Lecture 5

Here is a solution to the task of programming the evidence for \( \sim (P \lor \sim P) \). It was gratifying that students solved this. Here is a solution using the proof rules (that underpin Coq and Nuprl).

Recall that \( \sim (P \lor \sim P) \) is \( ((P \lor \sim P) \Rightarrow \bot) \Rightarrow \bot \).

\[
\vdash (P \lor \sim P) \Rightarrow \bot \\
f : (P \lor \sim P) \Rightarrow \bot \vdash \bot \\
\vdash (P \lor \sim P) \\
\vdash \sim P \\
p : P \vdash \bot \\
\vdash P \lor \sim P \\
\vdash P \\
w : \bot \vdash \bot \\
v : \bot \vdash \bot
\]

We compute \( \text{apseq}(f; \text{inl}(p); w) \) as \( w = f(\text{inl}(p)) \), then compute \( v \), \( \text{apseq}(f; \text{inr}(\lambda(p.\text{inl}(p)))); v) \). So \( v = f(\text{inr}(\lambda(p.f(\text{inl}(p))))) \).

The realizer, or program is

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

Whenever we have \( \text{apseq}(f; a, v.v) \) the result if \( f(a) \). If we have \( \text{apseq}(f; a; v.exp(v)) \) the result is \( \text{exp}(f(a)) \).

**The Predicate Calculus (First-Order Logic)**

We will now take up reasoning about logical operators called quantifies. They are written as \( \forall x \) and \( \exists x \), “for all \( x \)” and “exists \( x \)” respectively. We have used these in stating Euclid’s Theorem and the Goldbach conjectures.
**Euclid's Theorem:** There are an unbounded number of primes.

We can also say: For every number \( n \), we can find a prime number larger than \( n \).

\[
\forall n : \mathbb{N} \exists p : \mathbb{N}. (n < p \& \text{Prime}(p))
\]

We say that \( p \) is prime if and only if its only divisors are 1 and \( p \) itself. We exclude 1 nowadays so that we can state the **Fundamental Theorem of Arithmetic** in a crisp way: Every positive natural number greater than 1 is a product of primes, and that factorization is unique.

The atomic formulas of Predicate Logic are the atomic propositions \( A, B, C, \ldots \) that we have already seen along with predicate letters, \( A(x), A(x, y), A(x, y, z), \ldots B(x), B(x, y), B(x, y, z) \ldots \). These are atomic propositions indexed by elements of a domain of discourse \( D \). We take \( D \) to be any type, even an empty type.

In some accounts of the logic, most in fact, the type \( D \) is assume to have at least one element.

Another way to think about \( A(x), A(x, y), \ldots \) is as the ambiguous value of a **propositional function**

\[
A : D \to \text{Prop} \quad \text{and if } x \in D, A(x) \text{ is a proposition.}
\]

\[
A : D \times D \to \text{Prop} \quad \text{and if } x, y \in D, A(x, y) \text{ is a proposition.}
\]

\[
\vdots
\]

\[
A : D \times \cdots \times D \to \text{Prop}, \text{written } A : D^n \to \text{Prop}
\]

Typical examples for \( D \) the natural numbers \( \mathbb{N} = \{0, 1, 2, \ldots \} \) are these **predicates** or **relations**:

\[
A(x) \text{ is } x = 0 \text{ or } x = 1 \text{ or } x > 0
\]

\[
A(x, y) \text{ is } x = y \text{ or } x < y \text{ or } x > y
\]

We need to acquire the skill of translating ordinary English sentences about a domain of objects into predicates.

*See Simon Thompson’s book *Type Theory and Functional Programming* available for free at the CS5860 web page.

What is the meaning of these quantifies? Can we provide a computational meaning?

It is remarkable that we can give a completely natural computational meaning for these new logical operations using types. The explanation is again due to Brouwer. The formulation of the semantics using types in a precise way is a modern discovery that can be traced to a few logicians. The key contributors are Kleene, Heyting, Kolmogorov, Beth, and Howard. (Bill Howard is still active, and we just corresponded about this idea last night!)

We need two basic dependent types:
1. Generalize $A \rightarrow B$ to a family of types $B(x)$ to get $x : A \rightarrow B(x)$, those computable functions, which given $a \in A$ produce a value in type $B(a)$. Note, $B$ is indeed by $A$.

2. Generalize $A \times B$ to a family of types $B(x)$ indexed by $A$, denoted $x : A \times B(x)$. These are ordered pairs $<a,b>$ such that $b \in B(a)$.

Examples of valid quantified propositions:

1. $\forall x, y. A(x, y) \iff \forall y, x. A(x, y)$
2. $\exists x, y. A(x, y) \iff \exists y, x. A(x, y)$
3. $\exists x. A(x, x) \Rightarrow \exists x \exists y. A(x, y)$
4. $(\exists x. A(x) \lor \exists x. B(x)) \iff \exists x. (A(x) \lor B(x))$
5. $\exists x. (A(x) \& B(x)) \Rightarrow \exists x. A(x) \& \exists x. B(x)$
6. $\forall x. A(x) \lor \forall x. B(x) \Rightarrow \forall x. (A(x) \lor B(x))$
7. $\exists x. A(x) \Rightarrow \sim \forall x. \sim A(x)$
8. $\forall x. A(x) \Rightarrow \sim \exists x. \sim A(x)$
9. $\exists x. \sim A(x) \Rightarrow \sim \forall x. A(x)$

Finding evidence for $\exists x. A(x) \Rightarrow \sim \forall x. \sim A(x)$.

$\exists x. A(x) \Rightarrow \lambda(e. \lambda(na.spread(e; d, a.na(d)(a))))$

$\exists x. A(x)$

$\forall x. (A(x) \Rightarrow False)$

$\forall x. A(x)$

$\lambda(e. \lambda(na.spread(e; d, a.na(d)(a))))$

$e \in \exists x. A(x)$

$na \in \forall x. (A(x) \Rightarrow False)$

$na(d) \in A(d) \Rightarrow False$

$na(d)(a) \in False$

We define $\forall x. A(x)$ as $x : D \rightarrow A(x)$ and $\exists y. A(y)$ as $y : D \times A(y)$

It sometimes is suggestive to write the quantifiers as $\forall x : D.A(x)$ and $\exists x : D.A(x)$.

Let’s consider examples for $D$ the type of natural numbers $\mathbb{N}$:

$\forall x : \mathbb{N}. (x = 0 \lor \sim (x = 0))$

To know this we need a computable function that can produce evidence. While the exact details might not be entirely clear, we can produce a computable function to give this result:

$\exists x : \mathbb{N}. (x > 0)$
\[ \exists x : \mathbb{N}. (x = 0) \]
\[ \forall x : \mathbb{N}. \exists y : \mathbb{N}. (x < y) \]

How does this work?