CSE 341: Programming Languages

Spring 2005
Lecture 10 — Map, Fold, Curry

Partial application ("currying")

Recall every function in ML takes exactly one argument.

Previously, we simulated multiple arguments by using one n-tuple argument.

Another way: take one argument and return a function that takes another argument and ...

This is called "currying" after its inventor, Haskell Curry

Example:

```
val inorder3 = fn x => fn y => fn z =>
                 z >= y andalso y >= x
((inorder3 4) 5) 6
inorder3 4 5 6
val is_pos = inorder3 0 0
```


More currying idioms

Currying is particularly convenient when creating similar functions with iterators:

```
fun fold_old (f,acc,l) =
  case l of
    []     => acc
  | hd::tl => fold_old (f, f(acc,hd), tl)
fun fold_new f = fn acc => fn l =>
  case l of
    []     => acc
  | hd::tl => fold_new f (f(acc,hd)) tl
fun sum1 l = fold_old ((fn (x,y) => x+y), 0, l)
val sum2 = fold_new (fn (x,y) => x+y) 0
```

There's even convenient syntax: `fun fold_new f acc l = ...`

Currying vs. Pairs

Currying is elegant, but a bit backward: the function writer chooses which *partial application* is most convenient.

Of course, it's easy to write wrapper functions:

```
fun other_curry1 f = fn x => fn y => f y x
fun other_curry2 f x y = f y x
fun curry f x y = f (x,y)
fun uncurry f (x,y) = f x y
```