0.1 Introduction

This is a log file for the project. Here I'll explain my thought process and the things I've done during each week.

0.2 Week 1

0.2.1 Setting up the environment

Installed Coq. Use the command `sudo apt install coq`
Coq is a proof assistant. It works based on Curry-Howard correspondance which states that types are theorems and programs are proofs ie, when we are writing a proof it is essentialy a program, and the proof is said to be correct when the program is able to infer the type of all variables used.

For example consider $A \implies B$ which can be considered as a fucntion that takes things from A to B. but froma logical point of view this can be seen as given a proof for the proposition A this produces the proof for the proposition B.

- Coq uses constructive logic.
- implication as a function.
- true and false as inductivly defined propositions.
- negation as implication of false.

0.2.2 Interactive environment

The interactive environment can be brought up using the command `coqtop` . This brings a shell which looks like
`Coq >`
To exit the shell we can use the command `Quit.`
Remember that a fullstop is used to show it's the end of any command *(look above)*

Declarations

A declaration associates a name with a specification , the names follow the naming conventions of variables in any programming languages.
A specification is a formal expression which classifies the notion which is being declared, there are basically three types of specifications *logical propositions , mathematical collections , abstract types* .
Every valid expression associated with a specification , itself is a valid expression, called it's type $\tau\,(E)$.
We write $e : \tau\,(E)$ for the judgment e is of type E.
And type of a valid expression can be seen using the command `Check (expression).`
Eg:

```
Coq > Check 0.
      0 : nat
```

it says that 0 is a natural number.

```
Coq > Check nat.
      0 : Set
```

And nat is a Set.

```
Coq > Check Set.
      0 : Type
```

And Set is nothing but a type , So what is the type of type.

Surprisingly the answer is,

```
Coq > Check Type.
      0 : Type
```

ie, type is nothing but type.

0.2.3 CoqIDE

As a hello world program we can write a proof for the theorem . For all theorems if there exists a proof for it then there exists a proof for it. This can be written in coq as

```
Theorem my_first_proof : (forall A: Prop, A->A).
Proof.
 intros A.
 intros proof_of_A.
 exact proof_of_A.
Qed.
```

How the above written code works

Now that we have written a theorem . Let's see how Coq interprets it. Coq uses 3 different languages.

- The vernacular language manages definitions at the top level interactions. Each of its command starts with a capital letter: *"Theorem" , "Proof" , "Qed"*.
- The tactics language used to write proofs the commands starts with lower case letters *intros , exact , ...*
- The unnamed language of Coq terms is used to express what you want to prove. Its expressions use lots of operators and parentheses: $\forall A : Prop, A \Longrightarrow A$. (Technically, this language is a subset of the vernacular language, but it is useful to think of it as its own thing.)

Usually it's not the case that we'll write one theorem in vernacular language and tactics language together and just compiles it. In most of the cases we'll first write the vernacular definition and then write tactics one by one to prove whatever we have stated in the vernacular language.
This leads us to a point at which we need to know where we are at the proof. i.e. we need to know what all have we assumed and what all are left to prove. Coq IDE provides a window which shows exactly this . It can be accessed from the right side of the IDE

it will look something similar to :

```
A : Prop
proof_of_A : A
====================
A
```

all that is written above the bar is what we know to exist or what we've assumed to exist. These are called hypotheses. we refer to these as "the context" . Below the bar is what we are trying to prove called as "the current subgoal". "The goal" is what we are trying to prove and "The subgoal" is what we have to prove next.

Tactics !!
when we start out the proof. i.e. just after defining the theorem with the vernicular language our state will be.

```
======================
forall A : Prop , A->A
```