Quiz 8a

1. (3 points) Recall that in Scheme, for a function call, we are not guaranteed whether the order of evaluation of the arguments is from left to right or from right to left (or some other order). For example, the expression (- (count) (count)) where count is a freshly created counter could evaluate to 1 or -1. We now look at the following expression:

(– (foo (mystery 0))

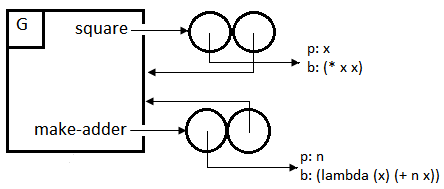
(foo (mystery 1)))

For each of the following cases, state whether we are guaranteed that the value of the expression does not depend on the order of evaluation of arguments.

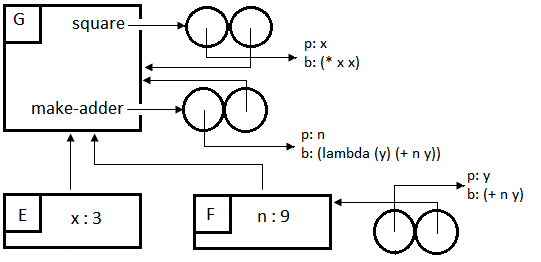
Note: “Functional, ignores its argument” means that the procedure is functional, and that nowhere in the body of the procedure is the argument used – for example, (lambda (x) 42).

|  |  |  |
| --- | --- | --- |
| Programming style of foo | Programming style of mystery | Answer (Yes/No) |
| Functional | Functional |  |
| Non-functional | Non-functional |  |
| Functional | Non-functional |  |
| Functional, ignores its argument | Non-functional |  |
| Non-functional | Functional |  |
| Non-functional | Functional, ignores its argument |  |

1. (1 + 1 + 1 + 1 points) Consider the initial environment diagram given below:



Suppose we then evaluated an expression, which evaluated without causing an error. This resulted in the following environment diagram:



* 1. How many times was a user-defined procedure called? How do you know?
  2. Which user-defined procedures were called? How many times? How do you know?
  3. Suggest what expression we typed in to get this environment diagram. (There are infinitely many possibilities; you only need to give one.)
  4. Is there any expression I can type into STk that will use the value of x defined in frame E? If yes, give such an expression. If no, explain why not. (Remember that when typing expressions into STk, we are in the global frame G.)

1. (3 points) Draw the environment diagram after all of the following expressions have been evaluated, and say what the final expression evaluates to:

> (define (make-nth n)

(lambda (lst)

(if (= n 0)

(car lst)

((make-nth (- n 1)) (cdr lst)))))

> (define second (make-nth 1))

> (second ‘(she loves you))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_