Learning Objectives

1. Explain the characteristics of continuations and functions written in continuation passing style (CPS).
2. Critique code that represents a failed attempt to convert a function to CPS.
3. Convert some functions to CPS using the supplied transformation rules.
4. Change the result of a computation by reordering continuations.

Characteristics

**Direct Style**

0 \(\text{fact } 0 = 1\)
1 \(\text{fact } n = n \times \text{fact } (n-1)\)
2
3 \(\text{sumList } [] = 0\)
4 \(\text{sumList } (x:xs) = x + \text{sumList } xs\)
5
6 \(\text{prodList } [] = 1\)
7 \(\text{prodList } (0:xs) = 0\)
8 \(\text{prodList } (x:xs) = x \times \text{prodList } xs\)

**CPS**

0 \(\text{fact } 0 k = k 1\)
1 \(\text{fact } n k = \text{fact } (n-1) (\lambda v \rightarrow k (n \times v))\)
2
3 \(\text{sumList } [] k = k 0\)
4 \(\text{sumList } (x:xs) k = \text{sumList } xs (\lambda v \rightarrow k (x + v))\)
5
6 \(\text{prodList } xx k = \text{aux } xx k k\)
7 \text{where} \(\text{aux } [] ks ka = ks 1\)
8 \(\text{aux } (0:xs) ks ka = ka 0\)
9 \(\text{aux } (x:xs) ks ka = \text{aux } xs (\lambda v \rightarrow ks (x \times v)) ka\)

**Problem 1**) What is the relationship between the return value of a direct-style function and the value passed into \(k\) in the CPS equivalent functions?

**Problem 2**) There are several anonymous functions in the CPS code. Their parameters are all named \(v\). What kinds of values are being passed into these parameters?
Problem 3) What percentage of the recursive calls are in tail form in the CPS functions?

Problem 4) We like to pretend that functions written in CPS `never return`. What justifies that?

Problem 5) Suppose we call direct style \texttt{prodList} with argument \([2,3,0,4,9,8,7]\). How many times will the multiplication operator be invoked? Suppose we call CPS \texttt{prodList} with the same list and continuation \texttt{print}. (The \texttt{print} function prints its argument to the screen and does not return any value\(^1\).)

Now It’s Ruined

Consider the following three programs. They are not in CPS.

\begin{align*}
\texttt{decList} & \ [ ] \ k \ = \ [ ] \\
\texttt{decList} & \ (x:xs) \ k \ = \ (x-1 : \ \texttt{decList} \ xs \ k) \\
\texttt{mink} & \ a \ b \ k \ = \ \texttt{if} \ a < b \ \texttt{then} \ k \ a \ \texttt{else} \ b \\
\texttt{minList} & \ [x] \ k \ = \ k \ x \\
\texttt{minList} & \ (x:xs) \ k \ = \ \texttt{minList} \ xs \ (\lambda v \rightarrow \ \texttt{mink} \ v \ x \ \texttt{id})
\end{align*}

Problem 6) For \texttt{decList}, \texttt{maxList}, and \texttt{minList}, explain why they are not in continuation passing style.

\(^1\)Well, it returns \texttt{unit} in the IO monad.
Conversion

Here are some of the conversion rules.

\[ C[f \text{ arg } = e] \Rightarrow f \text{ arg } k = C[e]_k \]

\[ arg \text{ is simple} \]
\[ C[a]_k \Rightarrow k a \]
\[ C[f \text{ arg}]_k \Rightarrow f \text{ arg } k \]
\[ C[f \text{ arg}]_k \Rightarrow C[arg]_k(\lambda v. f \ v \ k), \text{ where } v \text{ is fresh.} \]

\[ e_1, e_2 \text{ are simple} \]
\[ C[e_1 + e_2]_k \Rightarrow k(e_1 + e_2) \]
\[ C[e_1 + e_2]_k \Rightarrow C[e_1]_k(\lambda v_1 \rightarrow k(v_1 + e_2)) \text{ where } v \text{ is fresh.} \]
\[ C[e_1 + e_2]_k \Rightarrow C[e_1]_k(\lambda v_1 \rightarrow C[e_2]_k(\lambda v_2 \rightarrow k(v_1 + v_2))) \text{ where } v_1 \text{ and } v_2 \text{ are fresh.} \]

**Problem 7)** The phrase `where v is fresh` appears a lot here. Why do we need to be concerned about that?

**Problem 8)** Suppose we want to convert these functions into CPS. There are helper functions \( f, g, \) and \( h. \) During the conversion process, we are also going to convert \( f \) and \( g \) to CPS, but leave \( h \) in direct style.

\[ \text{foo a b } = f \ a + g \ b \]
\[ \text{bar x } y = h \ x + y \]
\[ \text{baz c } = 3 + c \]
\[ \text{quux d } = h \ (g \ d) \]

Which subexpressions in the code above are simple? Can you think of way to describe what being simple means in this context?
Convert To

**Problem 9)** Convert $\text{map}$ to CPS. Assume $f$ is written in direct style.

0. $\text{map } f \ [\ ] = [\ ]$
1. $\text{map } f \ (x:xs) = f \ x : \text{map } f \ xs$

**Problem 10)** Do it again, but this time assume $f$ is written in CPS and takes one continuation.

0. $\text{map } f \ [\ ] = [\ ]$
1. $\text{map } f \ (x:xs) = f \ x : \text{map } f \ xs$

**Problem 11)** Convert the following code to CPS, preserving the order of operations that would be used if Haskell were an eager language. Note: you will need a *nested continuation* to make this work.

0. $\text{min } a \ b = \text{if } a < b \ \text{then } a \ \text{else } b$
1. $\text{min4 } a \ b \ c \ d = \text{min } (\text{min } a \ b) \ (\text{min } c \ d)$
Reordering Computations

On the off chance we have extra time, here's something to try.

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 ever 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.
CPS Activity--- Reflector's Report

<table>
<thead>
<tr>
<th>Role</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager</td>
<td>Keeps team on track</td>
</tr>
<tr>
<td>Recorder</td>
<td>Records decisions</td>
</tr>
<tr>
<td>Reporter</td>
<td>Reports to Class</td>
</tr>
<tr>
<td>Reflector</td>
<td>Assesses team performance</td>
</tr>
</tbody>
</table>

1. What was a strength of your team's performance for this activity?

2. What could you do next time to increase your team's performance?

3. What insights did you have about the activity or your team's interaction today?

CPS Activity --- Team's Assessment (SII)

Manager or Reflector: Consider the objectives of this activity and your team's experience with it, and then answer the following questions after consulting with your team.

1. What was a strength of this activity? List one aspect that helped it achieve its purpose.

2. What is one thing we could do to improve this activity to make it more effective?

3. What insights did you have about the activity, either the content or at the meta level?