

1 Curried vs. Uncurried Functions

Digression

Use of parenthesis:

1. $(A \rightarrow B) \rightarrow C$ Give me a function from A to B and I'll give you C (Curried)
2. $A \rightarrow (B \rightarrow C)$ Give me a item of type A and I will give you a function that maps from B to C
3. $A * B \rightarrow C$ Give me a tuple(A,B) and I'll give you C(Uncurried)

Code samples for curried and uncurried functions...

NOTE: x is of type A, and y is of type B

let curry f = fun x → fun y → f(x,y)

let uncurry f = fun(x,y) → (f x) y or f x y

Properties: $\text{curry}(\text{uncurry}(A \rightarrow (B \rightarrow C))) = \text{curry}(\text{identity map})$
 $\text{uncurry}(\text{curry}((A * B) \rightarrow C)) = \text{uncurry}(\text{identity map})$

2 Continuation-Passing Style

First we develop a motivation for the CPS.

In the following examples: These are the definitions of fst and snd function.

*let fst(x,y) = x: ($\alpha * \beta$) → α*

*let snd(x,y) = y: ($\alpha * \beta$) → β*

```
let f n s =  
  if n = 0 then s  
  else let x = fst s  
        y = snd s  
        s' = (x-y, x+y)  
        in f (n-1) s'
```

Same as:

```
let f = fun n → fun s →  
  if n = 0 then s  
  else let x = fst s  
        y = snd s  
        s' = (x-y, x+y)  
        in f (n-1) s'
```

Let's say that we know n early so that we make our first attempt at staging this function.

```
n = 2
```

```
let f n <s> =  
  if n = 0 then s  
  else <let x = fst ~s  
        y = snd ~s  
        s'=(x-y,x+y)  
        in ~(f (n-1) <s'>>>
```

Specialize the function with $n = 2$:

```
<fun (a,b)→~(f 2 <(a,b)>>>
```

This function has type: $\text{int} \rightarrow (\text{int} * \text{int}) \rightarrow \text{int} * \text{int}$

When this code is executed, we get the following:

```
<fun s→  
  let x = fst s  
      y = snd s  
      s'=(x-y,x+y)  
      let x = fst s'  
          y = snd s'  
          s''= (x-y,x+y)  
      in s''>
```

This code seems a bit costly. Fst and snd operations are not cheap. Creating tuples is also a costly operation, and we have 9 costly operations. We do not need all of this; optimization can be done at this time.

IDEALLY WE WOULD WANT THE CODE TO LOOK LIKE:

```
<fun (a,b)→  
  let x = a - b  
      y = a + b  
      x' = x - y  
      y' = x + y  
  in (x',y')>
```

PROPOSED SOLUTION: Change the incoming type $\text{int} \rightarrow (\text{int} * \text{int}) \rightarrow \text{int} * \text{int}$

```
let f n s =
  if n = 0 then s
  else let  x = fst ~s
           y = snd ~s
           s'=(<~x-~y>,<~x+~y>)
         in f (n-1) s'

<fun(a,b)→~(let(x,y)= f 2 (<x>,<y>) in >(~x,~y)>>>
```

PROBLEM

This function has code duplication of costly functions which is also undesirable.

TRUE SOLUTION: Use CPS (for next time...)

EXERCISE: SEE IF GENERAL TRICKS FOR HANDLING CODE EXPLOSION IN THIS EXAMPLE.