OOPSLA 2025 Artifact (README.md)

Introduction/Overview

This document is an overview of the artifact for the paper: **Close is Good Enough: Component-Based Synthesis Modulo Logical Similarty**. The artifact is distributed as a source code repository link.

The file contains two sections: The **Getting Started** section gives the main steps for installing the dependencies and sub-modules, using OCaml package manager (spam) and `stack,` followed by introducing small scripts for building and running `Hegel` on some sample test cases.

The second section, **Step-by-step Instructions**, explains the directory's structure, how to run Hegel (and other baseline tools) on all the benchmarks in the paper (Both RQ1 and RQ2 in the paper), how to understand the results, and how to run Hegel on an individual synthesis task in different ablation modes.

Hardware Dependencies

- OS: Linux x86-64 machine
- Memory : > 8GB

Getting Started

Following are the main steps to build and run `Hegel`, the tool associated with the paper:

Building Hegel from Sources

We have successfully tried building Hegel on Linux (Ubuntu 22.04) and Mac(macOS Monterey). However, the artifact also runs earlier tools, `Synquid` and `Hoogle+,` which we were only able to build on Linux systems. Consequently, we discuss the instructions for the Ubuntu build and running.

Prerequisites

To build Hegel, the following dependencies must be installed:

- OCaml (Version >= 4.03)

```
#install opam
$ apt-get install opam

#environment setup
$ opam init
$ eval `opam env`

# install a specific version of the OCaml base compiler
$ opam switch create 4.03
$ eval `opam env`

# check OCaml installation
$ which ocaml
```

```
/Users/.../.opam/4.03.0/bin/ocaml

$ ocaml -version
The OCaml toplevel, version 4.03.0
```

- Z3 SMT Solver

```
$ opam install "z3>=4.7.1"
$ eval $(opam env)
```

- Menhir for parsing the specification language

```
$ opam install menhir
$ eval $(opam env)
```

- OCamlbuild version >= 0.12

```
$ opam install "ocamlbuild>=0.12"
$ eval $(opam env)
```

To Run the Evaluations.

- Python3

```
$ apt-get install python3
```

To Build `Synquid` and `Hoogle+`

- stack: (https://docs.haskellstack.org/en/stable/README/)
- Hoogle+: (https://github.com/davidmrdavid/hoogle_plus#readme)
- Synquid (https://github.com/nadia-polikarpova/synquid.git)

Building Hegel

After all the dependencies are installed, Hegel can be directly built using *ocamlbuild* using the script `build.sh` in the project root directory.

```
$ ./build.sh
```

The above build script will create a native executable `prudent.native` in the project's root directory.

Building Synquid

The details for building `Synquid` are provided by the authors in [1]. Following are a series of steps necessary steps.

```
$ cd synquid
$ stack setup && stack build
```

Building Hoogle+

The details for building `Synquid` are provided by the authors in [2]. Following are a series of steps necessary steps.

```
$ cd hoogle_plus
$ stack setup && stack build
```

Kick the tires: Test Running Hegel

Once all the dependences and the three tools are built as discussed above, we can run the following to start `kick-the-tires`:

```
$ python3 quick_test.py
```

The run will take a total of around `7 minutes`, as Synquid and Hoogle time out on various tests. This should produce files `./kick-tires.txt` and `./kick-tires-timings.txt`. The file `./kick-tires.txt` has rows of the following form:

```
./prudent_tests/unit/algorithmW/u_test1, Hegel, 2.15, 3, 4, 0
./synquid/test/hegel/u_test1, Synquid, 1000.00, 0, 0, 0
```

The above row reflects a test case similar to the benchmarks in Figure 11 in the paper. The columns, give the location for the test, the name of the tool (`Hegel` | `Synquid` | `Hoogle+`), the synthesis time in seconds (with a timeout depicted by a 1000.00 secs). This is followed by columns giving the number of conjuncts or disjuncts, the size of the synthesized solutions and the number of control-flow branches if any.

The `./kick-tires-timings.txt` contains only the synthesis timings for the runs.

The synthesized programs are located under the `./output/<test-location>` directory in the project's root directory. For instance, you will see a single program generated for `u_test2.spec` at

`output/u_test2.spec` with a single branch:

```
let rec goal    (z : Ty_list int a)  (size : int) : (Ty_list int a) =
 match z with
 | Nil    -> _lbv1
 | Cons    var_x3 var_xs4 -> _lbv1
```

Detailed Step-by-step Instructions for Full-evaliations

The following instructions explain:

1. The structure of the repository highlighting relevant files.
2. How to Run Hegel to generate synthesis time Figures 11, 12, and 14.
3. How to Run Hegel on individual synthesis problems.

Structure of the Artifact.

The source code for this artifact is available at prudent

The files and directories used in this artifact are:

- `quick_test.py` : a script to test the successful installation of Hegel
- `run_benchmarks.py` : a script to run Hegel for all the benchmarks in the paper producing results.
- `prudent_tests/Hegel/Hoogle+/**.spec` contains benchmarks in Figure 11.
- `prudent_tests/hegel/Cobalt+/**.spec` contains benchmarks in Figure 12

Running the evaluation using a push button.

```
$ cd project_root
$ python3 run_benchmarks.py
```

Once the script terminates, it will produce a files `results-full.txt` and `full-timings.txt` which contains entries for each benchmark in Fig 11 and 12 in the following format. (Showing the result for first Fig 11 benchmark `nth`). The time 1000 shows the case where the tool timed out.

```
Running Hegel ./prudent_tests/hegel/Hoogle+/nth.spec
6.97user 5.78system 0:13.06elapsed 97%CPU (0avgtext+0avgdata
42500maxresident)k
0inputs+28808outputs (0major+657795minor)pagefaults 0swaps
```

This will also generate the synthesis timings in the file `full-timings.txt`, for instance, for `Nth.spec`:

```
*******************************
./prudent_tests/hegel/Hoogle+/nth.spec_Hegel : 6.973712
```

and all columns of Fig 11 in `full-result.txt`,

```
./prudent_tests/hegel/Hoogle+/nth, 6.97, 3, 4, 0
```

Finally, the generated programs are at `output/nth.spec`; for `nth.spec`, it generates around 60 programs.

Reusability Guide

Outline of the Artifact Source

- `./main/prudent.ml` : Main entry for the tool.
- `./synlang/**` : The Synthesis language definitions $$\lambda_{syn}$$ in the paper.
- `./synthesis/**` : The Synthesis Algorithm(s)
- `./speclang/**` : The specification language of Refinement Types.
- `./prudent_tests/**`: Tests and Benchmarks contain benchmarks in Figure 11.

Running an individual synthesis query

The general usage for Hegel:

```
$ ./prudent.native [-bi] [cdcl] -k <maximum-path-length> <spec-file1>
```

To run Hegel to get the synthesis results using the complete Hegel mode ($\textcolor{blue}{blue}$ bar in Fig 14):

```
$ ./prudent.native -bi -cdcl -k <maximum-path-length> file.spec
```

- To run Hegel in Hegel(-S) mode ($\textcolor{yellow}{yellow}$ bar in Fig 14):

```
$ ./prudent.native -cdcl -k <maximum-path-length> file.spec
```

- To run Hegel in Hegel(-P) mode ($\textcolor{green}{green}$ bar in Fig 14):

```
$ ./prudent.native -bi -k <maximum-path-length> file.spec
```

- To run Hegel in Hegel(-ALL) mode ($\textcolor{red}{red}$ bar in Fig 14):

```
$  ./prudent.native -k <maximum-path-length> file.spec
```

Note about the paper under Major Revision

The paper is under `major revision,` and we will submit a new version of the artifact with experiments showing the number of SMT queries and size reductions for QTAs.

References

[1] https://github.com/nadia-polikarpova/synquid.git

[2] https://github.com/davidmrdavid/hoogle_plus#readme