University of Nottingham

School of Computer Science

G53IDS Interim Report

Formal Verification of Combinatorics Problems using Type Theory

Alessio Cauteruccio

psyagca@nottingham.ac.uk

4287379

Supervised by: Thorsten Altenkirch

Contents

1 Introduction 1

1.1 Project Background 1

1.2 Project Brief & Aims 1

2 Motivation & Related Work 2

3 Methodology 3

3.1 Preliminary Research 3

3.2 Development Process 3

4 Dissertation Structure 4

5 Implementation 5

5.1 Language Selection 5

5.2 Software Selection 5

5.3 Software Implementation 5

6 Progress Update 9

6.1 Progress To Date & Reflections 9

6.2 Planning for Second Half of Cycle 11

7 Bibliography 12

1 Introduction

1.1 Project Background

The definition of combinatorics according to the Oxford Dictionary is "*The branch of mathematics dealing with combinations of objects belonging to a finite set in accordance with certain constraints*"[1]. Combinatorics at its base level is the study of counting, and as such is of vital importance to the study and further development of computer science due to its involvement in both high-level reasoning of theorems as well as its low-level involvement in the development of subsequent solutions. Problems such as the pigeonhole principle and the theorem of friends and strangers are utilised throughout problem sets in computer science, such as Ramsey Theory (the theorem behind friends and strangers) used for problems such as packet switching, finding "*an application of Ramsey Theory in the design of a packet switched network, the Bell System signalling network*" [2]. In addition to this using Type Theory and its implementation in Agda allows the link between mathematical reasoning and and programmed solution to be as close as possible, as "*Computational type theory was assembled concept by concept over the course of the 20th century as an explanation of how to compute with the objects of modern mathematics*"[3]

1.2 Project Brief & Aims

The aim in this project is to develop formal verifications of select combinatorics theorems that hold specialised importance to the study and development of computer science, using Agda. The theorems that will be specified are the *'Pigeonhole Principle'* and the *'Theorem of Friends and Strangers'*. The project will focus on creating formal verification of the proofs of these problems using Agda, a functional programming language based on Type Theory that acts as a proof assistant where proofs are written in a functional programming style, as such creating a link between the mathematical proof and a real world programming implementation of the theorem. The key objectives of this project are:

1. To investigate key combinatorics theorems that are utilised within areas of computer science
2. To develop a working simulation of these combinatorics problems
3. To prove the correctness of these problems by using the simulation of them

2 Motivation & Related Work

There are many academic papers studying many combinatorics problems and various subsets of these problems such as proofs of the problems themselves, proofs of the complexities, comparisons between different problems and so on. The theory of these problems have also been extensively used in programmed solutions of other projects to prove their correctness, in error checking and other uses. However there doesn't seem to be many, if any, projects that actually provide proofs of these problems in a programming language, rather being either relatively simplistic mathematical proofs and real world examples or used trivially as part of other solutions. Therefore currently there isn't currently a project that aims to achieve the goals that this project does.

As the project continues and more research is undertaken whilst developing the solutions, any related works that will aid in comprehension of the topic will be provided in this section in the final report. However as of now my findings haven't come across any works that are closely related enough to warrant description here.

3 Methodology

3.1 Preliminary Research

Due to the nature of the project there aren't any previous projects providing complete answers in relation to the project. As such initial research needed to be development based by gaining insights from various existing papers and deriving answers from partial solutions. Due to this the research portion of the project involved two main portions:

1. **Initial Research:** At the very start of the project research was undertaken to gain a base understanding of the problems chosen to be researched namely '*Pigeonhole Principle'* and *'Theorem of Friends and Strangers'* as well as research into the language being used 'Agda'
2. **Ongoing Research:** As the project started and as it continues, research is undertaken in conjunction with development as due to the nature of the problem set it is necessary to utilise more research as problems in development arise. In addition to this by allowing research to continue alongside development it allowed development to commence earlier

As known at the start of the project Agda will be used to develop a solution and as such research around the language of Agda and how to develop in its style of functional programming is necessary from the outset.

3.2 Development Process

Throughout this project, I will be the only person working on the entire process from start to finish. Due to this it makes sense to adopt a development style suited to both myself and the project itself. I will therefore be utilising the *Waterfall* process however slightly adapted to the project where research and development will occur simultaneously but still in a linear fashion. For example, development of *Pigeonhole Principle* will occur simultaneously with research into that problem set, but this will be completed before the development and research of *Theorem of Friends and Strangers* begins in earnest

4 Dissertation Structure

As stated before, this project will be undertaken using a modified version of the *waterfall* methodology and as such will proceed in a mostly linear fashion. There will be sequential steps undertaken which will all aid in the completion of the project. The structure of the project will be as follows:

1. **Abstract**: A section for quantifying the goal and the result of the project
2. **Introduction:** A section to provide background on the project and set out the initial aims and objectives
3. **Motivation & Related Work:** A section introducing the context of the project, containing an insight into why this project is novel, and a look into existing related work (if any), to derive findings from that work and to identify why this project is sufficiently different
4. **Research:** A section allowing for findings from research to be collated, including any relevant designs created
5. **Implementation:** A section providing a comprehensive description of the software, development of the solutions including initial mathematical proofs and also including the language and software used for development and reasons pertaining to these
6. **Testing:** A section defining any testing necessary
7. **Evaluation and Personal Reflection:** An evaluation of how the project went as a whole, and an introspective reflection on my efforts throughout the process
8. **Conclusions and Further Work:** Final conclusions on the project and any potential further work that could be undertaken to finalise, further refine or extend the project
9. **Appendices:** Any additional information pertaining to the project
10. **Bibliography:** Any references used throughout the dissertation

5 Implementation

5.1 Language Selection

As stated before, the language being used in this project is Agda. Agda is a dependently typed functional programming language, however more pertinent to this project it is also a proof assistant. The nature of the language being able to both prove and implement these problems is useful to create the bridge between mathematical proofs and programming. There are other proof assistants such as Coq, however Agda provides a functional programming interface to create proofs whilst simultaneously being a functional language in it's own right

5.2 Software Selection

Agda is developed to be written into Emacs. The Agda Emacs mode comes with an input method for easily writing Unicode characters, and also allows Agda files to be dynamically type checked without full compilation. Agda can recognise unicode characters and as such characters such as Gamma can be utilised in solutions, creating a more mathematical looking system but still compiling in the functional language

5.3 Software Implementation

At this point of the project, there has been no translation of the proofs into Agda. There has been development of examples to aid my understanding of the language and to begin to construct proofs in Agda itself. A selection of these examples are as follows:

This first code block shows an initial file of creating functions in Agda to gain a base understanding of using the language

```
data ℕ : Set where -- \bN \omega ω Ω λ
  zero : ℕ
  suc : ℕ → ℕ -- \to -> \-> →

data Bool : Set where
  true : Bool
  false : Bool
-- C-c C-l
_+_ : ℕ → ℕ → ℕ
zero + n = n
suc m + n = suc (m + n)

_*_ : ℕ → ℕ → ℕ
zero * n = zero
suc m * n = n + (m * n)

max : ℕ → ℕ → ℕ
max zero y = y
max x zero = x
max (suc x) (suc y) = suc (max x y)
```

```
_-_ : ℕ → ℕ → ℕ
x - zero = x
zero - y = zero
(suc x) - (suc y) = x - y

eq : ℕ → ℕ → Bool
eq zero zero = true
eq (suc x) zero = false
eq zero (suc y) = false
eq (suc x) (suc y) = eq x y

div2 : ℕ → ℕ
div2 zero = zero
div2 (suc zero) = zero
div2 (suc (suc x)) = suc (div2 x)

div3 : ℕ → ℕ
div3 zero = zero
div3 (suc zero) = zero
div3 (suc (suc zero)) = zero
div3 (suc (suc (suc x))) = suc (div3 x)

rem2 : ℕ → Bool -- true if remainder is 1, false otherwise
rem2 zero = false
rem2 (suc zero) = true
rem2 (suc (suc x)) = rem2 x

_&&_ : Bool → Bool → Bool
b && true = b
b && false = false

power : ℕ → ℕ → ℕ
power x zero = suc zero
power x (suc y) = x * (power x y)

factorial : ℕ → ℕ
factorial zero = suc zero
factorial (suc x) = (suc x) * factorial x
```

This second code block shows how Agda can be used as a proof assistant (N.B. This file is still under construction)

```
open import Data.Nat
open import Relation.Binary.PropositionalEquality
open import Data.Empty
open import Data.Bool

simple : 2 ≡ 2
simple = refl
```

```
simple2 : 2 ≡ 3 → ⊥
simple2 ()

cong' : {A B : Set}(f : A → B) → {x y : A} → x ≡ y → f x ≡ f y
cong' f refl = refl

assAdd : (i j k : ℕ) → i + (j + k) ≡ (i + j) + k
assAdd zero j k = refl
assAdd (suc i) j k = cong suc (assAdd i j k)

commPlusZero : (j : ℕ) → j ≡ j + zero
commPlusZero zero = refl
commPlusZero (suc j) = cong suc (commPlusZero j)

commPlusSuc : (i j : ℕ) → suc (j + i) ≡ j + suc i
commPlusSuc i zero = refl
commPlusSuc i (suc j) = cong suc (commPlusSuc i j)

commPlus : (i j : ℕ) → i + j ≡ j + i
commPlus zero j = commPlusZero j
commPlus (suc i) j = trans (cong suc (commPlus i j)) (commPlusSuc i j) -- suc (j
+ i) ≡ j + suc i

assTimes : (i j k : ℕ) → i * (j * k) ≡ (i * j) * k
assTimes zero j k = refl
assTimes (suc i) j k = { }0

commTimes : (i j : ℕ) → i * j ≡ j * i
commTimes i j = { }1

-- commutative monoid.

distr : (i j k : ℕ) → i * (j + k) ≡ i * j + i * k
distr i j k = { }2

-- semiring

symm : {A : Set} → (a a' : A) → a ≡ a' → a' ≡ a
symm a .a refl = refl
```

In addition to these rough workings of proofs for the *Pigeonhole Principle* are in progress with a focus on finding a proof by induction due to this style of proof being more natural to transpose into Agda rather than other styles, for example using contradiction. Despite this, initially it has proved to be useful to construct various styles of proofs to consolidate each other.

Generalised Pigeonhole Principle: If N pigeons are placed into M pigeonholes, then there is at least one pigeonhole containing at least ⌈N/M⌉ pigeons. For example if 5 pigeons are to go into 4 pigeonholes then at least one pigeonhole will contain [5/4] = [1.25] 2 pigeons

Example 1: Proof by Contradiction
Suppose N pigeons are to go into M pigeonholes and N > M.
Assume there is no pigeonhole with at least N/M pigeons.
Therefore, it follows that at any pigeonhole X, N[X] < N/M

∴ N < X*(N/M)

∴ N < M*(N/M)

∴ N < N

However, given that the number of pigeons must be strictly equal to N it follows that the conclusion is false.
Hence there exists at least one pigeonhole containing at least N/M pigeons

Example 2: Intuitive Proof by Induction

If N + 1 pigeons are distributed amongst N pigeonholes, at least one pigeonhole will have N>1 pigeons

Base Case: N = 1, N + 1 = 2, there is only one pigeonhole and two pigeons therefore they must both be in the same pigeonhole

Inductive Step: Assume this is true for N = K. In pigeonholes 1 to K there must be a pigeonhole M containing =<2 pigeons

Prove for K+1: As we know the assumption holds for K, there is now one extra pigeonhole to distribute between. As such there are 3 cases:

1. **Pigeonhole K+1 contains 0 pigeons**: Therefore K+2 pigeons must be distributed between the first K pigeonholes, and as K+2 pigeons contains K+1 pigeons and as seen previously the statement holds for K holes and K+1 pigeons, this holds as true
2. **Pigeonhole K+1 contains 1 pigeon**: Therefore K+1 pigeons must be distributed between the first K pigeonholes, and as seen previously the statement holds for K holes and K+1 pigeons, therefore this holds as true
3. **Pigeonhole K+1 contains 2 pigeons**: Therefore K pigeons are to be distributed between the first K pigeonholes. This on it's own cannot prove the statement, but if we extend it to look at the K+1 hole as well, this contains K>1 pigeons as in the statement and as such holds as true also

In addition to these three cases hole K+1 could also have <2 pigeons, however Case 3 would still prove this to be correct. Therefore as the statement is proven for K+1 and as such N+1, the proof must be True.

6 Progress Update

6.1 Progress to Date & Reflections

Due to the nature of this project I understood from the outset that a significant portion of the start of the project would be dedicated to gaining an initial understanding of Agda, the environment that I would be working in. This was incorporated into the initial Gantt chart *(Fig 1.)* created. I also understood that due to my lack of knowledge of Agda and the requirements to develop the necessary solutions it would be best for the project to build in plenty of buffer space for any sections of the project that would prove to run-over their initial allocated time frame. This proved to be partially correct however not entirely descriptive of how the project has gone to this point.

As development started I realised that there was much more to learn in Agda than I first anticipated. In addition to needing to refresh my memory on functional programming as a whole (necessary as I had not truly used functional programming since first year), there are nuances to Agda that are separate to other functional languages (i.e. Haskell). Past this there it is necessary to learn how to prove in Agda. This proves to be far more difficult than I first anticipated and as such has taken far longer to get to grips with and I am still in the process of truly getting to grips with the process.

Overall the beginning stages of this project have still remained within the scope of my initial Gantt chart which I has been instrumental in judging where my progress has gotten to at regular intervals. However it was soon evident that the Gantt chart both needed updating in terms of its timings. I also felt it was necessary to add some sections to the Gantt chart to better represent the current structure of the project and to highlight areas which as the project continues have proved to be more integral than first anticipated. This has resulted in an updated Gantt chart *(Fig 2.)* that is a more fitting representation of the necessary components of the project at this point. In comparison between the two charts there is now a dedicated section for research of the problems that run in parallel with development of the problems. The section for learning Agda itself has also been split into 2 sections allowing a dedicated portion to follow learning proving in Agda. Past these changes the chart maintains the same structure utilising the modified *Waterfall* methodology as described above.

Very early in the project I also realised that a more rigorous work schedule was required to allocate the necessary time for the project so as to stay on course to meet other coursework deadlines, any revision necessary throughout the year and also external circumstances that may affect the time frame. It has shown that diligent timekeeping across all aspects of the course is crucial to remain on track.

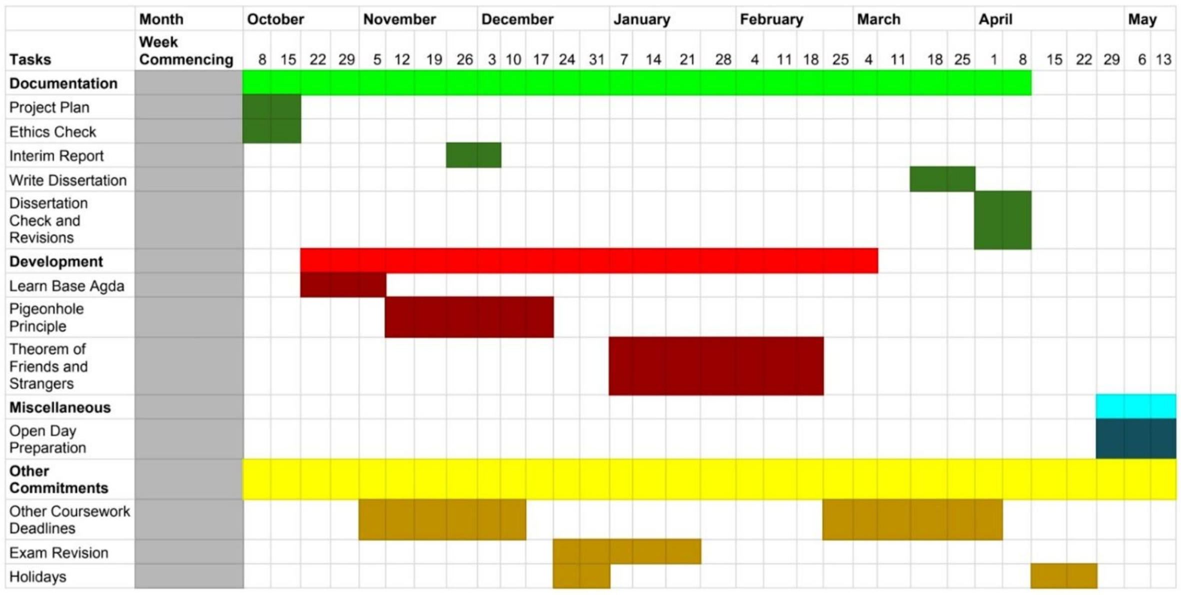

Figure 1: Initial Gantt Chart created at the start of the project

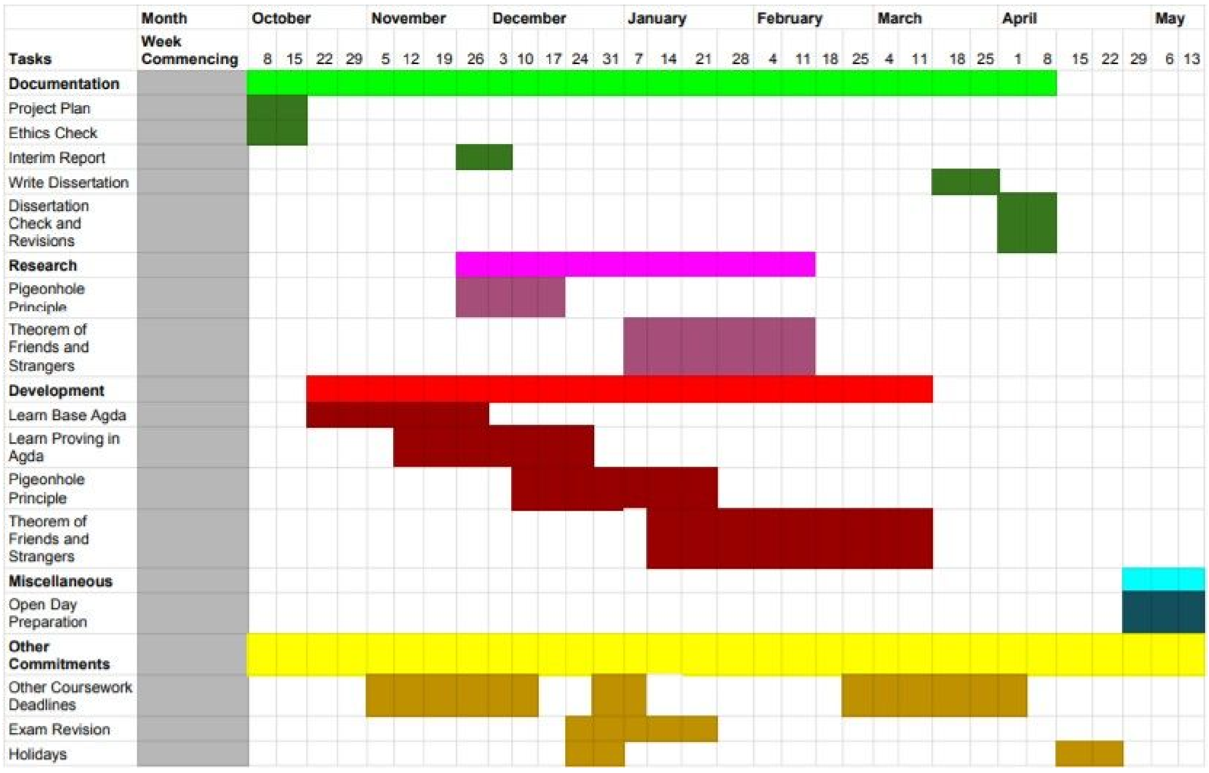

Figure 2: Updated Gantt Chart

At this interval of the project work has started relating to the pigeonhole principle. No actual coding for the solution has begun but research into the topic is underway and I am currently working on creating a proof using induction for the *Pigeonhole Principle.* It is currently an intuitive proof *(Example 2 in Section 5.3)* that I will first further formalise using mathematical induction and will then compose into Agda. The same process will later be used to formalise *Theorem of Friends and Strangers.* Prior to this in terms of development as stated previously time has been spent gaining an understanding of the language both in functional terms and as a proof assistant by completing examples and creating small proofs of my own.

Reflecting on my current progress in the project cycle, I find it hard to gauge my feelings on my progress. In one respect I can see how much I have learnt about Agda, about the problems that I will be quantifying and in functional programming and proofs as a whole, and as such I have a strong feeling that my work in this first half of the project cycle has served to lay a crucial foundation to carry forward into creating the solutions. In addition to this due to the nature of the project it is evident that there won't be masses of code and examples to provide as proof of progress in the project and this was a fact that I have known from the outset. Despite this, I feel as though progress has been slower than I would like, due to in equal measures external circumstances such as other courseworks and not allocating sufficient timekeeping or goal setting at the outset of the project, and taking longer than I would have liked to get up to speed with the language, proofs etc. This is something that although not ideal, was identified relatively early on in the project and I have worked to rectify to this point. I feel that the project is in a reasonably healthy state moving into the second half of the cycle, however due to the experience of this first half with a careful eye on stricter goal setting throughout to maintain development and hit target deadlines throughout.

6.2 Planning for Second Half of Cycle

The second half of the cycle will be focused almost solely on the development and implementation of the problems with research factored in. More specifically, the following will be completed:

1. **Pigeonhole Principle:**
    i. **Research:** Complete research on *Pigeonhole Principle*, collating any relevant resources and creating a mathematical proof by induction
    ii. **Implementation:** Implement proof into Agda
2. **Theorem of Friends and Strangers:**
    i. **Research:** Complete research on *Theorem of Friends and Strangers*, collating any relevant resources and creating a mathematical proof utilising a relevant proof strategy (induction, contradiction etc.)
    ii. **Implementation:** Implement proof into Agda
3. **Testing:** As of currently no testing is required as Agda can type check itself in development, however I will be investigating whether there is need for testing nearer the time of a complete solution
4. **Dissertation:** Completion of the components listed in *Dissertation Structure* and addition of any components not currently listed
5. **Evaluation Steps:** An evaluation of the work done will be undertaken to assess what future work could be done to further refine or extend the project

7 Bibliography

[1] Oxford Online Dictionary, accessed 18/10/2018;
https://en.oxforddictionaries.com/definition/combinatorics

[2] Applications of Ramsey Theory by Roberts, Discrete Applied Mathematics, Vol 9, 1984.;
http://www.cs.umd.edu/~gasarch/TOPICS/ramsey/ramsey.html

[3] Constable, Robert L. (2009), Computational Type Theory;
http://www.scholarpedia.org/article/Computational_type_theory