Continuations

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Objectives

You should be able to ...

It is possible to use functions to represent the *control flow* of a program. This technique is called *continuation passing style*. After today's lecture, you should be able to

- Explain what CPS is.
- Give an example of a programming technique using CPS.
- Write a recursive function using CPS.

Direct Style

Example Code

```
inc x = x + 1
double x = x * 2
half x = x `div` 2

result = inc (double (half 10))
```

- Consider the function call above. What is happening?

The Continuation

```
result = inc (double (half 10))
```

- We can ‘punch out’ a subexpression to create an expression with a ‘hole’ in it. `result = inc (double ⟦⟧)`
- This is called a *context*. After `half 10` runs, its result will be put into this context.
- We can call this context a *continuation*.

Making Continuations Explicit

- We can make continuations explicit in our code.

```haskell
cont = \ v -> inc (double v)
```

- Instead of returning, a function can take a *continuation argument*.

Using a Continuation

```haskell
half x k = k (x `div` 2)
result = half 10 cont
```

- Convince yourself that this does the same thing as the original code.

Properties of CPS

- A function is in *Direct Style* when it returns its result back to the caller.
- A *Tail Call* occurs when a function returns the result of another function call without processing it first.
  - This is what is used in accumulator recursion.
- A function is in *Continuation Passing Style* when it passes its result to another function.
  - Instead of returning the result to the caller, we pass it forward to another function.
  - Functions in CPS "never return."
- Let's see some more examples.

Comparisons

Direct Style

```
inc x = x + 1
double x = x * 2
half x = x `div` 2

result = inc (double (half 10))
```

CPS

```
inc x k = k (x + 1)
double x k = k (x * 2)
half x k = k (x `div` 2)
id x = x
result = half 10 (\v1 ->
         double v1 (\v2 ->
         inc v2 id))
```

CPS and Imperative Style

- CPS look like imperative style if you do it right.

CPS

```
result = half 10 (\v1 ->
         double v1 (\v2 ->
         inc v2 id))
```

Imperative Style

```
    v1 := half 10
    v2 := double v1
result := inc v2
```

The GCD Program

```haskell
gcd a b | b == 0 = a
        | a < b  = gcd b a
        | otherwise = gcd b (a `mod` b)
```

gcd 44 12 ⇒ gcd 12 8 ⇒ gcd 8 4 ⇒ gcd 4 0 ⇒ 4

- The running time of this function is roughly $\mathcal{O}(lg\ a)$.

GCD of a List

```
gcdstar [] = 0
gcdstar (x:xs) = gcd x (gcdstar xx)

> gcdstar [44, 12, 80, 6]
2
> gcdstar [44, 12]
4
```

- Question: What will happen if there is a 1 near the beginning of the sequence?
- We can use a continuation to handle this case.

Continuation Solution

```
gcdstar xx k = aux xx k
  where aux [] newk = newk 0
        aux (1:xs) newk = k 1
        aux (x:xs) newk = aux xs (\res -> newk (gcd x res))

> gcdstar [44, 12, 80, 6] report
2
> gcdstar [44, 12, 1, 80, 6] report
1
```

Other Topics

- Continuations can simulate exceptions.
- They can also simulate cooperative multitasking.
  - These are called co-routintes.
- Some advanced routines are also available: call/cc, shift, reset.