Getting Started with UCLID5

January 26, 2018

1 Introduction

UCLID5 is a modeling language that supports verification and synthesis. The UCLID5 toolchain aims to:

1. Enable modeling of finite and infinite state transition systems.
2. Verification of safety and hypersafety (k-safety) properties on these systems.
3. Allow syntax-guided synthesis of models and model invariants on these transitions systems.

This document serves as introduction to UCLID5 modeling language and toolchain.

1.1 Getting Started: A Simple UCLID5 Model

```
1  module main {
2    // Part 1: System description.
3    var a, b : int;
4
5    init {
6      a = 0;
7      b = 1;
8    }
9    next {
10     a, b = b, a + b;
11   }
12
13   // Part 2: System specification.
14   invariant a_le_b: a <= b;
15
16   // Part 3: Proof script.
17   control {
18     unroll (3);
19     check;
20     print_results;
21   }
22 }
```

Example 1.1: A UCLID5 model that computes the Fibonacci sequence.

A simple UCLID5 module that computes the Fibonacci sequence is shown in Figure 1.1. Let us walk through each line in this model to understand the basics of UCLID5.

The top-level syntactic structure in UCLID5 is a `module`. All modeling, verification and synthesis code in UCLID5 is contained within modules. In Figure 1.1, we have defined one `module` named `main`. This module starts on line 1 and ends on line 18. The module can be conceptually split into three parts: a system model, a specification and proof script. In the example, these three conceptual parts are also kept separate in the code.[1] The following subsections will describe each of these sections of the module.

The System Model

This part of a UCLID5 module describes the functionality of the transition system that is being modeled: it tells us *what the system does*.

The first item of interest within the module `main` are *state variables*. These are declared using the `var` keyword. The module `main` declares two state variables: `a` and `b` on line 2. These are both of type `int`, which corresponds to mathematical integers.[2]

The `init` block appears next and spans lines 4 to 7. It defines the initial values of the states variables in the module. We see that `a` is initialized to 0 while `b` is initialized to 1.

The `next` block appears after this and it defines the transition relation of the module. In the figure, the next statement spans from lines 8 to 10; `a` is assigned to the (old) value of `b`, while b is assigned to the value `a + b`.

The System Specification

The specification answers the question: *what is the system supposed to do?*.

In our example, we have a single `invariant` that comprises that entire specification. Line 12 defines this `invariant`. It is named `a_le_b` and as the name suggests, it states that `a` must be less than or equal to `b` for every reachable state of the system.

The Proof Script

The third and final part of the UCLID5 module is a set of commands to the UCLID5 verification engine. These tell how we should go about proving[3] that the system satisfies itself specification.

The proof script is contained within the `control` block. The commands here execute the system for 3 steps and check whether all of the systems properties (in this case, we only have one invariant: `a_le_b`) are satisfied for each of these steps.

The command `unroll` executes the system for 3 steps. This execution generates four *proof obligations*. These proof obligations ask whether the system satisfies the invariant `a_le_b` in the initial state and in each of the 3 states reached next. The `check`

[1] This is not required by the UCLID5 syntax but is a good design practice.

[2] Mathematical integer types, as opposed to the machine integer types present in languages like C/C++ and Java, do not have a fixed bit-width and do not overflow.

[3] Note we are using a very broad definition of the word prove here to refer to any systematic method that gives us assurance that the specification is (perhaps partially) satisfied.

command *checks* whether these proof obligations are satisfied and the `print_results` prints out the results of these checks.

1.2 Installing UCLID5

Public releases of the UCLID5 can be obtained at: `https://github.com/uclid-org/uclid/releases`. For the impatient, the short version of the installation instructions is: download the archive with the latest release, unzip the archive and add the 'bin/' subdirectory to your `PATH`.

More detailed instructions for installation are as follows.

1.2.1 Prerequisites

UCLID5 has two prerequisites.

1. UCLID5 requires that the **Java Runtime Environment** be installed on your machine. You can download the latest Java Runtime Environment for your platform from `https://www.java.com.com`.
2. UCLID5 uses the Z3 SMT solver. You can install Z3 from: `https://github.com/Z3Prover/z3/releases`. Make sure the 'z3' or 'z3.exe' binary is in your path after Z3 installed. Also make sure, the shared libraries for libz3 and libz3java are in the dynamic library load path (`LD_LIBRARY_PATH` on Unix-like systems).

1.2.2 Detailed Installation Instructions

First, down the platform independent package from `https://github.com/uclid-org/uclid/releases`.

Next, follow these instructions which are provided for the bash shell running on a Unix-like platform. Operations for Micosoft Windows, or a different shell should be similar.

- Unzip the archive.

  `$ unzip uclid-0.9.zip`.

  This should produce output similar to the following.

```
Archive:  uclid-0.9.zip
  inflating: uclid-0.9/lib/default.uclid-0.9.jar
  inflating: uclid-0.9/lib/com.microsoft.z3.jar
  inflating: uclid-0.9/lib/org.scala-lang.scala-library-2.12.0.jar
  inflating: uclid-0.9/lib/org.scala-lang.modules.scala-parser-combinator
  inflating: uclid-0.9/lib/org.scalactic.scalactic_2.12-3.0.1.jar
  inflating: uclid-0.9/lib/org.scala-lang.scala-reflect-2.12.0.jar
  inflating: uclid-0.9/bin/uclid
  inflating: uclid-0.9/bin/uclid.bat
```

- Add the `uclid` binary to your path.

  ```
  $ export PATH=$PATH:$PWD/uclid-0.9/bin/
  ```

- Check that the `uclid` works.

  ```
  $ uclid
  ```

  This should produce output similar to the following.

  ```
  $ uclid

      Usage: uclid [options] filename [filenames]
      Options:
        -h/--help : This message.
        -m/--main : Set the main module.
        -d/--debug : Debug options.

  Error : Unable to find main module.
  ```

1.2.3 Running UCLID5

2 UCLID5 Grammar

TODO: Fix this.

Default settings

⟨*statement*⟩ ::= ⟨*ident*⟩ '=' ⟨*expr*⟩
| 'for' ⟨*ident*⟩ '=' ⟨*expr*⟩ 'to' ⟨*expr*⟩ 'do' ⟨*statement*⟩
| '{' ⟨*stat-list*⟩ '}'
| ⟨*empty*⟩

⟨*stat-list*⟩ ::= ⟨*statement*⟩ ';' ⟨*stat-list*⟩ | ⟨*statement*⟩

Increase the two lengths

⟨*statement*⟩ ::= ⟨*ident*⟩ '=' ⟨*expr*⟩
| 'for' ⟨*ident*⟩ '=' ⟨*expr*⟩ 'to' ⟨*expr*⟩ 'do' ⟨*statement*⟩
| '{' ⟨*stat-list*⟩ '}'
| ⟨*empty*⟩

⟨*stat-list*⟩ ::= ⟨*statement*⟩ ';' ⟨*stat-list*⟩ | ⟨*statement*⟩