Lists in the λ-calculus

Simon Winwood

July 30, 2010

We will represent list objects as functions taking an argument for the *cons* case, and one for the *nil* case, so lists will, in general, have the form

$$\lambda f\ n.\ \ldots$$

Thus, in the *nil* case we can just return the second argument.

$$nil = \lambda f\ n.n$$

In the *cons* case, we need to pass the list elements to the first function, along with something for the tail of the list

$$cons = \lambda x\ xs.\lambda f\ n.f\ x\ (xs\ f\ n)$$

Note that we need to pass f and n to xs.
To implement *map*, that is,

$$map\ f\ [x_1, \ldots, x_n] = [f\ x_1, \ldots, f\ x_n]$$

we note that, in the *nil* case, we simply want *nil* again. In the *cons* case, we want

$$map\ f\ (cons\ x\ xs) = cons\ (fx)(map\ f\ xs)$$

hence

$$map = \lambda fxs.xs\ (\lambda x\ xs'.cons\ (fx)\ xs')\ nil$$

The *foldl* function

$$foldl\ f\ i\ [x_1, \ldots, x_n] = f\ x_1\ (f\ x_2\ (f\ x_3\ (\ldots (f\ x_n\ i)))\ldots)$$

is rather simpler, and is left as an exercise.