Foundations and Pragmatics of Four Automated Systems for Interactive Proof-Assistance and Certifiably-Safe Programming

Rationale and contents:

The need to certify ever-more complex software artifacts has spurred an unprecedented development of methodologies for analysis and verification based on principles of mathematical logic. The overarching goal is to ascertain that software artifacts, which implement complex algorithms or model interacting processes in wider computing environments, satisfy formally specified properties (typically related to safety or security, but not only). Success of these formal methodologies in real-world applications has depended on their being automated, partially or fully, and on their scaling up to large software packages beyond the ability of human agents to analyze and verify by hand.

In this course, we aim to take a closer look at four systems: Coq, Agda, Lean, and ATS, which are all under active development but mature enough to give an advance experience of what to expect from automated, or semi-automated interactive, formal verification of large and complex software. These four systems share a common theoretical core, primarily based on what are called dependently-typed lambda-calculi; but they also diverge significantly in many other respects, both in their theoretical foundations and their pragmatics.¹

While Coq and Lean are primarily interactive proof assistants, or were initially designed as such, Agda and ATS aim to create environments for certifiably safe programming. Nevertheless, all four systems can be used for both purposes: interactive proof assistance and certifiably safe programming, as well as combinations of both that fall under different headings that are as varied as: safe systems programming, computer-aided cryptographic proofs, formally-certified compilers, and many others from across the computer-science field.

About 2/3 of the course will be devoted to foundational aspects of the systems under consideration, and about 1/3 to their pragmatics and use in very simple applications. Material on foundations will be drawn from two main references:

(1) A tutorial on “Dependent Types” by D. Aspinall and M. Hofmann, which is Chapter 2 in the book Advanced Topics in Types and Programming Languages by B. Pierce, and

(2) A selection of chapters from the book Lectures on the Curry-Howard Isomorphism by M.H. Sorensen and P. Urzyczyn, culminating with what is called the Calculus of Inductive Constructions.

For purposes of comparison and experimenting with the four systems, we will apply them to simple use cases drawn from: the book Certified Programming with Dependent Types by A. Chlipala (examples in Coq), the book Verified Functional Programming in Agda by A. Stump (examples in Agda), the book Theorem Proving in Lean by L. de Moura et al (examples in Lean), and the website The ATS Programming Language by H. Xi (examples in ATS).

¹ These four by no means exhaust the list of automated or semi-automated interactive systems that aim at achieving similar goals of formal analysis and verification. Most notably, there are several systems – for example, Isabelle and HOL, and extensions based on them – whose core foundation is first-order and higher-order logic with competing claims of successes in several areas of computer science. Interesting though it may be, this is material for another graduate-level course.
Organization and logistics:

There are no formal prerequisite for this course. However, it will be conducted as a first-year graduate-level course, combining lecture style (presentations by the instructor) and seminar style (presentations by a subset of the students). Both registered students and auditors will be welcome, and should have a background and mathematical maturity equivalent to a completed Bachelor’s degree in CS – or else get instructor’s permission.

There will be a total of 27 lectures and 14 lab sessions, spread over 14 weeks in the course of the Spring 2019 semester at BU, starting on 22 January 2019 and ending on 2 May 2019.

*Meeting time:* Lectures on Tuesday and Thursday at 11 am, lab sessions on Wednesday at 2:30 pm.  
*Meeting place:* TBA (one of the classrooms in the MCS building at BU).

Performance in the course will be entirely based on homework assignments, some involving (more theoretical) hand exercises and some (more practical) implementation exercises in Coq, Agda, Lean, and ATS. There will be no mid-term and no final exam. Over the last 4-6 weeks of the semester students will undertake a small project on a topic of their own choosing or from a list of possible topics provided by the teaching staff, which will span the spectrum from theoretical (e.g. a report on a recent research article related to dependently-typed lambda-calculi) to the pragmatic (e.g. in-depth use of one of the four systems on some application or a comparative study of two or more of the four systems).

For inquiries, send email to:  
Assaf Kfoury (kfoury@bu.edu) and Kinan Dak Albab (babman@bu.edu).