Journal Report 18

2/17/20-2/21/20
Rahul Mittal
Computer Systems Research Lab
Period 2, White

Daily Log

Tuesday February 18

Start attempting to learn Coq, go to Coq website, look around for a ”Getting Started”-type thing, can’t find one, google Coq tutorial, find one, start reading, install CoqIDE, continue reading.

Thursday February 20

Continue reading Coq tutorial, get to natural numbers.

Timeline

Date	Goal	Met
February 3	Be able to find all implications between statements in Colorado proofs	No, worked on figuring out hanging quantity
February 17	Be able to find all implications between statements in Colorado proofs, start work on combinatorics foundation in Coq	Finished implications for Colorado 1, 2, 3, not started Coq
February 24	Start work on combinatorics foundation in Coq	Read most of Coq tutorial
March 2	Finish Coq tutorial, go back to Haven’s thesis on using Coq for combinatorics, express Colorado 1 in Coq	
March 9	Complete Coq combinatorics foundation	

Reflection

This week's log was very short, especially in contrast to my previous logs, but ultimately there wasn't much to write about since I was just reading a tutorial. It might seem a bit wasteful to spend a whole week on a tutorial (especially since I didn't even finish it), but I think it's extremely important for me to understand Coq well considering how important it is to my project - and considering how weird it is at least initially.

Coq is actually very easy to understand once you understand it once you know the basics of how it works. What I've been saying to everyone for the past year without actually knowing Coq is true - it is basically like a programming language. More specifically, at least the way I understand it, it's based heavily on destructuring statements in various ways: for example, if you're trying to prove the statement 'A -> B', then you can use the so-called tactic 'intros proof_of_A', and you will now have the object 'proof_of_A' of type 'A', and have as your goal to prove 'B', which you might eventually do by getting an object 'proof_of_B' of type 'B' and then using the tactic 'exact proof_of_B'. In this way you decomposed the statement 'A -> B' into what it means: you've proven 'A', and now you need to prove 'B'.

Here's a proof in Coq of the fact that logical and is commutative just to give a more thorough example of what Coq is like:

```
Theorem and_commutes__again : (forall A B, A /\ B -> B /\ A).
Proof.
  intros A B.
  intros A_and_B.
  destruct A_and_B as [ proof_of_A proof_of_B].
  refine (conj _ _).
    exact proof_of_B.
    exact proof_of_A.
Qed.
```

The 'refine' tactic is another example of destructuring in Coq: you're refining the goal of proving 'B /
A' (B and A) into the two goals of proving 'B' and 'A' which form the 'conj' constructor for a proof of 'B /
A'. That's another thing I found interesting - in Coq, when you define new types of statements, you declare zero or more constructors for them. From the proving things perspective, a constructor is a way that you can prove that type of statement, but what's more interesting is that from a programming perspective, this is almost exactly like constructors in a normal programming language. When you define a new type of a statement, you're literally defining a new type - all statements in Coq are types, with objects of the type being proofs of the statement. So constructors are literally just constructors of object of that type, as in any other programming language.

Now knowing at least the basics of how Coq works, I can say that it's become much clearer to me how using it to prove the combinatorial statements that I want my project to prove is going to work (and hopefully it should become even clearer once I finish the section on Peano's axioms). My plan for next week (after I finish the tutorial) is to go back to the thesis by Andrew Haven that got me started on this project in which he uses Coq to solve the full house problem. Ideally I can just mostly use his work on combinatorics in Coq instead of having to write everything myself, but at the very least it should give me an idea of how to go about proving combinatorial statements in Coq.

If you're interested, the full tutorial is located at https://coq.inria.fr/tutorial-nahas.