Another Use for CPS

A function is a tail-call in an expression if the value returned by the function is the value returned by the entire expression. So:

- $f$ is not a tail-call in $(+ (f 4) 3)$
- $f$ is a tail-call in $(f (+ 4 3))$
- $g$ is not a tail-call in $(f (g 6))$ but $f$ is.

Tail-calls use stack space more efficiently because they are linear. Non-tail-calls result in tree-like behavior. Remember the recursive versions of factorial and fibonacci?

Converting Non-Tail calls to Tail-Calls

Let’s start with this function definition and use:

\[
\text{(define } (f x) (+ x 3))
\]
\[
(* (f 5) 2)
\]

To convert the non-tail-call, we’d need to move it out of the multiplication, leaving a hole: $(* \text{ hole } 2)$. Let’s abstract this to:

\[
\text{(lambda } (\text{hole}) (* \text{ hole } 2))
\]

Converting Non-Tail calls to Tail-Calls

Now we need to transform the function $f$ in the following way: the multiplication function we abstracted represents the rest of the computation. We need to pass this to the transformed $f$ as the receiver.

\[
\text{(define } (f/k x k) (k (+ x 3))) ;; where k is the receiver}
\]
\[
(f/k 5 \text{(lambda } (\text{hole}) (* \text{ hole } 2)))
\]

The transformed function $f/k$ is now in tail-call position. Confirm that this expression returns the same answer as before.
The Method Summarized, Part 1

The receiver $k$ in the example represents an implicit use of a continuation.

To convert a non-tail call to function $f$ in expression $E$ do this:

- Copy the original call to $f$ to the top-level (outside all other expressions)
- Replace the entire call to $f$ in $E$ with a new variable (like hole).
- Enclose the revised $E$ in a lambda expression with the new variable as its parameter

The Method Summarized, Part 2

- Create a new version of $f$ (such as $f/k$) that takes a function (of one argument) as its additional argument and calls that function on the original body of $f$
- Modify the copied call to $f$ to now call $f/k$ passing the lambda expression as the last argument.
- Not done yet! Make sure that all calls in the body of $f/k$ are tail-calls, converting those that aren’t recursively

Try this Yourself

```scheme
(define (g x) (* x 6))
(define (h y) (+ y 5))
(+ 4 (g (h 3)))
```

- Convert to h/k and g first, then h/k and g/k

As mentioned before, we can CPS any function. Try these:

```scheme
(define (sum-area w1 h1 w2 h2)
  (+ (rect-area w1 h1)
      (rect-area w2 h2)))
(sum-area 2 4 6 8)
```

CPS in the presence of Cond

To CPS-convert a `cond` statement:

- If the questions are all tail-calls, send the answer of each to the continuation $k$ then check whether the new answer expressions have any tail calls.
- If a question has a non-tail-call, put the call outside the `cond` and replace the question with the "hole" (lambda argument). Then continue as before.