Machine Problem Set 6

Assigned: April 24, 2006
Due: May 1, 2006

Objectives and Background

This MP will give you some practice with the use of continuations.

Problem 1  CPS Transform (20 points)

Write a function, called cps-trans, performs lexical transforms on functions (lambda expressions). Functions will be represented as lists of lists and symbols.

You should also handle all the expressions mentioned on p. 305 of the book, except let and letrec expressions.

proc expressions will be lambda expressions.

The primitive operations are +, -, *, and eq?.

The only conditional expression you will have to deal with is an if. You may assume that the then and else expressions will always occur.

If you need fresh variables, you may safely assume that no variables starting with 'k' will exist in the input.

\[
\begin{align*}
> \ (\text{cps-trans}) \ (\text{lambda} \ \text{(x)} \ (\text{if} \ (\text{eq?} \ x \ 0) \ 1 \ (* \ x \ (\text{fact} \ (- \ x \ 1)))))) \\
\ \ (\text{lambda} \ \text{(x k)} \ (\text{if} \ (\text{eq?} \ x \ 0) \ (\text{k} \ 1) \ (\text{fact} \ (- \ x \ 1) \ (\text{lambda} \ \text{(kr)} \ (\text{k} \ (* \ x \ kr)))))) \\
> \ (\text{define} \ \text{fact} \ (\text{eval} \ (\text{cps-trans}) \ (\text{lambda} \ \text{(x)} \ (\text{if} \ (\text{eq?} \ x \ 0) \ 1 \ (* \ x \ (\text{fact} \ (- \ x \ 1))))))) \\
> \ (\text{fact} \ 5 \ (\text{lambda} \ \text{(x)} \ x)) \\
120
\end{align*}
\]

Note: As the example above demonstrates, you can test the correctness of your transforms by evaluating the returned expression and calling the function with an identity function as the initial continuation.

You may wish to read section 8.4 in the book before starting this. You should not have to parse Scheme code, since it’s Scheme code - you’re writing abstract syntax trees. However, in many cases, the details of what to do will be similar.
Problem 2  Tree Search (20 points)

We will represent trees n-ary trees containing key/item pairs as a list. The first element of
the list will be a cons cell whose car is the key, and whose cdr is the item. The remaining
elements of the list will be the children of the node. Implement a search function that takes
a key and a tree, and returns a pair consisting of the first item found, and a zero argument
function that when called will continue the search. If no items are found, the function should
return nothing.

Use call-with-current-continuation to implement this function.

> (search 'q '(((q . a) ((a . b) ((q . b)))))
(a . <cont1>)
> (<cont1>)
(b . <cont2>)
> (<cont2>)
<no return value>

Hint: You will probably want to use continuations to return your answers. Normally, this
probably would not be necessary, but since we’re asking you to backtrack, this will probably
turn out to be necessary. Since the continuation will return whatever value you give it to
the original call-with-current-continuation call, you will have to generate a new continuation
anytime the zero argument function that continues the search is called. It may be useful to
store the return continuation as a global variable - it’s not strictly necessary, but may be
conceptually easier.

Problem 3  Co-routines (20 points)

This problem is only required for graduate students and may be considered extra credit,
otherwise.

Implement co-routines by providing two functions start and yield. start will take a list
of zero argument procedures, and run them as co-routines. Start with the first procedure in
the list and continue down the list each time control is yielded by a procedure. Loop back
to the beginning of the list when the end is reached. yield will be called by the procedures
and will yield control to the next procedure to be run.

> (define p1 (lambda () (display 'a) (yield) (p1)))
> (define p2 (lambda () (display 'b) (yield) (p2)))
> (start (list p1 p2))
ababababababababab...
Handin

You should hand in a single file named ‘mp6.scm’ with the implementations of the above functions. The names of the functions and number and order of arguments of the functions should be the same as in the problems.

Please see the CS 421 FAQ web page for handin instructions.