Carson Kim
Ling 185A - HW #1
1/14/2020

```
1. let x = 4 + 5 in (3 * x)
      (3 * (4 + 5))  --------- let reduction
      27 ----- arithmetic

2. (\x -> 3 * x) (4 + 5)
      (3 * (4 + 5) ---- lambda reduction
      27 ---- arithmetic

3. ((\x-> (\y -> x + (3 * y))) 4 ) 1
      (\y -> 4 + (3 * y)) 1 ---- lambda reduction
      4 + (3 * 1) ----- lambda reduction
      4 + 3 ----- arithmetic
      7 ----- arithmetic

4. let x = 4 in (let y = 1 in (x + (3 * y)))
      let x = 4 in x + 3 * 1 --- let reduction
      let x = 4 in x + 3 --- arithmetic
      4 + 3 --- let reduction
      7 --- arithmetic

5. let x = 4 in (let y = 1 + x in (x + (3 * y)))
      let x = 4 in x + (3 * (1 + x)) --- let reduction
      4 + (3 * (1 + 4)) -- let reduction
      19 --- arithmetic

6.  ((\x -> (\y -> x + (3 * x))) 4) 1
      (\y-> 4 + (3 * 4) 1 --- lambda reduction
      4 + (3 * 4) --- lambda reduction
      4 + 12 --- arithmetic
      16 --- arithmetic

7. ((\x -> (\y -> y + (3 * y))) 4) 1
      (\y -> y + (3 * y)) 1 --- lambda reduction
      1 + (3 * 1) --- lambda reduction
      1 + 3  --- arithmetic
      4 --- arithmetic
```

```
8. (\y -> y + ((\y -> 3*y) 4)) 5
      (\y -> y + (3 * 4)) 5 --- lambda reduction
      5 + (3 * 4) --- lambda reduction
      5 + 12 --- arithmetic
      17 --- arithmetic

9. (\y -> ((\y -> 3*y) 4) + y) 5
      (\y -> (3 * 4) + y) 5 --- lambda reduction
      (3 * 4) + 5 --- lambda reduction
      12 + 5 --- arithmetic
      17 -- arithmetic

10. (\x -> x * (let x = 3*2 in (x + 7)) + x) 4
      (\x -> x * ((3*2) + 7) + x) 4 --- let reduction
      4 * ((3*2) + 7) + 4 --- lambda reduction
      (4 * 13) + 4 --- arithmetic
      52 + 4 --- arithmetic
      56 --- arithmetic

11. g ((let x = 4 in (\y -> x + y)) 2)
      g ((let x = 4 in (x + 2))) --- lambda reduction
      g (4 + 2) --- let reduction
      g 6 --- arithmetic
      (\z-> z + 4) 6 --- substitution from file
      6 + 4 -- lambda reduction
      10 -- arithmetic

12. let fn = \x -> (let y = 3 in x + y) in fn 4
      (\x -> (let y = 3 in x + y)) 4 --- let reduction
      (\x -> (x + 3)) 4 --- let reduction
      4 + 3 --- lambda reduction
      7 --- arithmetic

13. let fn = (let y = 3 in \x -> x + y) in fn 4
      (let y = 3 in \x -> x + y) 4 --- let reduction
      (\x -> x + 3) 4 --- let reduction
      4 + 3 --- lambda reduction
      7 --- arithmetic

14. f ((\fn -> fn Rock) (\x -> whatItBeats x))
      f ((x -> whatItBeats x) Rock) --- lambda reduction
      f (whatItBeats Rock) --- lambda reduction
```

f (\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}) Rock --- substitution from file

f (case Rock of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}) --- lambda reduction

f Scissors --- case reduction

\s -> case s of {Rock -> 334; Paper -> 138; Scissors -> 99} Scissors --- substitution from file

case Scissors of {Rock -> 334; Paper -> 138; Scissors -> 99} --- lambda reduction

99 --- case reduction

15. **((\f -> (\x -> f (f x))) whatItBeats) Paper**

(\x -> whatItBeats(whatItBeats x)) Paper --- lambda reduction

whatItBeats (whatItBeats Paper) --- lambda reduction

whatItBeats (\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper} Paper) --- substitution from file

whatItBeats (case Paper of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}) ---- lambda reduction

whatItBeats Rock ---- case reduction

(\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}) Rock --- file substitution

(case Rock of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}) Rock --- lambda reduction

Scissors --- case reduction

16. **whatItBeats (case Paper of {Rock -> Paper; Paper -> Rock; Scissors -> Scissors})**

```
\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}
(case Paper of {Rock -> Paper; Paper -> Rock; Scissors -> Scissors})
---substitution from file

\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}
Rock --- case reduction

case Rock of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper} ---
lambda reduction

Scissors --- case reduction
```

17. **(case (Win Rock) of {Draw -> whatItBeats; Win z -> (\s -> Scissors)}) Paper**

```
(\s -> Scissors) Paper --- case reduction
Scissors --- lambda reduction
```

18. case **(Win (whatItBeats Rock)) of {Draw -> n; Win x -> (n + f x)}**

```
(n + f (whatItBeats Rock)) --- case reduction

(n + f (\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors ->
Paper} Rock)) --- substitution from file

(n + f (case Rock of {Rock -> Scissors; Paper -> Rock; Scissors ->
Paper} Rock)) --- lambda reduction

n + f Scissors -> case reduction

n + (\s -> case s of {Rock -> 334; Paper -> 138; Scissors -> 99})
Scissors --- substitution from file

n + (case Scissors of {Rock -> 334; Paper -> 138; Scissors -> 99}) ---
lambda reduction

n + 99 --- case reduction
1 + 99 --- substitution from file
100 --- arithmetic
```

19. **let y = 2 in (case (Win (whatItBeats Rock)) of {Draw -> n; Win y -> (n + f y)} + y)**

```
(case (Win (whatItBeats Rock)) of {Draw -> n; Win y -> (n + f y)} + 2) ---
let reduction

(n + f whatItBeats Rock) + 2) --- case reduction

(n + f (\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}
Rock)) + 2 ---- substitution from file

(n + f (case Rock of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper})) +
2 --- lambda reduction

(n + f Scissors) + 2 -- case reduction

(1 + (\s -> case s of {Rock -> 334; Paper -> 138; Scissors -> 99}) Scissors
)) + 2 --- substitution from file

(1 + (case Scissors of {Rock -> 334; Paper -> 138; Scissors -> 99}) Scissors
)) + 2 --- lambda reduction

(1 + (99)) + 2 --- case reduction

102 --- arithmetic
```

Index of comments

1.1 multiple arithmetic evaluations in one step.

-1.5

2.1 Watch out for the parentheses. The first step could only be a let reduction but not a lambda reduction.
-1