The Curry is Spicy

Eddy Varela & Taylor Beebe

November 2018

1 Curried v. Uncurried

```
let curry (f: ('a * 'b) -> 'c) : ('a -> ('b -> 'c)) = fun a -> fun b -> f(a,b)
let uncurry (f: 'a -> ('b -> 'c)) : (('a * 'b) -> 'c) = fun (a,b) -> f a b

let f : ((int * int) -> int) = fun (x:int,y:int) -> x+y
let f1 :(int->(int -> int)) = fun a b -> a - b

let test1 = uncurry(curry(f))
let test2  = curry(uncurry(f1))
```

After running this code in the F# interpreter, we find that the function definitions and verification typecheck.

Thus, we are guaranteed that the function works. But why? This is the part you want to go home, read over and have your brain melt.

For our curried definition, we take we take in a tuples of functions. Then we unwrap the functions in the tuple and execute the outer function with the tuple of newly defined functions.

For our uncurried definition, we find that we are taking in a function that takes in two functions as parameters and create a new function on the tuple. Then we execute the outer function on the two functions seperately.

Thus we see that the uncurried function is doing the inverse of the curried function. It was helpful to have explicit definitions of the types in the function definitions to see the return types are the input type of the inverse function.

For this reason, if we curry a function and then uncurry the result, or vice versa, we will be left with the original function.