Environment Model Rules

- Here are the complete rules for the environment model:

Every expression is either an atom or a list.

At any time there is a *current frame*, initially the global frame.

I. Atomic expressions.

   A. Numbers, strings, `#T`, and `#F` are self-evaluating.

   B. If the expression is a symbol, find the *first available* binding. (That is, look in the current frame; if not found there, look in the frame "behind" the current frame; and so on until the global frame is reached.)

II. Compound expressions (lists).

If the car of the expression is a symbol that names a special form, then follow its rules (II.B below). Otherwise the expression is a procedure invocation.

   A. Procedure invocation.

      Step 1: Evaluate all the subexpressions (using these same rules).

      Step 2: Apply the procedure (the value of the first subexpression) to the arguments (the values of the other subexpressions).

      (a) If the procedure is compound (user-defined):

         a1: Create a frame with the formal parameters of the procedure bound to the actual argument values.

         a2: Extend the procedure's defining environment with this new frame.

         a3: Evaluate the procedure body, using the new frame as the current frame.

         *** ONLY COMPOUND PROCEDURE INVOCATION CREATES A FRAME ***

      (b) If the procedure is primitive:

         Apply it by magic.

   B. Special forms.

      1. `Lambda` creates a procedure. The left circle points to the text of the `lambda` expression; the right circle points to the defining environment, i.e., to the current environment at the time the `lambda` is seen.

         *** ONLY `LAMBDA` CREATES A PROCEDURE ***

      2. `Define` adds a *new* binding to the *current frame*.

      3. `Set!` changes the *first available* binding (see I.B for the definition of "first available").

      4. `Let` = `lambda` (II.B.1) + invocation (II.A)

      5. `(define (...)  ...)` = `lambda` (II.B.1) + `define` (II.B.2)

      6. Other special forms follow their own rules (`cond`, `if`).