Artifact for Paper
UCLID5: Multi-Modal Formal Modeling, Verification, and Synthesis

Elizabeth Polgreen[1,2], Kevin Cheang[1], Pranav Gaddamadugu[1], Adwait Godbole[1], Kevin Laeufer[1], Shaokai Lin[1], Yatin A. Manerkar[1,3], Federico Mora[1], and Sanjit A. Seshia[1]

[1] UC Berkeley
[2] University of Edinburgh
[3] University of Michigan

1 Setup

The submission tarball has the following structure:

```
├── License.txt
├── Readme.txt
├── artifact.pdf
├── packages
│   └── sbt_1.5.5_all.deb
└── uclid
    ├── License.txt
    ├── Readme.txt
    ├── tool_paper_examples
    │   ├── example_<name>.sh scripts
    │   ├── export_paths
    │   ├── uclid
    │   ├── z3
    │   ├── delphi
    │   ├── cvc4
    │   ├── Fib
    │   ├── Control
    │   ├── Keystone
    │   ├── OperAxUhb
    │   ├── TrainSystem
    │   └── Readme.md
    ├── src
    └── <others>
```

The `packages` directory contains the `sbt` Debian package which is required to build **UCLID5** from source. This is completely optional and it is not required for running **UCLID5** using the binaries that we have provided. The `uclid` directory is a snapshot of the **UCLID5** GitHub repository at commit `add commit`. The `tool_paper_examples` folder is self contained for running the tool on the examples with the provided binaries (see Running with prebuilt binaries below). To begin, copy the `packages` and `uclid` directories into the `$HOME` directory of the VM.

Running with prebuilt binaries (recommended) The `tool_paper_examples` directory is self contained with prebuilt **UCLID5** and external solver binaries as well as the relevant examples mentioned in the paper. Please change move into this directory and follow the instructions given in Section 2.

Building from source The supplied `sbt` Debian package is only required if you wish to build **UCLID5** from source. Start by installing `sbt` by running `sudo dpkg -i sbt_1.5.5_all.deb` from the `packages` directory. Then from the `uclid` directory, follow the instructions given at `https://github.com/uclid-org/uclid#compiling-uclid5`. After doing so, you can either build the **UCLID5** binary or run examples directly from within the sbt prompt.

2 Running the examples

Move into the `tool_paper_examples` directory. This directory consists of the following:

- Prebuilt binaries for the **UCLID5** tool and the external solvers `z3`, `cvc4` and `delphi` in their respective directories
- Five test-examples directories: `Fib`, `Control`, `TrainSystem`, `OperAxUhb`, and `Keystone`
- Run scripts for each of the above examples, named as `run_<name>.sh`
- A `export_paths` sourcefile

Startup instructions To begin, source the `export_paths` script. This should set permissions for running all the binaries and add required paths to the `$PATH` variable.

```
source export_paths
```

To run example `examplename`, run the script `./run_<examplename>.sh` from the `tool_paper_examples` directory. Here `examplename` corresponds to: `Fib`, `Control`, `TrainSystem`, `OperAxUhb`, and `Keystone`. We now describe each of these examples in order:

2.1 Fib

This example corresponds to Figure 4 in our submitted paper. It demonstrates how to use UCLID5's synthesis syntax.

The model in fib.ucl represents a simple Fibonacci sequence with a partial proof of the property that the numbers in the sequence always increase. The proof is partial because the property is not inductive. UCLID5 is able to complete the proof using a syntax-guided synthesis engine. **Running run_Fib.sh should take around 10 seconds.**

2.2 Control

This example corresponds to Figure 5 in our submitted paper. It demonstrates how to use UCLID5's integration with oracles: it uses the oracle `is_stable`, provided in the `Control` sub-directory. The model in test-control.ucl represents a Linear Time Invariant system with two state variables. The system is specified using the following matrices:

$$A = [0.901224922471, 0.000000013429; 0.000000007451, 0.000000000000]$$
$$B = [128, 0]$$

The UCLID model finds a two values for the controller: k0 and k1. The invariants specify that the controller should stabilize the system (i.e., the eigenvalues should fall within the unit circle, whilst the system states remain within safe bounds up to a finite unrolling bound. For more example controllers see `https://ssvlab.github.io/dsverifier/dssynth-toolbox/index.html`. **Running run_Control.sh should take around 45 seconds.**

2.3 TrainSystem

This example corresponds to Figure 9 and 10 in our submitted paper. It demonstrates the hybrid approach of combining operational and axiomatic modeling using UCLID5.

The model describes a train system written in Lingua Franca (LF), a polyglot coordination language for building deterministic reactive systems. The source code is in `TrainSystem.lf`. The state transition of the model takes an operational approach by using the `init` and `next` keywords in UCLID5, while the semantics of LF are specified using a set of axioms. The model shows a flaw in the train system under verification, by exposing that a bad state, "the train moves while the door is open," is reachable. UCLID5 illustrates the flaw by returning a counterexample to the user. **Running run_TrainSystem.sh should take around 30 seconds.**

2.4 OperAxUhb

This example corresponds to Figure 8 in our submitted paper, namely the combined operational-axiomatic model of a microarchitecture whose pipelines fetch instructions in program order. The example code contains `uhb_common.ucl`, the embedding of μspec in UCLID5 (parts of which are shown in Figure 7 of our submitted paper). It also contains the code of the model depicted in Figure 8[4] of our paper in `operAx.ucl`. The `operAx.ucl` file also contains two properties (`FetchHBNextExecute` and `WBFifo`). The control block of `operAx.ucl` checks these properties using bounded model checking for traces up to size 5. The first property (`FetchHBNextExecute`) is maintained by a combination of the operational and axiomatic constraints, and so will always be true. The second property (`WBFifo`) is not maintained by the model, and the `bmc` command duly detects a counterexample trace of length 5 for it. **Running run_OperAxUhb.sh should take around 30 seconds.**

[4] Note that the `fifoFetch` axiom differs very slightly from Figure 8 of our paper due to a small inaccuracy in the submitted paper's code.

2.5 Keystone

Running run_Keystone.sh should take around 1 hour.