Advanced Progamming Languages	**Lecture 20**
CS 6110 Spring 2015	**Wed. March 11, 2015**

Lecture 20

Topics

1. Problem Set 3 – two problems presented today.
2. Motivation for "equational reasoning" in type theory, *new results*.
3. Goodstein approach – some philosophy and history, online with lecture notes.
4. Goodstein recursive arithmetic.

---

1. Problem Set 3

   (a)

   We have used primitive recursion with simple types,

   e.g. $add : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$

   $$\begin{cases} add\ 0\ y = y \\ add\ S(x)\ y = S(add\ x\ y) \end{cases}$$

   Prove that *add* and *mult* as defined before are total functions on $\mathbb{N}$, e.g. have type $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$. We could also define the type $\mathbb{N} \times \mathbb{N}$ of ordered pairs of numbers, $< n, m >$. In this case we could assign the type $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$

   (b)

   We can define *higher-order* primitive recursion as follows:

   $R\ a\ b\ 0\ =\ a$
   $R\ a\ b\ S(n)\ =\ b\ n\ (R\ a\ b\ n)$

   Where $a \in \alpha,\ b \in \mathbb{N} \to \alpha \to \alpha,\ 0 \in \mathbb{N},\ S : \mathbb{N} \to \mathbb{N}$

   $R :\ \alpha \to (N \to \alpha \to \alpha) \to (\mathbb{N} \to \alpha)$

   Define $\sum_{i=0}^{n} f(i)$ with higher-order primitive recursions.

   Give the types. Prove that functions defined by higher order recursion from (total) computable functions are total. Take α to be $\mathbb{N}$ for the proof.

2. Motivation for equational reasoning.

   A great deal of reasoning in type theory is equational over equations of the type $t_1 = t_2 \in T$ for various types T and for various notions of equality, such as:

   $$\begin{array}{lll} t_1 \sim t_2 & \text{in} & \text{Base} \\ t_1 \sim t_2 & \text{in} & \overline{\mathbb{N}} \text{ (over partial types)} \\ t_1 = t_2 & \text{in} & \mathbb{N} \text{ (over total types)} \end{array}$$

   Abhishek and Dr. Rahli gave us a deeper account for partial types in Nuprl. The line of research came from CS6110 in 2012.

   A simple example of this style of reasoning goes back to Skolem in 1934 – that's why logic is strong in Sweden. We examine the 1957 approach of R.L. Goodstein for numbers and recursive number theory.

   He uses only primitive recursion and *double recursion*.

   $$\begin{cases} G(0,n) \text{ is a given function, say } f(n) \\ G(m+1,0) = a(m, G(m, b(m))) \\ G(m+1,n+1) = c(m,n,G(m,d(m,n,G(m+1,n))),G(m+1,n)) \end{cases}$$

   Goodstein starts with a very cogent account of how he conceives of the type $\mathbb{N}$. We discuss this a bit. You are urged to read the account included with these notes.

Expressing mathematics in a programming language

We use *add*, *mult*, and these functions to define logic:

$$\begin{array}{ll} pred(0) = 0 & monus(x,0) = x \\ pred(S(x)) = x & monus(x,S(y)) = pred(monus(x,y)) \end{array}$$

Let $x \dot{-} y$ abbreviate $monus(x,y)$.

Define the positive difference $|x,y|$ by $|x,y| = (x \dot{-} y) + (y \dot{-} x)$.

Define $a = b$ for a and b numbers to be a true proposition iff $|a,b| = 0$.

Goodstein defines $\alpha(|a,b|)$ as the *number* of the proposition $a = b$.

We can define the logical operators on propositions as follows.

Let p and q be propositions $a = b$ and $c = d$ respectively. Then

$$\begin{array}{lll} \sim p & \text{is} & (1 \dot{-} |a,b|) = 0 \\ p \,\&\, q & \text{is} & (|a,b| + |c,d|) = 0 \\ p \vee q & \text{is} & (|a,b| \cdot |c,d|) = 0 \\ p \to q & \text{is} & \sim p \vee q \\ p \leftrightarrow q & \text{is} & (p \to q) \,\&\, (q \to p) \end{array}$$

Goodstein defines these "quantifiers":

$A_x^n(f(x)=0)$	for all x from 0 to n	$f(x)=0$
$E_x^n(f(x)=0)$	for some x from 0 to n	$f(x)=0$
$L_x^n(f(x)=0)$	the least x from 0 to n	$f(x)=0$

We can validate *mathematical induction*:

$$[p(0)\,\&\,A_x^n(p(x)\rightarrow p(x+1))]\rightarrow p(n)$$