Advanced Progamming Languages	**Lecture 7**
CS 6110 Spring 2015	**Weds. Feb. 4, 2015**

Lecture 7

Topics

1. Review structural induction for λ-terms.
2. What is the right equality on λ-terms? Comments on Dr. Rahli's lecture 6.
3. Combinators
   - Another programming language (universal)
   - Dictionary of combinators
4. Equations involving combinators
5. Translating λ-terms into combinators.

---

1. **Structural induction for inductive types.**

   To prove assertions of the $\forall x : \lambda term.P(x)$ we can use the induction on the structure of λ-terms given by their inductive definition. Recall

   $$\lambda\text{-term} = Variable | \, Abstraction | \, Application$$

   We took the variables, Var, to be $v_0, v_1, v_2...$ The abstractions are $\lambda(v.t)$ where v is a Var and t a λ-term. The applications are $ap(t_1; t_2)$ where t_i are terms. This is an informal "type definition" in the informal language of the lectures and course notes. We need the definition of the computation system before we can formally define *types*.

   From the informal definition, we can see the *inductive structure*.

   $$t = v \mid \lambda(v.t) \mid ap(t_1; t_2)$$
   $$t = v \mid \lambda(v.t) \mid ap(t; t)$$

   It's okay to use in this format.

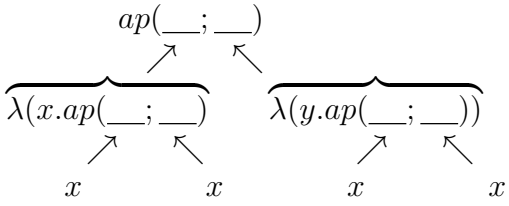
Aside: The Coq definition of λ-terms is also given like this.

$$
\begin{aligned}
&\mathit{Inductive}\ \ tm : \mathit{Type}\ :=\\
&\quad \mid tvar : id \to tm\\
&\quad \mid tapp : tm \to tm \to tm\\
&\quad \mid taps : id \to tm \to tm
\end{aligned}
$$

So we form elements such as $tvac(x_0),\ tapp(tvar(x_0))(tvar(x_0))$

$taps\ x_0\ (tvar\ x_0)$ is $\lambda(x_0.x_0)$

We can see a tree like structure:



In Nuprl's Constructive Type Theory (CTT), we use a *co-inductive type* and require a finite bound, that is the tree is potentially unbounded.

2. **Comments on Dr. Rahli's Lecture 6**

We have seen Barendregt's notions of equality on λ-terms. There are already these equalities:

$t_1 \equiv t_2$ Syntactic identity

$t_1 =_\alpha t_2$ α-equality

$t_1 =_\beta t_2$ α and β equality

But what is the fundamental mathematical equality? Doug Howe (Cornell PhD, 1988) found it! We and others call it Howe's Squiggle Equality, $t_1 \sim t_2$.

Either both terms diverge or both have the same outer operator and their subterms are $\sim$ equal. This is close to *Kleene equality* on practical computable functions $f \simeq g$, $f(x) \simeq g(x)$ for all $x \in \mathbb{N}$.

3. **Combinators**

One remarkable fact about the λ-calculus is that it is expressive enough to define the natural numbers and all partial recursive (Turing computable) functions on them. One uses the *Church numerals* $c_n = \lambda f x.f^n(x)$.

What is even more remarkable is that there is a much simpler and related computational basis, combinators. The idea has its origins in the 1924 work of Schönfinkel who thought that expressing logic using variables was a poor (misguided) idea. He said that the logical fact that $\sim p \Rightarrow (p \Rightarrow q)$ is not about p or q or anything denoted by those letters! Variables are a linguistic construct rather than a logical one.

According to Schönfinkel $\sim p \Rightarrow (p \Rightarrow q)$ is about the relationship between the logical operators (functions) $\sim$ and $\Rightarrow$ (implication).

This idea led to the definition of functions without using bound variables. He developed a function calculus in terms of operators we call *combinators*.

Definition The combinatorial terms, CL-terms, are inductively defined as:

(*i.*) Variables $x_0, x_1, x_2, ...$

(*ii.*) Constants **S, K**, and

(*iii.*) If P and Q are CL-terms then so is (PQ), their application.

Here are the key equations in the CL-theory:

1. $P = P$
2. $(P = Q) \Rightarrow (Q = P)$
3. $P = Q, Q = R \Rightarrow P = R$
4. $P \Rightarrow Q \Rightarrow PR = QR$
5. $P = Q \Rightarrow RP = RQ$
6. $\mathbf{K}PQ = P$
7. $\mathbf{S}PQR = PR(QR)$

Equation 7 is written in Barendregt's cute way. Why is this cute? Does it give a mnemonic for the rule?

There is a simple translation of these combinators to λ-terms. We have seen them before.

$$\mathbf{K}^\circ = \lambda(x.\lambda(y.x)) \qquad \mathbf{S}^\circ = \lambda(x.\lambda(y.\lambda(z.xz(yz))))$$

A combinator is *closed* iff there are no variables in it, as in these examples.

Dictionary of Combinators

$\mathbf{I} \equiv \mathbf{SKK}$	$\mathbf{I}^\circ = \lambda(x.x)$
$\mathbf{B} \equiv \mathbf{S(KS)K}$	$\mathbf{B}^\circ = \lambda(x, y, z.x(yz))$
$\mathbf{W} \equiv \mathbf{SS(KI)}$	$\mathbf{W}^\circ = \lambda(x, y.xyy)$
$\mathbf{C} \equiv \mathbf{S(BBS)(KK)}$	$\mathbf{Y}^\circ = \lambda(f.\lambda(x.f(xx))\lambda(x.f(xx))$
$\mathbf{Y} \equiv \mathbf{WS(BWB)}$	$\mathbf{\Omega} = \lambda(x.xx)\lambda(x.xx)$

Can this be even simpler, a single basis element? Yes, as we discuss later.
What is the astonishing fact about all this?

Translating λ-terms to combinators (Section 5, p.47):

(i) $[x].X \equiv \mathbf{I}$ if $X \equiv X$

(ii) $[x].X \equiv \mathbf{K}X$ if X is an atom distinct from x, e.g. $[x].Y \equiv \mathbf{K}Y$

(iii) $[x].X \equiv \mathbf{S}([x].Y)([x].Z)$ if $X = YZ$

If $n > 1$ then

(iv) $[x_1, ..., x_n].X \equiv [x_1].([x_2, ...x_n].X)$

Notice that $[x].X$ *does not contain the variable* x. The next step is this:

Let a, b be λ-terms

1. $\bar{a} \equiv a$ if a is a variable or a primitive constant
2. $(\overline{ab}) \equiv \bar{a}\bar{b}$
3. $\overline{\lambda(x.a)} \equiv [x].\bar{a}$

The translation from combinators to λ-terms is easy. See Barendregt p.159 and index of symbols p.605.

$\mathbf{S}^\circ = \lambda x, y, z.xz(yz)$

$\mathbf{K}^\circ = \lambda x, y.x$

$\mathbf{B}^\circ = \lambda xyz.x(yz)$

$\mathbf{W}^\circ = \lambda xy.xyy$

$\mathbf{C}^\circ = \lambda xyz.xzy$

$\mathbf{y}^\circ = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$

$\mathbf{\Phi}^\circ = \lambda xyzw.x(yz)(yw)$

$\mathbf{\Omega}^\circ = (\lambda x.xx)(\lambda x.xx)$

Basis of all life is 20 amino acids and the laws of chemistry.
Basis of all computation can be two combinators and the rules of reduction.
We'll see in the next lecture that the basis can be even smaller!