# ENGG4812 – Thesis Proposal

### Student: Jacob Freeman

### Student Number: 44852708

### Supervisor: Graeme Smith

## Topic

This project will develop VS Code extensions which allow users to write and check weakest precondition proofs on Dafny code. This will allow future students of CSSE3100/7100 to readily format their assignments, and for tutors of CSSE3100/7100 to readily check assignments for correctness. It will also provide a valuable tool for software developers wanting to understand the behaviour of complex code.

## Questions

Should I be analyzing other formal method tools, and how their design may apply to the VS code extension tool?

Do you have any links to vs code extensions codebases that are similar and may be useful?

Should I analyze the syntax and logic required for wpp in dafny?

# Introduction

Dafny is a high-level validation language that determines the correctness of code. Its main purpose is to validate the occurrence of runtime errors while also validating exactly what the programmer wanted their code to do. Annotations are used within the codes implementation to logically reason and prove the accuracy of the code. These annotations can be implemented abstractly using expressions that summarize how the code operates. The expressions that encompass these annotations of the code are proven for correctness over the body of the code. Overall, Dafny reduces the need to write error-free code and instead write error-free annotations. Generally writing error-free annotations is easier and less error prone to implement due to their concise format. [1]

## Problem

The most basic form of these annotations used in Dafny are preconditions and postconditions, amongst other more complex annotations. These conditions specify the state the code and its variables should be in before and after the code has run. Dafny, proves the correctness of these annotations by running weakest precondition proofing over the code to determine that the precondition matches the postcondition given the code in between. Dafny performs this proof ‘under the hood’, providing complicated error messages throughout the code.

## Thesis

The University of Queensland’s CSSE3100 Reasoning about programming course, many students interested in learning how the code is verified, especially in Dafny’s case, calculate these proofs line-by-line throughout the code, making the code and the weakest precondition reasoning increasingly difficult to read. This thesis aims to provide students a tool to check semantics in their weakest precondition proofs in real time and for tutors to mark the logic within these weakest precondition proofs in real time.

## Solution

Dafny can be installed and run on the visual studio code application on windows using a VS code extension. Further VS code extensions can be developed and implemented to conduct weakest precondition semantic checking for students and weakest precondition logic checking for tutors.

# Background Theory

## CSSE3100 Reasoning About Programming

Course Requirements

Course Experiences

Course Needs

## Dafny

Syntax

Method

A method should be called with a method call ‘method’ and a method name with all words within the function name having capitals e.g. MaxNum. If a function requires inputs, these inputs will need to be specified with a name and a type and be positioned inside a set of brackets after the function name and separated by a colon; multiple inputs are separated by a comma e.g. MaxNum(a: int, b: int). If a function requires an output the inputs will be followed by a ‘returns’ with the returned objects name and type being specified the same as the inputs e.g. returns (n: int). Overall the specification of the function is method MaxNum(a: int, b: int) returns (n: int). The body of the code is then contained within [2].

method MaxNum(a: int, b: int) returns (n: int)

{

Body of code

}

Logical Statements

Semantics

Loops and Arrays

Preconditions and Postconditions

Weakest Precondition Proof

Invariants

Termination

Functions

Classes

## Boogie

## Z3 SMT Solver

## VS Code Extensions

## 2.6 Formal Methods Teaching

Examples

Introduction to Concepts

Massive Exercises

Teaching Abstract Skills

Small Projects

Formal Engineering Methods

Tool support

Tools are a vitally important when teaching formal methods as they are effective at helping students write, execute and test programs. Although, past experiences with teaching course with SOFL and VDM indicate teaching results in some circumstances. There are reasons for this; students need to learn formal methods like learning English as a second language, which requires students to learn syntax and semantics for that particular formal method. This is possible due to the small scale of the exercises posed to students. This also increases the student’s knowledge of abstraction techniques as it removes the temptation to copy, which is easy to do on a computer. Students also need to read and understand the formal methods without a computer running it. Essentially, handwriting is more effective than using a tool that will automatically format the style and format of the specifications. Tool support is however useful for productivity and reducing the chances of reducing mistakes. Tools such as IFAD VDMTools and SOFL GUI. Often students go through a hard time learning formal methods using handwritten techniques and then when given access to a tool, students are extremely happy due to the tools automation. [2]

IFAD VDMTools

SOFL GUI

SPIN

SMV

LTSA

Dealing with time constraint

# Timeline

## Gantt Chart

|  |  |
| --- | --- |
| Week | Task |
| Semester 2 2021 | |
| 0 | Select Task |
| 1-3 | Outline Requirements with Supervisor |
| 1-5 | Background Theory and Proposal |
| 1-5 | Proposal |
| 4-11 | Student Extension |
| 11 | Present Progress |
| Semester 1 2022 | |
| 1-12 | Tutor Extension |
|  |  |
|  |  |

# Ethics Assessment

# Bibliography

[1] K. R. M. L. Jason Koenig, *Getting Started with Dafny: A Guide*, p. 30. [Online]. Available: <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/12/krml220.pdf>. Accessed on: 15/7/2021.

[2] K. T. Shaoying Liu, Toshinori Hayashi, Toshihiro Nakayama, "Teaching Formal Methods in the

Context of Software Engineering " 2014.