Buffalo’s Guide to Simple Continuation Passing Style

Preparation –

The goal of CPS is to convert some recursive procedures to non-recursive ones. To start you’ll need:

1. The code of the procedures you intend to change – sometimes it’s just one, other times its one plus some helpers
2. A newly created datatype we will call “continuation”. This datatype will initially have only one flavor – the “all-done” flag which is by tradition called init-k.
3. A newly created procedure we will call (by tradition) apply-k. It takes 2 parameters k – the continuation type, and v – a value. If given init-k, thus function should return v.

Then we will convert the functions one by one. If you can, start with the innermost utility functions so you can test them individually.

For each function…

Add a new parameter k to its parameter list. Then look at all the places where the function returns a result. There are 3 possibilities:

# Option 1: The result is calculated directly

By directly I mean “without any other CPS function being involved”. Could look like this: (cons p1 p2). But cannot look like this (cons p1 (my-cool-cps-function p2)).

In this case know what the result of the function should be. But there might be other work that still needs doing stored in the continuation. To do it, call apply-k with the value that would normally be returned e.g.

(cons p1 p2) -> (apply-k k (cons p1 p2))

# Option 2: The result is a CPS procedure call in tail position

In tail position means “I call another CPS procedure and that is the overall result with no additional work required after”. Could look like this: (my-cool-cps-function (cons p1 p2)). But cannot look like this (cons p1 (my-cool-cps-function p2)).

In this case, we can rely on that other CPS function to deal with continuation when the final result is calculated. So we simply pass k along and return that.

(my-cool-cps-function (cons p1 p2)) -> (my-cool-cps-function (cons p1 p2) k)

# Option 3: The result has a CPS procedure call in non-tail position…or several

It could look like this (cons p1 (my-cool-cps-function p2)). If there is several, identify the chronically first call. This is the call that will be in tail position when the conversion is done.

3a. Add a new flavor to the continuation type. This data-wise, this will take as parameters any data you need to continue work after your get the result of that first call. That will always include k, but maybe other stuff.

(define-datatype continuation continuation?

...

[stepX (p1 number?) (k continuation?)]

...

3b. Add a new entry in the apply-k case when the newly created flavor is the current k. Modify the code so that v used as a replacement for the result of the tail call. All other data should be retrieved from the continuation.

(define apply-k

(lambda (k v)

(cases continuation k

...

[stepX (p1 k)

(cons p1 v))]

3c. Modify the original function code to call the original function in tail position, and pass your new flavor as the k

(my-cool-cps-function p2 (stepX p1 k))

3d. If there was more than one non-tail cps function call, apply-k will have the remaining non-tail code. Apply steps 3a-d again to eliminate transform it into tail for and another new continuation flavor.

An example would be like this (cons (my-cool-cps-function p1) (my-cool-cps-function p2))