Lecture 20

Lectured by Prof. Caldwell and scribed by Sunil Kothari

December 8, 2008

1 Review

```
Main> :t subst
subst :: ([Char],Term) -> Term -> Term
Main> :r
```

$$\Lambda ::= x \mid MN \mid \lambda x.M$$

It turns out that this is a turing-complete but we want to add pairs as a means of constructing lambda terms so

$$\Lambda ::= x \mid MN \mid \lambda x.M \mid < M, N > \mid Spread(M; x, y.N)$$

$< \lambda x.x, y >$ is a pair in our lambda calculus
So what is *spread* ?
In *spread* $(M; x, y.N)$, x and y are bound variables.

Recall that $(\lambda x.M)N \rightsquigarrow_\beta M[x := N]$. For example, $(\lambda x.x)N \rightsquigarrow_\beta x[x := N] = N$. The term $(\lambda x.M)\ N$ is also called a *redex*. This is the computation mechanism in lambda calculus.
So, *beta* is given as:

```
beta (Ap (Abs x m) n) = subst (x,n) m
beta t = t
```

```
beta (V "x")
x::Term

beta (Ap (Abs "x"  (V "x")) (V "N")
N::Term
```

Definition 1 (Fixpoint). *x is a fixpoint for f if $f\ x = x$.*

Spread is actually a destructor for pairs. The computation over spread terms is defined by the *spread_rule*.
$spread(< M, N >; x, y.M') \rightsquigarrow M'[x := M, y := N]$ and
$spread(M; x, y.N) \rightsquigarrow spread(M; x, y.N)$ if M is not a pair.

We can define more primitive destructor for pairs in terms of spreads
$fst\ p = spread\ (p; x, y.x)$
$snd\ p = spread\ (p; x, y.y)$

$$\begin{array}{rcl} fst\ < M, N > & = & spread\ (< M, N >; x, y.x) \\ & = & (x[x := M, y := N]) \\ & = & M[y := N] \\ & = & M \end{array}$$

We can do swap easily with spread.

$$swap\ p = spread\ (p, x, y. < y, x >)$$

NOTE: For some reason I was unable record all that was mentioned in the lecture. You should also look at HW 16 description for more material related to this lecture.