CS3110 Spring 2016 Lecture 26: Course Review

Robert Constable

Topics

- **Simple study guidelines**

  I am making up the exam, so *read the lecture notes* and review problem sets.

  Don't be in the position of student X in prelim: didn't attend lectures and didn't read the notes, so was totally unprepared.

  There will be an *essay question* related to the main points of the course:

  - Why OCaml is valuable.
  - Value of OCaml at Jane Street.
  - Relationship of OCaml to Coq proof assistant.

  Look for relationship to proof assistants Coq and Nuprl.

  - The type system: partial vs. total, type inference vs. proving typehood
  - Types and logic

  Think about the future role of CS in the world.

  - Role of PL innovation – *autocatylitic process*
  - The complexity of our tasks: building software vs building bridges

    Roman bridge in Portugal, "We built this bridge to last forever."

    Cyber security circa 2020

- **Logic and Types**
  - Here is how to solve $\sim\sim(\alpha \vee \sim\alpha)$ (done on the blackboard and sketched below)
  - There will be some question of the form "find a member of this type, if there is one, view the element as evidence (proof) that the typing relationship holds."
  - Know what *partial* types are
  - Know what *dependent* types are
  - Think about the "broader roles of types" illustrated by the $10M NSF Expeditions project for using Coq in CS education and the proposed $15B for cyberphysical weapons systems.
- Problem set material.
- Ties to the major trusts and challenges of CS.

Logic example

	$\vdash \sim\sim(\alpha \vee \sim\alpha)$	
	$\vdash ((\alpha \vee (\alpha \to \text{Void})) \to \text{Void}) \to \text{Void}$	by $\lambda(f.\_\_)$
$f : (\alpha \vee (\alpha \to \text{Void})) \to \text{Void}$	$\vdash \text{Void}$	by applying f
	$\vdash (\alpha \vee (\alpha \to \text{Void}))$	prove right side
	$\vdash \alpha \to \text{Void}$	$\lambda(x.\_\_)$
$x : \alpha$	$\vdash \text{Void}$	by applying f again
	$\vdash \alpha \vee (\alpha \to \text{Void})$	prove left side
	$\vdash \alpha$	by x

The program is $\lambda(f.\text{ap}(f; \text{R}(\lambda(x.\text{ap}(f; \text{L}(x))))))$.

The extract of this proof done in Nuprl looks like this:

$$\lambda f.(f\ (\text{inr}(\lambda p.(f(\text{inl}\ p)))))$$

where $f : \sim(P \vee (\sim P)) \equiv ((P \vee (\sim P)) \Rightarrow \text{False})$ and $p : P$. The full proof of this in Nuprl can be found here `http://www.nuprl.org/MathLibrary/LogicalInvestigations/theorem10.html`.

OCaml illustrates well the value of an expressive yet simple type system. Extending the system to include dependent types makes it vastly more expressive, capable of rendering most concepts from modern mathematics. To treat mathematics it is also critical to go beyond *partial types* and include the total types.

One of the remarkable benefits of adding dependent total types is that it is possible to fully express constructive logic. Contrary to popular belief, this logic is more expressive and more general than so called classical logic based on "truth values." Instead of truth values, constructive logic is grounded in *computational evidence*. It includes classical (Boolean valued) logic as a special case. That is possible because the "classical logical operators" can be expressed using the constructive ones. For example $\alpha \vee \sim\alpha$, is expressed as $\sim\sim(\alpha \vee \sim\alpha)$. The classical *existential quantifier* is defined as $\sim\forall x{:}\alpha.\ \sim\beta(x)$.

We briefly discussed *dependent types* since they are used in the Agda, Coq, Nuprl, and F* proof assistants. In a previous version of the course we used the following dependent type only for specifications: $x{:}\ \alpha$ *where* $b(x)$ for $b{:}\alpha \to$ bool. We mentioned that $x : \alpha * \beta(x)$ is another useful dependent type.

We can "simulate" the dependent type $x{:}\ \alpha$ *where* $b(x)$ as follows. Consider the dependent type $x{:}\ int$ *where* $x \geq 0$. We can get the effect using `nat:fun x → if x < 0 then -x else x`. Here is OCaml code that uses this method.

```
# let nat = fun x -> if x < 0 then -x else x;;
val nat : int -> int = <fun>

# let rec add (x:int)(y:int) : int = if x = 0 then y else (add (x-1) y) + 1;;
val add : int -> int -> int = <fun>

# add (nat (-5)) 6;;
- : int = 11

# add (-5) 6;;
Stack overflow during evaluation (looping recursion?).
```

Topics definitely covered on the final exam

1. OCaml operational semantics. (Lecture 2)

   In addition, we cover *lazy evaluation* as an option as discussed in comparing *fix* and *efix*. (Lecture 20)

   Previous versions of the course included defining a function that counts the steps of evaluation for a simple subset of the language, e.g. OCalf.

2. OCaml types and type checking rules. (Lectures 3, 4, 6, 7, 21)

3. Definition of the *constructive reals* and the operations of addition and multiplication. Method for proving $x \leq y$, Lemma 2.18, and the method of creating an irrational number by diagionalization over the rationals, Theorem 2.19. (Lectures 11, 12, 13, 15)

4. Executing convex hull on a list of distinct real numbers. (Lectures 16, 17)

5. What is a proof assistant? What kinds of information were provided with the Nuprl proof assistant? (Lecture 5, 8, 9, 18, 19, 25)

6. Binary search tree operations. (Lectures 18, 19)

7. Understand the leader election in a ring protocol and 2/3 consensus. (Lectures 22, 23, 25)

8. How can consensus protocols fail to give answers? (Lectures 22, 23, 25)

9. What is a message sequence diagram? What is local order in such a diagram? What is the causal order relationship? (Lectures 22, 23, 25)

10. Be able to prove the agreement and validity properties for 2/3 consensus. (Lectures 22, 23, 25)

Historical context and historical moments for computer science

Comparison to physics and biology – the 20th century sciences

Einstein	Turing (did his PhD at Princeton)
Heisenberg	McCarthy (MIT then Stanford)
Bohr	Newell (CMU)
Planck	Simon (CMU)
Oppenheimer	Hoare (Oxford)
Bethe	Hartmanis (Cornell)
	Hopcroft (Cornell)
	(An American science)

The next fifty years – some speculation, some issues, some dangers

A.I. exploring the universe
A.I. augmented humans
The cyber security issue.

> "In the game of life and evolution there are three players at the table: human beings, nature, and machines.
>
> I am firmly on the side of nature. But nature, I suspect, is on the side of the machines."
>
> -George B. Dyson
> *Darwin among the Machines, the evolution of global intelligence*, 1997.

George Dyson born in Ithaca, NY in 1953, son of the physicist Freeman J. Dyson.

> "The scientist both seeks and makes truth."