Artifact Evaluation for Data-Driven Inference of Representation Invariants

ANDERS MILTNER, Princeton University
SASWAT PADHI, UCLA
TODD MILLSTEIN, UCLA
DAVID WALKER, Princeton University

1 INTRODUCTION

In this pdf, we:

- Describe the steps for installation of the artifact (§2).
- Describe the file structure of the artifact repository (§3).
- Describe how to run the program (§4).
- Describe how to test individual sections of the paper (§5).

2 INSTALLATION

The below table provides a description of the install step, the command we performed for that step (if applicable), and the relevant version (if applicable).

Description of Step	Command (if applicable)	Version (if applicable)
Download OS		`Ubuntu 18.04.4`
Download Opam	`sudo apt install opam`	`1.2.2-6`
Initialize Opam	`opam init`	
Switch OCaml Version	`opam switch 4.07.0+flambda`	`4.07.0+flambda`
Setup Switch in Current Shell	``eval `opam config env` ``	
Install Dune	`opam install dune`	`1.2.1`
Install Core Dependencies	`opam depext conf-m4.1`	
Install Core	`opam install core`	`0.11.3`
Install Ppx_deriving	`opam install ppx_deriving`	`4.2.1`
Install Async	`opam install async`	`0.11.0`
Install CTypes Dependencies	`opam depext conf-pkg-config.1.1`	
Install Libffi-dev	`sudo apt install libffi-dev`	`3.2.1-8`
Install CTypes	`opam install ctypes`	`0.14.0`
Install CTypes-Foreign	`opam install ctypes-foreign`	`0.4.0`
Install Menhir	`opam install menhir`	`20180905`
Install Pip	`sudo apt install python-pip`	`9.0.1-2.3 ~ubuntu1.18.04.1`
Install EasyProcess	`pip install easyprocess`	`0.2.10`
Install MatPlotLib	`pip install matplotlib`	`2.2.5`

3 FILE STRUCTURE

In the following descriptions, $ will refer to the repository root. In other words, on the VM, $ means ~/HanoiArtifactEvaluation.

$/README.pdf	This document.
$/code	Folder containing the source code for Hanoi.
$/code/benchmarks	Folder containing the benchmarks used for data collection.
$/code/benchmarks-no-helper	Folder containing the benchmarks without helper functions added.
$/code/Makefile	File containing a number of commands for building and testing Hanoi.
$/code/paper_run	Folder containing data and graphs from the run used in the submitted version of the paper.
$/code/HanoiCmdLine.exe	(generated) Executable for running Hanoi.
$/code/generated_data/generated_data.csv	(generated) File containing the run data.
$/code/generated_graphs/times.eps	(generated) File containing the graph of time vs benchmarks completed.

4 RUNNING

One can run individual benchmarks with `HanoiCmdLine.exe [filename]`.

Running `HanoiCmdLine.exe -help` will provide a variety of command-line options to run Hanoi with different options.

Running `python generate-data.py HanoiCmdLine.exe [folder] [file1] [file2] [file3]` will run all the benchmarks within `folder`, and save them to `generated_data/generated_data.csv`. It will generate a table corresponding to Figure 7 at `generated_data/fig_7.txt`. It will generate a table corresponding to the individual runs present in Figure 8 at `generated_data/fig_8.txt`. It will run Hanoi with the Myth and Fold synthesizer on `[file1]` and `[file2]`, and save the results to `generated_data/heap_no_helper_no_module.csv`. Lastly, it will run Hanoi on `[file3]` with a timeout of 120 minutes, and will save the results to `generated_data/bst_data.txt`. This command can be run with just the `[folder]` argument provided; with `[folder]`, `[file1]`, and `[file2]` provided; and with `[folder]`, `[file1]`, `[file2]`, and `[file3]` all provided.

Running `python transform-data.py [file]` will make a time vs benchmarks completed graph for the benchmark results in a csv file.

5 VALIDATION

5.1 Section 4

To review our implementation, one should validate that our description of the implementation, and our algorithm, match that of the paper.

Our syntax for expressions and values are defined in `$/code/src/Expr.ml` and `$/code/src/Value.ml`. The semantics are defined in `$/code/src/Eval.ml`. The typechecking rules are defined in `$/code/src/Typecheck.ml`. Our verifiers are in `$/code/src/EnumerativeLR.ml` and `$/code/src/EnumerativeVerifier.ml` for our verification of the triangle relation and for postcondition verification. We call into Myth at `$/code/src/MythSynthesizer.ml`, and Myth is held in the `$/code/myth` repository. Our overall algorithm is implemented in `$/code/src/VPIE.ml`, with the function `learnVPreCondTrueAll`.

5.2 Section 5

To generate the data for Section 5, one must run the `make generate-all` command from within `$/code`. This will create all of `HanoiCmdLine.exe`, `generated_data.csv`, and `times.eps`. This data generation is restartable – once a benchmark has been completely run, it will be saved in the csv. To rerun a benchmark, merely delete that benchmark row from the csv file. This can be used to validate the graphs and tables in the evaluation section of the paper. This command will take about a day to complete.'

By default, we set up the script with a timeout 30 minutes, and to run each test only once (to take less time for the evaluators). The evaluators can change the timeout to t seconds if they update `TIMEOUT_TIME` in `generate_data.py` to $t + 5$. The evaluators can change each test to run n times if they update `REPETITION_COUNT` in `generate_data.py` to n. Increasing timeout time or repetition count increases the time it takes to run the benchmarks.

In what follows, we describe what should be validated in each subsection, how to perform validation, and possible differences that may arise between your local results and the results presented in the paper.

Subsection 5.1. To validate our benchmark suite, one can validate that our benchmarks are as described. The `micro` repository corresponds to the "other" benchmarks.

Subsection 5.3. One can validate Table 7 by comparing the table present in `code/generated_data/fig_7.txt` to Table 7. The some of the generated test names are slightly incorrect. All the benchmarks of the form "micro/[test_name]" should be "other/[test_name]". In addition, the tests named "coq/sorted-list-::-heap.ds" and "coq/sorted-list-::-heap+binfuncs" should be named "coq/maxfirst-list-::-heap.ds" and "coq/maxfirst-list-::-heap+binfuncs". Manual inspection of these files will show the correct invariant is one where the maximum element is the first element (not sorted, as would be indicated by the generated test name).

Subsection 5.4. One can validate bst by looking at the data generated in `generated_data/bst_data.txt`. This data is shaped in the following way:

```
[invariant]
;
[synthesis calls]
;
[verification calls]
;
[max synthesis time]
;
[max verification time]
;
[total synthesis time]
;
[total verification time]
;
[invariant size]
;
[total run time]
```

If the program does not terminate in 120 minutes, the file will say `t/o`. If the program terminates with no output (indicative of running out of memory), the file will say `m/o`. If the program terminates with an error, it will say "error:" followed by the error. Note that, while the paper describes the computer as having 16GB of RAM, macOS is very good at using swap space and compressing memory. If using a Linux of Windows system, one may need to use up to 50GB of RAM to get anything other than `m/o`.

One can validate that the fold synthesizer can solve `vfa/tree-::-priqueue` and `vfa/tree-::-priqueue+binfuncs` without using helper functions and module functions by looking at the file `generated_data/heap_no_helper_no_module.csv`. The column describing the time to complete using the Myth is `Myth` and the column describing the time to complete using the fold synthesizer is `Fold`. One can validate that the Myth synthsizer and Fold synthesizer preform comparably on benchmarks by looking at `generated_data/generated_data.csv` and comparing the MythTime and FoldTime columns.

Subsection 5.5. To validate the graph contained in this section, one should compare the graph in `$/code/generated-graphs/times.eps` to Figure 8. For fine-grained information on the graph, one can look at the file `$/code/generated_data/fig_8.txt`.

In OneShot, there may be a number of "incorrect"s and "err"s. The word "incorrect" means that the synthesized invariant was incorrect, as determined by comparing to the output of a correctly synthesized invariant from Myth. The word "err" means that the postcondition was over two instances of the abstract type, so we cannot merely "run" the postcondition on the elements of the concrete type.

With a sufficiently fast computer, one may find that ∧Str succeeds on one or two benchmarks (one of bst_::_set and bst_::_set+hofs) that Hanoi does not succeed on. This success is a false positive, and is only returned due to the unsoundness of our verifier. One can validate that this response is a false positive by running `./HanoiCmdLine -conj-str benchmarks/coq/bst_::_set.ds` or `./HanoiCmdLine -conj-str benchmarks/coq/bst_::_set+hofs.ds` (depending on which has the false positive) from within the code directory and checking the output. To check the output, one should manually ensure that trees that satisfy the invariant also satisfy the postcondition, and should ensure that performing any operation on a tree that satisfies the predicate will result in another tree satisfying the predicate.