Artifact Evaluation for Bottom-up Synthesis of Recursive Functional Programs using Angelic Execution

ANDERS MILTNER, UT Austin, USA
ADRIAN TREJO NUÑEZ, UT Austin, USA
ANA BRENDEL, UT Austin, USA
SWARAT CHAUDHURI, UT Austin, USA
ISIL DILLIG, UT Austin, USA

1 INTRODUCTION

In this paper, we:

- Describe the steps for installation of the artifact (§2).
- Describe the steps for the "kick-the-tires" phase (§3).
- Describe the file structure of the artifact repository (§4).
- Describe how to run the program (§5).
- Describe how to validate the experiments (§6).

2 INSTALLATION

There are two methods of artifact installation, via virtual machine and via manual installation. Virtual machine installation is the recommended method. The username for this virtual machine is `burst` and the password is also `burst`.

2.1 Virtual Machine

We have provided a virtual machine via google drive here. We tested our virtual machine using VirtualBox (available for download here).

If one is using VirtualBox installation, they can proceed as follows:

(1) Select the *New* button at the top of the VirtualBox application.
(2) Choose a Name for your installation. We used BurstArtifactEvaluation. The Type is Linux, and the Version is Ubuntu 64-bit.
(3) Select a memory allotment. We gave ours 12300 MB of RAM.
(4) Select "Use an existing virtual hard disk file." Select the downloaded and extracted vdi file.
(5) Click "Create"

2.2 Manual Installation

The other method for installation (not recommended for artifact evaluation) is to manually install the code. Table 1 provides a description of the install step, the command we performed for that step (if applicable), and the relevant version (if applicable).

3 KICK-THE-TIRES

For the kick-the-tires phase, we ask that the committee first follow the installation instructions in Section 2.

Authors' addresses: Anders Miltner, UT Austin, Austin, TX, USA, amiltner@cs.utexas.edu; Adrian Trejo Nuñez, UT Austin, Austin, TX, USA, atrejo@cs.utexas.edu; Ana Brendel, UT Austin, Austin, TX, USA, anabrendel@utexas.edu; Swarat Chaudhuri, UT Austin, Austin, TX, USA, swarat@cs.utexas.edu; Isil Dillig, UT Austin, Austin, TX, USA, isil@cs.utexas.edu.

Description of Step	Command (if applicable)	Version (if applicable)
Download OS		`Ubuntu 20.04.3`
Download opam	`sudo apt install opam`	`2.0.5`
Initialize opam	`opam init`	
Switch opam version	`opam switch create 4.10.0+flambda`	`4.10.0+flambda`
Set up opam env	`eval $(opam env)`	
Install dune	`opam install dune`	`2.9.1`
Install core	`opam install core`	`v0.14.1`
Install ppx_deriving	`opam install ppx_deriving`	`5.2.1`
Install menhir	`opam install menhir`	`20210419`
Install bark	`opam install bark`	`0.1.4`
Install pip	`sudo apt install python3-pip`	`20.0.2`
Install easyprocess	`pip3 install easyprocess`	`0.3`
Install pandas	`pip3 install pandas`	`1.3.3`
Install curl	`sudo apt install curl`	`7.68.0`
Install stack	`curl -sSL https://get.haskellstack.org/ | sh`	`2.7.3`
Install libz3-dev	`sudo apt install libz3-dev`	`4.8.7`
Install sdkman	`curl -s "https://get.sdkman.io" | bash`	`5.12.4`
Install java jdk	`sdk install java $(sdk list java | grep -o "8\.[0-9]*\.[0-9]*\.hs-adpt" | head -1)`	`8.0.292`
Install sbt	`sdk install sbt`	`1.5.5`
Download codebase	`git clone https://github.com /amiltner/BurstArtifactEvaluation.git`	

Table 1. Manual installation steps.

After, please run the command `make kick-the-tires` in the `BurstArtifactEvaluation` directory. If this command finishes, outputting `Tires kicked successfully!` as the last line, then the basic functionality of the artifact is ensured.

4 FILE STRUCTURE

The file structure of important files and folders is described in Table 2.

5 RUNNING

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

Running `./BurstCmdLine.exe -help` will provide a variety of command-line options to run Burst with different options.

6 VALIDATION

The only data to validate is present in Section 8, Evaluation.

To generate all the generated data, one should run `make generate-all` from the root of the directory. This will create csv files summarizing the runs in the `$/burst/generated-data/`, `$/leon/generated-data/`, and `$/synquid/generated-data/` directories. This will also create `.out` files in the `$/burst/benchmarks/`, `$/leon/benchmarks/`, and `$/synquid/benchmarks/` directories. This command can be stopped and restarted, and it will pick up where it left off. If one

Location in Directory	Description
`$/README.pdf`	This document.
`$/burst/`	This folder contains the BURST and SMYTH tools.
`$/burst/Makefile`	This Makefile contains a number of commands for building and testing BURST.
`$/burst/code_description.txt`	This file describes the layout of the BURST codebase.
`$/burst/benchmarks/`	This folder contains the BURST benchmark suite.
`$/burst/benchmarks/logical/`	This folder contains the BURST **Logical** benchmark specifications.
`$/burst/benchmarks/io/`	This folder contains the BURST **IO** benchmark specifications.
`$/burst/benchmarks/ref/`	This folder contains the BURST **Ref** benchmark specifications.
`$/burst/manual-data/`	Contains manual data describing whether or not a synthesized program is correct.
`$/burst/BurstCmdLine.exe`	(generated) Executable for running BURST.
`$/burst/generated-data/`	(generated) Contains data generated from experiments.
`$/leon/`	This folder contains the LEON tool.
`$/leon/Makefile`	This Makefile contains a number of commands for building and testing LEON.
`$/leon/benchmarks/`	This folder contains the BURST **Logical** benchmark suite, written in a form compatible with LEON
`$/leon/manual-data/`	Contains manual data describing whether or not a synthesized program is correct, and whether or not it satisfies the postcondition.
`$/leon/generated-data/`	(generated) Contains data generated from experiments.
`$/synquid/`	This folder contains the SYNQUID tool.
`$/synquid/Makefile`	This Makefile contains a number of commands for building and testing SYNQUID.
`$/synquid/benchmarks/`	This folder contains the BURST **Logical** benchmark suite, written in a form compatible with SYNQUID
`$/synquid/manual-data/`	Contains manual data describing whether or not a synthesized program is correct, and whether or not it satisfies the postcondition.
`$/synquid/generated-data/`	(generated) Contains data generated from experiments.

Table 2. File structure description. In the location column, $ refers to the repository root. In other words, in the VM, $ means `/BurstArtifactEvaluation/`.

wants to remove the generated files and restart, they can run `make hyper-clean` from the root of the directory.

The `.out` files can be manually inspected to ensure that they are the desired function. If they are, then `Correct` column should be labelled with `\correct` in the manual data CSVs. If they are not, then `Correct` column should be labelled with `\incorrect` in the manual data CSVs. If the program times out, then the `Correct` column should be labelled with `\na` in the manual data CSVs. Furthermore, LEON will sometimes generate code that does not satisfy the specification (the other tools will generate programs that are not desired, but do satisfy the specification). If this is the case, then the `LeonSatisfies` column should be labelled `n`, otherwise it should be labelled `y`. The manual data from our paper run is provided by default in these files.

After the command `make generate-all` has been run, and the manual-data files have been updated, running `python3 aggregate-data` will create tables in the `$/generated-data/` folder. There is two types of files here, CSV files and `_pretty.txt` files. The CSV files are more useful if viewing data through a CSV viewer like Excel. The `_pretty.txt` files are more useful if viewing data through a text editor like vim, or through the command line via cat. The `io` files correspond to Figure 11. The `ref` files correspond to Figure 12. The `logical` files correspond to Figure 13. The `ablation` files correspond to Figure 16.

6.1 Possible Inconsistencies with the Paper Findings

Both Burst and Leon have nondeterminism present, so there is expected to be some variability in runtimes. This is more evident when running on different hardware and operating system than the original tests (for example, there is more variability when running tests on a Linux VM, rather than directly on a Intel chip running MacOS).

Furthermore, we found that we incorrectly computed the percent correct in the ablation Figure 16. Note that in Figure 13, we found that Burst completed 41/45 logical benchmarks. However, Figure 16 incorrectly shows that Burst completed 42/45.