**Advisor:** Zhong Shao  
**Abstract:** Formal verification is the process of verifying the correctness of a computer program, using the logical reasoning techniques of pure mathematics. As we depend upon software, built up from programs, in more and more situations in daily life, it becomes ever more important that this software be reliable and trustworthy. In the most crucial applications, it is not enough to be satisfied that some human programmer has visually stepped through the code and determined it to be bug-free. For most significant projects today, which consist of thousands, if not millions of lines of code, such a claim is outright impossible. The solution is to use computers themselves to aid in the systematic verification of a program.

More specifically, interest in formal verification has motivated the development of proof assistants, a term used to describe a class of software which aids in proof development. Proofs assistants provide an environment in which proof writers may manage hypotheses and claims, an interface between real written code and the logical constructions necessary for reasoning about it, and, most importantly, the ability to check the validity of a proof according to the assistant’s internal logic engine. Perhaps the best known proof assistant is Coq, which was developed starting in 1984 by Thierry Coquand and Gerard Huet of inria. Coq’s applications range from the verification of real-world software such as CompCert or CertiKos, to being a computational aid in proving famous mathematical theorems such as the 4-color theorem or Feit-Thompson theorem.

This semester, I have been learning of the basics of formal verification using Coq, through *Software Foundations* by Pierce et al. In this report, I give a summary of what I have learned, which should be accessible to anyone familiar with basic mathematical logic (mainly induction) and a bit of functional programming. I begin with an overview of the basic logical ideas underlying Coq, with an emphasis towards the powerful reasoning derivable from inductively-defined objects. Next, I explain how Coq may be harnessed to reason about programs, using a tiny toy language, *Imp*, to model general principles found in many languages: variables, assignment, conditionals, and loops. Finally, I give a brief description of formal verification in CertiKos, a verified operating system developed by Yale’s FLINT research group, directed by Zhong Shao, who is also the supervisor of my project.