Discussion 10: November 11th

Reminders:

- Quiz due today
  - Covered topics: Operational Semantics, CFGs, Lambda Calculus
  - As a result, today's discussion will be shorter in case students want to take the quiz in the class.
- Exam 2 on coming Thursday. Logistics post here
- Exam review on coming Monday: Hybrid mode of delivery. Logistics post here

Operational Semantics

Formal semantics of a PL: Mathematical description of meaning of programs in a PL.

Terminologies in Operational Semantics

```
    A; e1 => v1   A;e2 => v2   v3 is v1 + v2
    ---------------------------------------
              A; e1 + e2 => v3
```

- **Expression**: A program that evaluates to a value
- **Value**: A result of an expression
- **Environment**: A mapping from variables to values
- **Hypothesis**: A set of rules that describe the meaning of expressions
- **Judgement**: A statement with expressions and values (`e => v`). The expression `e` evaluates to the value `v` .

Operational Semantics Exercise

Solve the following problems using operational semantics:

```
Evaluate 1 + (2 + 3) = 6 using the following hypotheses:

                                      e1 => v1   e2 => v2   v3 is v1 + v2
       ------------                  ---------------------------------------
         n  => n                               e1 + e2 => v3


   1. 1 + (2 + 3) => 6
       That is the second rule.


      ------------------------------------------------------------------------------------
                                    1 + (2 + 3) => 6
       e1 = 1
       e2 = (2 + 3)
       v3 = 6

   2. 1 => 1
       That is the first rule.

        ----------
```

```
          1 => 1
      ------------------------------------------------------------------------------
                                  1 + (2 + 3) => 6
```

3. (2 + 3) => 5
   That is the second rule.

```
       ----------     ---------------------------------------
          1 => 1                  2 + 3 => 5
      ------------------------------------------------------------------------------
                                  1 + (2 + 3) => 6
```

   e1 = 2
   e2 = 3
   v3 = 5

4. 2 => 2, 3 => 3
   Those are the first rule.

```
                      --------  --------
                       2 => 2    3 => 3     5 is 2 + 3
       ----------     ---------------------------------------
          1 => 1                  2 + 3 => 5
      ------------------------------------------------------------------------------
                                  1 + (2 + 3) => 6
```

5. Final step:
   Compute the result

```
                      --------  --------
                       2 => 2    3 => 3     5 is 2 + 3
       ----------     ---------------------------------------
          1 => 1                  2 + 3 => 5                          6 is 1 + 5
      ------------------------------------------------------------------------------
                                  1 + (2 + 3) => 6
```

6. We are done.

Evaluate the expression using the given hypotheses: A; let y = 1 in let x = 2 in x => 2

```
                                                        A(x) = v
           ------------                               -------------
             n  => n                                   A; x => v

    A; e1 => v1   A, x: v1; e2 => v2         A; e1 => v1   A; e2 => v2   v3 is v1 + v2
   ----------------------------------      ------------------------------------------
      A; let x = e1 in e2 => v2                        A; e1 + e2 => v3
```

1. A; let y = 1 in let x = 2 in x
   That is the third rule.

```
    ------------------------------------------------------------------------------------
                                A; let y = 1 in let x = 2 in x => 2
    e1 = 1
    e2 = let x = 2 in x
    v2 = 1
```

2. 1 => 1
    That is the first rule.

```
     -----------
      A; 1 => 1
    ------------------------------------------------------------------------------------
                                A; let y = 1 in let x = 2 in x => 2
```

3. let x = 2 in x
    That is the third rule. NOTICE HOW WE UPDATED THE ENVIRONMENT.

```
     -----------     ------------------------------------------------------------------
      A; 1 => 1                          A, y: 1; let x = 2 in x => 2
    ------------------------------------------------------------------------------------
                                A; let y = 1 in let x = 2 in x => 2

    e1 = 2
    e2 = x
    v2 = 2
```

4. 2 => 2
    That is the first rule.

```
                    -----------------
                     A, y: 1; 2 => 2
     -----------     ------------------------------------------------------------------
      A; 1 => 1                          A, y: 1; let x = 2 in x => 2
    ------------------------------------------------------------------------------------
                                A; let y = 1 in let x = 2 in x => 2
```

5. x
    That is the second rule. NOTICE HOW WE UPDATED THE ENVIRONMENT.

```
                                                        A, y: 1, x: 2 (x) => 2
                    -----------------              ----------------------------------------
                     A, y: 1; 2 => 2                       A, y: 1, x: 2; x => 2
     -----------     ------------------------------------------------------------------
      A; 1 => 1                          A, y: 1; let x = 2 in x => 2
    ------------------------------------------------------------------------------------
                               A; let y = 1 in let x = 2 in x => 2
```

6. We are done.

Lambda Calculus

Lambda Calculus Terminologies

```
\x. \y. x + y + a
```

- **Lambda Expression**: A lambda expression is a function that takes an argument and returns a value.
- **Free Variable**: A variable that is not bound by a lambda expression.
- **Lambda Abstraction**: A lambda abstraction is a function that takes an argument and returns a function.
- **Alpha Conversion**: A process of renaming a variable in a lambda expression to avoid name conflict. Does not change the meaning of the expression. **Do not rename free variables**
- **Beta Reduction**: A process of substituting a lambda expression for a variable in a lambda abstraction.

Lambda Calculus Exercise

Things to keep in mind: 1. Alpha conversion: Do not rename free variables 2. Explicit Parentheses: Scope of a variable extends to far right or the first `)` seen. 3. Lambda Calculus is left-associative. 4. Beta reduction: Keep applying the functions until you can't anymore.

Solve the following problems using lambda calculus:

1. Make parentheses explicit in the following lambda expressions:

```
1. a b c
    Left-associative: (a b) c
    Left-associative: ((a b) c)

2. \a. \b. a b
    Far right: (\a. \b. a b)
    Far right: (\a. (\b. a b))
    Left-associative: (\a. (\b. (a b)))

3. \a. a b \a. a b
    Far right: (\a. a b \a. a b)
    Far right: (\a. (a b \a. a b))
    Left-associative: (\a. ((a b) \a. a b))
    Far right: (\a. ((a b) (\a. a b)))
    Left-associative: (\a. ((a b) (\a. (a b))))
```

2. Identify the free variables in the following lambda expressions:

```
1. \a. a b a
    Explicit Parentheses are helpful.
    (\a. ((ab) a))

    Since the scope extends to the far right, 'a' cannot be free as both 'a's are bound.
    'b' does not. It is free.

2. \a. \b. a b
    (\a. (\b. (a b)))

    Since both 'a' and 'b' are bound, no free variables.

3. \a. (\b. a b) a b
    (\a. (((\b. a b) a) b))

    'b's scope is limited to the first ')' while that for 'a' extendts to far right.
    Hence, the last 'b' is free.
```

3. Perform alpha conversion on the following lambda expressions:

```
1. \a. \a. a
    Identify the free variables and definition of each variable.
    No free variables. So, we choose to rename either or both definitions of 'a'.

    Some equivalent ways to convert:
        \a. \b. b
        \b. \a. a
        \b. \c. c

2. (\a. a) a b
    Identify the free variables and definition of each variable.
    'a' and 'b' at the end are free. So, we cannot rename them. So, we'd have to rename the first 'a'.

    The only way possible is:
        (\c. c) a b


3. (\a. (\a. (\a. a) a) a) a
    Identify the free variables and definition of each variable.
    Last 'a' is free. But, we have multiple definitions of 'a'. So, we need to rename all definitions

    Some equivalent ways to convert:
        (\f. (\b. (\c. c) b) f) a
        all combinations of 3 different letters.
```

4. Perform beta reduction on the following lambda expressions:

1. (\a. a b) x b
    Again, explicit parentheses are helpful.
    (((\a. (a b)) x) b)

    Take x and plug it into the first parameter, here \a.
    (x b) b

    Irreduecible, further.

    We are done.

2. (\a. b) (\a. \b. \c. a b c)
    Explicit parentheses:
    ((\a. b) (\a. (\b. (\c. ((a b) c)))))

    When you appying the second group to the first, there are no 'a's to substitute.
    That is the only possible application.

    b

    We are done.

    NOTE: You can reduce such expressions wittily if you make parentheses explicit first.

3. (\a. a a) (\a. a a)

    Explicit parentheses:
    ((\a. (a a)) (\a. (a a)))

    When you apply the second group to the first, you get the second parameter twice because of two 'a
    This makes it an infinite loop.
    Hence, it is irreducible further.

    (\a. (a a)) (\a. (a a))

    We are done.