Lecture 3

Topics

1. Since there are new students, this lecture will recapitulate points made in Lecture 1 from a week ago.

   (a) Reasoning about programs typically requires a specification language giving the intent of the program in a declarative language. This will include establishing its *type*.

   We can also reason only about programs behavior – how long does it run, is the code well organized, e.g. modular? This reasoning is guided by program structure.

   Large programs have modular structure, and we might reason about program structure, "in the large", e.g. the chores of data types, components, communication, etc.

   We are mainly interested in *richly typed logical specifications*.

   (b) It is increasingly worthwhile to use *formal reasoning* in a style that can be supported and checked by a proof assistant. This is especially effective for complex tasks or in *critical systems*.

   A recent verification success is from the HACMS project at DARPA (High assurance cyber physical systems). This year the red team failed to compromise the Air team's drone, even though the red team had the code. This was a stunning demonstration.

   (c) Formal reasoning by modern proof assistants such as Agda, Coq, Nuprl, F*, HOL, etc. involves considerable *automated reasoning assistance* from computers.

   (d) A comprehensive treatment of these topics will involve precise formal semantics for Programming Languages (PLs), precise formal analysis of data types, and synthesis of algorithms from proofs.

   (e) Proof assistants are also playing an increasing role in mathematics:

   Solving open problems.

   Checking hard (and long) proofs.

   Constructive proof assistants essentially solved the *foundations problem* for mathematics, how to avoid inconsistent theories and paradoxes, e.g. Russell's paradox.

   In so doing, they created a new branch of mathematics, *formal mathematics*. We will explore this topic as well.

(f) Not only did the constructive provers carry out much of the mathematical foundations program (of Hilbert, Herbrand, Weyl, Poincaré, etc.), they also provide the basis for a *foundational theory of computer science*. For example, proof assistants use a very advanced programming language with a completely formal semantics. Compilers can be verified, operating system kernels can be verified, much of computer science theory can be formalized and verified.

(g) There are many first-rate research problems involving proof assistants:

- Foundations of Homotopy type theory.
- Synthesizing distributed protocols.
- Verifying small distributed systems.

(h) We will teach elements of Coq and Nuprl, especially the Coq model of Nuprl. We will explore the converse question – *model Coq's Calculus of Inductive Constructions (CIC) in Nuprl's Constructive Type Theory (CTT).*

(i) We might illustrate proposal writing.

(j) Several good research questions and thesis topics will be discussed.

2. A look at the logic books – they could all be formalized in Agda, Coq, or Nuprl.

(a) Stephen Cole Kleene, *Introduction to Metamathematics*, 1952.

A classic, reprinted twelve times, last in 2000.

(b) J.R. Shoenfield, *Mathematical Logic*, 1967. (points to Kleene's book)

Formal systems, generalized inductive definitions.

(c) Jean H. Gallier, *Logic for Computer Science*, 1986.

An extensive treatment of proof systems and automated reasoning.

(d) J.L Bell & M. Machover, *A Course in Mathematical Logic*, 1977.

Organized around *Mathematical structures* (à la Bourbaki).

(e) H. Hermes, *Introduction to Mathematical Logic*, 1973.

"Mathematics is about proving theorems"

(f) George S. Boolos & Richard Jeffrey, *Computability and Logic*.

Chapter 9 is FOL done only semantically. The book is a classic in many ways, crisp treatments of key topics.

(g) Ian Chiswell & Wilfrid Hodges, *Mathematical Logic*, Oxford 2007.

"Short and sweet", metavariables, inductive definitions, *natural deduction*. They note that logic could be like presented Boolean algebra.

(h) Smullyan, *First-Order Logic*, Dover 1995. (Springer 1968).

A 155 page gem, full of magic and deep insights.

(i) Simon Thompson, *Type Theory and Functional Programming*, 1991.

This is a good resource for our course.

Chapter 1 Logic (natural deduction).

(j) Iman Poernomo, John Crossley, & Martin Wirsing, *Adapting Proofs-as-Programs, the Curry-Howard Protocol*, 2005.

3. Styles of Proof

(a) Hilbert style – see example from Kleene's book.

(b) Gentzen natural deduction style

$A\&B \Rightarrow A$

Proof

Assume $h : A \,\&\, B$

A by and-elimination

Qed

(c) Gentzen sequent style – top down refinement style

	$\vdash A \,\&\, B \Rightarrow A$	by $\lambda(x.\_\_\_)$
$x : A \,\&\, B$	$\vdash A$	by spread$(x;\, a, b.___)$
$a : A,\ b : B$	$\vdash A$	by a

The extract is the "program" $\lambda(x.\text{spread}(x;\, a, b.a))$

(d) Gentzen sequent style – bottom up

$x : A \,\&\, B$	spread$(x;\, a.b.___)$
$a : A,\ b : B$	use a
$a : A$	
$A \,\&\, B \Rightarrow A$	spread$(x;\, a, b.a)$