Programming Abstractions

Lecture 9: Fold right

Stephen Checkoway

Lots of similarities between functions

`(sum lst)`

```
(define (sum lst)
  (cond [(empty? lst) 0]
        [else (+ (first lst)
                 (sum (rest lst)))]))
```

Lots of similarities between functions

`(length lst)`

```
(define (length lst)
  (cond [(empty? lst) 0]
        [else (+ 1
                 (length (rest lst)))]))
```

Lots of similarities between functions

```
(map proc lst)
```

```
(define (map proc lst)
  (cond [(empty? lst) empty]
        [else (cons (proc (first lst))
                    (map proc (rest lst)))]))
```

Lots of similarities between functions

(remove* x lst)

```
(define (remove* x lst)
  (cond [(empty? lst) empty]
        [(equal? x (first lst)) (remove* x (rest lst))]
        [else (cons (first lst)
                    (remove * x (rest lst)))]))
```

Let's rewrite this one to look more like the others

```
(define (remove* x lst)
  (cond [(empty? lst) empty]
        [else (if (equal? x (first lst))
                  (remove* x (rest lst))
                  (cons (first lst)
                        (remove* x (rest lst))))]))
```

Some similarities

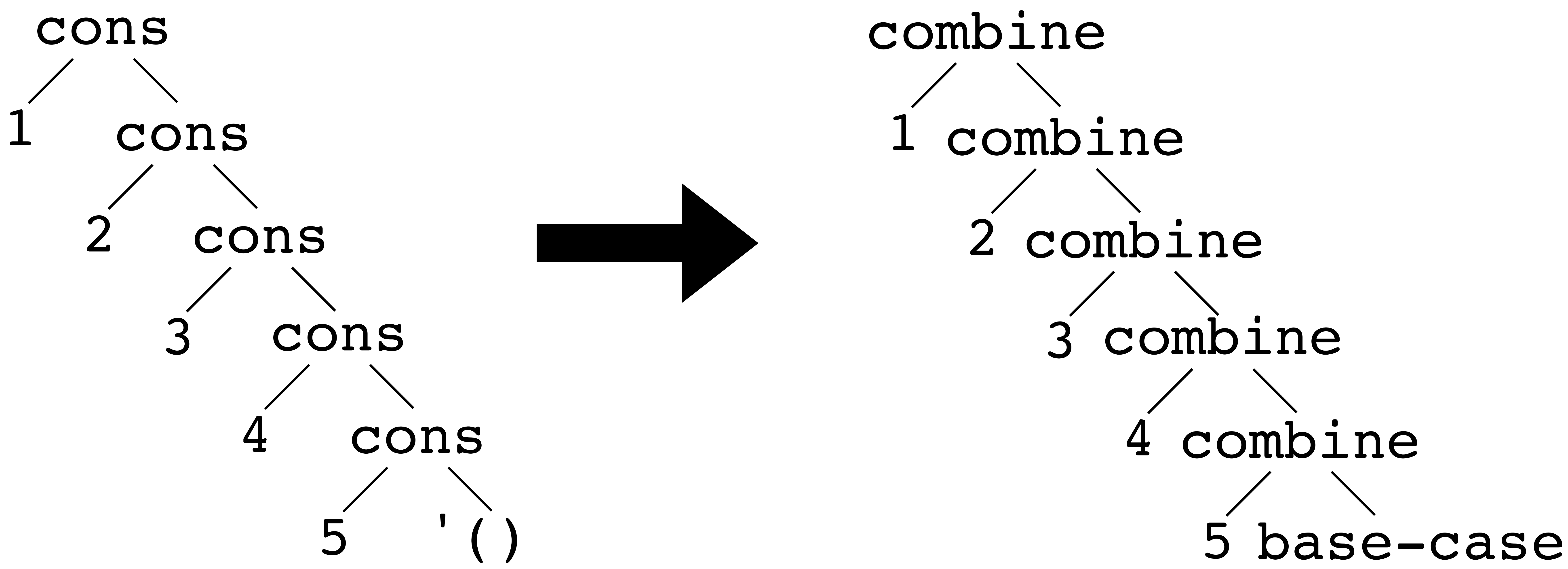
Basic structure is the same (rewriting slightly)

```
(define (fun … lst)
  (cond [(empty? lst) base-case]
        [else
         (let ([head (first lst)]
               [result (fun … (rest lst))])
           (combine head result))]))
```

Function	base-case	(combine head result)
sum	0	(+ head result)
length	0	(+ 1 result)
map	empty	(cons (proc head) result)
remove*	empty	(if (equal? x head) result (cons head result))

Abstraction: fold right

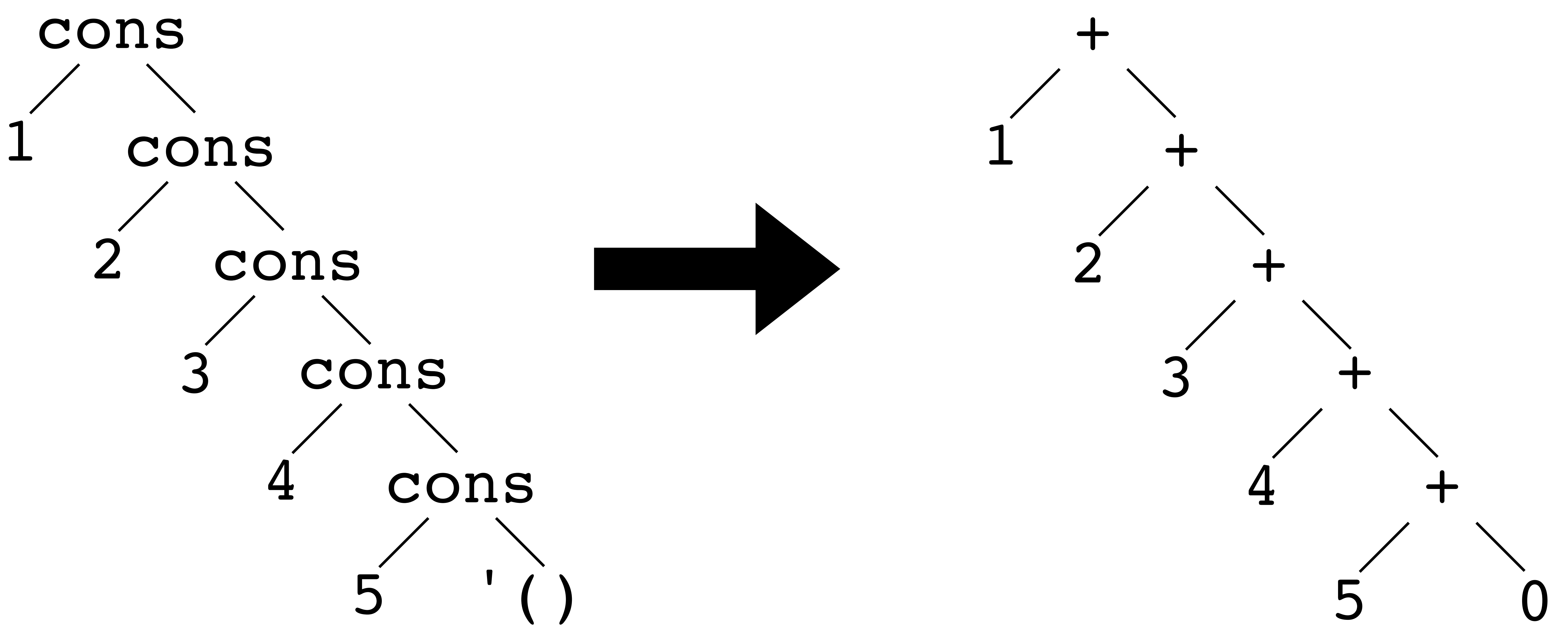
`(foldr combine base-case lst)`



sum as a fold right

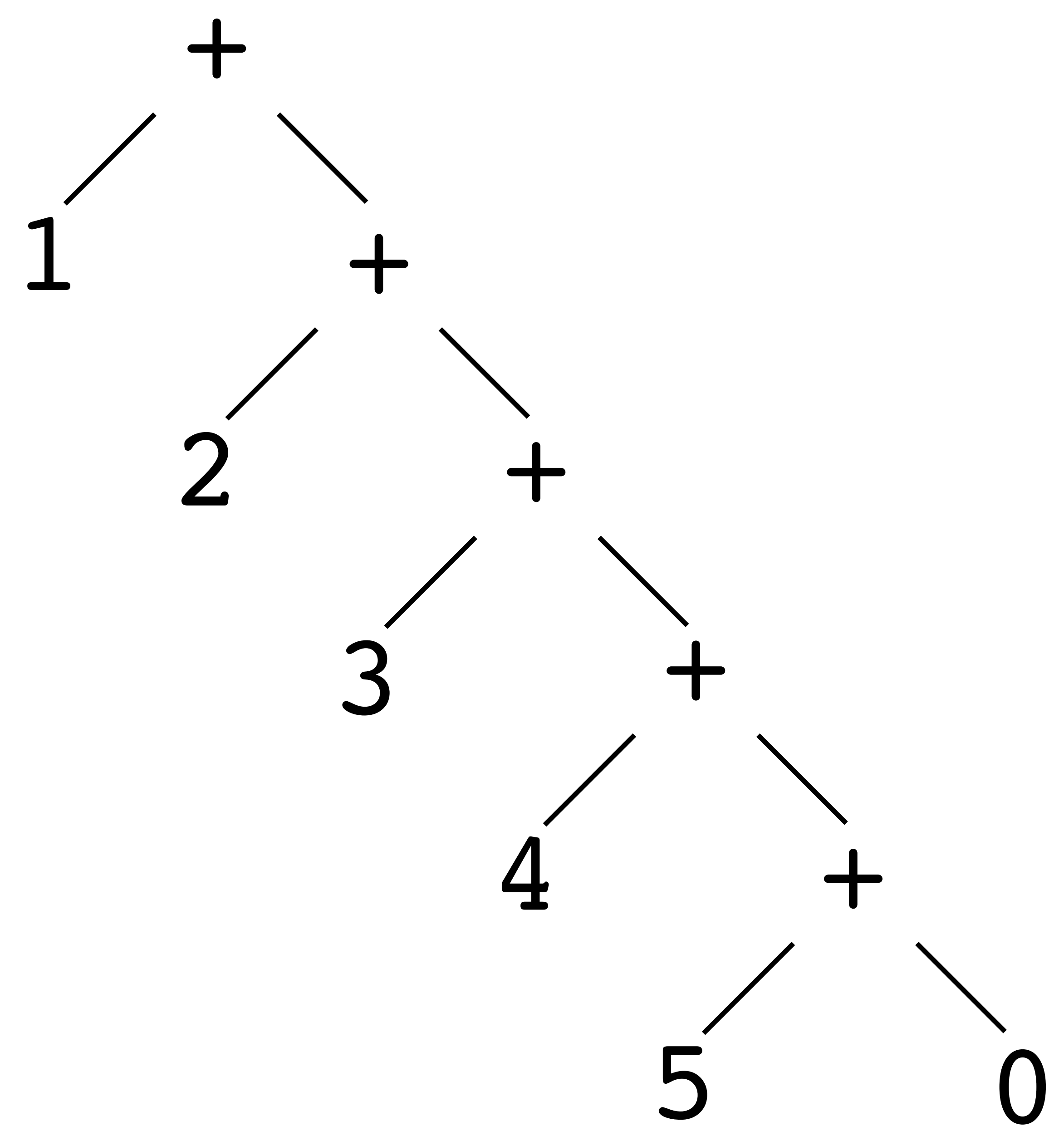
`(foldr combine base-case lst)`

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



Print out the arguments

```
(foldr (λ (x acc)
         (let ([result (+ x acc)])
           (printf "(+ ~s ~s) => ~s~n" x acc result)
           result))
       0
       '(1 2 3 4 5))
```

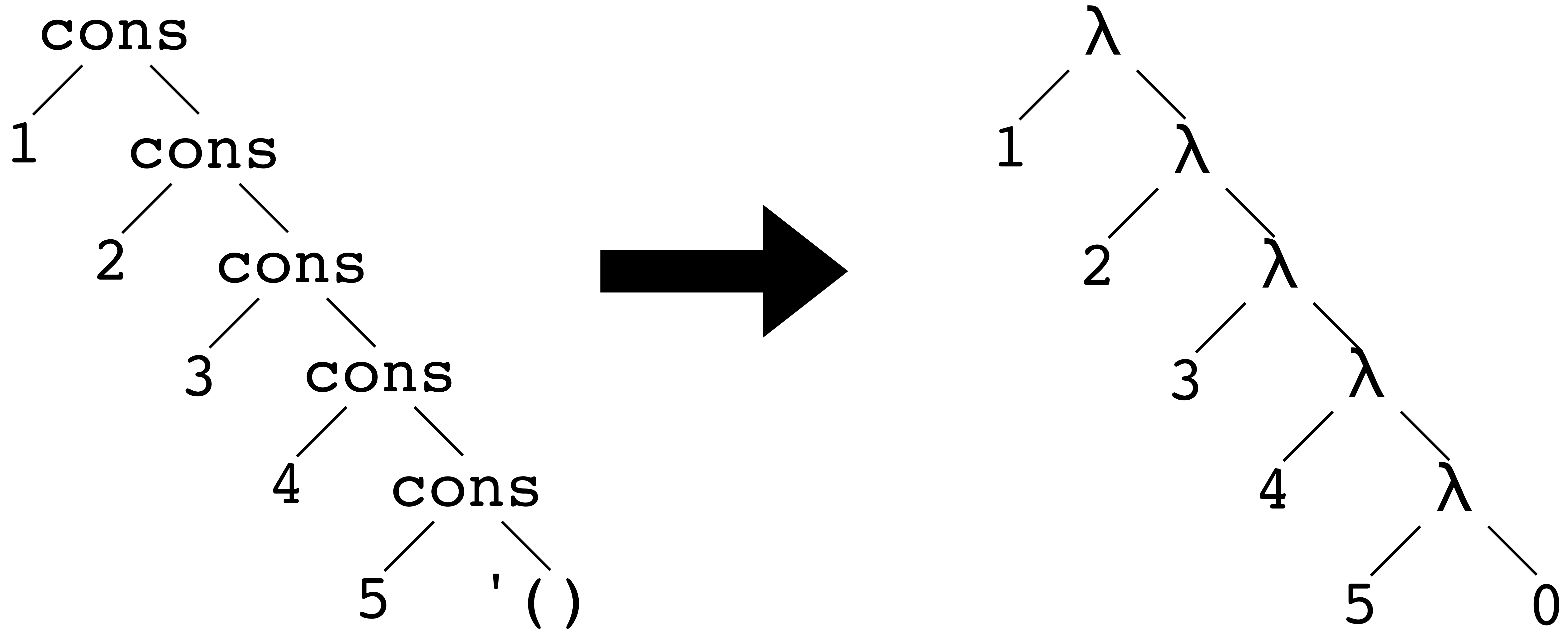


```
(+ 5 0) => 5
(+ 4 5) => 9
(+ 3 9) => 12
(+ 2 12) => 14
(+ 1 14) => 15
```

length as a fold right

`(foldr combine base-case lst)`

```
(define (length lst)
  (foldr (λ (head result) (+ 1 result)) 0 lst))
```



map and remove* as fold right

(foldr combine base-case lst)

```
(define (map proc lst)
  (foldr (λ (head result)
           (cons (proc head) result))
         empty
         lst))

(define (remove* x lst)
  (foldr (λ (head result)
           (if (equal? x head)
               result
               (cons head result)))
         empty
         lst))
```

Consider the procedure

```
(define (foo lst)
  (foldr (λ (head result)
           (+ (* head head) result))
         0
         lst))
```

What is the result of `(foo '(1 0 2))`?

A. `'(1 0 2)`

B. `'(5 4 4)`

C. `5`

D. `1`

E. None of the above

Consider the procedure

```
(define (bar x lst)
  (foldr (λ (head result)
           (if (equal? head x) #t result))
         #f
         lst))
```

What is the result of `(bar 25 '(1 4 9 16 25 36 49))`?

A. `'(#f #f #f #f #t #f #f)`

B. `'(#f #f #f #f #t #t #t)`

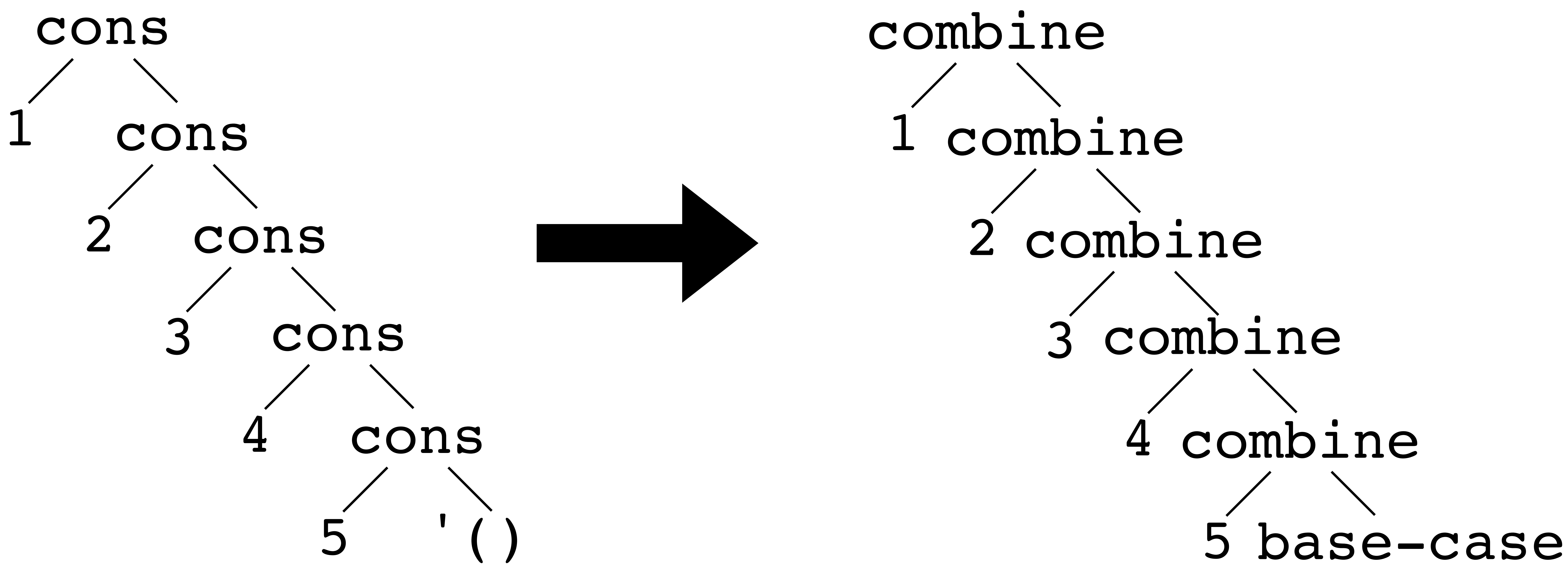
C. `#f`

D. `#t`

E. None of the above

Let's write `foldr`

`(foldr combine base-case lst)`


cons
1
cons
2
cons
3
cons
4
cons
5
'()
combine
1
combine
2
combine
3
combine
4
combine
5
base-case