ENGR489 Meeting Minutes
QuickCheck for Whiley

Janice Chin
Primary Supervisor: David Pearce
Secondary Supervisor: Lindsay Groves

1 March 20th 2018

1.1 Present

- David Pearce

1.2 Action points were achieved since the last meeting

None as this is the first meeting.

1.3 Action points have yet to be achieved since the last meeting

None as this is the first meeting.

1.4 Action points were agreed to for the next meeting

1. Install Whiley command line tool.
2. Read research papers about QuickCheck to determine strategies for generating test cases.

1.5 Discussion Points

Objective 1: Take Whiley program and generate test cases from it using the Whiley compiler.

Assuming each Whiley program has specifications for each function.

To investigate:

- Learn and understand Whiley
- Whiley jar file for the compiler
- Whiley web project

- Maven, pom.xml for dependencies for the generator
- Need to know strategies for generating test cases

2 March 26th 2018

2.1 Present

- David Pearce

2.2 Action points were achieved since the last meeting

- Read some research papers about QuickCheck and other testing tools (DART, ArbitCheck)
- Read Whiley Getting Started Guide and started reading language specifications
- Got WhileyLabs working on my machine, could not get Whiley working on the command line.

2.3 Action points have yet to be achieved since the last meeting

None

2.4 Action points were agreed to for the next meeting

- Finish draft project proposal and send to supervisors to get it checked (by 30/3/2018).
- Reading more research papers. Suggest reading papers about American Fuzzy Lop.
- Begin implementation of project. The first steps are to create a Java project with the necessary hooks to access the Whiley compiler. And, then to be able to read a compiled Whiley file (*.wyil) to identify functions to test.

2.5 Discussion Points

Whiley can also pass functions as an argument to other functions.

- What testing method am I using? Random testing? Property testing?
  - Firstly, use random testing using the properties of the function. Pre-condition to for candidate values and post-condition to check test passed or failed. Then will look at limitations of the testing method

and decide whether to expand on the testing method (such as using dynamic symbolic execution).

- What types to use in test generation? Primitive, recursive etc
  - Primitive types: bool, byte, int, real, null, any, void
  - Array
  - Later on: records (closed), union
  - Even later: Recursive types
  - Extra (may not be implemented) - references, functionss
- Testing functions only or also test methods? Methods have side effects.
  - Test for functions first then by methods.
- Using pre- and post-conditions or property syntax?
  - Do not need to worry as the interpreter should evaluate the pre- and post-conditions. Properties are only used by the verifier and are specified in the pre- and post-conditions.

3 April 9th 2018

3.1 Present

- David Pearce

3.2 Action points were achieved since the last meeting

- Read more research papers. Of particular note is MoreBugs, an extension to QuickCheck which generalises test cases so test cases that discover the same bug do not occur again.
- Created base implementation of QuickCheck for Whiley. This does
  - Generates Int (specific Int) and random Bool
  - Check candidate values are valid based on the precondition. Note: Does not generate a new test to replace the candidate values that do not pass the precondition.
  - Validates tests using postcondition
- Finished project proposal

3.3 Action points have yet to be achieved since the last meeting

None

3.4 Action points were agreed to for the next meeting

- Start on writing background survey for preliminary report.
- Consider the design of QuickCheck for Whiley. Look into more iteration strategies for generating test values.

3.5 Discussion Points

- Wanted to know about wyal.util.SmallWorldDomain as it also uses a Generator.
  - Finds counterexamples for the Whiley Theorem Prover.
  - Exhaustive iteration through possible values in a small range
- Technique for writing a related works section in the report is to write a paragraph summary of each paper and then use those paragraphs as a basis for the section. Should be high level.
- Keep the test case generator flexible to allow different testing techniques.
- Want to compare random vs exhaustive testing.
- Function generation can be difficult. Can be inefficient if functions call other functions. Could just create return values for the functions that conform to the functions post-condition. However, need to consider tradeoff between performance and accuracy.

4 April 16th 2018

4.1 Present

- David Pearce

4.2 Action points were achieved since the last meeting

- Implemented exhaustive test generation for booleans and integers.
- Started background survey but have not done much of this.

4.3 Action points have yet to be achieved since the last meeting

Background survey for preliminary report

4.4 Action points were agreed to for the next meeting

- Array, record and nominal generation

4.5 Discussion Points

- Nominal types. Look at NameResolver and Flowtype checker. Ignore constraints when generating the tests. Like type synonyms.
- Bounded random state space. Selecting n samples, using basic probability across the state space.
- Arrays should be limited by its array size. This should be a constant upper limit. Random generation of elements. The arrays takes a generator element.
- Records. Print name of the field. Don't know field names of .... This generates n number of fields.
- Recursive types. Need to know if the type is recursive. Infinite data type e.g. binary tree. Limit it like an array. Cyclic generators? e.g. type list is {int data, null — List next }
- Integer range paper - For figuring out ranges to use during generation.

5 April 30th 2018

5.1 Present

- David Pearce

5.2 Action points were achieved since the last meeting

- Array generation. Size of elements are bounded between 0 and 3.
- Nominal type generation.
- Record generation for closed records only.
- Null generation

5.3 Action points have yet to be achieved since the last meeting

None

5.4 Action points were agreed to for the next meeting

- Union generator
- Start on integer range analysis

5.5 Discussion Points

- ResolutionError occurs when a name of a type cannot be found. To resolve, need to import the correct files used such as the Whiley standard library.
- Look at Knuth's Algorithm S for random sampling.
- Null type primarily used for recursive types.
- When generating union types, need to fairly select values of each type. E.g. for int—bool, select an int then a bool then an int etc.

6 May 7th 2018

6.1 Present

- David Pearce

6.2 Action points were achieved since the last meeting

- Completed union generation
- Fixed some bugs with nominal invariants not being applied
- Completed integer range analysis ONLY for nominal types that wrap an integer. I.e. nat is (int x) where x >0.

6.3 Action points have yet to be achieved since the last meeting

None

6.4 Action points were agreed to for the next meeting

- Run the valid and invalid tests written for testing the Whiley compiler. To see what your tool does and to check that: valid tests don't produce counter examples, and invalid tests (ideally) do. Don't expect every invalid test will find a counter example though.
- Continue working on integer range analysis

6.5 Discussion Points

- Whiley does implicit casting. E.g. type nat is (int x) where 'a' >x will verify. In Whiley, a char is an integer from 0 to 255.
- A nominal type can have multiple invariants due to multiple where clauses.

- Implies (== >) is a
- Each type could have a range. This means that the Generators for types would mirror the hierarchy for Ranges. An array would have a range for the elements itself and the length of the array.
- Trying to add integer ranges for all elements in an array is difficult. For example, type intArr is (int[] x) where all { i in 0...|x| | x[i] >0 } . Could possibly pattern match on this case?
- implies (== >) for x ==>y is !(x) || (x && y)
- iff (<== >) for x <== >y is equality i.e. (x == y)