Getting Started

The software artifact is packaged as an Open Virtual Appliance (.ova) file [https://en.wikipedia.org/wiki/Open_Virtualization_Format] with Xubuntu 21.04. The virtual machine has been set up to use 4 cores and 8gb of RAM. We recommend that the reviewers use a machine with a least 4 cpu cores.

The credentials are:

> username: pldi22-reviewer
>
> password: pldi22ae

The synthesis tool can be found in the virtual appliance `/home/pldi22-reviewer/Synduce`. All the commands given in this README are meant to be run from that directory. Note that the tool is named Sebis in the paper for anonymity reasons, and the artifact builds upon and extends the tool Synduce. In this artifact, we use the name Synduce for the tool.

The tool used in this artifact is available publicly on Github. The specific version used here can be accessed at https://github.com/synduce/Synduce/releases/tag/pldi22.

Running the tool on simple benchmarks

To get started, open a terminal and change directory to the root folder of the tool `/home/pldi22-reviewer/Synduce`. The tool has already been compiled, but the README.md in the tool directory contains general instructions on how to build the tool and run it. We have prepared a script for the reviewers to use to check that the installation is working as expected. Running the following command:

```
$./0-kick-the-tires.sh
```

Runs the tool on a small set of 24 benchmarks. All benchmarks should pass. All benchmarks have a solution, except the last one. The last line of the output should look like:

```
 All 24 benchmarks passed in 3.9 s.
```

The last benchmark is unrealizable, and it is expected to fail in that configuration. The running time on the reviewer's machine may vary, but it should be within 10s. The reviewer cam also run the tool on individual benchmarks, for example:

```
$ ./Synduce benchmarks/constraints/bst/count_lt.ml
```

Should print a solution in less than a second.

Generating the figures from the experimental data

We included the data generated during our experimental evaluation in `benchmarks/data/exp/paper_results.csv`. Running the script

```
$./0-report-paper-results.sh
```

generates the figures and the tables presented in the paper. The last lines of output of the script should be:

```
Number of benchmarks: 140 (45 unrealizable cases)
Timeout value: 400 s
.____________________________________.
|              SE2GIS  SEGIS+UC  SEGIS  |
| Realizable       93        70     70  |
| Unrealizable     44        25      0  |
|   Total         137        95     70  |
|____________________________________|

See tables and figures in paper-results/
```

The reviewer should be able to see the figures and tables presented in the paper in the `paper-results` folder. Note that the paper contains a typo and reports that we have 44 unrealizable benchmarks. The 44 on line 1143 should read 45.

Step-by-step Instructions

This section contains step-by-step instructions to reproduce the results presented in the paper. More precisely, we provide instructions to reproduce:

- the sequence of calls to the tool described in Section 2, leading the programmer to find a solution for the problem of optimizing the count-between function on binary search trees.
- the experimental results that are presented in Section 8 in Figures 3 and 4. These results are repeated in Appendix C.2 and C.3 in table form (Table 1 and 2).

Reproducing the example of Section 2

The benchmarks corresponding to the different versions of the sketch for the `countbtw` example have been grouped in the folder `benchmarks/count_between_example/`. There are 4 different unrealizable benchmarks, and 1 realizable one. At each step, the reviewer can check that:

- the tool answers realizable (there is a solution) or unrealizable (there is provably no solution),
- the tool provides useful feedback to the user: either a solution or an explanation as to why the benchmark is unrealizable.

1. The first example (`benchmarks/count_between_example/1.ml`) is the first instantiation of the problem presented in Section 2. The reference function and the target recursion skeleton are the ones given on page 3 of the paper. This problem is unrealizable, there is no solution to the sketch. Running the tool on that benchmark: `./Synduce benchmarks/count_between_example/1.ml` The tool should give a hint as to why it is unrealizable. The reviewer should expect an answer within 1s, and the before-last line of the output should be:

> On input Node(i, p, p0), g1 should have access to g p0

2. The second example is the instance where the programmer has fixed the input of `g1` given the information returned by the tool. The problem is still unrealizable. Running `./Synduce benchmarks/count_between_example/2.ml`, the reviewer should expect an answer within 1s, and the before-last line of the output should be:

> On input Node(i, p, p0), g2 should have access to g p

1. In the third example, the inputs of g1 and g2 have been fixed. The problem is still unrealizable. A quick answer can be obtained using the Symbolic CEGIS algorithm by running `./Synduce benchmarks/count_between_example/3.ml --segis`. The tool answers that the benchmark is unrealizable within a second. To get a more detailed answer, run `./Synduce benchmarks/count_between_example/3.ml`. The tool produces the following hint in less than a minute:

> On input Node(i, p, p0), g3 should have access to g p

4. In the fourth example, the programmer added `(g l)` as an argument to `g3`. Running `./Synduce benchmarks/count_between_example/4.ml --segis` produces the "unrealizable" answer in less than a second. Running `./Synduce benchmarks/count_between_example/4.ml` produces an answer with arguably more useful information in about 4min (the tool has to discover many invariants):

> On input Node(i, p, p0), g3 should have access to g p0

5. In the fifth example, the programmer added `(g r)` as an argument to `g3`. The problem is realizable. Running `./Synduce benchmarks/count_between_example/5.ml` produces a solution in less than a second (the solution should include a definition for the functions `g0`, `g1`, `g2`, `g3`). The solution should be functionally equivalent to the one given in the paper.

Reproducing the experimental results of Section 8

The reviewer can reproduce the experimental results presented in Section 8 by executing the scripts `1-short-experiments.sh` (1-2 hours) or `2-full-experiments.sh` (6-12 hours) in the root directory. The first script is a shorter evaluation process (1-2 hours) while the second script should only be run by a reviewer that wants to obtain more complete results. From these experiments, we expect the reviewer to verify our claims that:

- Fig. 3: our algorithm SE2GIS solves more benchmarks than the baseline we implemented in SEGIS+UC. This is also the result reported in the small table on page 11.
- Fig. 4: when both SE2GIS and SEGIC+UC solve a given benchmark, the relative performance of each algorithm depends on the benchmark, and there is no general trend (no algorithm is generally

faster).

Short Experimental Setup

The script `1-short-experiments.sh` reproduces only partially the results presented in the paper, but runs in a shorter amount of time (1-2 hours).

Running:

```
./1-short-experiments.sh
```

launches a series of tests. For each benchmark in the set of 140 benchmarks, the scripts runs the tool with the SE2GIS (default) algorithm and the SEGIS+UC algorithm, with a timeout of 60s. The expected output during this phase should be two lines for each benchmark.

Once all the tests have been executed, the script will generate the figures that correspond to the experimental data in the `short-results` folder:

- fig3or10.pdf corresponds to Figures 3 and 10 of the paper,
- fig4or9.df corresponds to Figures 4 and 9 of the paper,
- fig11.pdf corresponds to Figure 11 of the paper (in the Appendix).
- table1.pdf corresponds to Table 1 of the paper (and table2.pdf to Table2).

The main difference in the evaluation process of this script compared to the one used to generate the results presented in the paper is that:

- only SE2GIS and SEGIS+UC algorithms are run. The SEGIS algorithm runs in similar time to SEGIS+UC on realizable benchmarks, and fails on unrealizable benchmarks. The experimental information gained from running this algorithm compared to SEGIS+UC is not significant. Therefore, we removed the tests in this evaluation setting.
- A timeout of 60s is used instead of 400s.
- Each benchmark is only evaluated once for each algorithm, instead of 10 times in the paper.

The reviewer should expect the last lines of the output of the script to be:

```
Number of benchmarks: 140 (45 unrealizable cases)
Timeout value: 60 s
._____________________________________.
|              SE2GIS  SEGIS+UC  SEGIS  |
| Realizable       91        66      0  |
| Unrealizable     42        23      0  |
|   Total         133        89      0  |
|_____________________________________|

See tables and figures in short-results/
```

The exact numbers will vary. Since SEGIS is not run in this experimental setup, all the related numbers will be 0.

Full Experimental Setup

The script `2-full-experiments.sh` follows a similar evaluation approach as the previous one, but runs all algorithms with a 400s timeout. Running this script can take between 6 to 12 hours depending on your machine. Running:

```
./2-full-experiments.sh
```

launches a series of tests and then a script to generate the figures corresponding to the experimental data. The figures are written to the `full-results` folder. As opposed to the previous script, both SEGIS and SEGIS+UC are run on each benchmark and the timeout is increased to 400s. The only difference between this evaluation process and the one used to produce the results presented in the paper is that each benchmark is run only once for each synthesis algorithm, instead of 10 times in the paper.

Building the tool and running your own benchmarks

The tool directory contains a `README.md` file with instructions on how to build the tool, documentation and more information on how to write input problems for Synduce. The documentation of the source code is in `docs/index.html`.