6.001 recitation 14 4/6/06

- environment model

Dr. Kimberle Koile

How does the environment model differ from the substitution model?
What's an environment? How do we represent it?
What's an "enclosing environment"?
What does the overall shape of environment diagrams connote about a language?

Eval

. **name:** look up name in the lowest frame of the current environment, and if find it, return the value, otherwise do the lookup in the sequence of parent frames.

. **(lambda (params) body):** create a "double bubble" with code pointer to params and body, and env pointer to the lowest frame of the current environment.

. **(define name value):** evaluate value and create or replace the binding for name in the (lowest frame of the) current environment.

. **(set! name value):** evaluate value and replace the closest binding for name in the environment frame sequence, starting with the lowest frame of the current environment

. (proc args …): evaluate proc and args, then do the apply steps

. Otherwise, follow the correct rule (if, cond, etc.).

Apply

1. Drop a new frame.
2. Link frame ptr of new frame to (lowest frame of the) environment referenced by env pointer of double bubble.
3. Bind params of double bubble in the new frame.
4. Eval the body in the new frame.

1. (define x 5)

2. (define (foo n)
   (+ x n))

3. (foo 1) =>
   _____

4. (define (bar a)
   (define x a)
   (foo 1))

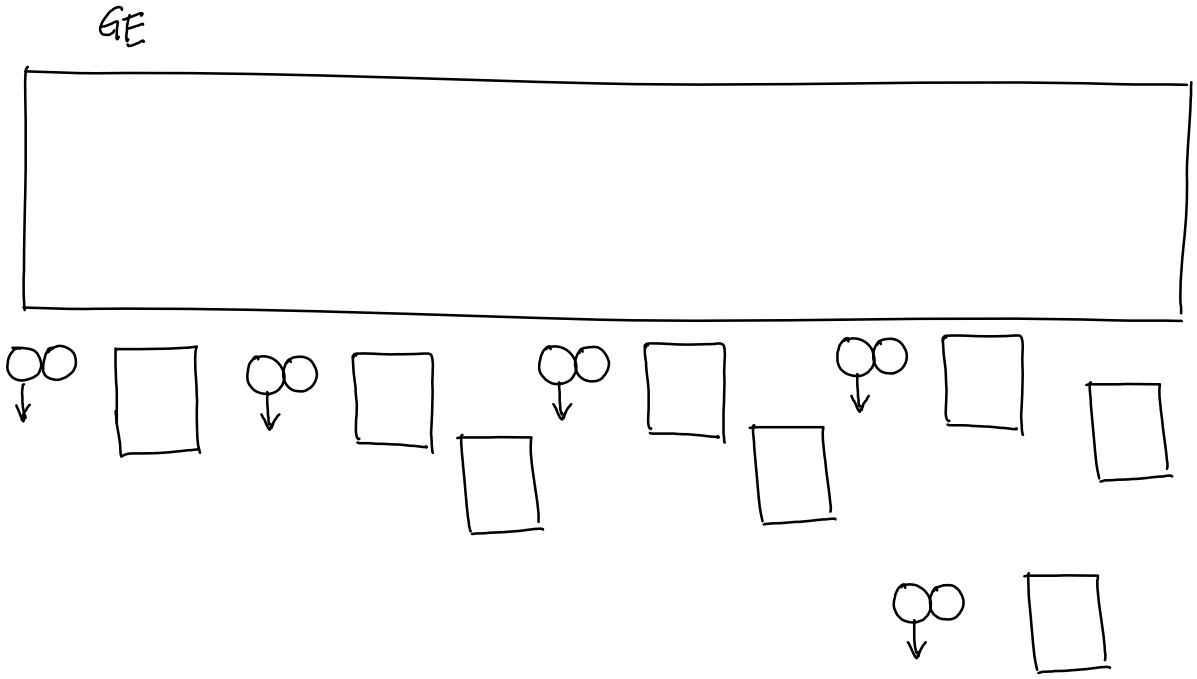
5. (bar 10)
   =>_____

6. (define (baz b)
   (set! x b)
   (foo 1))

7. (baz 20) =>_____

8. (define (blah c)
   (let ((x c))
   (foo 1)))

9. (blah 100)



Let turns into a lambda that makes a local frame: desugar the let into the lambda, then immediately apply the lambda:

e.g (let ((a 0)) (foo a)) => ((lambda (a) (foo a)) 0)| GE

Lambda becomes a procedure object in the **current** environment (i.e., the lambda is captured by the current environment). So even if applied elsewhere, free variables are looked up in the **environment of definition**, not the environment of call.

Set! works on the nearest frame in the current environment sequence that contains a binding for x; this may not be the current frame. Causes an error if a binding for x does not exist.

Define always works on the current frame only. Replaces a binding for x if it existed previously or creates a new binding for x if it did not exist previously.

Example of set! vs define:

1. (define x 0)

2. (define f
   (lambda (y)
   (define x (+ y 10)) x))
3. (define g
   (lambda (y)
   (set! x (+ y 10)) x))
4. (f 5) =>_____

   x => ____

5. (g 5) =>_____

   x => ____