXor.alive_simplifyAndOrXor2515' depends on
417-axioms: [propext, Classical.choice, Quot.sound] -/
418:#guard_msgs in #print axioms alive_simplifyAndOrXor2515
419-
420-/-
--
532-/-- info: 'AliveHandwritten.Select.alive_simplifySelect764' depends on axioms:
533-[propext, Classical.choice, Quot.sound] -/
534:#guard_msgs in #print axioms alive_simplifySelect764
535-
536-end Select
```

A.5 Paper Code Examples

- *Figure 1, 2* is at `SSA/Projects/FullyHomomorphicEncryption/PaperExamples.lean`.
- *Figure 3, 4* is at `SSA/Core/Framework.lean`.
- The statement and proof of `denote_rewritePeephole` can be found at `SSA/Core/Framework.lean`.
- The definitions and proofs of DCE can be found in `SSA/Projects/DCE/DCE.lean`, and of CSE in `SSA/Projects/CSE/CSE.lean`.
- The examples for bitvector rewrites in found at `SSA/Projects/InstCombine/PaperExamples.lean`.
- The hand-written examples are found at `SSA/Projects/InstCombine/AliveHandwrittenLargeExamples.lean`.
- All FHE examples can be found in `SSA/Projects/FullyHomomorphicEncryption/PaperExamples.lean`.

A.6 Miscellanous Docker Usage

To copy files for inspection from the docker container into the host, keep the container running, and in another shell instance, use the `docker cp` command to copy files from within the container out to the host:[1]

[1] For more about `docker cp`, please see: (`https://docs.docker.com/engine/reference/commandline/cp/`)

```
$ docker container ls # find   ID
$ docker cp <CONTAINERID>:<PATH/INSIDE/CONTAINER> \
            <PATH/OUTSIDE/CONTAINER>
```