CPS Activity

Mattox Beckman

Conversion

$$C[\![f\ arg\ =\ e)]\!] \Rightarrow f\ arg\ k\ =\ C[\![e]\!]_k$$

$C[\![a]\!]_k \Rightarrow\ k\ a$
$C[\![f\ arg]\!]_k \Rightarrow\ f\ arg\ k$
$C[\![f\ arg]\!]_k \Rightarrow\ C[\![arg]\!]_{(\lambda v.f\ v\ k)}$, where v is fresh.
$C[\![e_1 + e_2]\!]_k \Rightarrow\ k(e_1 + e_2)$
$C[\![e_1 + e_2]\!]_k \Rightarrow\ C[\![e_1]\!]_{(\lambda v -> k(v+e2))}$where v is fresh.
$C[\![e_1 + e_2]\!]_k \Rightarrow\ C[\![e_1]\!]_{(\lambda v_1 -> C[\![e_2]\!]_{\lambda v_2 -> k(v_1+v_2)})}$where v_1 and v_2 are fresh.

Convert To

Convert the following functions to CPS:

1.
```
sumList [] = 0
sumList (x:xs) = x + sumList xs
```

2. Assume f is written in direct style.

```
map f [] = []
map f (x:xs) = f x : map f xs
```

3. Assume `f` is written in CPS and takes one continuation.

```
map f [] = []
map f (x:xs) = f x : map f xs
```

4. Convert the following code to CPS.

```
min a b = if a < b then a else b
min4 a b c d = min (min a b) (min c d)
```

More CPS Tranforms

5. Write the CPS transform for the `if` expression. You will need two cases.

Reordering Computations

6. Suppose you have a calculator which has an accumulator and a list of instructions. `Add i` adds `i` to the accumulator, and `Sub i` subtracts `i` from the accumulator.

```haskell
data Calc = Add Integer
          | Sub Integer
    deriving (Eq,Show)
```

The only problem is that our accumulator cannot be negative! Use continuations to fix this.

Here's the original calculator:

```haskell
calc xx = aux 0 xx
  where aux a [] = a
        aux a ((Add i):xs) = aux (a+i) xs
        aux a ((Sub i):xs) = aux (a-i) xs
```

Hint: you will need *two* continuations to make this work.