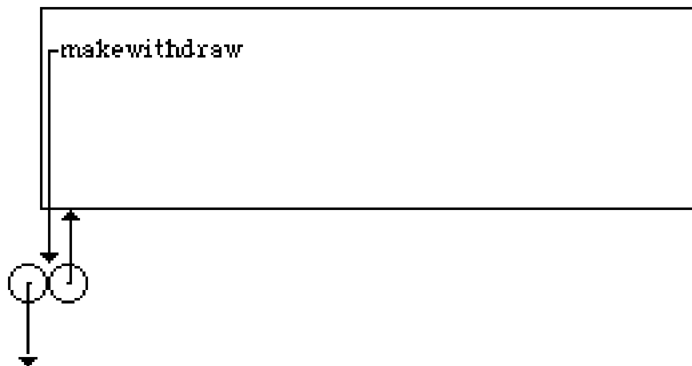
Ex. 3.10 page 248.

Throughout this analysis, we will refer to rules 1 and 2 on page 238 as 1 and 2. We will refer to the first bullet item on page 240 as 3 and to the second bullet item on page 240 as 4. In order to apply these rules, we will write the definition and the let in their lambda forms. With this, we have:

```
(define makewithdraw
  (lambda (initamt)
    ((lambda (balance)
       (lambda (amount)
         (if (>= balance amount)
           (begin
             (set! balance (- balance amount))
             balance)
           "insufficient funds")))
     initamt)))
```

When we execute this definition at the top level, we must evaluate the (lambda (initamt) ... in the global environment and bind it to the identifier makewithdraw. By rule 4, we get



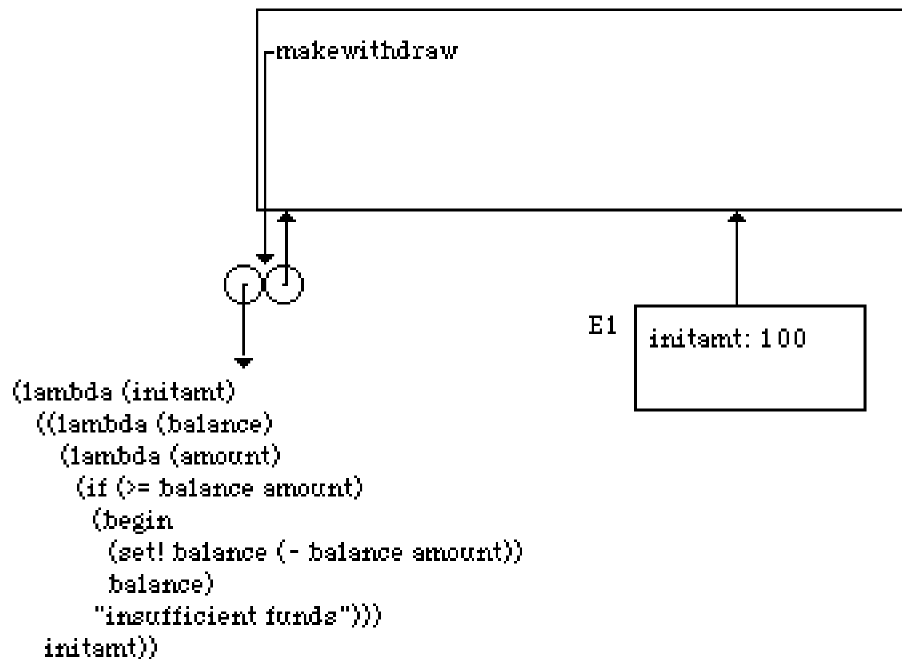

```
(lambda (initamt)
    ((lambda (balance)
       (lambda (amount)
         (if (>= balance amount)
           (begin
             (set! balance (- balance amount))
             balance)
           "insufficient funds")))
     initamt))
```

Now when we evaluate (define w1 (makewithdraw 100)), we must evaluate the combination (makewithdraw 100) in the global environment and bind the result returned to w1. To evaluate (makewithdraw 100) , we use rule 1 first to evaluate its subexpressions. makewithdraw evaluates to the procedure object depicted above; 100 evaluates to 100. Now rule 2 says to apply the procedure object to 100. That is, we

must evaluate:

```
( (lambda (initamt)
    ((lambda (balance)
      (lambda (amount)
        (if (>= balance amount)
          (begin
            (set! balance (- balance amount))
            balance)
          "insufficient funds")))
     initamt))
 100)
```

in the global environment because that is where the procedure objects environment pointer points. In order to do this we follow rule 3 and create an environment E1 in which we bind initamt to 100 and then evaluate the body ((lambda (balance) ...) in it. So, we have:

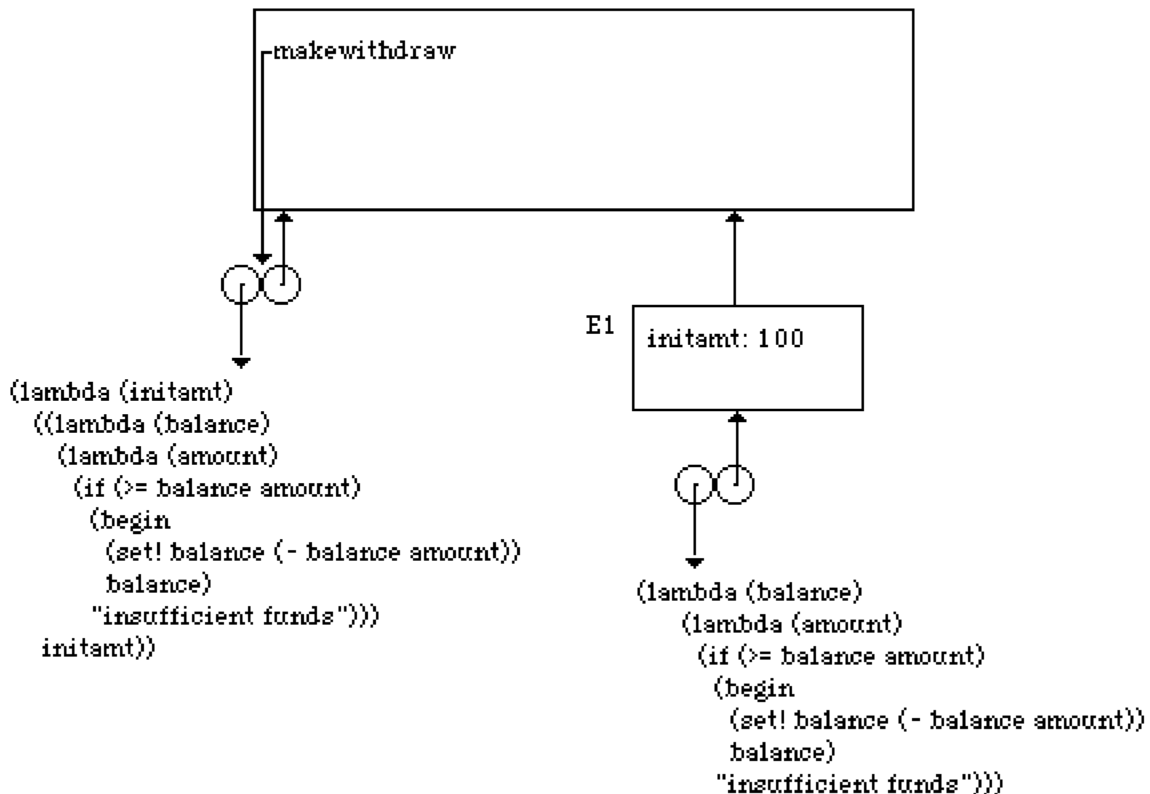


And we must evaluate

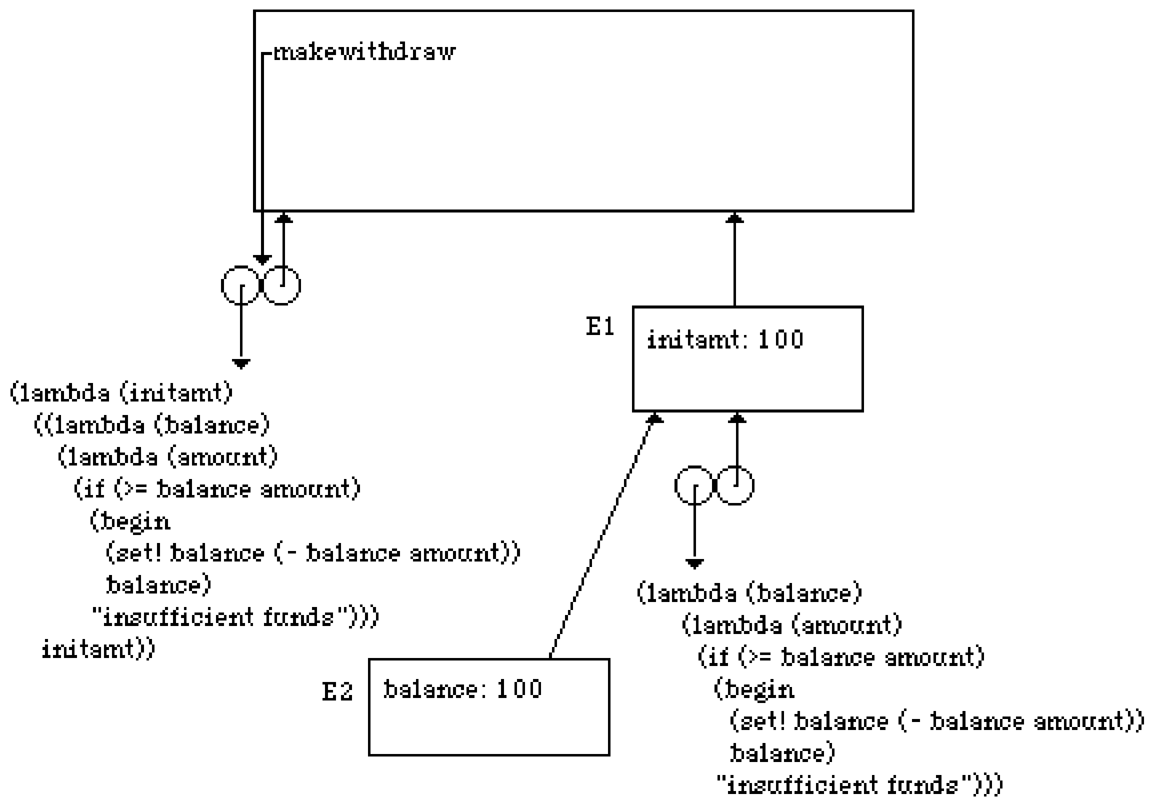
```
((lambda (balance)
      (lambda (amount)
        (if (>= balance amount)
          (begin
            (set! balance (- balance amount))
            balance)
          "insufficient funds")))
    initamt)
```

in the environment E1. So we apply rule 1 and evaluate (lambda (balance) ...) and

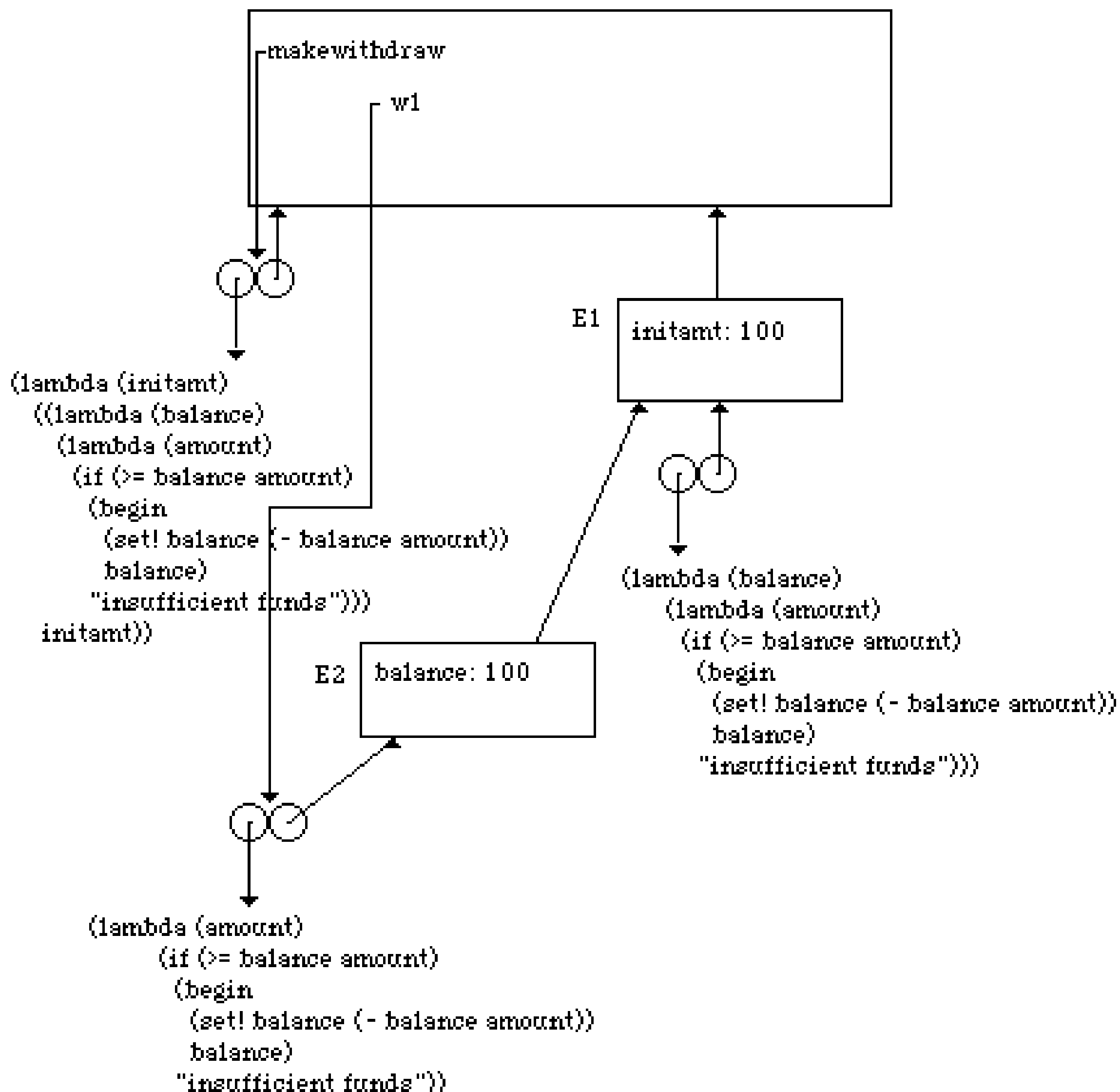
initamt in E1. To evaluate (lambda (balance) ...), we use rule 4 and get a new procedure object shown below whose environment pointer points to E1. Evaluating initamt in E1 returns 100.



Now by rule 2 we must apply (lambda (balance) ...) to 100 in environment E1. Rule 3 tells us how. Create a new environment E2 in the context of E1 and bind balance to 100 there. Then evaluate (lambda (amount) ...) in E2. So we must evaluate (lambda (amount) ...) in the environment E2 shown below.



To evaluate (lambda (amount) ...) in E2, we use rule 4 to create a new procedure object shown below that gets bound to w1, as shown below:



We have finished the evaluation of (define w1 (makewithdraw 100).

Now let us evaluate (w1 50).
Rule 1 says to evaluate the subexpressions of (w1 50). w1 evaluates to the procedure object (pointed to by w1 in the diagram above)

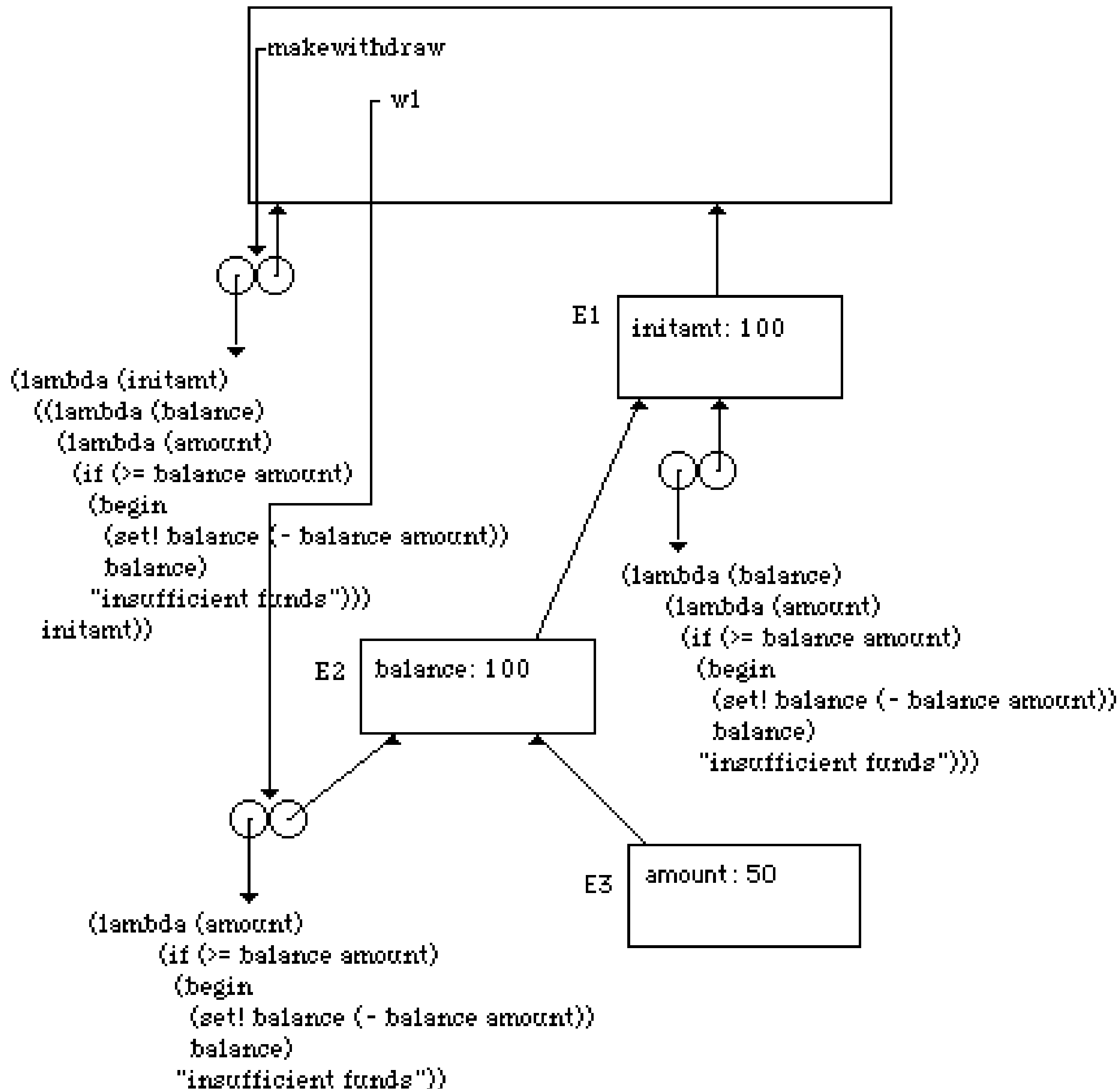
```
(lambda (amount)
        (if (>= balance amount)
          (begin
            (set! balance (- balance amount))
            balance)
          "insufficient funds"))
```

whose environment is E2. 50 evaluates to 50. So by rule 2, we must apply (lambda (amount) ...) to 50 in E2. Following rule 3, we must construct an

environment, E3, in which we bind 50 to the parameter amount and then evaluate

```
(if (>= balance amount)
        (begin
          (set! balance (- balance amount))
          balance)
        "insufficient funds")
```

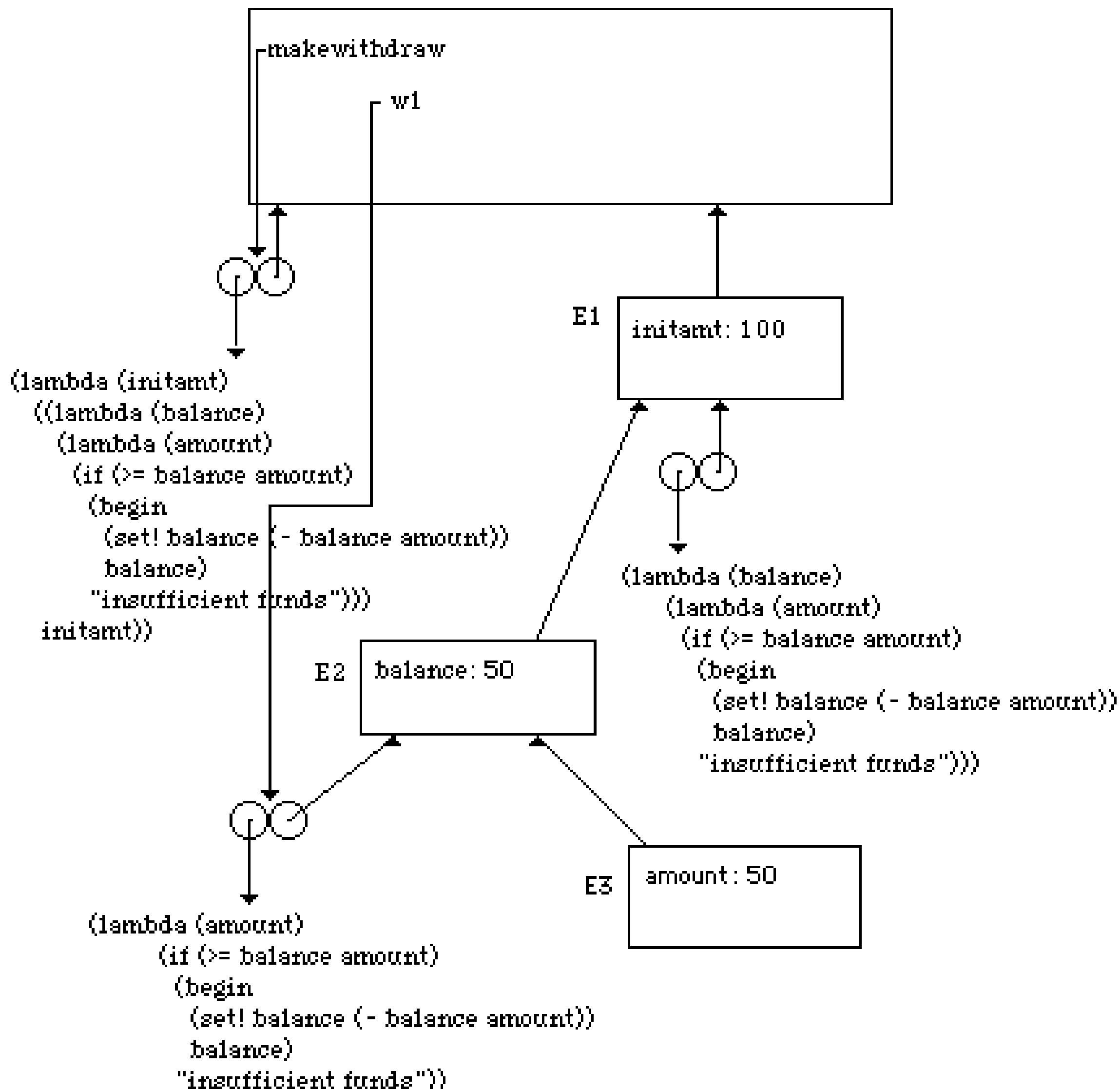
in E3. So we construct:



Now we evaluate:

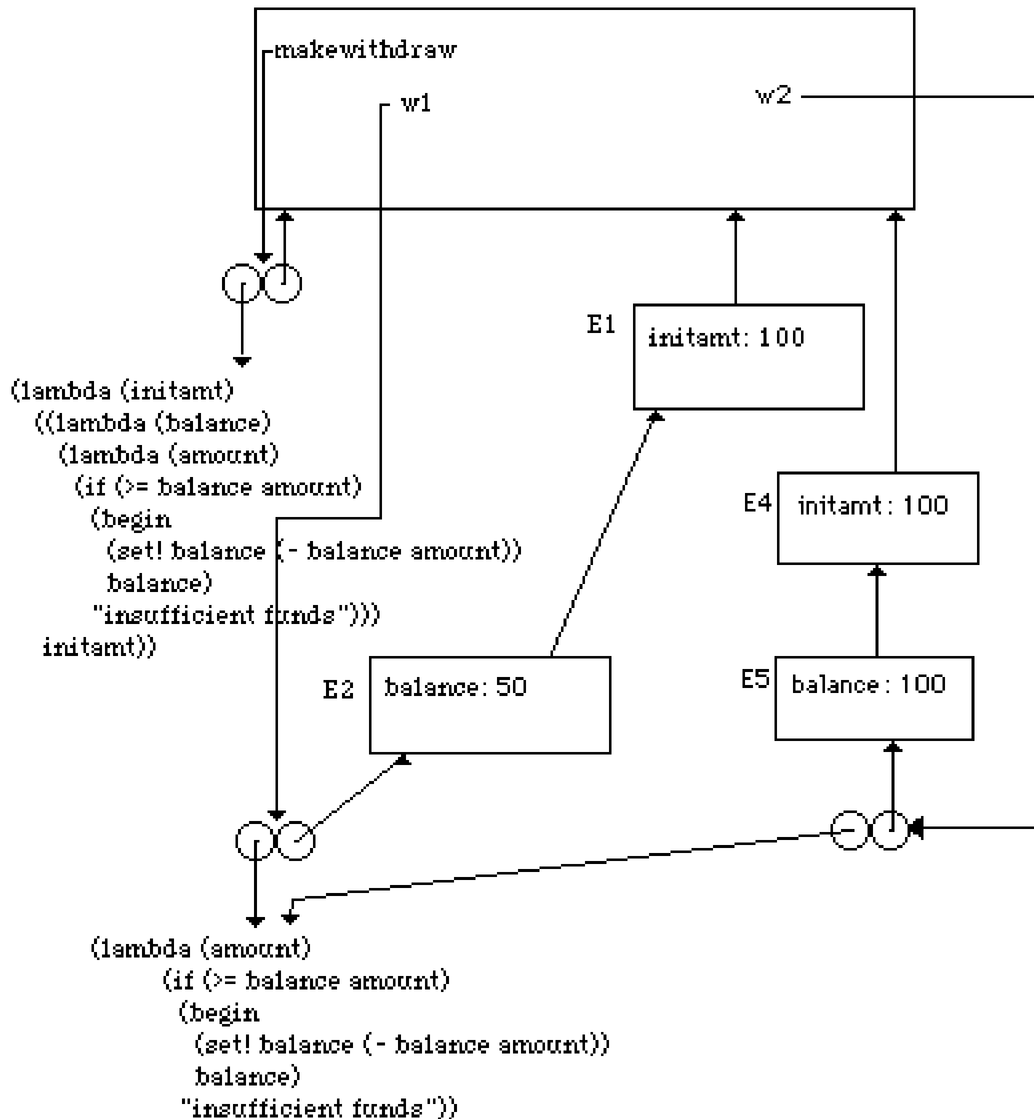
```
(if (>= balance amount)
        (begin
          (set! balance (- balance amount))
          balance)
        "insufficient funds")
```

in E3. In this environment and its containing environments, balance is found to have value 100 in E2 and amount has value 50 in E3. So the if condition is true. This means that the set! is executed which changes the value of balance in E2 to 50. Then the values of balance in E2 (now 50) is returned. The environment diagram upon completion of (w1 50) looks like:



Note that E3 is no longer reachable from the global environment. Also the procedure object whose code begins (lambda (balance)... is not part of an evaluation in progress nor is it reachable from the golbal environment. This means that this procedure object and E3 are eligable for garbage collection. Since I need the space, I

will no longer show them as part of my diagram. We now want to evaluate: (define w2 (makewithdraw 100)). As above when we defined w1, we will follow the same set of rules to create environments E4 and E5 and a procedure object that points to E5 as illustrated below:



I have intentionally shown the procedure objects for w1 and w2 sharing exactly the same code. This is not required. It is up to the particular implementation. However, it may be done. What is important to see is that even if they share exactly the same code, w1 and w2 are different procedure objects because they are evaluated in different environments. In fact, evaluating (w1 25) and (w2 25) would give different results at this point.

The problem asks us to show that the version of makewithdraw in ex 3.10 will have

the same behavior as the version on the previous pages of the text. First note that the only difference is that the ex3.10 version puts an environment between the environment containing balance and the global environment. This inbetween environment contains the binding for initamt. initamt was used to set the initial values of the balances, but since the bodies of w1 and w2 do **not** use initamt in **either** case, this inbetween environment can not make any difference to any calculation. (If the bodies had used initamt, then the two makewithdraws could create semantically different procedures.)