Paper 583 Artifact

ANONYMOUS AUTHOR(S)

1 GETTING STARTED GUIDE

This section presents a quick setup guide along with a sanity check that should ideally take about 30 minutes.

1.1 Creating virtual environment

```
python3 -m venv probe
probe\Scripts\activate.bat (Windows)
source probe/bin/activate (Ubuntu/MacOS)
```

1.2 Clone and Compile

```
cd probe
chmod +x ./build
./build
```

1.3 Sanity Check

The following command runs Probe on a single SyGuS file, this command should output a synthesized program in about 5 seconds.

```
java -cp target/scala-2.12/probe-assembly-0.1.jar
sygus.ProbeMain src/test/benchmarks/string/exceljet1.sl
```

Then we proceed to run a set of 10 random benchmarks which are given in Fig.13 of the paper -

```
python3 artifact.py string --expt sanity
```

This should print out the synthesized programs which are given in Fig. 13 of the paper. The synthesized programs match the ones in the figure but the synthesis times might not since the logging framework adds more time and hardware might vary.

1.4 CVC4 installation

If CVC4 is not already installed, the build_cvc4 needs to run to install it. Linux/MacOS users should be able to execute the following command in about 2 seconds -

```
timeout 10 cvc4 --lang=sygus1 src/test/benchmarks/string/11604909.sl
```

Windows Users should be able to execute the following command in similar time -

```
timeout 10 cvc4-1.7-win64-opt.exe src/test/benchmarks/string/11604909.sl
```

2 STEP BY STEP INSTRUCTIONS

This section highlights the steps to run each of the experiment in the paper with different entry points. We evaluated Probe on 82 synthesis benchmarks from the EuPhony string benchmark suite. The string benchmarks can be found in src/test/benchmarks/string/ directory.

2020. 2475-1421/2020/1-ART1 $15.00
https://doi.org/

2.1 Experiments from Section 6.2

This section covers the experiments from Section 6.2 of the paper that evaluates the effectiveness of the just-in-time learning approach. We measure the time to find a solution to each of the 82 benchmarks in our benchmark set, for three strategies we refer to as Probe (guided search), size based enumeration and height based enumeration (unguided). We log the partial solutions that are used for updating the PCFG during enumeration in the results/probe-out/<benchmark>.iter files.

Run Probe on a single SyGuS file

```
java -cp target/scala-2.12/probe-assembly-0.1.jar
sygus.ProbeMain src/test/benchmarks/string/exceljet1.sl
```

Output format for all experiments in this section

```
"file", "program", "time taken (secs)", "size (number of AST nodes)",
"ites (number of ite operators)"  (ites only if strategy is probe)
```

Run Probe on the entire SyGuS string benchmarks with a default timeout of 10 minutes

```
python3 artifact.py string --strategy probe --expt perf
```

Run Probe on the entire SyGuS string benchmarks with a timeout of 10 seconds

```
python3 artifact.py string --strategy probe --expt perf --timeout 10
```

Since running for 10 minutes per benchmark would take a long time (about 6 hours), there is an option to run it with a lower timeout. The results of these experiment get saved in the results/probe.csv file and a more detailed log is outputted in the results/probe-out directory.

Run size based on the entire SyGuS string benchmarks

```
python3 artifact.py string --strategy size --expt perf
```

The results of this experiment get saved in the results/size.csv file.

Run height based on the entire SyGuS string benchmarks

```
python3 artifact.py string --strategy height --expt perf
```

The results of this experiment get saved in the results/height.csv file.

Larger Grammar Experiment

In addition to our regular benchmark suite, we also ran Probe and the size-based enumeration on an extended grammar with 48 string literals, 11 int literals and all other available operations. The benchmarks can be found in src/test/benchmarks/larger-grammar directory. To run Probe and size-based on the larger grammar experiment, execute the following commands with a timeout of your choice.

```
python3 artifact.py string --strategy probe --expt extgrammar
python3 artifact.py string --strategy size --expt extgrammar
```

The results of this experiment get saved in the results/probe-larger.csv and results/size-larger.csv files.

Plotting the graphs Fig 15 and Fig 14

```
python3 -m pip install numpy matplotlib
```

Fig. 15 can be plotted using the command below and gets stored as figures/probe-vs-unguided.png. This command requires that all the three result files from running Probe, size based enumeration and height based enumeration *i.e.* probe.csv, size.csv and height.csv resp. are present in the results directory.

```
python3 plot-times-unguided.py
```

Fig. 14 for the larger grammar experiment can be plotted using the command below and gets stored as figures/probe-larger.png. This command requires that the files from running Probe and

size based enumeration on the extended grammar *i.e.* probe-larger.csv and size-larger.csv are present in the results directory.

```
python3 plot-times-larger.py
```

2.2 Experiments from Section 6.3

This section provides a guide to reproduce the experiments from Section 6.3 that involves comparison of Probe to EuPhony and CVC4.

Running CVC4

Running CVC4 takes about 4 hours to complete and the results should be directed to /probe/results/cvc4.csv for plotting graphs later. The results for this experiment are already in the results/cvc4.csv file and can be verified by running on a subset of benchmarks if time constraints. The timeout for the experiment can be changed by tweaking the time on line 5 of syguscomp.sh.

```
cd probe/src/test/benchmarks
(Linux/MacOS) ./cvc4-syguscomp.sh > ~/probe/results/cvc4.csv
(Windows) ./cvc4-syguscomp-win.sh > ~/probe/results/cvc4.csv
```

Running EuPhony

EuPhony solutions for string benchmarks have already been provided in the results/euphony.csv file and can be verified by cloning and running EuPhony using the following commands.

```
git clone https://github.com/wslee/euphony.git
cd euphony/
./build
. bin/setenv
./artifact string --timeout 600 --only_euphony
```

The results of running EuPhony get stored in the euphony/euphony-out directory. Fig. 16 can be replicated by plotting the comparison graph -

```
python3 plot-times-others.py
```

2.3 Experiments from Section 6.4

Size of Solutions generated

This section reproduces the plots in Fig.17 which compare the solution sizes of Probe with size-based enumeration, EuPhony and CVC4.

```
python3 plot-scatter-relative-size.py (Fig.17 (a))
python3 plot-scatter-relative-euphony.py (Fig.17 (b))
python3 plot-scatter-relative-cvc4.py (Fig.17 (c))
```

Generalization Accuracy

This section reproduces the table in Fig.19 which compares the generalization accuracy of Probe and CVC4. The benchmarks for this experiment can be found in src/test/benchmarks/accuracy-expt. The goal is to learn a program using the normal version of the benchmark and test the accuracy of the synthesized program using the long version of the benchmark that has additional examples. The solutions to the normal version for Probe and CVC4 are provided in results/probe-train.csv and results/cvc4-train.csv as they have already been verified in the performance experiments. The generalization experiment for Probe and CVC4 can be executed as :

```
python3 artifact.py string --strategy probe --expt accuracy
python3 artifact.py string --strategy cvc4 --expt accuracy
```

The results are found in results/probe-accuracy.csv and results/cvc4-accuracy.csv, the second column in these files indicates the accuracy.

Case splitting

This section reproduces the plot in Fig.18 which indicates the number of ite operators per number of examples for Probe and CVC4. The following command reproduces the plot in figures/ite.png if the files results/probe.csv and results/cvc4.csv are present.

```
python3 ites.py
```

2.4 Additional Experiments

These experiments are not included in the paper but will be added in the final version. The BitVector benchmarks can be run using the command

```
python3 artifact.py bitvec --expt perf --strategy probe
```