

CSci 360, Fall 2004, Assignment 9

This assignment, worth 40 points, is due Wednesday, November 10, at the beginning of class. You should submit a paper copy of your lambda expressions to my office or at the beginning of class.

For this assignment, I recommend continuing to use *Lambda Calculator*, which (as you may recall) you can start by typing the following at the command line after you have the file “math.lmb.”

```
unix% lambda-calc math.lmb &
```

In this assignment, we will work with the following technique for representing lists.

$$\begin{aligned}() &\equiv \lambda c.\lambda n.n \\(1) &\equiv \lambda c.\lambda n.c\ 1\ n \\(2\ 1) &\equiv \lambda c.\lambda n.c\ 2\ (c\ 1\ n)\end{aligned}$$

Note the similarity of this definition to the definition of integers: I happen to have used c rather than s for the first parameter, and n rather than z for the second parameter, but that’s not a real difference. (I’m thinking of c as standing for *cons* and n as standing for *null* — whereas with the Church numerals I’m thinking of s as standing for *successor* and z as standing for *zero*.) The only real distinction is that the c function takes two parameters rather than just one — with new parameter being the first, which is an element of the list.

We can define *null* to refer to the empty list (i.e., $\lambda c.\lambda n.n$). Notice how *null* and 0 (i.e., $\lambda s.\lambda z.z$) are exactly the same: They both take two parameters and return the second argument. This is also the same as *false*. In fact, in classical Lisp implementations all these three values are the same. Scheme, however, in an attempt to have a stronger notion of type, departs from this by treating them as different.

Note: In the following functions, your definitions may assume that types are correct: That is, if the function expects a list as a parameter, then you need not worry about what your definition does if the argument is not in fact a list.

1. Define the *cons* function, which takes an element and a list and returns a new list whose elements begin with the parameter element and are followed with the elements of the parameter list.

$$\mathit{cons}\ 1\ \mathit{null} \Rightarrow \lambda c.\lambda n.c\ 1\ n$$

2. Define the *car* function, which returns the first element of its parameter list.

The *cdr* function is considerably more difficult. The easiest way is to base it on the *pred* (or *dec*) function that returns a Church numeral that is one less than its argument — which is rather difficult to understand. In case you’re interested, here’s the definition, but you should not use anything like it for this assignment.

$$\mathit{cdr} \equiv \lambda \ell.\lambda c.\lambda n.\ell\ (\lambda e.\lambda f.\lambda g.g\ e\ (f\ c))\ (\lambda g.n)\ (\lambda x..\lambda y.y)$$

3. Define the *null?* function, which returns *true* if the parameter list is empty and *false* otherwise.
4. Define the *length* function, which returns the number of elements in the parameter list. Naturally, you’ll make use of the arithmetic functions defined in class and in the *math.lmb* file.

For this, and for the preceding functions, you may *not* use the Y combinator defined in class (which we used in defining the factorial function) in these definitions.