F.G.H.
6/11/85

LispCourse #35: Solutions to Homework #34

Apologies

Homework #34 was poorly planned out. In doing the homework myself, I ran into a number of conceptual rough spots that I did not point out in the problem specifications.

Some of these problems are:

1) I forgot to account for numbers (and arrays, etc.) that evaluate to themselves. In LC.Eval, LITATOMs should have theior value looked up on the stack, LISTPs should be evaluated as per the rules, and everything else (including numbers) should evaluate to themselves. Its this last clause I left out of my description.

2) I did not account for NLAMBDA functions. In the LC.Eval function, NLAMBDA functions have to be handled specially, since their arguments should not be evaluated before LC.Apply is called. On page 8 of the solution printout is a magic function LC.NlambdaP that determines if its argument is the name of an NLAMBDA function.

3) As part of the LC.Apply function, I specified that if the function definition was not a list, then you should use the standard Interlisp APPLY procedure instead of your own. This is true, except it doesn't quite work for functions that use free variables.

   What happens is this: For functions with LISTP definitions, you bind variables on your own stack. For non-LISTP definitions, you use Interlisp which has its own stack. When Interlisp wants to look up a free variable, it looks on its own stack. But the variable might have last been bound on your stack. Therefore, Interlisp will either not find the binding or find an old binding.

   Solution: Before you call the Interlisp APPLY, you have to bind on the interlisp stack all of the variables that are bound on your stack. One way to do this is to construct a LET or PROG statement with the necessary bindings and containing the appropriate APPLY statement. You can then Interlisp EVAL this LET/PROG statement. This solution is captured in the function LC.LispApply on page 8 of the solution printout. In my LC.Apply, I called this function rather than calling APPLY directly to get around this problem.

4) Things get dull after you call LC.LispApply, since thereafter Lisp is doing all the EVALs and APPLYs. Therefore, you want to call standard Interlisp function as late as possible so that you can watch functions with LISTP definitions (i.e., one you have defined) being evaluated.

   LC.CountAtoms has a COND at its top level. Therefore, most of the work in evaluating LC.CountAtoms ends up being done by Lisp and not by your LC.Eval and LC.Apply. I have one function, LC.CountAtoms1, that works this way. But I also wrote a second function, LC.CountAtoms2, that uses my own version of COND called LC.Cond. When I LC.Eval this second CountAtoms much more of the work is done by my evaluator because I bomb into Lisp much later in the evaluation process.

Solutions

Attached.

Sample Runs

Attached.