Fossil 2.0 Repeatability Instructions

Welcome to the Fossil 2.0 repeatability instructions. This document describes how to reproduce the results presented in the paper "Fossil 2.0: Formal Certificate Synthesis for the Verification and Control of Dynamical Models", as submitted to HSCC 2024. Here, we detail the requirements of the repeatability artifact, how to install them and how to reproduce the results presented in the paper.

System Requirements

- Due to an architectural incompatibility of one of our dependencies, machines using an ARM processor are not supported. This includes Apple M series Macs, Raspberry Pis and other ARM-based machines. We are sorry for the inconvenience, but this is a hard requirement of one of our dependencies. For more information, see this issue.
- This artifact uses Docker. We assume Docker is already installed on the host machine. If not, please follow the instructions here, Docker Desktop may also be useful for Mac users.
- Ideally, the host machine is running Linux. We have tested the artifact on Ubuntu 22.04 (which the Docker image is based on) and MacOS Sonoma 14.1.
- Exact memory requirements are unknown, but we recommend at least 8GB of RAM. The artifact has been tested on machines with 8GB of RAM and 16GB of RAM. We have not tested it on machines with less than 8GB of RAM.
- Please ensure you have at least 5GB free disk space. The Docker image is ~ 2.2GB, so a good internet connection is also required.

A Note on Reproducibility

We have made every effort to ensure that the results presented in the paper are reproducible. However, we note that the results may vary slightly due to the following factors:

- PyTorch (which we use to train the neural networks) does not guarantee reproducible results across different releases or machines. Since our results are each repeated 10 times, we expect the results in terms of success rate to be very similar, but the exact numbers may vary slightly. Similarly, the presented times may vary slightly. For details, see PyTorch documentation.
- Computational resources. PyTorch and the SMT solvers used are CPU-dependent; the results presented in the paper were obtained on a laptop with a 6 core Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz. Only a single core was used for the experiments. We expect the results to be similar in magnitude on other machines, but the exact times may vary slightly. We tested this RE on a 2.4 GHz Quad-Core Intel Core i5 and found the results to be similar in magnitude, but slightly slower.

A Note on Docker

Depending on the nature of your docker install, you may need to run the commands below with superuser privileges using `sudo`. If you are unsure, try running the commands without `sudo` first. If you get a permissions error, try again with `sudo`. We assume by default that your Docker install does not require `sudo`.

Installation

The artifact consists of the following files:

- A copy of the paper, `fossil2.0.pdf`
- A copy of this document, `repeatability_instructions.md`

We request that you create a directory on your host machine to store the results of the experiments. This directory will be mounted into the Docker container when running the experiments, so that the results are available on the host machine. First, we create a working directory called `fossil` on the host machine:

```
# mkdir fossil
# cd fossil
```

All commands from this point on should be run from the `fossil` directory. We then create a subdirectory called `results` to store the results of the experiments:

```
# mkdir results
```

For your convenience, we have created a Docker image that contains all the dependencies required to run Fossil 2.0. To install the Docker image, run:

```
# docker pull aleccedwards/fossil:latest
```

Alternatively, if using the Zenodo artifact you can install the Docker image from the tarball provided. First, extract the tarball:

```
# tar -xvf fossil2.0.tar.gz
```

Then, load the Docker image:

```
# docker load -i fossil2.0.tar

If you want to change the tag of the image, you can run:

```console
# docker tag aleccedwards/fossil:latest fossil:latest
```

Repeating the Experiments

There are two sets of experiments and one reproducible Figure presented in the paper:
```

- *Table 2*, which compares the performance of Fossil 2.0 with Fossil 1.0 on a set of benchmarks. We include a script to repeat the Fossil 2.0 experiments from this table.
- *Table 3*, which contains the results of Fossil 2.0 on a set of 17 benchmarks. We include a script to repeat the Fossil 2.0 experiments from this table.
- *Figure 4*, which depicts an example of a certificate synthesised by Fossil 2.0. We include instructions to reproduce this figure.

Reproducing Table 2

To reproduce the results from Table 2, run:

```
# docker run -it --rm -v $(pwd)/results:/fossil/results \
  fossil python3 -m experiments.fossil1-comparison
```

This command should take approximately 10 minutes to run (Table 2 indicates 7.5 minutes of run time). It wil produce the following three files:

- raw_fossil1_comparison.csv: the raw data from the experiments. This contains a row for each run of each benchmark, with a lot of additional information. This is not presented in the paper, but is kept for completeness.
- fossil1_comparison.csv: the data from the experiments, averaged over 10 runs, presented in the format of Table 2
- fossil1_comparison.tex: the data from the experiments, averaged over 10 runs, presented in the format of Table 2, in LaTeX format

A preview of the table is also printed to the console.

Reproducing Table 3

To reproduce the results from Table 3, run:

```
# docker run -it --rm -v $(pwd)/results:/fossil/results \
 fossil ./run_hscc_experiments.sh
```

This command should take approximately 1-2 hours to run (Table 3 indicates ~1 hour of run time). It will produce the following three files:

- raw_tool_results.csv: the raw data from the experiments. As before, this contains a row for each run of each benchmark, with a lot of additional information. This is not presented in the paper, but is kept for completeness.
- tool_paper_tab.csv: the data from the experiments, averaged over 10 runs, presented in the format of Table 3
- tool_paper_tab.tex: the data from the experiments, averaged over 10 runs, presented in the format of Table 3, in LaTeX format

A preview of the table is also printed to the console.

Reproducing Figure 4

To reproduce Figure 4, run:

```
# docker run -it --rm -v $(pwd)/results:/fossil/results \
  fossil fossil experiments/benchmarks/cli/ctrl_rar_sine.yaml --plot
```

This takes about 1 minute to run and will produce two pdf files in the `results` directory:

- One showing the phase portrait of the system, with the zero contours of the certificate shown as dashed lines
- One showing a surface plot of the certificate functions.

The exact appearance of the figures may vary slightly from the paper due to the reasons outlined in the "A Note on Reproducibility" section.

Running Fossil 2.0 as a User

For the convenience of the repeatability reviewers, we include details on how Fossil 2.0 may be ran as a user with its different interfaces. This is not required to repeat the experiments, but include this information for reviewers who may wish to experiment with Fossil 2.0.

Since the purpose of this section is to interact with Fossil as a user, it is best here to enter the Docker container in an interactive mode. To do this, run:

```
# docker run -it --rm -v $(pwd)/results:/fossil/results \
  fossil bash
```

This will drop you into a bash shell in the Docker container. From here, you may run Fossil 2.0 as a user. All commands from this point should be run from inside the Docker container, in the `/fossil` directory (which you should be in already). If you wish to be able to edit the example files, you may wish to install a suitable text editor (e.g. vim, emacs, nano) using the package manager `apt-get`. nano is installed by default.

Command Line Interface

To run Fossil 2.0 from its command line interface, you must provide a configuration file. This is a .yaml file, and we include an example in the image, filename `lyap-test.yaml`.

For example, to run Fossil 2.0 on the benchmark `lyap-test.yaml`, which must be in the current directory, run:

```
# fossil lyap-test.yaml
```

This benchmark should finish is less than 30 seconds, and Fossil 2.0 will report that a Lyapunov function has been synthesised (and hence the specification is satisfied).

Python API

Fossil 2.0 may also be interfaced with using Python3. We include an example of this in the image provided, entitled `example.py`. This is not required to repeat the experiments, but may be useful for reviewers who wish to experiment with Fossil 2.0.

As before, make sure the file `example.py` is in the current directory.

To run the example, run:

```
# python3 example.py
```

This example will take approximately 1 minute to run, after which it will report that a barrier certificate has been synthesised (and hence the specification is satisfied).

Extending Fossil 2.0

Fossil 2.0 is designed to be easily extensible in terms of synthesising different certificates for verifying additional properties to those described in the paper. We have included the salient continuous-time certificates as part of Fossil 2.0, but envisage that users may wish to extend Fossil 2.0 to synthesise other certificates. We do not recommend attempting to do this for the purposes of the repeatability review as it is designed for advanced users and will require a non-trivial amount of time, but include this information for completeness. Defining a new certificate involves inheriting from the `Certificate` class from the certificate module. This class has several key methods that must be implemented:

- compute_loss: computes the loss function for the certificate to encourage a network to satisfy its conditions. This is the function that is minimised during training.
- learn: the overall learn method for the certificate.
- get_constraints: returns the (negation of) the conditions that define the certificate as SMT-formulae.

Examples of these certificates may be seen in the `certificate.py` file in the public Fossil repo. Once defined, a custom certificate may be used by setting the CertificateType parameter to "CUSTOM", and passing the custom certificate class to the `CUSTOM_CERTIFICATE` parameter of CegisConfig. We include an example of this in the `custom_cert.py` file, which defines a custom Lyapunov certificate. Ensure this file is in the current directory, and run:

```
# python3 custom_cert.py
```

This example should take less than 30 seconds to run, and Fossil 2.0 will report that a function satisfying the custom certificate has been synthesised.