Intro to Formal Methods	Lecture 6
CS 5860 Fall 2014	Weds. Sept. 17

Lecture 6

Reading: Please read the notes on proof rules posted on the course web page: *Proof Rules for Intuitionistic First-Order Logic in Refinement Style.* Also read Simon Thompson Chapter 4, pages 67-95. This chapter covers the subject of Lectures 2-6 in this course. Also read Thompson section7.2 on the *subset type*, pgs 267-270.

Task: Confirm that all evidence, programs, and data that is created using our proof rules is *uniform*, that is, it is evidence for the formula for all possible atomic propositions. For instance, $\lambda(x.x)$ is evidence for $A \Rightarrow A$ for all possible choices of specific propositions A. The evidence is *polymorphic.*

We noted in lecture 5 that $(P \vee \sim P)$ can not have polymorphic evidence. If there is evidence, it depends on the proposition, as in $(x \leq 0) \vee (x > 0)$ or $Prime(4) \vee \sim Prime(4)$.

Conclude that $P \vee \sim P$ can not be proved using our rules. We say that it is not uniformly valid because we can demonstrate that there can't be polymorphic evidence. We discussed this simple observation in class.

*I would be interested in reading how you would express this observation if you wanted to include it among your Problem Set 2 answers.

Here is another example of how to derive solutions to programming problems expressed as types. Recall that we discussed that proof search by this method for the propositional calculus $(\&, \vee, \Rightarrow, False)$ will succeed if there is evidence. Moreover, we can arrange a search that will halt and report that there is no evidence if there is none. Investigating and implemeting such an algorithm would be a very strong course project.

We will consider two derivations for the same formula:

$\vdash (P \Rightarrow Q) \Rightarrow (Q \Rightarrow \bot) \Rightarrow (P \Rightarrow \bot)$	by $\lambda(f.\_\_\_)$
$f : (P \Rightarrow Q) \vdash (Q \Rightarrow \bot) \Rightarrow (P \Rightarrow \bot)$	by $\lambda(g.\_\_\_)$
$g : (Q \Rightarrow \bot) \vdash (P \Rightarrow \bot)$	by $\lambda(p.\_\_\_)$
$p : P \vdash \bot$	by $apseq(f; \_\_\_; w.\_\_\_)$, note $f(p) = w$
$w : Q \vdash \bot$	by $apseq(g; \_\_\_; v.\_\_\_)$
$\vdash Q$	by w
$v : \bot \vdash \bot$	by v

The program is $\lambda(f.\lambda(g.\lambda(p.g(f(p)))))$

Here is another derivation where we use g before f:

$\vdash (P \Rightarrow Q) \Rightarrow (Q \Rightarrow \bot) \Rightarrow (P \Rightarrow \bot)$	by $\lambda(f.\_\_)$
$f : (P \Rightarrow Q) \vdash (Q \Rightarrow \bot) \Rightarrow (P \Rightarrow \bot)$	by $\lambda(g.\_\_)$
$g : (Q \Rightarrow \bot) \vdash (P \Rightarrow \bot)$	by $\lambda(p.\_\_)$
$p : P \vdash \bot$	by $apseq(g;\_\_;v.\_\_)$
$\vdash Q$	by $apseq(f;\_\_;w.\_\_), (f(p) = w)$
$\vdash P$	by p
$w : Q \vdash \bot$	by w
$v : \bot \vdash \bot$	by v

We know $w = f(p)$ and $v = q(w)$, so the realizer is $\lambda(f.\lambda(g.\lambda(p.g(f(p)))))$

Further review of lecture 5 with additional comments:

1. How did Brouwer argue that $\sim\sim (P \vee \sim P)$ is valid?
2. The types for $\forall x$ and $\exists x$ are called *dependent types*. Why?

   They are very useful in programming, but so far only Agda, Coq, and Nuprl use them in their programming languages. They are too "expensive" and for Nuprl, type checking is not decidable, one must prove that a typing is correct.

We will examine another very useful dependent type not available in Agda and Coq, the *subset type* or *refinement type*, $\{x : D | P(x)\}$. See the Thompson realizer on this type.