Tutorial 9: Solutions

CS 135 Fall 2007

November 9, 2007

1 Local!

1.
```
(define (f x) (add1 x))
(f (local ((define (f x) (sub1 x)))
     (f 5)))
```

⇒

```
(define (f x) (add1 x))
(define (f1 x) (sub1 x))
(f (local ()
     (f1 5)))
```

⇒

```
(define (f x) (add1 x))
(define (f1 x) (sub1 x))
(f (f1 5))
```

⇒

```
(define (f x) (add1 x))
(define (f1 x) (sub1 x))
(f (sub1 5))
```

⇒

```
(define (f x) (add1 x))
(define (f1 x) (sub1 x))
(f 4)
```

⇒

```
(define (f x) (add1 x))
(define (f1 x) (sub1 x))
(add1 4)
```

⇒

```
(define (f x) (add1 x))
(define (f1 x) (sub1 x))
5
```

2.
```
(define a 2)
(define (f x)
  (local (define (a 5))
    (sqr a)))
(f -10)
```

⇒

Syntax error: Need an extra set of parenthesis around localdefinitions.

3.
```
(define (f x)
  (local ((define (g y)
            (cond [(empty? y) 5]
                  [else (* (first x)
                           (f (rest y)))])))
    (g (rest x))))
(f (list 2 3 4 5 6))
(f (list 1 2))
```

⇒

```
(define (f x)
  (local ((define (g y)
            (cond [(empty? y) 5]
                  [else (* (first x)
                           (f (rest y)))])))
    (g (rest x))))
(local ((define (g y)
```

```
  (cond [(empty? y) 5]
[else (* (first (list 2 3 4 5 6))
 (f (rest y)))]))))
  (g (rest (list 2 3 4 5 6))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y)
  (cond [(empty? y) 5]
[else (* (first (list 2 3 4 5 6))
 (f (rest y)))]))
(local ()
  (g1 (rest (list 2 3 4 5 6))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y)
  (cond [(empty? y) 5]
[else (* (first (list 2 3 4 5 6))
 (f (rest y)))]))
(g1 (rest (list 2 3 4 5 6)))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y)
  (cond [(empty? y) 5]
[else (* (first (list 2 3 4 5 6))
 (f (rest y)))]))
(g1 (list 3 4 5 6))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
```

```
(define (g1 y)
  (cond [(empty? y) 5]
[else (* (first (list 2 3 4 5 6))
 (f (rest y)))]))
(cond [(empty? (list 3 4 5 6)) 5]
      [else (* (first (list 2 3 4 5 6))
       (f (rest (list 3 4 5 6))))])
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(cond [false 5]
      [else (* (first (list 2 3 4 5 6))
       (f (rest (list 3 4 5 6))))])
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(cond [else (* (first (list 2 3 4 5 6))
       (f (rest (list 3 4 5 6))))])
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(* (first (list 2 3 4 5 6))
   (f (rest (list 3 4 5 6))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(* 2
   (f (rest (list 3 4 5 6))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(* 2
   (f (list 4 5 6)))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(* 2
   (local ((define (g y)
     (cond [(empty? y) 5]
   [else (* (first (list 4 5 6))
    (f (rest y)))])))
     (g (rest (list 4 5 6)))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y)
  (cond [(empty? y) 5]
[else (* (first (list 4 5 6))
 (f (rest y)))]))
(* 2
   (local ()
     (g2 (rest (list 4 5 6)))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y)
  (cond [(empty? y) 5]
[else (* (first (list 4 5 6))
```

```
 (f (rest y)))]))
(* 2
   (g2 (rest (list 4 5 6))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y)
  (cond [(empty? y) 5]
[else (* (first (list 4 5 6))
 (f (rest y)))]))
(* 2
   (g2 (list 5 6)))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y)
  (cond [(empty? y) 5]
[else (* (first (list 4 5 6))
 (f (rest y)))]))
(* 2
   (cond [(empty (list 5 6)) 5]
 [else (* (first (list 4 5 6))
  (f (rest (list 5 6))))]))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(* 2
   (cond [false 5]
 [else (* (first (list 4 5 6))
  (f (rest (list 5 6))))]))
```

```
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(* 2
   (cond [else (* (first (list 4 5 6))
  (f (rest (list 5 6))))]))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(* 2
   (* (first (list 4 5 6))
      (f (rest (list 5 6)))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(* 2
   (* 4
      (f (rest (list 5 6)))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(* 2
   (* 4
      (f (list 6))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(* 2
   (* 4
      (local ((define (g y)
(cond [(empty? y) 5]
      [else (* (first (list 6))
       (f (rest y)))]))))
(g (rest (list 6))))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y)
  (cond [(empty? y) 5]
[else (* (first (list 6))
 (f (rest y)))]))
(* 2
   (* 4
      (local ()
(g3 (rest (list 6))))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y)
  (cond [(empty? y) 5]
[else (* (first (list 6))
 (f (rest y)))]))
(* 2
```

```
   (* 4
      (g3 (rest (list 6)))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y)
  (cond [(empty? y) 5]
[else (* (first (list 6))
 (f (rest y)))]))
(* 2
   (* 4
      (g3 empty)))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y)
  (cond [(empty? y) 5]
[else (* (first (list 6))
 (f (rest y)))]))
(* 2
   (* 4
      (cond [(empty? empty) 5]
    [else (* (first (list 6))
     (f (rest empty)))]))))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
```

```
(* 2
   (* 4
      (cond [true 5]
    [else (* (first (list 6))
     (f (rest empty)))])))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(* 2
   (* 4
      5))
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(* 2
   20)
(f (list 1 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
40
(f (list 1 2))
```

⇒

```
(define (f x) ...)
```

```
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
40
(local ((define (g y)
  (cond [(empty? y) 5]
[else (* (first (list 1 2))
 (f (rest y)))])))
  (g (rest (list 1 2))))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(define (g4 y)
  (cond [(empty? y) 5]
[else (* (first (list 1 2))
 (f (rest y)))]))
40
(local ()
  (g4 (rest (list 1 2))))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(define (g4 y)
  (cond [(empty? y) 5]
[else (* (first (list 1 2))
 (f (rest y)))]))
40
(g4 (rest (list 1 2)))
```

⇒

```
(define (f x) ...)
```

```
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(define (g4 y)
  (cond [(empty? y) 5]
[else (* (first (list 1 2))
 (f (rest y)))]))
40
(g4 (list 2))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(define (g4 y)
  (cond [(empty? y) 5]
[else (* (first (list 1 2))
 (f (rest y)))]))
40
(cond [(empty? (list 2)) 5]
      [else (* (first (list 1 2))
       (f (rest (list 2))))])
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(define (g4 y) ...)
40
(cond [false 5]
      [else (* (first (list 1 2))
       (f (rest (list 2))))])
```

⇒

```
(define (f x) ...)
```

```
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(define (g4 y) ...)
40
(cond [else (* (first (list 1 2))
               (f (rest (list 2))))])
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(define (g4 y) ...)
40
(* (first (list 1 2))
   (f (rest (list 2))))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(define (g4 y) ...)
40
(* 1
   (f (rest (list 2))))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(define (g4 y) ...)
40
(* 1
   (f empty))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(define (g4 y) ...)
40
(* 1
   (local ((define (g y)
     (cond [(empty? y) 5]
   [else (* (first empty)
    (f (rest y)))])))
     (g (rest empty))))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(define (g4 y) ...)
(define (g5 y)
  (cond [(empty? y) 5]
[else (* (first empty)
 (f (rest y)))]))
40
(* 1
   (local ()
     (g5 (rest empty))))
```

⇒

```
(define (f x) ...)
(define (g1 y) ...)
(define (g2 y) ...)
(define (g3 y) ...)
(define (g4 y) ...)
(define (g5 y)
```

```
  (cond [(empty? y) 5]
[else (* (first empty)
 (f (rest y)))]))
40
(* 1
   (g5 (rest empty)))
```

⇒

Semantics error: `(rest empty)` has no meaning.

4.
```
(define a 2)
(local ((define b 4)
        (define a (add1 b)))
  (+ a b))
(+ a b)
```

⇒

```
(define a 2)
(define b1 4)
(local ((define a (add1 b1)))
  (+ a b1))
(+ a b)
```

⇒

```
(define a 2)
(define b1 4)
(define a1 (add1 b1))
(local ()
  (+ a1 b1))
(+ a b)
```

⇒

```
(define a 2)
(define b1 4)
(define a1 (add1 4))
(local ()
  (+ a1 b1))
(+ a b)
```

⇒

```
(define a 2)
(define b1 4)
(define a1 5)
(local ()
  (+ a1 b1))
(+ a b)
```

⇒

```
(define a 2)
(define b1 4)
(define a1 5)
(+ a1 b1)
(+ a b)
```

⇒

```
(define a 2)
(define b1 4)
(define a1 5)
(+ 5 b1)
(+ a b)
```

⇒

```
(define a 2)
(define b1 4)
(define a1 5)
(+ 5 4)
(+ a b)
```

⇒

```
(define a 2)
(define b1 4)
(define a1 5)
9
(+ a b)
```

⇒

```
(define a 2)
(define b1 4)
(define a1 5)
9
(+ 2 b)
```

⇒

Semantics error: `b` has no meaning.

2 Functional Abstraction using Family Trees

See t9-solns.scm.