University of North Carolina at Greensboro — Handout 3
CSC 495/693: Software Foundations — November 1, 2016
Prof. Stephen R. Tate

Graduate Student Projects

This handout provides information for graduate students taking this class as CSC 693.

As stated on the syllabus, all graduate students are required to complete an independent project in the class. These projects are required to be non-trivial applications of Coq to producing certified software for a useful purpose. There are three ultimate deliverables for each project: executable code that performs a useful task, Coq proofs that show that the code does what it is intended to do, and a project notebook that documents your work on the project. This last part is similar to an "engineering notebook" that is common in engineering development, or a "lab notebook" that is common lab science work, and serves to document not only successful approaches but also exploratory ideas that did not work out. Documenting the things you have tried, and decisions you have made (and why you made them) is an important part of a mature approach to project development. The writing in the project notebook isn't expected to be formal writing, but should clearly describe the work you have done and decisions you have made.

Possible topics and approaches: The only real requirement here is that your project be centered on useful code (not just isolated trivial functions like we've been looking at in the *Software Foundations* book) that is proved correct in Coq. Below are several suggestions for approaches, along with one or more project ideas in each category.

- The "useful code" can be written in Coq itself, properties proved about it in Coq, and then the code can be exported to a more standard and higher-performance functional language like ML (Coq supports this). Some code lends itself to this better than other code – in particular, code structured around recursion is going to be easier to work with than iterative code. For example, you could create a binary search tree data structure, and functions for searching, inserting, and deleting data from the binary search tree. Or you could implement recursive sorting algorithms such as mergesort and quicksort.

- The "useful code" could be in the limited imperative programming language introduced in the *Software Foundations* book. Since we haven't gotten to this part of the book yet, it would require reading ahead, but it would allow you to work with code that is closer to the style of programming you have seen before. Any standard algorithm could be interesting here, although keeping data structures simple is important. For example, standard dynamic programming algorithms for subsequence problems or for least cost edit distance might be feasible in this setting.

- Finally, you could include a manual (non-verified) rewriting step that takes useful code, converts it into Coq functions that capture its functionality, and prove properties that are satisfied by the Coq functions. The manual rewriting step clearly leaves a gap in the guarantees provided for the code, but this could be seen as one step in a larger project that would eventually close that gap with Coq-certified translation. For example, a challenging (but highly useful and important) project could involve manually transforming portions of the OpenSSL "big number" library to Coq and proving properties about it. This would involve proofs that relate the big number representation in OpenSSL to the Coq integer types, and then a proof that the OpenSSL addition function works properly (similar to the way the `binary_commute` and `binary_inverse` exercises showed that operations on your binary data type "worked correctly"). While the OpenSSL addition function is simple, isolated parts of the multiplication function could be worked on as well, but the entire multiplication function is too complex to tackle for a class project.

The project uses the following timeline (all parts should be turned in through Canvas by midnight on the due date). Note that the final project due date is December 13, in order to give you the maximum amount of time possible to work on your project. However, keep in mind that this is during final exams when you will have a fairly busy schedule of project and exam dates, so you need to manage your time sensibly.

Topic/Project Selection – due Tues., Nov. 8 (10 points). Within the next week I would like you to decide on a basic project idea. I'm happy to discuss any ideas you would like to explore, if you are uncertain if or how they could be used as a project. You should write a brief, one-paragraph summary that describes: what "useful code" will be at the core of your project, and what you plan to prove about that code.

Progress Report – due Tues, Nov. 29 (25 points). At this stage you should submit your current Coq file and project notebook. I am not expecting these to be complete or "polished" at this point, so don't worry about submitting a work-in-progress. If you have any particular uncertainties in your project that you would like advice on, put a note in your project notebook and highlight it. The main point is to give me a status report so that I can provide you with guidance that will help you complete the project.

Final Submission – due Dec. 13 (65 points). This is the due date for all project materials: All code and Coq files, as well as your project notebook, with a final entry in the notebook that gives a full status report on the project describing what was completed/accomplished, and what was not (or what the next steps would be if the project were continuing).