Matt Might
University of Utah
matt.might.net

Functional Programming

Today

- SpaceMonkey on Thursday!
- P3 is out; any questions?
- Functional programming
- Reduction-based interpreters

Functional programming?

Functions as values

(Higher-orderness)

Closures

Purity

Equational reasoning

And often...

Laziness

Lists

Trees

Pattern-matching

Powerful types

Iteration v. Recursion

```
function fact(n) {
  var a = 1 ;
  for (var i = n; i > 0; --i) {
    a = a * i ;
  }
  return a ;
}
```

```
function fact(n) {
   if (n == 0)
      return 1 ;
   else
      return n*fact(n-1) ;
}
```

```
function fact(n) {
  return n == 0 ? 1 : n*fact(n-1) ;
}
```

```
function fact(n) n == 0 ? 1 : n*fact(n-1) ;
```

```
function fact(a,n) n == 0 ? a : fact(a*n,n-1) ;
```

Morphisms v. Recursion

map

‘(1 2 3) => ‘(2 3 4)

```
(define (inc-list lst)
  (if (null? lst)
      '()
      (cons (+ 1 (car lst))
            (inc-list (cdr lst)))))
```

```
(define (inc-list lst)
  (map (λ (x) (+ x 1)) lst))
```

```
(define (inc-list lst)
  (if (null? lst)
      '()
      (cons (+ 1 (car lst))
            (inc-list (cdr lst)))))
```

```
(define (map f lst)
  (if (null? lst)
      '()
      (cons (f (car lst))
            (map f (cdr lst)))))
```

folds

‘(1 2 3 4) => 10

```
(define (sum lst)
  (if (null? lst)
      0
      (+ (car lst) (sum (cdr lst)))))
```

```
(define (sum lst)
  (if (null? lst)
      0
      (+ (car lst) (sum (cdr lst)))))
```

```
(define (sum lst)
  (foldr + 0 lst))
```

```
(define (sum lst)
  (if (null? lst)
      0
      (+ (car lst) (sum (cdr lst)))))
```

```
(define (foldr f z lst)
  (if (null? lst)
      z
      (f (car lst) (foldr f z (cdr lst)))))
```

```
(foldr f z '(1 2 3)) = (f 1 (f 2 (f 3 z))))
```

```
(foldr cons '() '(1 2 3)) =
```

```
(foldr cons '() '(1 2 3)) = '(1 2 3)
```

```
(foldr * 1 '(1 2 3)) =
```

```
(foldr * 1 '(1 2 3)) = 6
```

```
(foldr f z '(1 2 3)) = (f 1 (f 2 (f 3 z))))
```

```
(foldl f z '(1 2 3)) = (f (f (f 1 z) 2) 3)
```

```
(define (foldl f z lst)
  (if (null? lst)
      z
      (foldl f (f (car lst) z) (cdr lst))))
```

reduce

```
(define (sum lst)
  (foldr + 0 lst))
```

```
(define (sum lst)
  (reduce + lst))
```

filter

‘(1 10 8 7 5) => ‘(1 7 5)

```
(define (odds? list)
  (cond
    [(null? list)                '()]
    [(odd? (car list))           (cons (car list)
                                       (odds? (cdr list)))]
    [else                        (odds? (cdr list))]))
```

```
(define (odds? list)
  (filter odd? list))
```

```
(define (odds? list)
  (cond
    [(null? list)                '()]
    [(odd? (car list))           (cons (car list)
                                       (odds? (cdr list)))]
    [else                        (odds? (cdr list))]))
```

```
(define (filter matches? list)
  (cond
    [(null? list)                '()]
    [(matches? (car list))       (cons (car list)
                                       (filter matches? (cdr list)))]
    [else                        (filter matches? (cdr list))]))
```

More than lists!

```
(struct tree (left value right))
```

```
(define (tree-map f t)
  (cond
    [(null? t)  '()]
    [else       (tree (tree-map f (tree-left t))
                      (f (tree-value t))
                      (tree-map f (tree-right t))))]))
```

Match

```
(define (tree-map f t)
  (cond
    [(null? t)  '()]
    [else       (tree (tree-map f (tree-left t))
                      (f (tree-value t))
                      (tree-map f (tree-right t))))]))
```

```
(define (tree-map f t)
  (match t
    ['()                '()]
    [(tree l v r)     (tree (tree-map f l)
                            (f v)
                            (tree-map f r))]))
```

Laziness

Postpone computation until necessary

value

```
(λ () value)
```

(value)

streams

```
(define nats (stream-cons 1 (stream-map (λ (x) (+ x 1)) nats)))
```

```
(stream-take nats 5) => '(1 2 3 4 5)
```

```
(define primes (stream-filter prime? (stream-rest nats)))
```

```
(define primes (stream-filter prime? (stream-rest nats)))
```

```
(stream-take primes 7) => '(2 3 5 7 11 13 17)
```

Reduction interpreters

program => program’

Call-by-value

Call-by-name

Mini-language