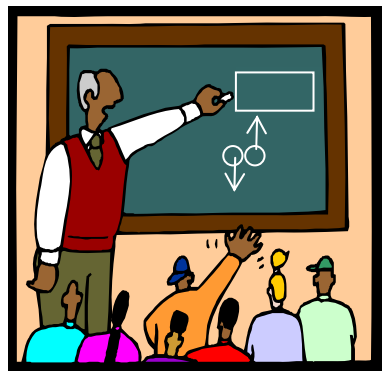
6.001 recitation 15 4/11/06

- environment diagrams (cont'd)



Dr. Kimberle Koile

environment diagram review

Eval

. **name:** look up name in the (lowest frame of the) current environment. If you find it, return the value, otherwise do the lookup in the parent frames of the current environment.

. **(lambda (params) body):** create a "double bubble" with code pointer to params and body, and env pointer to 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 chain, 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.

environment diagram example (solution)

1. (define x 5)

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

3. (foo 1) => _____

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

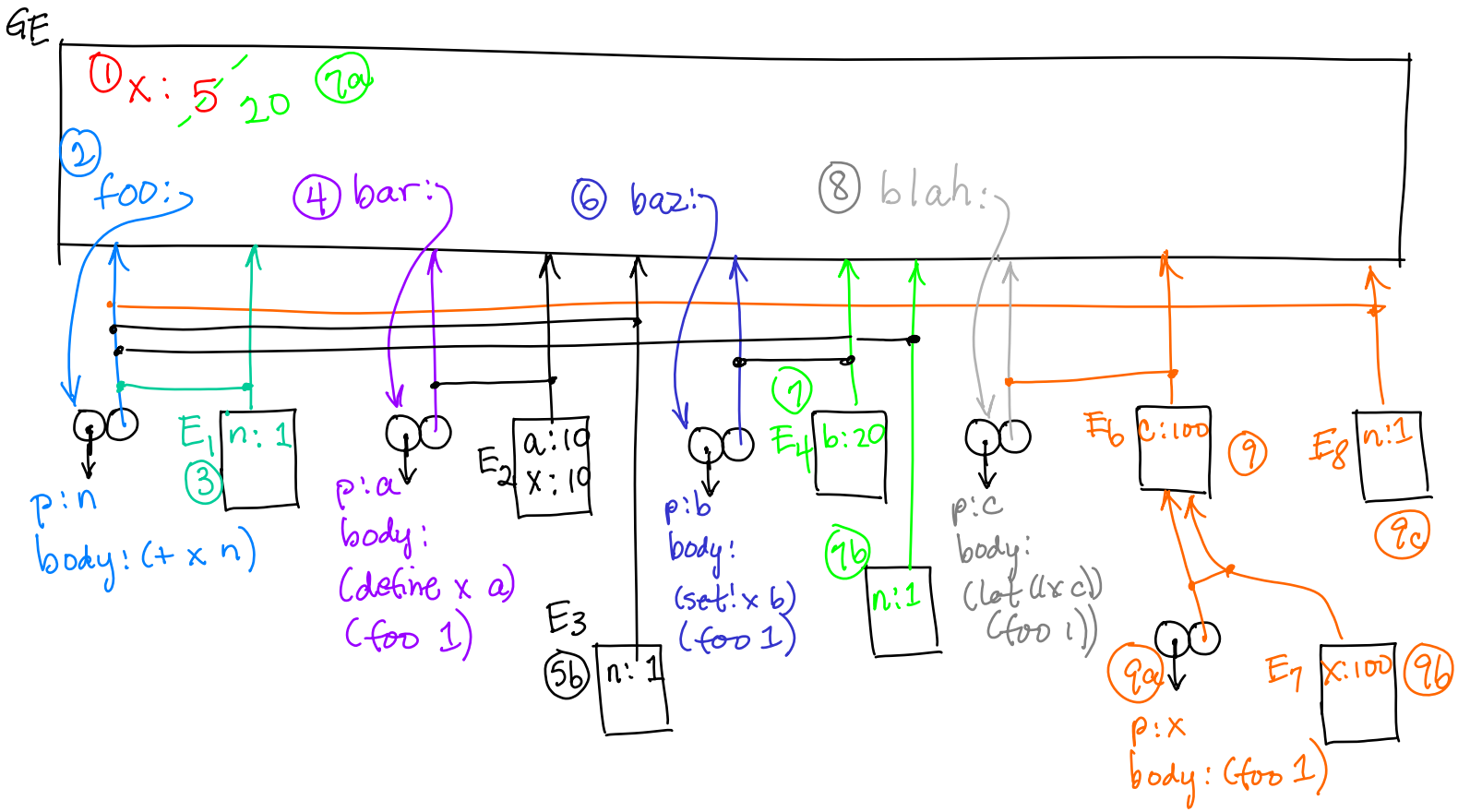
5. (bar 10) =>_____

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

7. (baz 20) =>_____

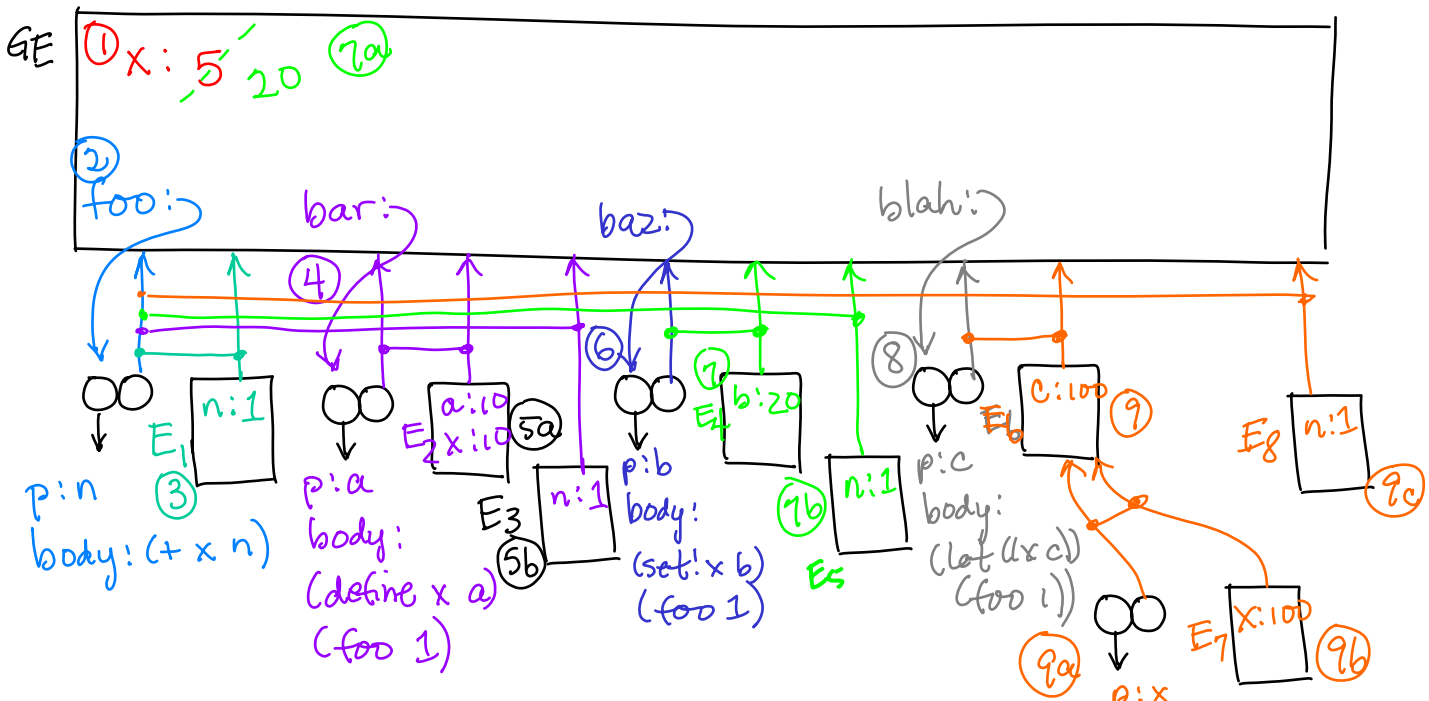
8. (define (blah c)
   (let ((x c))
   (foo 1)))

9. (blah 100)



solution (cont'd)

1. (define x 5)
2. (define (foo n)
   (+ x n))
3. (foo 1) => 6
4. (define (bar a)
   (define x a)
   (foo 1))
5. (bar 10) => 6
6. (define (baz b)
   (set! x b)
   (foo 1))
7. (baz 20) => 21
8. (define (blah c)
   (let ((x c))
   (foo 1)))
9. (blah 100) => 21



3 $(foo\ 1)|_{GE}$
$(+\ x\ n)|_{E_1}$
$(+\ 5\ 1)|_{E_1} \Rightarrow 6$

5 $(bar\ 10)|_{GE}$
5a $(define\ x\ a)|_{E_2}$
$(foo\ 1)|_{E_2}$
5b $(+\ x\ n)|_{E_3}$
$(+\ 5\ 1)|_{E_3} \Rightarrow 6$

Note: For dynamic scoping, E_3 would point to E_2, the calling frame.

7 $(baz\ 20)|_{GE}$
7a $(set!\ x\ b)|_{E_4}$
7b $(foo\ 1)|_{E_4}$
$(+\ x\ n)|_{E_5}$
$(+\ 20\ 1)|_{E_5} \Rightarrow 21$

9 $(blah\ 100)|_{GE}$
$(let\ ((x\ c))\ (foo\ 1))|_{E_6} \Rightarrow ((\lambda (x)\ (foo\ 1))\ 100)$
9a, 9b

9c $(foo\ 1)|_{E_7}$
$(+\ x\ n)|_{E_8}$
$(+\ 20\ 1)|_{E_8} \Rightarrow 21$

let, lambda, set!, define solution

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 chain that contains a binding for x; this may not be the current frame. Causes an error if a bining 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:

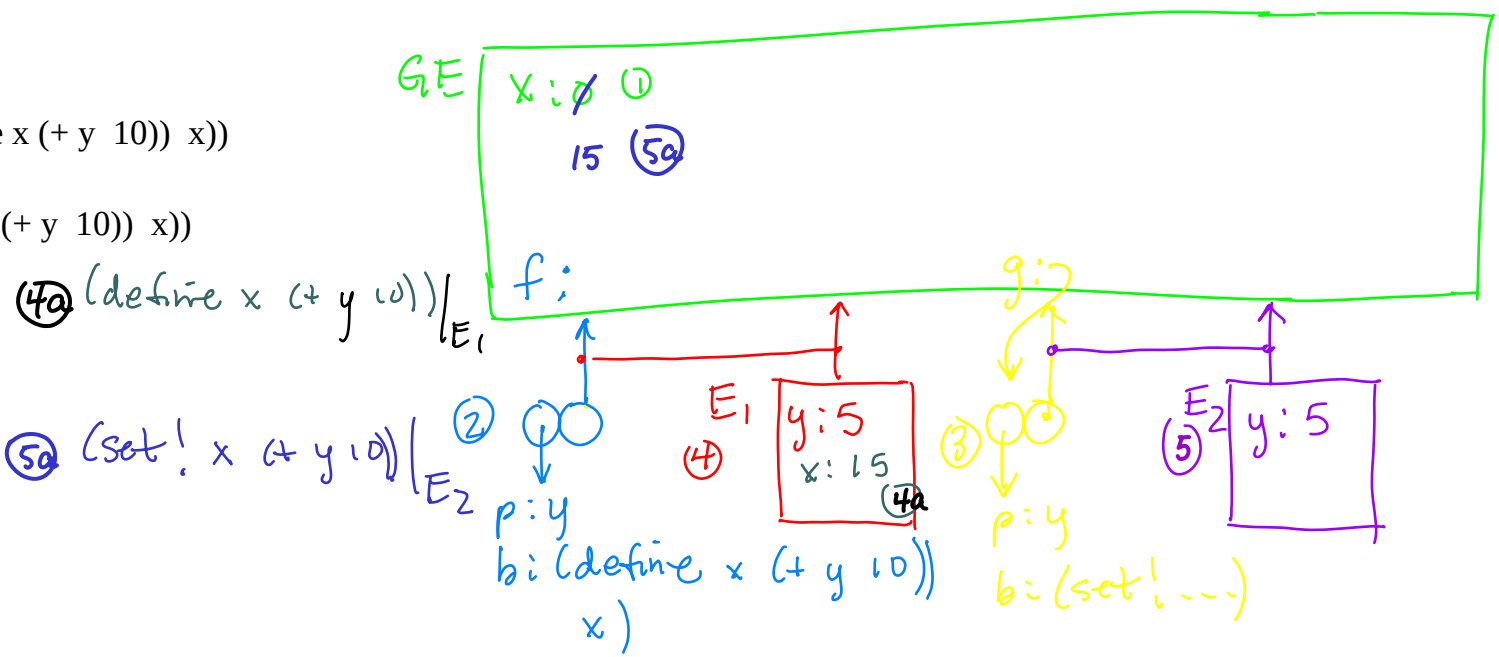
```
(1) (define x 0)
(2) (define f
      (lambda (y)  (define x (+ y  10))  x))
(3) (define g
      (lambda (y)  (set! x (+ y  10))  x))
```

(4) (f 5) => 15

x => 0

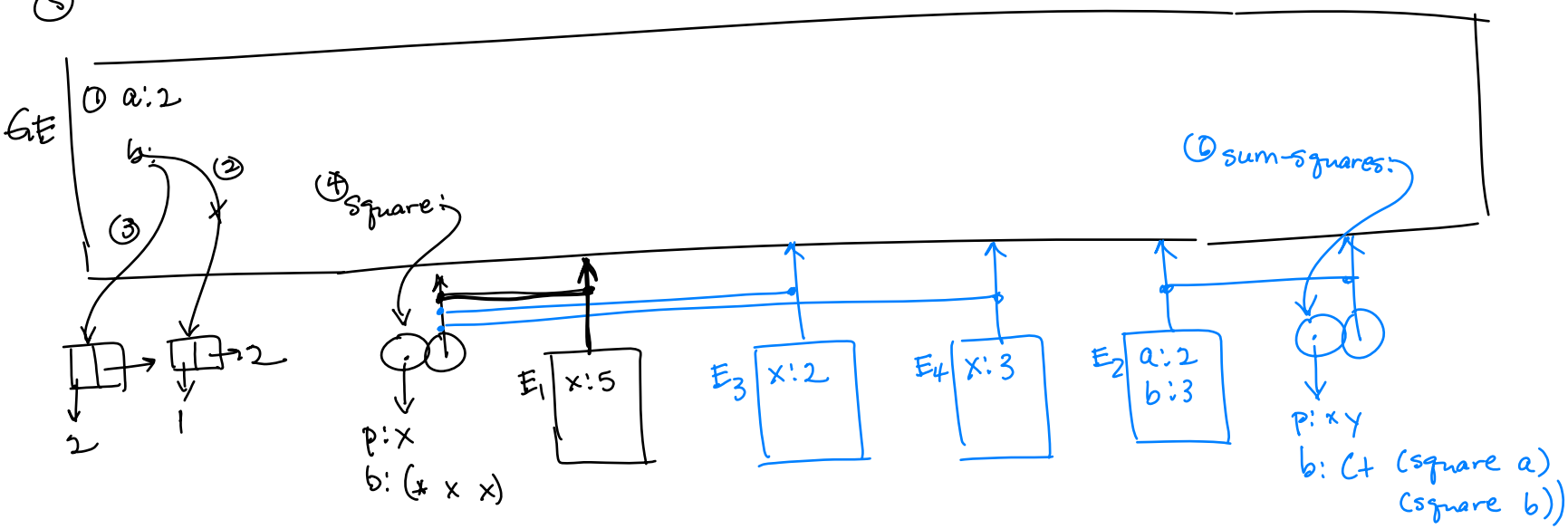
(5) (g 5) => 15

x => 15



practice problems

1. ①(define a 2)
②(define b (cons 1 a))
③(set! b (cons 2 b))
④(define square (lambda (x) (* x x)))
⑤(square 5)



Problem 1:

(square 5)|GE

(* 5 5)|E1 ⇒ 25

Problem 2: (sum-squares 2 3)|GE

(+ (square 2) (square 3))|E2

(square 2)|E2

(* 2 2)|E3 ⇒ 4

(square 3)|E2

(* 3 3)|E4 ⇒ 9

(+ 4 9)|E2 ⇒ 13

2. Add to previous diagram;
(define (sum-squares a b) (+ (square a) (square b)))
(sum-squares 2 3)

practice problems

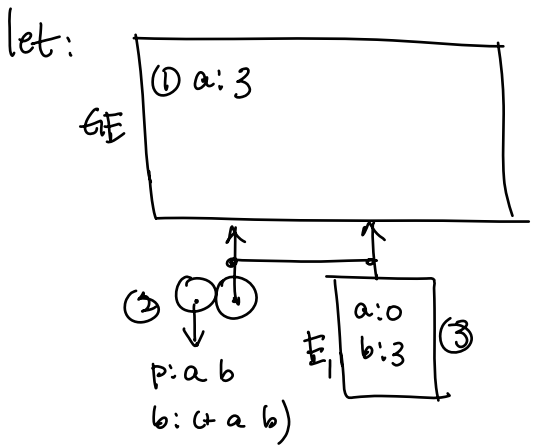
3. (define a 3)

① a. (let ((a 0) (b a)) (+ a b)) ⇒ ② desugar let ③ apply λ $((\lambda\ (a\ b)\ (+\ a\ b))\ 0\ a)|_{GE}$

b. (let* ((a 0) (b a)) (+ a b)) Recall that let* desugars into nested lets: (let ((a 0)) (let ((b a)) (+ a b)))

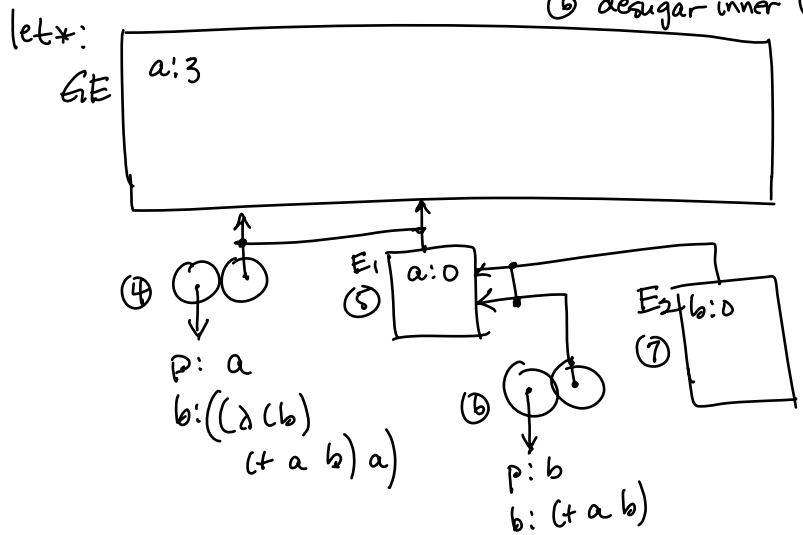
$\Rightarrow$ $((\lambda\ (a)\ ((\lambda\ (b)\ (+\ a\ b))\ a))\ 0)|_{GE}$

④ desugar outer let
⑤ apply λ
⑥ desugar inner let
⑦ apply λ



$(+\ a\ b)|_{E_1}$

$(+\ 0\ 3) \Rightarrow 3$



$((\lambda\ (a)((\lambda\ (b)\ (+\ \overset{0}{\cancel{a}}\ b))\ a)\ 0)|_{GE}$

$((\lambda\ (b)\ (+\ 0\ \overset{0}{\cancel{b}}))\ 0)|_{E_1}$

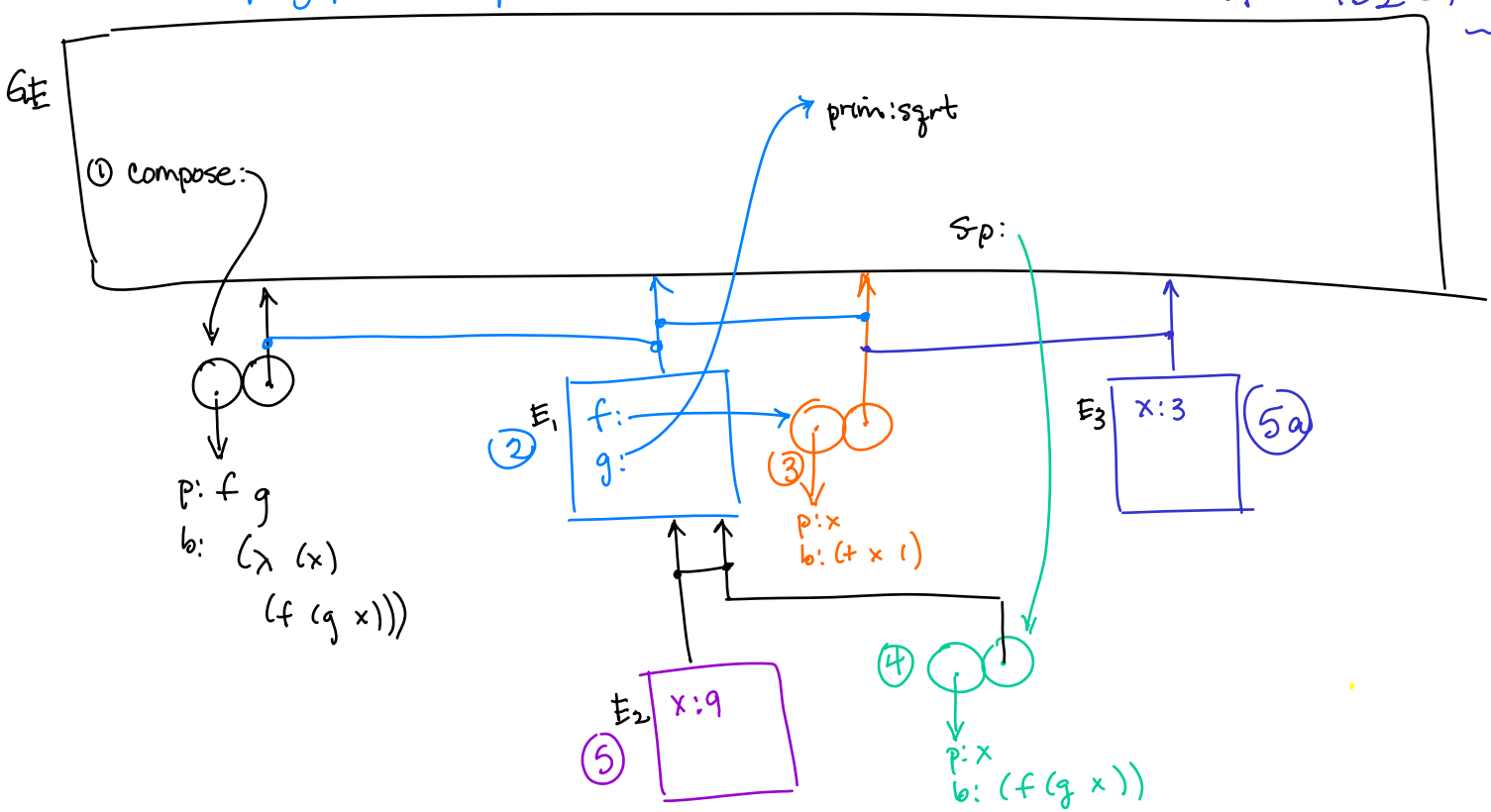
$(+\ 0\ 0)|_{E_2} \Rightarrow 0$

practice problems

4.
```
(define (compose f g)
    (lambda (x) (f (g x))))
(define s-p (compose (lambda (x) (+ x 1)) sqrt))
(s-p 9)
```

① ② apply proc compose ③ eval args ④ eval body ⑤

⑤ (s-p 9) | GE
(f (g x)) | E2
⑤a eval arg (g x) | E2, drop E3 + link to g's env
(f 3) | E2 ⇒ 4

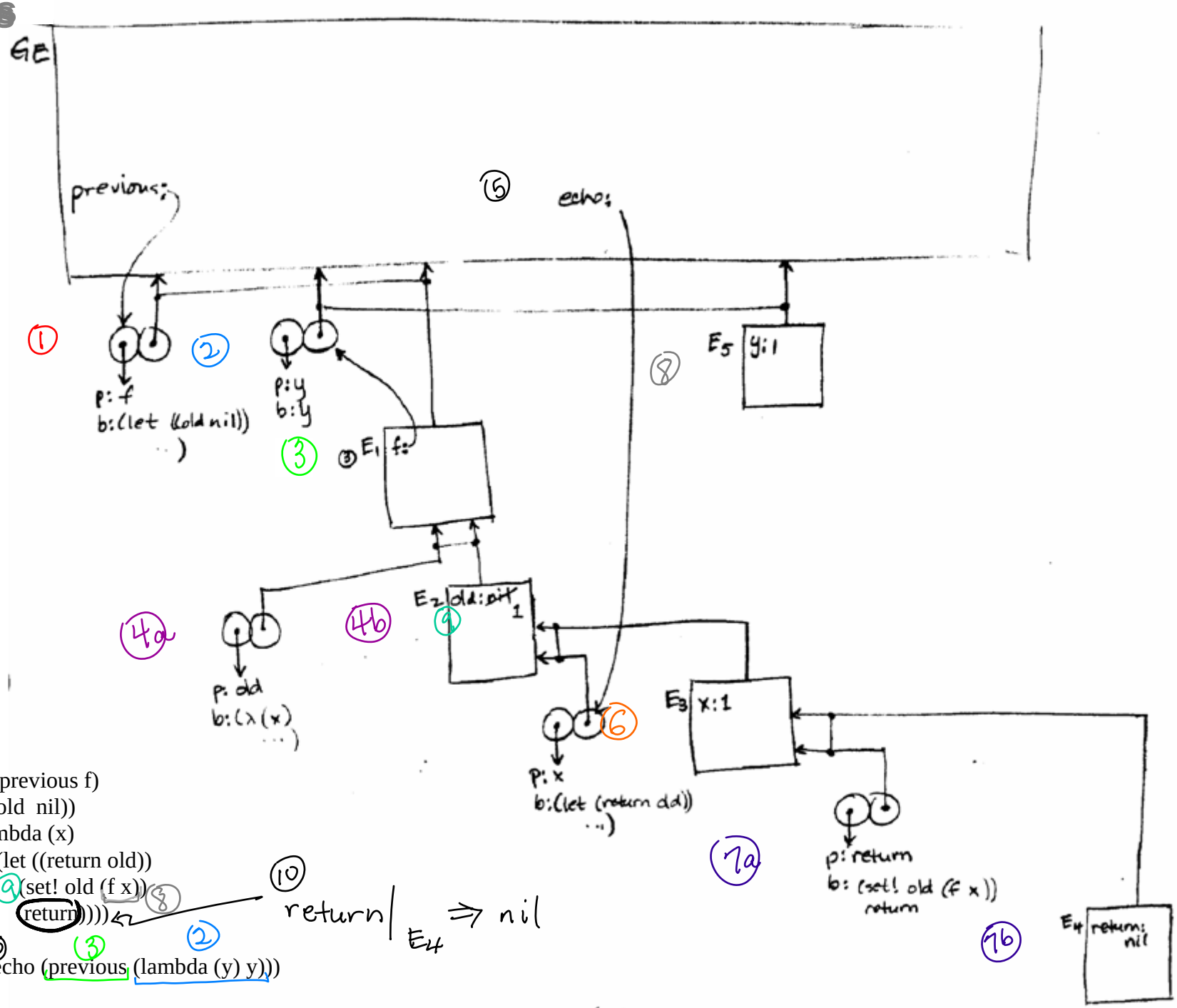


practice problems

5.
```
(define (previous  f)
        (let  ((old  nil))
             (lambda  (x)
                  (let ((return  old))
                       (set!  old (f  x))
                       return))))
(define  echo (previous  (lambda (y)  y)))
(echo 1)
```

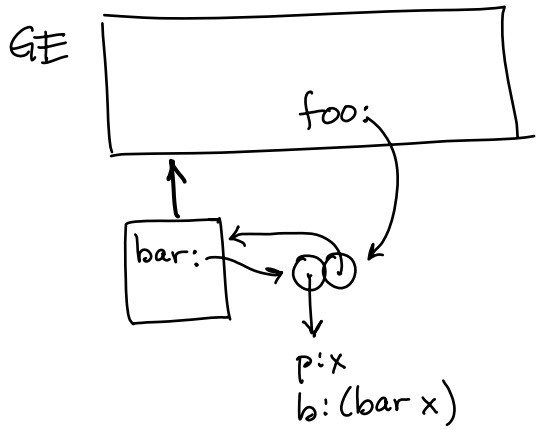
See next slide.

previous


GE
previous:
echo:
p: f
b: (let ((old nil)) ...)
p: y
b: y
E1 f:
E5 y: 1
p: dd
b: (λ (x) ...)
E2 old: pit 1
p: x
b: (let (return dd)) ...)
E3 x: 1
p: return
b: (set! old (f x))
return
E4 return: nil
return | E4 ⇒ nil
(define (previous f)
(let ((old nil))
(lambda (x)
(let ((return old))
(set! old (f x))
return))))
(define echo (previous (lambda (y) y)))
(echo 1)

practice problems

6. What code made this environment diagram? Some parts my not be drawn.




```
(define foo
  (let ((bar nil))
    (set! bar (lambda (x) (bar x)))
    bar))

OR

(define foo
  (let ()
    (define bar (lambda (x) (bar x)))
    bar))
```