Lambda Calculus Cheat Sheet – continued

CS 68

April 3, 2011

1 Booleans

$$\begin{aligned} \texttt{true} &= \lambda u.\ \lambda v.\ u. \\ \texttt{false} &= \lambda u.\ \lambda v.\ v. \\ \texttt{cond} &= \lambda u.\ \lambda v.\ \lambda w.\ u\ v\ w. \end{aligned}$$

2 Data structures

$$\begin{aligned} \texttt{Pair} &= \lambda m.\ \lambda n.\ \lambda b.\ \texttt{cond}\ b\ m\ n. \\ \texttt{fst} &= \lambda p.\ p\ \texttt{true} \\ \texttt{snd} &= \lambda p.\ p\ \texttt{false} \end{aligned}$$

3 Natural numbers and arithmetic

$$\begin{aligned} \texttt{0} &= \lambda\ s.\ \lambda\ z.\ z. \\ \texttt{1} &= \lambda\ s.\ \lambda\ z.\ s\ z. \\ \texttt{2} &= \lambda\ s.\ \lambda\ z.\ s\ (s\ z). \\ &\ldots \\ \texttt{n} &= \lambda\ s.\ \lambda\ z.\ s\ (s\ (s\ (\ \ldots (s\ z)\ \ldots))), \end{aligned}$$

where the right hand side of the last line includes n occurrences of s.

Notice that $\texttt{n}\ f\ x = f^n(x)$

The successor function adds an extra application of successor to a number.

$$\texttt{Succ} = \lambda\ n.\ \lambda\ s.\ \lambda\ z.\ s\ (n\ s\ z).$$

Thus $\texttt{Succ n} = \texttt{n+1}$. With successor, it is easy to define addition, multiplication, and a test for zero:

$$\begin{aligned} \texttt{Plus} &= \lambda\ n.\ \lambda\ m.\ \lambda\ s.\ \lambda\ z.\ m\ s\ (n\ s\ z). \\ \texttt{Mult} &= \lambda\ n.\ \lambda\ m.\ m\ (\texttt{Plus}\ n)\ 0. \\ \texttt{isZero} &= \lambda\ n.\ n\ (\lambda\ x.\ \texttt{false})\ \texttt{true}. \end{aligned}$$

Thus

$$\begin{aligned} \texttt{Plus n m} &= \lambda\ s.\ \lambda\ z.\ \texttt{m}\ s\ (\texttt{n}\ s\ z) \\ &= \lambda\ s.\ \lambda\ z.\ s^n(s^m(z)) \\ &= \lambda\ s.\ \lambda\ z.\ s^{n+m}(z) \\ &= \texttt{n+m}. \end{aligned}$$

while

```
Mult n m  =  m (Plus n) 0
          =  (Plus n)^m 0
          =  n*m.
```

Also, `isZero 0` = `true` because the constant function $\lambda\ x.$ `false` is never applied, while `isZero n` = `false` when $n > 0$ because the constant function will be applied at least once.

Finding the predecessor of a number is tricky. Start by providing a new encoding of numbers as pairs:

`PZero` = `<0,0>` = `Pair 0 0`

`PSucc` = $\lambda\ n.$ *`Pair (snd n) (Succ(snd n))`*

Therefore n is encoded as `Pair (n-1) n`.

Now define the predecessor function:

`Pred` = $\lambda n.$ `fst` (n `PSucc PZero`).

The definition of subtraction is easily obtained.

4 Recursion

Define

`Y` = $\lambda f.\ (\lambda x.\ f(xx))(\lambda x.\ f(xx))$

`Y` is called the fixed-point combinator because for all g, we get `Y` g = g(`Y` g). Thus `Y` g is a *fixed point* of g for any function g.

Recall the definition of factorial:

fact = $\lambda\ n.$ `cond` (`isZero` n) `1` (`Mult` n (*fact* (`Pred` n)))

Define F by

`F` = $\lambda\ f.\ \lambda\ n.$ `cond` (`isZero` n) `1` (`Mult` n (f (`Pred` n)))

then `F`(*fact*) = *fact*. Because *fact* is a fixed point of `F`, define

`fact` = `Y F`.

Here are some values of `fact`.

```
fact 0  =  (F (fact)) 0      because fact is a fixed point of F
        =  cond (isZero 0) 1 (Mult 0 (fact (Pred 0)))      expanding F
        =  1     by the definition of cond


fact 1  =  (F (fact)) 1     because fact is a fixed point of F
        =  cond (isZero 1) 1 (Mult 1 (fact (Pred 1)))     expanding F
        =  Mult 1 (fact (Pred 1))     by the definition of cond
        =  fact 0     by the definition of Mult and Pred
        =  1     by the above calculation


fact 2  =  (F (fact)) 2     because fact is a fixed point of F
        =  cond (isZero 2) 1 (Mult 2 (fact (Pred 2)))     expanding F
        =  Mult 2 (fact (Pred 2))     by the definition of cond
        =  Mult 2 (fact 1)     by the definition of Pred
        =  Mult 2 1     by the above calculation
        =  2     by the definition of Mult
```