Rules for Environment Diagrams

Remember: The purpose of drawing environment diagrams is to visualize how Scheme evaluates expressions.

Preparational work: Start by drawing the Global Environment, set it as your Current Environment.

The Golden Rules:

1. *Calling a lambda procedure makes frames*, pointing to the right bubble of the `lambda`.
2. *Making a lambda procedure makes 2 bubbles*, the left holding information and the right pointing to the current frame.

The Complete Rules:

1. If the expression is a **constant**, self-evaluate. There is no need to change anything.
2. If the expression is a **symbol**, find the binding in Current Environment. If it's not found, go to the environment that the current environment points to and try to and it there. Repeat process until the binding is found. Error if no binding found in the Global Environment.
3. If the expression is a **procedure call** then:
   (a) If the procedure being called is a **`lambda` procedure**, follow these lambda-proc call rules:
      i. Evaluate the arguments by recursively applying these same rules
      ii. Find the right bubble of the procedure we're calling, then create a new environment that points to the frame that the right bubble points to and bind formal parameters to their respective arguments. *This is the only place you make frames.*
      iii. Set the new environment as your current environment, evaluate the body of the procedure by recursively applying these same rules
      iv. Go back to the previous current environment after procedure call finishes.
   (b) If the procedure being called is a **primitive procedure**, then follow these rules:
      i. Evaluate the procedure and arguments by recursively applying these same rules
      ii. Apply the procedure by magic
4. If the expression is a **lambda expression**, then create two bubbles. The left bubble should point to the parameters and body, and the right bubble points to the *current environment*.
5. If the expression is a **define expression**, convert so it does not use the syntactic shortcut of lambdas, evaluate the second argument, then add a new binding in the current environment to the value.
   (a) To convert: `(define (foo args) body)` ⇒ `(define foo (lambda (args) body))`
6. If the expression is a **set! expression**, then change the first available binding.
7. If the expression is a **let expression**, convert it to a lambda procedure call, then follow the lambda-proc call rules.
   (a) To convert: `(let ((a b) (c d)) body)` ⇒ `((lambda (a c) body) b d)`