Devin Liu

HW #1

1. let x = 4 + 5 in (3 * x) Given
   => let x = 9 in (3 * x) Arithmetic
   => (3 * 9) Let Reduction
   => 27 Arithmetic

2. (\x -> 3 * x) (4 + 5) Given
   => (\x -> 3 * x) (9) Arithmetic
   => 3 * 9 Lambda Reduction
   => 27 Arithmetic

3. ((\x -> (\y -> x + (3 * y))) 4) 1 Given
   => (\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))) Given
   => let x = 4 in (x + (3 * 1)) Let Reduction
   => 4 + (3 * 1) Let Reduction
   => 4 + 3 Arithmetic
   => 7 Arithmetic

5. let x = 4 in (let y = 1 + x in (x + (3 * y))) Given
   => let x = 4 in (x + (3 * (1 + x))) Let Reduction
   => 4 + (3 * (1 + 4)) Let Reduction
   => 4 + (3 * 5) Arithmetic
   => 4 + 15 Arithmetic
   => 19 Arithmetic

6. ((\x -> (\y -> x + (3 * x))) 4) 1 Given
   => (\y -> 4 + (3 * 4)) 1 Lambda Reduction
   => 4 + (3 * 4) Lambda Reduction (No free variables)
   => 4 + 12 Arithmetic
   => 16 Arithmetic

7. ((\x -> (\y -> y + (3 * y))) 4) 1 Given
   => (\y -> y + (3 * y)) 1 Lambda Reduction (No free variables)
   => 1 + (3 * 1) Lambda Reduction
   => 1 + 3 Arithmetic

=> 4 Arithmetic

8. (\y -> y + ((\y -> 3*y) 4)) 5 Given
   => (\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 Given
   => (\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 Given
    => (\x -> x * (let x = 6 in (x + 7)) + x) 4 Arithmetic
    => (\x -> x * (6 + 7) + x) 4 Let Reduction
    => 4 * (6 + 7) + 4 Lambda Reduction
    => 4 * 13 + 4 Arithmetic
    => 52 + 4 Arithmetic
    => 56 Arithmetic

11. g ((let x = 4 in (\y -> x + y)) 2) Given
    => (\z -> z + 4) ((let x = 4 in (\y -> x + y)) 2) Substitution from file
    => (\z -> z + 4) ((\y -> 4 + y) 2) Let Reduction
    => (\z -> z + 4) (4 + 2) Lambda Reduction
    => (\z -> z + 4) (6) Arithmetic
    => 6 + 4 Lambda Reduction
    => 10 Arithmetic

12. let x = 5 in (\z -> x * z) Given
    => (\z -> 5 * z) Let Reduction

    *This line results in an error because the lambda function has no value for lambda reduction.

13. f ((\fn -> fn Rock) (\x -> whatItBeats x)) Given
    => 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

14. ((\f -> (\x -> f (f x))) whatItBeats) Paper Given
=> (\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
Substitution from file
=> case Rock of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}
Lambda Reduction
=> Scissors Case Reduction

15. whatItBeats (case Paper of {Rock -> Paper; Paper -> Rock; Scissors -> Scissors})
Given
=> whatItBeats Rock Case Reduction
=> (\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}) Rock
Substitution from file
=> case Rock of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}
Lambda Reduction
=> Scissors Case Reduction

16. (case (Win Rock) of {Draw -> whatItBeats; Win z -> (\s -> Scissors)}) Paper
Given
=> (\s -> Scissors) Paper Case Reduction
=> Scissors Lambda Reduction (No free variables)

17. case (Win (whatItBeats Rock)) of {Draw -> n; Win x -> (n + f x)} Given
=> case (Win ((\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors ->
Paper}) Rock)) of {Draw -> n; Win x -> (n + f x)}
Substitution from file
=> case (Win (case Rock of {Rock -> Scissors; Paper -> Rock; Scissors ->

Paper})) of {Draw -> n; Win x -> (n + f x)}

Lambda Reduction

=> case (Win Scissors) of {Draw -> n; Win x -> (n + f x)} Case Reduction

=> (n + f Scissors) Case Reduction

=> (1 + f Scissors) Substitution from file

=> (1 + (\s -> case s of {Rock -> 334; Paper -> 138; Scissors -> 99}) Scissors)

Substitution from file

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

Lambda Reduction

=> 1 + 99 Case Reduction

=> 100 Arithmetic

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

Given

=> let y = 2 in (case (Win ((\s -> case s of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper}) Rock)) of {Draw -> n; Win y -> (n + f y)} + y)

Substitution from file

=> let y = 2 in (case (Win (case Rock of {Rock -> Scissors; Paper -> Rock; Scissors -> Paper})) of {Draw -> n; Win y -> (n + f y)} + y)

Lambda Reduction

=> let y = 2 in (case (Win Scissors) of {Draw -> n; Win y -> (n + f y)} + y)

Case Reduction

=> let y = 2 in (n + f Scissors + y) Case Reduction

=> let y = 2 in (1 + f Scissors + y) Substitution from file

=> let y = 2 in (1 + (\s -> case s of {Rock -> 334; Paper -> 138; Scissors -> 99}) Scissors + y)

Substitution from file

=> let y = 2 in (1 + (case Scissors of {Rock -> 334; Paper -> 138; Scissors -> 99}) + y)

Lambda Reduction

=> let y = 2 in (1 + 99 + y) Case Reduction

=> (1 + 99 + 2) Let Reduction

=> 100 + 2 Arithmetic

=> 102 Arithmetic