Tutorial 4: Solutions

CS 135 Fall 2007

October 5, 2007

1.
```
(and (symbol? 'hello)
     (= (- 5 1) (* 2 3))
     (/ "a string" "another string"))
```

⇒

```
(and true
     (= (- 5 1) (* 2 3))
     (/ "a string" "another string"))
```

⇒

```
(and (= (- 5 1) (* 2 3))
     (/ "a string" "another string"))
```

⇒

```
(and (= 4 (* 2 3))
     (/ "a string" "another string"))
```

⇒

```
(and (= 4 6)
     (/ "a string" "another string"))
```

⇒

```
(and false
     (/ "a string" "another string"))
```

⇒

```
false
```

2.

```
(define a (+ 2 3))
(define (foo2 x)
  (cond [(or (> x 1)
     (< x -1))
 (sqr x)]
[(zero? x) 1]))
(foo2 a)
(foo2 (/ a a))
```

⇒

```
(define a 5)
(define (foo2 x)
  (cond [(or (> x 1)
     (< x -1))
 (sqr x)]
[(zero? x) 1]))
(foo2 a)
(foo2 (/ a a))
```

⇒

```
...
(foo2 5)
(foo2 (/ a a))
```

⇒

```
...
(cond [(or (> 5 1)
   (< 5 -1))
       (sqr 5)]
      [(zero? x) 1])
(foo2 (/ a a))
```

⇒

```
...
(cond [(or true
       (< 5 -1))
           (sqr 5)]
          [(zero? x) 1])
    (foo2 (/ a a))
```

⇒

```
    ...
    (cond [true
           (sqr 5)]
          [(zero? x) 1])
    (foo2 (/ a a))
```

⇒

```
    ...
    (sqr 5)
    (foo2 (/ a a))
```

⇒

```
    ...
    25
    (foo2 (/ a a))
```

⇒

```
    ...
    25
    (foo2 (/ 5 a))
```

⇒

```
    ...
    25
    (foo2 (/ 5 5))
```

⇒

```
    ...
    25
    (foo2 1)
```

⇒

```
...
25
(cond [(or (> 1 1)
   (< 1 -1))
       (sqr 1)]
      [(zero? 1) 1])
```

⇒

```
...
25
(cond [(or false
   (< 1 -1))
       (sqr 1)]
      [(zero? 1) 1])
```

⇒

```
...
25
(cond [(or (< 1 -1))
       (sqr 1)]
      [(zero? 1) 1])
```

⇒

```
...
25
(cond [(or false)
       (sqr 1)]
      [(zero? 1) 1])
```

⇒

```
...
25
(cond [(or)
       (sqr 1)]
      [(zero? 1) 1])
```

⇒

```
...
25
(cond [false
        (sqr 1)]
       [(zero? 1) 1])
```

⇒

```
...
25
(cond [(zero? 1) 1])
```

⇒

```
...
25
(cond [false 1])
```

⇒

```
...
25
(cond)
```

⇒

Semantics error (no substitution rule for `(cond)`)

3.
```
(define (foo3 5)
  (+ 1 5))
(/ (foo3 5)
   0)
```

⇒

Syntax error: 5 is not a valid variable name

Note: The division by zero is not even considered because the syntax error is seen first.

4.
```
(define-struct name (first middle last))
(define (foo4 nme)
  (name-middle (+ nme 1)))
(name-last (make-name "James" "A" "Garfield"))
```

⇒

```
(define-struct name (first middle last))
(define (foo4 nme)
  (name-middle (+ nme 1)))
"Garfield"
```

Note: The function `foo4` will generate a semantics error on *every* function call, but since it is never called, there is no error here.

5.
```
(define (foo5 x)
  (cond [(= 1 x) 2]
[else
  (* 2
     (foo5 (sub1 x)))]))
(foo5 3)
(foo5 -2)
```

⇒

```
...
(cond [(= 1 3) 2]
      [else
(* 2
   (foo5 (sub1 3)))])
(foo5 -2)
```

⇒

```
...
(cond [false 2]
      [else
(* 2
   (foo5 (sub1 3)))])
(foo5 -2)
```

⇒

```
...
(cond [else
(* 2
   (foo5 (sub1 3)))])
(foo5 -2)
```

⇒

```
...
(* 2
   (foo5 (sub1 3)))
(foo5 -2)
```

⇒

```
...
(* 2
   (foo5 2))
(foo5 -2)
```

⇒

```
...
(* 2
   (cond [(= 1 2) 2]
 [else
   (* 2
      (foo5 (sub1 2)))]))
(foo5 -2)
```

⇒

```
...
(* 2
   (cond [false 2]
 [else
   (* 2
      (foo5 (sub1 2)))]))
(foo5 -2)
```

⇒

```
...
(* 2
   (cond [else
   (* 2
      (foo5 (sub1 2)))]))
(foo5 -2)
```

⇒

```
...
(* 2
   (* 2
      (foo5 (sub1 2))))
(foo5 -2)
```

⇒

```
...
(* 2
   (* 2
      (foo5 1)))
(foo5 -2)
```

⇒

```
...
(* 2
   (* 2
      (foo5 1)))
(foo5 -2)
```

⇒

```
...
(* 2
   (* 2
     (cond [(= 1 1) 2]
   [else
     (* 2
        (foo5 (sub1 1)))])))
(foo5 -2)
```

$\Rightarrow$

```
...
(* 2
   (* 2
     (cond [true 2]
   [else
     (* 2
        (foo5 (sub1 1)))])))
(foo5 -2)
```

$\Rightarrow$

```
...
(* 2
   (* 2
      2))
(foo5 -2)
```

$\Rightarrow$

```
...
(* 2
   4)
(foo5 -2)
```

$\Rightarrow$

```
...
8
(foo5 -2)
```

$\Rightarrow$

Semantics error: infinite loop