Artifact for Interactive Bit Vector Reasoning using Verified Bitblasting

.1 Artifact check-list (meta-information)

A Introduction

This artifact contains the infrastructure and tooling necessary to display the performance of the verified bitblaster introduced by the paper for all the benchmarks presented.

- **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 (16 physical cores and 128GB of RAM recommended).
- **Output:** Key theorems of the paper will be built and shown to have no unsound axioms.
- **How much disk space required (approximately)?:** 80GB
- **How much time is needed to prepare workflow (approximately)?:** 1hr
- **How much time is needed to complete experiments (approximately)?:** 8hr (on recommended hardware)
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** Apache License 2.0
- **Archived (provide DOI)?:** 10.5281/zenodo.15755236

A.1 Performance

We test the performance of the verified bitblaster `bv_decide` against three benchmarks:

- `InstCombine` benchmark, extracted from LLVM's peephole verifier
- `HackersDelight` benchmark, containing bit-vector theorems extracted from the first two chapters of Hackers' Delight
- `SMT-LIB` benchmark, containing the problems from SMT-LIB's 2024 Competition

The artifact contains the benchmarks, the scripts to evaulate `bv_decide`'s performance, and the scripts to reproduce the plots in the paper.

B Hardware Dependencies

Podman or Docker are necessary to run our artifact. The container image has all dependencies needed to compile our framework with Lean4. TODO: add info

C Getting Started

Access the docker image from 10.5281/zenodo.15755236. On the docker image, run the following commands to get started:

```
$ docker load -i oopsla25-bv-decide.tar
$ docker run --name oopsla25-bv-decide -it oopsla25-bv-decide
# | This clears the build cache,
# | fetches the maths library from the build cache,
# | and builds our framework.
$ cd /code/lean-mlir && lake clean && lake exe cache get && lake build
# Run experiments InstCombine, HackersDelight, and SMT-LIB
```

Author's Contact Information:

```
# Collect the output and generate the plots and statistics
$ artifacts/oopsla25-bv-decide/run.sh
```

Alternatively, one can access the image from DockerHub (https://hub.docker.com/r/abdoo8080/oopsla25-bv-decide-base/) with the command `docker pull abdoo8080/oopsla25-bv-decide-base:v1`. A long-term archive of the Docker hub image will be uploaded to Zenodo, but we support dockerhub as the primary form of access.

The above commands will run a small subset of the experiments, collect the output, and generate the plots. The results of running the experiments will be stored in the `bv-evaluation/results`, collected data will be stored in `bv-evaluation/raw-data`, plots will be stored in `bv-evaluation/plots`, and tables in `bv-evaluation/tables`. Navigate to each directory to see the results of the experiments. To view the plots, copy `bv-evaluation/plots` out of the container via the following command:

```
$ docker cp oopsla25-bv-decide:/home/user/lean-mlir/bv-evaluation/plots ./plots
```

D Experiments Reproduction

Three main scripts are involved in the reproduction of our results and plots:

- `compare.py benchmark` runs our bitblaster as well as the solver we compare against for `benchmark`. The results obtained from this run are saved in `bv-evaluation/results`. Note that the number of problems solved by `bv_decide` might slightly change depending on the performance of the machine in relation to the timeout set for the SAT/SMT solvers.
- `collect.py benchmark` collects and analyzes the results obtain for `benchmark` and stores everything in `bv-evaluation/raw-data`.
- `plot.py benchmark` plots the results obtained from `benchmark`'s run, including the plots presented in the paper.
- `collect-stats-bv-decide.py` collects all the statistics regarding our evaluation, and in particular the numbers we describe in the paper.

D.1 Verifying the results of `InstCombine`

As an example, to reproduce the results of `InstCombine` using 8 threads for the experimental run and 1 repetition, the sequence of commands to run are:

```
$ docker load -i oopsla25-bv-decide.tar
$ docker run --name oopsla25-bv-decide -it oopsla25-bv-decide
$ cd /home/user/lean-mlir && lake clean && lake exe cache get && lake build
# Run experiments for InstCombine
$ /home/user/lean-mlir/bv-evaluation/compare.py instcombine -j8 -r5
# Collect InstCombine data
$ /home/user/lean-mlir/bv-evaluation/collect.py instcombine
# Plot InstCombine data
$ /home/user/lean-mlir/bv-evaluation/plot.py instcombine
```

Figure 9 is in `plots/bv_decide_stacked_perc_instCombine.pdf` and
Figure 7 is in `plots/cumul_problems_llvm_instcombine_solved_data.pdf`.

D.2 SMT-LIB

The SMT-LIB benchmark set contains 46191 benchmarks. Running the full experiment to reproduce the results in the paper will take a long time, even in a cluster. Instead, we recommend running the experiments on a subset of the benchmarks. The following command will run the experiments on 500 random benchmarks from the SMT-LIB benchmark set, using 16 threads with 20 minutes timeout and 8GB of memory per job (should take ~8 hours):

```
$ docker load -i oopsla25-bv-decide.tar
$ docker run --name oopsla25-bv-decide -it oopsla25-bv-decide
$ cd bv-evaluation
# Run experiments for SMT-LIB
python3 compare.py smtlib -n500 -j16 -t1200 -m8192
# Collect SMT-LIB data
python3 collect.py smtlib
# Plot SMT-LIB data
python3 plot.py smtlib
# Collect SMT-LIB stats (e.g., slowdown and % of solved problems compared to Bitwuzla)
python3 collect-stats-bv-decide.py
```

Note that the above commands will run the experiments on a subset of the benchmarks, and will not exactly reproduce the results in the paper. However, the results should resemble the ones in the paper and would eventually converge to the same results if run on the full set of benchmarks. For reviewers with lower hardware resources, we recommend running the experiments on a smaller subset of the benchmarks, e.g., 200 benchmarks.