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, functions
- 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)

7 May 13th 2018

7.1 Present

- David Pearce

7.2 Action points were achieved since the last meeting

- Integer range analysis for integers in records and array sizes
- Tested QuickCheck on the Whiley Valid and Invalid tests with various statistics.

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

None

7.4 Action points were agreed to for the next meeting

- Start introduction for preliminary report and a bit of the background survey.
- Integer range analysis for nominals

7.5 Discussion Points

- Whiley properties(predicates in Dafny) are used for verification.
- Start writing preliminary report. You can look into expanding the project proposal.
  - Introduction = The purpose of the project
  - Background = Briefly about Whiley? About other test frameworks.

- Technical Discussion = High level. Could make a class diagram of the generators. Start with the basics first and then go into more depth such as the extensions of the tool (integer range analysis).
- Data - What results you have so for
- Request for feedback - Can ask if your method for evaluation is good? Evaluators may want more concrete evidence in the form of statistics.

8 May 21st 2018

8.1 Present

- David Pearce

8.2 Action points were achieved since the last meeting

- Integer range analysis for nominals
- Started preliminary report - currently done 5 pages. 1 page introduction, 2 pages of background, 2 pages work completed.

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

None

8.4 Action points were agreed to for the next meeting

1. Complete preliminary report

8.5 Discussion Points

- Introduction - Include what have you done so far? What you are doing? Could consider including objectives.
- Background
  - Should write about tools similar to QuickCheck. Consider looking into and writing about JCrasher.
  - Write more about Whiley - relate how specifications can be used in testing. Static verification. Do not write about theorem prover but could specify about counter examples. Assume reader has never heard of and/or used Whiley before.
- Future plan
  1. Evaluation of the tool needs to be done.

2. Method/function calls within methods/functions are expensive. Therefore, need to optimise the performance of calling these functions/methods - call optimisation. One technique is instead of calling the function/method, is to just generate a return value. Problem if it is a method call as a method could modify a variable without returning it e.g. sorting an array without returning it.
3. Performance comparision between executing functions/methods normally VS call optimisation approach.
4. Mutation testing - mutating existing Whiley files. Mutating a file means to change some aspect of the file. Examples: changing operators, change forall to a sum, change if to a while statement, delete statements. This is to be able to check that the mutated file would have a different result than the original file. Where to apply mutation, on the .whiley or the .wyil file? Could do this on the .wyil file but then need to convert back into the .whiley file?
5. Could do recursive type generation, would need to limit the size of the value generated.
6. Don't need to do function generation and open record generation. A possible future extension to the tool.

- Sent draft of preliminary report to Dave, who has given feedback about it.

9 May 28th 2018

9.1 Present

- David Pearce

9.2 Action points were achieved since the last meeting

Completed Preliminary Report

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

None

9.4 Action points were agreed to for the next meeting

- Byte generator
- Use Algorithm S for random test generation
- Start looking into how functions/methods can be optimised

9.5 Discussion Points

- No meeting for next week. Next meeting on June 11th. Same meeting time during exam period.
- Fix typos, move listing captions below, add frames around code.
- Could expand on how generators work in QuickCheck
- Randoop more concentrated on constructors. Expand more about feedback-directed test generation. Think about talking how it generates values for parameters.
- Say more about random test generation, what was considered?
- Better explain generators
- Talk about issues encountered during implementation e.g. the difficulty of integer range analysis, recursive types, open records.
- Add time taken to run the tests. Why are two different integer ranges used in the tests (to limit number of combinations).
- Example test techniques in request for feedback

10 June 11th 2018

10.1 Present

- David Pearce

10.2 Action points were achieved since the last meeting

- Byte generator
- Implemented Algorithm S for random test generation. Was not sure if this should be added in as there could be bias in the results due to sampling without replacement.
- Bounded recursive type generation - bounded to a depth of 3

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

None

10.4 Action points were agreed to for the next meeting

- Start optimising function calls. This is by replacing the execution of the function/method with a randomly generated value. If same function is called again with the same arguments, it should ideally return the same value.

10.5 Discussion Points

- Algorithm S - random testing. Would be good to talk about the design and tradeoffs in the final report! Included should be fairness issue, performance cost, how it was implemented.
- For random testing, it might be good to look at `http://homepages.ecs.vuw.ac.nz/%7Edjp/files/GPCE17_preprint.pdf`
- Record returned values for functions. Same inputs == same outputs. executeInvoke should be called. Override the interpreter.
- How is it affecting the data for function optimisation? Invalid tests - are you finding the bugs?
- Ideally by the end of the mid-tri break (in tri2), should finish experiments for evaluation.
- Mutation testing should be done before the mid-tri break.
- QuickCheck versus verification for finding bugs and performance is interesting.
- Future extensions: References and methods
- This code does not verify, gets empty type error. Due to spaces VS indentation.

```
type Fun is function(int) -> int

function map(int[] items, Fun fn) -> int[]
requires |items| > 0
requires all { i in 0..|items| | items[i] > 0}:
    int i = 0
    while i < |items|
    where 0 <= i && i <= |items|:
        items[i] = fn(items[i])
        i = i + 1
return items
```

11 June 18th 2018

11.1 Present

- David Pearce

11.2 Action points were achieved since the last meeting

Created function optimisation for un-recursive functions by random generation.

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

None.

11.4 Action points were agreed to for the next meeting

- Start preliminary work with the experiments. Record statistics when running tests, and compare function call optimisation and caching against Whiley test suite.
- Run Whiley tests to see if there are invalid tests. Also run WyBench tests from the develop branch.
- Add flags for different parameters like whether to enable and disable function optimisation.

11.5 Discussion Points

- Encountered several problems when trying to implement function optimisation
    1. For the Interpreter, I had to copy a lot of methods into my own version of the Interpreter due to the private scope. Ideally, the methods would have the protected scope so I would only need to override the methods I require. Dave says this should be okay.
    2. Recursive functions are a problem! Especially when the invariants call a recursive function as this causes a StackOverflow due to an infinite loop. Currently, this has been disabled if the function is calling itself (e.g. factorial function). Not too sure how to solve this as I need to be able to detect the difference between a normal function call and recursive function call. E.g. if there are two functions foo() and bar() where foo calls bar and bar calls foo until some condition is met. Should try and disable this.

12 July 3rd 2018

12.1 Present

- David Pearce

12.2 Action points were achieved since the last meeting

- Began experimenting on Whiley valid and invalid tests.
- Fix bugs discovered from testing.
- Attempted to test the WhileyBench tests, could not get this to work due to other libraries required during compiling.

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

12.4 Action points were agreed to for the next meeting

- Execute the WyBench tests. See if function optimisation impacts the performance of these tests. Require Dave to send the jar and wyil files for the remaining WyBench tests.
- Fix more bugs discovered from testing.

12.5 Discussion Points

- Having problems importing libraries during compiling. Needed for the WyBench tests. Dave managed to get the wystd library working by using the correct version.
- How to format command line arguments as input? Not necessary to format command line arguments nicely. Only would be useful if it was widely used.

13 July 16th 2018

13.1 Present

- David Pearce

13.2 Action points were achieved since the last meeting

- Able to run the WyBench tests by using the Standard Library and the wybench.jar file provided.
- Discovered a few bugs in the WyBench tests.

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

None

13.4 Action points were agreed to for the next meeting

- For the bugs discovered, add issues to the WyBench GitHub. Then fix these issues by submitting a pull request. This can then be addressed as evidence in the final report.
- Fix any further bugs in the tool, adding missing specifications and refactoring where appropriate.
- Investigate mutation testing. Setup the framework to read in a WyIL file, apply a mutation to some expression within that file, and the print out the file using wyc.io.WhileyPrettyPrinter. Copy the code for that from GitHub into your project and fix it.

13.5 Discussion Points

- Received feedback on the Progress Report. Dave said not to worry too much about the feedback. Especially the function optimisation since I have done this already.
- Presentation on Friday July 20th for the Programming Language user group at 12pm, CO255. Prepare a presentation between 5 to 10 minutes long, around 3 slides. Short introduction, maybe present data and explain about the generators.
- Take an Whiley as input, produce mutated file as output (Whiley). Consider how many to apply mutations to apply and where they should be applied. Could implement this as a applyMutation function, where you take an arbitrary expression and mutate it. Check the mutated file can be compiled.

14 July 24th 2018

14.1 Present

- David Pearce

14.2 Action points were achieved since the last meeting

- Added issues on GitHub to fix Whiley tests
- Fixed more bugs in the tool, and better error reporting when a test fails
- Started looking into mutation testing.

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

None

14.4 Action points were agreed to for the next meeting

- Generating references - do simple cases first then consider what happens with aliasing.
- Generating functions - should be similar to function optimisation.
- Testing methods

14.5 Discussion Points

- In the report, write about caching values (memoisation). How it affects performance?
- References should start at a random value (RValue.Cell). The Reference points to RValue.Cell.
- Problem with references is aliasing where two references can point to the same cell. For example, e.g. foo(&int, &int, &int) can have different combinations such as 1, 1, 1 (all point to same cell), 0, 1, 0 (first and last point to same cell) etc.
- Have enough tests for the evaluation. Leave out mutation testing for now as it seems like another side project. If we do decide to do this, then we may do a quick implementation of it (e.g. just replacing operators).

15 July 31st 2018

15.1 Present

- David Pearce

15.2 Action points were achieved since the last meeting

- Can now test on methods
- Reference generation without aliasing
- Function generation - just generates a return value randomly using the return parameters. Similar to generating a record.

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

None

15.4 Action points were agreed to for the next meeting

- Record statistics from method calls.
- Want to try and optimise performance of the tool, by reducing the number of tests if all combinations have been exhausted. Only for functions as the methods could alter global variables, therefore, the number of method calls matter.
- Add extra command line arguments for memoisation.
- Try and prepare structure for the final report.

15.5 Discussion Points

- Next week, will start the final report. Could take parts of the progress report and put it in the final report
- Have a lot of material for evaluation, the statistics from testing and the issues raised on GitHub.
- Need to check if the return type for the function generated follows the type constraints. E.g. function(int) -¿ (nat) where the nat needs to be an int >= 0.
- For the functions generated, their return calls could also be memosised? (Hmmm... the function always returns the same value, so calling a generated function shouldn't be costly?). Since this is not implemented, this can be added as a future extension in the report.
- Also in the report, can talk about why aliasing was not done for reference generation and how it could have been approached.

16 August 7th 2018

16.1 Present

- David Pearce

16.2 Action points were achieved since the last meeting

- Added flag for memoisation. Turns out it makes testing a lot faster. Due to methods being executed.
- Stop test execution if all possible test combinations have been generated.
- Allow memoisation on Properties too
- Started looking into the final report.
- Started executing statistics for methods. Still need to do some more due to some tests taking a long time. May need to tweak function optimisation.

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

None

16.4 Action points were agreed to for the next meeting

- Start final report. Concentrate on the implementation and evaluation sections. Want to have a vision of what the final report will look like.
- Want to know how you interpret the various statistics in the evaluation. For example, could have the statistics in a table. GitHub issues could also be represented as a table of what issue was found, how it was identified, where it was corrected etc.
- Send draft to Dave of what was written in the morning. (Plan to send this through by Monday morning, Tuesday morning at the latest!)

16.5 Discussion Points

- Realised I didn't use Knuth Algorithm S for random test case generation. So I implemented it. Problem: Very slow when there are a few tests but a large number of combinations such as when conducting function optimisation. As it has to iterate through X number of combinations, using probability to get the N number of tests. Previously, it would just iterate through the generators (where generators did execute Algorithm S), calling generate.
- Could output statistics into a latex format to be easily copied and pasted into the report.
- Bullet point what you want to write in the relevant sections in the report.
- Limitations part based on feedback on the progress report. Testing can't find everything however, verification can. What are the limitations of the tools in the research paper? Other limitations include the performance of the tool.

17 August 14th 2018

17.1 Present

- David Pearce

17.2 Action points were achieved since the last meeting

- Started implementation chapter - need to write function optimisation and memoisation
- Started figuring out what to do for the evaluation
- Attempted to fix random test problems - turns out that a high number of combinations 751,747,177 takes 12 seconds to execute 1 test. For 100 tests that is 20 minutes long. Therefore, need another way to select tests, maybe randomly generate and add to a set until it reaches the number of tests required?

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

None

17.4 Action points were agreed to for the next meeting

- Continue writing the report
- Revise report based on feedback on the draft version of the report.
- Split implementation into design chapter. Design is a high level of what you have done. In the implementation can then why you did it in such and such way. For example, fairness of unions.
- Make diagrams stand out better

rename Integer range analysis

17.5 Discussion Points

- Don't worry about the random test generation time problem. This can be talked about in the report.
- Investigate why memoisation is failing for the Whiley invalid tests.