VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

QuickCheck for Whiley

Janice Chin

Supervisors: David Pearce, Lindsay Groves

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honour.

Abstract

This document is a project proposal for the project, QuickCheck for Whiley. It describes an engineering problem and how this project will solve the problem. It will also explain how to evaluate my proposed solution and any resource requirements needed for this project.

1. Introduction

Testing is an important process in software development as it helps detect the presence of bugs. However, writing and running tests can be tedious and costly. Furthermore, it is difficult to write tests for all possible cases therefore, obscure bugs may not be detected. An automated test-case generator called QuickCheck was implemented in Haskell to solve these issues. This tool creates a large number of tests using user-defined properties and generating random input values.

This project is about implementing an automated test-case generator for the programming language, Whiley based on the QuickCheck tool.

2. The Problem

Whiley is a programming language, developed to verify code and eliminate errors using formal specifications. Ideally, programs written in Whiley should contain as few errors. To achieve this goal, Whiley contains a verifying compiler which employs the use of specifications written by a developer to check for common errors such as accessing an index of an array which is outside its boundaries.

Currently, the verifying compiler has limitations when evaluating complex pre- and post-conditions. For example, a post-condition could be falsely identified as not holding by the verifying compiler even though the program does meet the post-condition. Therefore, this project aims to implement an automated test-case generator in Whiley to improve software quality and increase confidence in unverifiable code.

3. Proposed Solution

The automated test-case generator needs to read a Whiley program and then generate tests for functions in the program. Randomised input values that adhere to the function's preconditions will be used in testing. The success of each test is determined by using the function's postconditions. The test results should show the percentage of tests that passed and the inputs used in the failed tests.

Several components developed for the Whiley language will need to be used for the automated test-case generator.

After basic test-case generation is completed, more complex types can be generated for testing and further enhancements to the tool can be made.

Possible enhancements include:

- Weight on the frequency of test values
- Shrinking failed test cases
- Classifying tests
- Using dynamic symbolic execution

Regular meetings (weekly or fortnightly) with the primary supervisor, David Pearce will be held to assist in this project.

Timeline

A gantt chart of the project can be found on the project's repository.

Trimester 1

Output	Estimated Time	Start Date	Complete by
Produce bibliography	2 weeks	19/3/18	9/4/18 (Week 5)
Produce project proposal	2 weeks	19/3/18	9/4/18 (Week 5)
Implement an automated test-case generator for Whiley programs with only the types: bool, byte, int, real, null	3 weeks	19/3/18	16/4/18 (Week 6)
Implement automated test-case generation for types: void, array, union and records	3 weeks	16/4/18	14/5/18 (Week 9)
Produce preliminary report	3 weeks	14/5/18	10/6/18 (Week 12)
Extend the automatic test generator to be able to generate other types or use methods for better test case distribution (weighting, classification)	3 weeks	11/6/18	2/7/18 (Exam period)

Trimester 2

Output	Estimated Time	Start Date	Complete by
Produce slides for presentation of preliminary report	2 weeks	2/7/18	16/7/18 (Week 1)
Extend the automatic test generator to include other methods of testing (symbolic) or be able to generate other types	3 weeks	16/7/18	6/8/18 (Week 4)
Produce draft of final report	4 weeks	6/8/18	15/9/18 (Week 7)
Complete implementation of automated test generator	3 weeks	15/9/18	5/10/18 (Week 10)
Finalise final report	2 weeks	6/10/18	21/10/18 (Week 12)
Prepare slides for final presentation	3 weeks	22/10/18	16/11/18 (Exam period)

4. Evaluating your Solution

To evaluate this tool, we will insert bugs into a small benchmark set and see whether they can be uncovered by the tool.

5. Resource Requirements

No special resources are required. Only the use of the ECS laboratories is required.