Advanced Progamming Languages	Lecture 37
CS 6110 Spring 2015	Wed. April 29, 2015

Lecture 37

Topics

1. PS5 is posted and due Friday May 8, next Friday. The posted material on Turing machines should be helpful as well as the recommended reading and the lecture notes.
2. The main topic of this lecture is the *connection between logic and types*. It is described under various names such as propositions-as-types, proofs-as-programs, the Curry-Howard isomorphism, the semantics of evidence.
3. We start with a look at universes and quotient types and comments on PS5.

Comments on PS5

Please note the *reading from Thompson*. In particular read the sections on type theory and read Problem 1 to go along with the lecture.

We can summarize the type theory so far with this table. Note, we can think of this theory as a relatively simpe extension of the typed λ-calculus in the style of Curry, that is, terms do not come with types. This table is also given inthe notes from Kreitz posted with lecture 35.

Types	Canonical Elements	Noncanonical elements
$A \to B$	$\lambda(x.b)$	$ap(f;a)$
$x{:}A \to B(x)$	$\lambda(x.b)$	$ap(f;a)$
$A \times B$	$<a,b>$	$spread(p;x,y.u)$
$x{:}A \times B(x)$	$<a,b>$	$spread(p;x,y.u)$
$A + B$	$inl(a),\ inr(b)$	$decide(p;l.u;c.v)$
$Void$	(none)	$any(t)$
$\mathbb{Z}$	decimal integers	$+$, -, $*$, $\div$, rem
$\{x{:}A\|B\}$	elements of A that satisfy B	
$A//x,y.E(x,y)$	elements of A	noncanonical elements of A

Universes

Every type belongs to a universe. These are also called *large types* for $\mathbb{U}_1$, *very large types* for $\mathbb{U}_2$, and very very large types for $\mathbb{U}_3$, etc., and the hierarchy goes on: $\mathbb{U}_1 \in \mathbb{U}_2 \in \mathbb{U}_3 \in \mathbb{U}_4 \in ... \in \mathbb{U}_i \in \mathbb{U}_{i+1}$,...

All universes are closed under the basic type constructors, and they are *cumulative*, i.e.

$\mathbb{N} \in \mathbb{U}_1,\ \mathbb{N} \in \mathbb{U}_2,$ ...

$\mathbb{N} \times \mathbb{N} \in \mathbb{U}_1,\ \mathbb{N} \times \mathbb{N} \in \mathbb{U}_2,\ \mathbb{N} \to \mathbb{N} \in \mathbb{U}_2,$ ...

If $A \in \mathbb{U}_1,\ B : A \to \mathbb{U}_1$ then $x : A \to B(x) \in \mathbb{U}_1,\ x : A \times B(x) \in \mathbb{U}_1$.

We get most of our work done in $\mathbb{U}_1$ and $\mathbb{U}_2$. N. de Bruijn suggested that almost all mathematics can be done already at $\mathbb{U}_3$.

Quotient Types (Background on quotient types – please read Thompson 7.5 page 279-281)

Equality is critical to the very definition of types. Recall Bishop's definition of a set, it applies to types but Bishop wanted to make his mathematics read well for mathematicians, so he used the word set. So $\mathbb{Z}$ has a standard equality relation $0 = 0 \in \mathbb{Z},\ 1 = 1 \in \mathbb{Z},\ -1 = -1 \in \mathbb{Z}$.

The *noncanonical* elements of a type are those expressions built from its constructors, thus for $\mathbb{Z}$ we have 1+1, $2 * 3,\ 2^3,\ f(3)$ for computable function f. These expressions are in $\mathbb{Z}$ iff they reduce to a *canonical member*, e.g. $2^3 = 8 \in \mathbb{Z},\ 2 - 5 = -3 \in \mathbb{Z}$.

We want to be able to "change the equality" on a type, as in defining the integers modulo 2. For example 0=2 mod 2, 1=3 mod 2, etc. Note, $x = y$ mod n is an equivalence relation.

We can picture this particular quotient types as:

$$\begin{pmatrix} 0 \\ 1\text{-}1 \\ 1+1 \\ 2 \\ 2 * 2 \\ 4 \\ \vdots \end{pmatrix} \begin{pmatrix} 1 \\ 2\text{-}1 \\ 2+1 \\ 3+2 \\ \vdots \end{pmatrix}$$

We first define the new equality as an *equivalence relation* on $\mathbb{Z}$. For example $x = y$ mod 2 means $\exists n.\mathbb{N}.(x - y =_{\mathbb{Z}} 2 \cdot n)$. Note, we use $=_{\mathbb{Z}}$ to define $x = y$ mod 2.

What would $\mathbb{Z}$ mod 3 look like?

We denote these types by the quotient operation on a type. For the type $\mathbb{Z}$, we could have $\mathbb{Z}//(x = y \bmod 2)$, $\mathbb{Z}//(x = y \bmod 3)$, $\mathbb{Z}//x, y : (x = y \bmod 5)$, etc. All are different equalities, defining different types.

Propositions-as-types Principle[1]

In looking at programming logics with block structure formats for natural deduction, we noticed a neat connection between implication and the function types. Recall the rule for implication.

$A \Rightarrow B$	$A \to B$
Proof	**Function**
Assume $x : A$	$x : A$
$\vdots$	$\vdots$
B	$b(x) : B$
Qed	**End**

Consider how we would prove these propositions.

$A \Rightarrow A$	$A \& B \Rightarrow A$	$A \Rightarrow A \vee B$
Proof	**Proof**	**Proof**
Assume $x : A$	Assume $x : A \& B$	Assume $x : A$
$\vdots$	$\vdots$	$\vdots$
A by x	B by $right(x)$[2]	$inl(x)$
Qed	**Qed**	**Qed**

In these simple examples, the proof is building *computational evidence* for the proposition, telling us how to provide evidence for why we believe/know the propsition.

Universal Quantifier Rules

Introduction Rule

$\forall x : D . P(x)$
Proof
Arbitrary $x : D$
$\vdots$
$P(x)$ by $p(x)$
Qed

$\lambda(x.p(x)) \in \forall x : D . P(x)$

Elimination Rule

If $\forall x : D . P(x)$ is known, then given a $d \in D$, we can conclude $P(d)$. The justification is that we know $f \in \forall x : D . P(x)$, then $f(d) \in P(d)$.

[1] I think this is one of the most fascinating features of type theory. It has a long history, probably originating with L.E.J. Brouwer from about 1907, roughly the time Russell was developing type theory. I learned about it from Kleene (actually a visitor trying to impress Kleene) as *realizability.* This idead made it into CS in 1971.

[2] The official Nuprl term is $spread(x; a, b.a)$.