# A Constructive Formalisation of Hoare Logic using the Interactive Theorem Prover Agda

Project Report
Word Count XXXX

Fraser L. Brooks 1680975
Supervisor: Vincent Rahli

Submitted in partial fulfillment
of the requirements for the degree of
Master of Science
(Computer Science)

at the
University of Birmingham
School of Computer Science
July 2021

## Abstract

Problem, Approach, What you Produced, Evaluation, What it all means. Nullam enim nisi, elementum eu pellentesque nec, facilisis id tellus. Sed erat sem, maximus vel fermentum et, fringilla quis est. Aliquam tempus nunc ac velit sollicitudin condimentum. Duis sed rutrum tellus. Curabitur rutrum finibus justo ut malesuada. Nullam tincidunt scelerisque iaculis. Quisque tempor massa id urna elementum, sit amet condimentum tellus euismod. Integer est eros, posuere et lacus finibus, pretium aliquam libero.

Mauris scelerisque aliquam vehicula. Fusce id sodales lacus, vitae eleifend ex. Nulla facilisi. Maecenas placerat sem imperdiet ex pellentesque, in ultricies odio vulputate. Nulla facilisi. Donec eget suscipit sapien. Aenean ipsum neque, cursus quis magna nec, porttitor viverra enim. Nulla tellus augue, convallis at mattis eget, pellentesque et odio. Etiam suscipit, libero nec pretium posuere, leo erat posuere enim, non accumsan nisi mi ac libero. Morbi tortor diam, venenatis vitae nisi non, vulputate hendrerit diam. Nam eget nulla turpis.

Donec posuere mi id pellentesque volutpat. Proin ultricies diam ut velit ultricies congue. Duis molestie aliquet lectus a sodales. Integer mollis sed leo in commodo. Suspendisse potenti. Etiam nec libero quis sapien porttitor vehicula. Proin eleifend dolor egestas, dapibus leo at, pulvinar velit. Proin id erat a turpis accumsan iaculis non sit amet purus. Integer porta, eros non elementum bibendum, libero eros elementum turpis, at fringilla sem sapien hendrerit mi.

Praesent consequat ut mi vel ullamcorper. Nullam a nisi bibendum, ultrices risus at, volutpat neque. Quisque tincidunt ac elit sed pellentesque. Integer sed tristique lectus. Pellentesque lacinia pellentesque magna in viverra. Aenean pharetra sit amet quam non molestie. Nam dictum quam sit amet eros sodales interdum. Morbi porttitor lectus lorem. Sed sagittis ante est, at tempor mauris dictum a. Suspendisse nec est vitae augue porta posuere quis eu velit. Nullam euismod nunc ut eleifend congue. Aliquam fermentum, lectus vel mollis tempor.

## Contents

## 1 Introduction

This is some text. That will not be in my report. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortisfacilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdietmi nec ante. Donec ullamcorper, felis non sodales.. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortisfacilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdietmi nec ante. Donec ullamcorper, felis non sodales...

## 2 Preliminaries & Literature Review

### 2.1 Weakest Precondition

How is one to give a semantics to computation? One answer is to provide a model of computation that denotes how a mechansim, a computation, or program[1] is to be computed; such as a Turing machine, or a FSA. Another way however is to describe what the mechanism, computation, or program can *do* for us; that is, specifiying what input states it can accept, and what states it will produce.

OR: given a desired output (of states) specifying, how (read, in which input states), if at all, it can produce the desired output.

This approach gives us a way of specifying a . . . without caring about the eventual form of the mechanism if indeed it ever takes form at all!

Computation as traversing the state space. Descartes - Cartesian product - [Dijkstra,p12]

Weakest precondition (according to Dijskstra) is unique when considered as a state space, but multiple predicates could denote the same space. (i.e. $x == y$ and $y == x$)

Strongest postcondition? [Gries, exercise 4 section 9.1]

'If for a given $P$, $S$, and $R$, $P \Rightarrow wp(S, R)$ holds, this can often be proved without explicit formulation — or, if you prefer, "computation", or "derivation" — of the predicate $wp(S, R)$'

Note that in the text [dijkstra], $wp(S, R)$, is used interchangeably as a predicate and as the state space that said predicate captures. With our constructive formalisation however, this lack of precision is not possible nor

[1]the three words here being used synonymously

desired, so we end up with the, perhaps superfluous, distinction between predicates and the state space that they describe. Meaning that under our formalisation, $wp(S, F)$ is empty when considered as a state space, but inhabited when considered as a predicate (inhabited uniquely by $F$ itself). This exposition also explains why $<< F >> S << Q >>$ is an inhabited type, as $F$ *is* a valid precondition of any computation for any postcondition (think absurd function, or bluff function). << Actually explained by the fact that what we have formalised is the weakest *liberal* precodnition!

Weakest Liberal Precondition is what has actually been formalised! (Need to work out the translation)

7 regions of the statesapce. As such, we can — if we wish — give a semantics to the notion of a derterministic mechanism as one in which the last four regions of the state space are empty.

## 2.2 Hoare Logic

## 2.3 Agda

## 2.4 Constructive Mathematics

'Agda is a constructive mathematical system by default, which amounts to saying that it can also be considered as a programming language for manipulating mathematical objects.' - MHE

## 2.5 Formal Proof

## 2.6 Applications

# 3 Specification

## 3.1 Obfuscating Interfaces

Might have also wanted to abstract away expression language (page 42 surface properties (Ligler))

x and y refer to possibly identical identifiers, while z is guaranteed to be distinct from both x and y. With more time a more expansive interface would be given allowing for as many identifiers as were necessary to reason about the desired program along with a mechanism for obtaining free identifiers from an expression, with the identifiers being represented as natural numbers, a new *free* identifier could always be generated by summing the numerical value of the identifiers present in the supplied expression, or in a given list.

## 3.2 Exppresion Language

Carving up state space. Every predicate denotes a subset of the statespace (which in our case is infinite).

(day = 23) Dijkstra's example

T/F, x == 2

Relationship between logical operators and set theoretic operators i.e. $\wedge \Leftrightarrow \cap$

Ought to have differentiated between non stuck-ness and termination. I.e. D(E) as domain of expression E, to eliminate divide by zero and non-defined variables in an expression, as that is a problem that can be handled distinctly from termination (i.e. (I think anway) that given a state S, and an expression E, one can deterministically/decidably determine whether or not it is a WFF).

## 3.3 Language

A SΔ is one of the following:

```
-- Commands/Programs/Mechanisms/Statements
-- Defined as ‘SΔ’ (read ‘State transformer’)
-- to emphasise all these different meanings
data Block : Set
data SΔ : Set where
  SKIP : SΔ
  WHILE_DO_ : Exp → Block → SΔ
  IF_THEN_ELSE_ : Exp → Block → Block → SΔ
  _:=_ : Id → Exp → SΔ

data Block where
  -- Terminator:
  _; : SΔ → Block
  -- Separator:
  _;_ : SΔ → Block → Block
```

### 3.4 Axioms & Rules

## 4 Implementation

### 4.1 Constructive Termination

### 4.2 Small Step Evaluation with Fuel

### 4.3 Termination Splitting

### 4.4 Axiom & Rules in Agda

### 4.5 Using the System to Reason about Programs

Reasoning about swap was done for constants but a similar proof could be implemented with any expression as long as the variables to be swapped were also swapped across the whole expression. For example:

```
< x == y + 1 > $\leftarrow$ sub x for z

z := x (z == y + 1) $\leftarrow$ sub y for x

x := y (z == x + 1) $\leftarrow$ sub z for y

y := z <y == x + 1 >
```

Hello, some maths: $\ll \mathtt{P} \gg \mathtt{C} \ll \mathtt{Q} \gg$ test test hello hello

## 5 Evaluation

### 5.1 Agda & Beautiful Verbosity

Getting better at working with Agda — thanks to unicode suport, psychological bias of aesthetic but incorrect signature.

## 5.2 Missteps

Downsides: Having to define all logical manipulations in the interface. Some mechanism for making this less painful would be nice. Equally however, said logical manipulations are not really of the main concern. Just because nothing can be postulated for the proof to have a computational meaning, doesn't mean we need be bound by this restriction. Indeed, there is little reason for us to care about whether or not our proofs have this computational context so long as we trust the parts we omit, and as these ommisions are oft simple logical manipulations, leaving them only in the record but not actually proved would be perfectly sensible and allow more focus to be on the manipulation of Hoare triples to reason about programs.

Actually, if I was doing it again, it would have been very sensible to not bother with implementing the interfaces at all. It was probably not a worthwhile use of my time to prove De Morgans law, or the commutativity of boolean and, in Agda when I could have instead focused on the more salient parts of the code base.

## 5.3 Future Work

Gries page 164 'a fine balance between the two' . . . but! automation, Infer,

. . . BUT WHAT ABOUT FUN??

parse a C program and create formal proof in background. Complain if fail

If the Agda code is to work as a library, there had ought to be some functionality for allowing potential users to add to Data-Implementation.agda, to define their own logical rules.

## 5.4 Conclusion

Hoare's surprise at test case success (see retrospective)

Not all that useful in practice, other tools are far more sophisticated and far better suited to practical applications, whether that be the verified software toolchain for reasoning about embedded software that needs to be correct, or Infer for catching a litany of bugs before they make their way into production. There's not a lot of room to claim that this Agda library has any real practical purpose. But that doesn't mean no purpose.

Constructive mathematics and interactive theorem provers are not front and center in mainstream mathematics, and perhaps never will be, but one oft talked about benefits of constructive mathematics is quite simply that it is fun to do [link MHE blog post] and it is on that note that one finds the most compelling use for this Agda library; it's actually a lot of fun — if you're of a particular sort — to reason about even the most simple of programs. I certainly found it enjoyable to reason about the swap program and have Agda check my workings for me.

Never been so intimately aquainted with the three lines of code comprising the swap function.

## 6 Appendix

## References

[1] Edsger W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[2] David Gries. *The Science of Programming.* Texts and Monographs in Computer Science. Springer, 1981.

[3] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[4] Charles Antony Richard Hoare. Viewpoint retrospective: An axiomatic basis for computer programming. *Communications of the ACM*, 52(10):30–32, 2009.

[5] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In John Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon, 1970.

[6] George T. Ligler. A mathematical approach to language design. In *Proceedings of the 2nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '75, page 41–53, New York, NY, USA, 1975. Association for Computing Machinery.

[7] George T. Ligler. The assignment axiom and programming language design. In *Proceedings of the 1976 Annual Conference*, ACM '76, page 2–6, New York, NY, USA, 1976. Association for Computing Machinery.

[8] John C Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pages 55–74. IEEE, 2002.