Discussion 10 (4/1)

- If you didn't have time to review lexing and parsing from last week, do that first! Let them ask questions as well
- Evaluation / Operational Semantics
    - Goal of operational semantics: to add meaning to whatever we just parsed, to evaluate it
    - A basic operational semantic rule has 3 parts:
        - Expression
        - Hypothesis
        - Result
    - Example 2
        - For the second rule
            - Input expression is e1 + e2
            - Hypothesis is the stuff above the line (3 hypotheses here)
            - Result is n3
            - By these rules, given two expressions to add, we get n3 by the hypotheses
    - Why do we need this? When we write programs, we need formal rules to show this
    - To show that our program is following the rules, we can draw a tree that evaluates the given expression
    - Example 2
        - We are evaluating 1 + (2 + 3) => 6, so that will be at the bottom of the tree
        - This maps to the second rule, so looking at the second rule, we see e1 = 1 and e2 = (2 + 3)
        - First, we evaluate e1 = 1 using the first rule (1 => 1)
        - To evaluate e2 = (2 + 3), we will need to use the second rule again to break it down. For this second instance, e1 = 2 and e2 = 3.
        - We can evaluate e1 = 2 and e2 = 3 using the first rule again (2 => 2, 3 => 3)
        - We get the result, 5, by summing n1 and n2 (5 is 2 + 3), so the whole expression (2 + 3) evaluates to 5
        - We return to the original evaluation to get 6 is 1 + 5
        - Thus, the whole expression evaluates to 6!

$$\dfrac{1 \Rightarrow 1 \qquad \dfrac{2 \Rightarrow 2 \quad 3 \Rightarrow 3 \quad 5 \text{ is } 2+3}{2+3 \Rightarrow 5} \qquad 6 \text{ is } 1+5}{1+(2+3) \Rightarrow 6}$$

        -
    - This was a pretty simple example, but in the context of programming we also will have an environment of variables we could draw from
    - This is usually represented by an uppercase A, followed by any variables stored in the environment. We call this evaluating the expression under environment A

- Example 4
    - The rule for let expressions starts with environment A for the first hypothesis, but after processing the first hypothesis we see our environment A now also contains x : v1
    - Now, if e2 needs information about x, we can reference v1 from the environment
- Keeping track of environments will be crucial for P4b, and will help in understanding real programming environments
- Example 4
    - We are evaluating A; let y = 1 in let x = 2 in x => 2, so that is our input expression
    - This matches to the rule for let expressions, so we start by evaluating e1 under A, e1 being 1
    - Note that A corresponds to whatever A we started with, so it could already have existing variables in it. We would have to keep those unless we explicitly have a rule that says otherwise
    - Now our environment contains y : 1, and we want to evaluate e2, which is the second let statement
    - We use the let expression rule again to evaluate e2, but this time the environment will contain y : 1 already
    - Now we will evaluate e2 for the second let expression, which is x
    - This time, we will need to use the updated environment to get the value of x, which is 2
    - Thus, overall we evaluate the expression to 2

$$\dfrac{A;\ 1 \Rightarrow 1 \qquad \dfrac{A, y:1;\ 2 \Rightarrow 2 \qquad \dfrac{A, y:1, x:2\ (x) = 2}{A, y:1, x:2;\ x \Rightarrow 2}}{A, y:1;\ \text{let } x = 2 \text{ in } x \Rightarrow 2}}{A;\ \text{let } y = 1 \text{ in let } x = 2 \text{ in } x \Rightarrow 2}$$

    -
- What if we have two let statements that both declare x? Then, our environment would have two instance of x values. We would shadow by only using the most recent (rightmost) value of x