Exercise 3.11.

In section 3.2.3 we saw how the environment model described the behavior of procedures with local state. Now we have seen how internal definitions work. A typical message-passing procedure contain both of these aspects. Consider the bank account peocedure of section 3.1.1:

```
(define (make-account balance)
  (define (withdraw amount)
    (if (>= balance amount)
        (begin (set! balance (- balance amount))
               balance)
        "Insufficient funds"))
  (define (deposit amount)
    (set! balance (+ balance amount))
    balance)
  (define (dispatch m)
    (cond ((eq? m 'withdraw) withdraw)
          ((eq? m 'deposit) deposit)
          (else (error "Uknown request -- MAKE-ACCOUNT"
                       m))))
  dispatch)
```

Show the environment structure generated by the sequence of interactions

```
(define acc (make-account 50))

((acc 'deposit) 40)
90

((acc 'withdraw) 60)
30
```

Where is the local state for `acc` kept? Suppose we define another account

```
(define acc2 (make-account 100))
```

How are the local states for the two accounts kept distinct? Which parts of the environment structure are shared between `acc` and `acc2`?

Answer.

Figure 1 shows the point in creating an account `acc` using the expression:

```
(define acc (make-account 50))
```

where the symbol `acc` has been bound to the internal procedure `dispatch` in environment E1. Observe the structure of environment. `Make-account` is a symbol in the global environment that is bound to a procedure object whose associateed environment is the global environment. When `make-account` was called, a new environment E1 was formed, subordinate to the global environment, in which the parameter `balance` is bound to 50. The body of `make-account` was then evaluate in E1. Since the first expression in the body of `make-account` is

```
(define (withdraw amount)
    (if (>= balance amount)
        (begin (set! balance (- balance amount))
               balance)
        "Insufficient funds"))
```

evaluating this expression defined the procedure `withdraw` in the environment E1. Similarly, `deposit` and `dispatch` were defined as procedures in E1. Additionally, defining the symbol `acc` created a binding in the global environment and associated it with the internal procedure `dispatch` in E1.

After the local procedures were defined, the expression

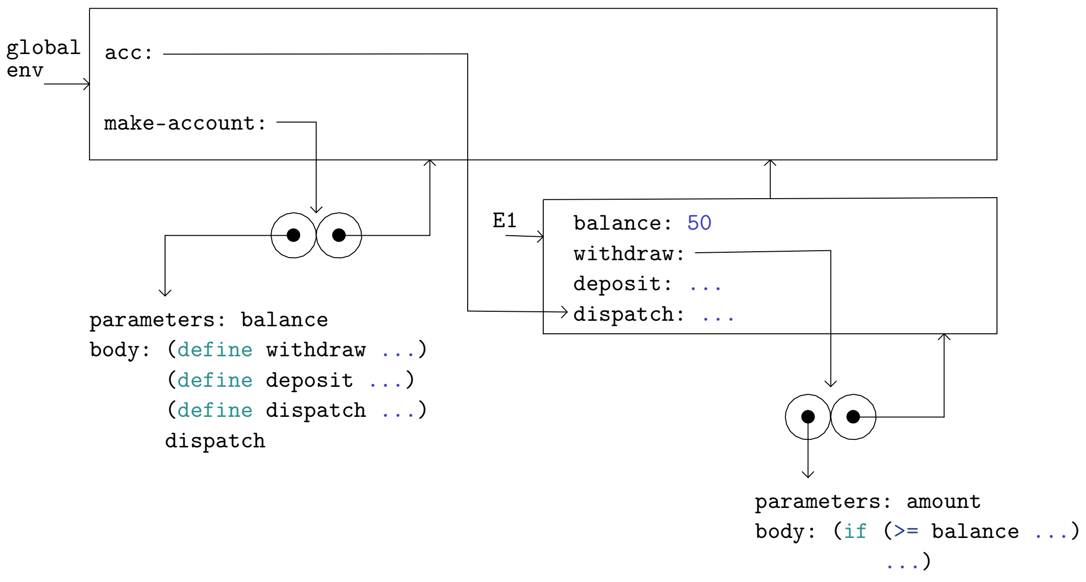

Figure 1. Environments created in evaluating `(define acc (make-account 50))`.

```
  ((acc 'deposit) 40)
```

was evaluated, still in environment E1. We first evaluated the operator subexpression. This established a new environment E2 in which `m`, the parameter of `dispatch` was bound to `'deposit`. This caused `dispatch` in turn returned the procedure `deposit` which was later applied to `40` by seting up another environment E3. Figure 2 shows the point in evaluating `((acc 'deposit) 40)`.

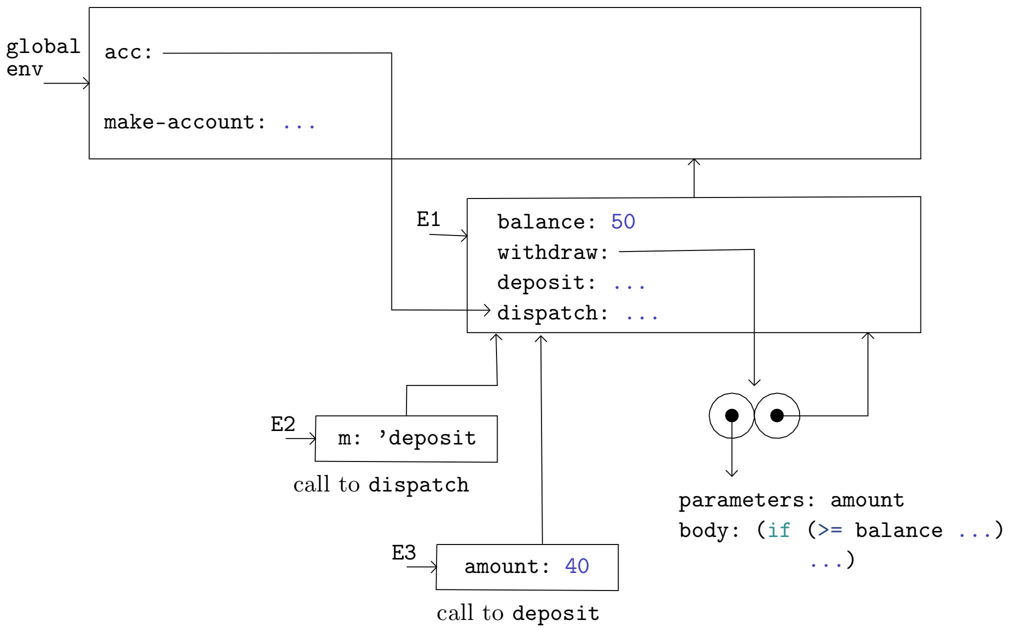

Figure 2. Environments in evaluating `((acc 'deposit) 40)`.

When the `set!` is executed, the binding of `balance` in E1 is changed. At the completion of the call to `(acc 'deposit)`, `balance` is `90`, and the frame that contains `balance` is still pointed to by the procedure object `acc`. The frames that bound `m` and `amount` is no longer exist, since the procedure call that constructed it has terminated. The next time `acc` is called, this will build new frames that binds `m` and `amount` whose enclosing environment is E1. We see that E1 serves as the "place" that holds the local state variable for the procedure object `acc`. Figure 3 shows the situation after the call to `(acc 'deposit)`.

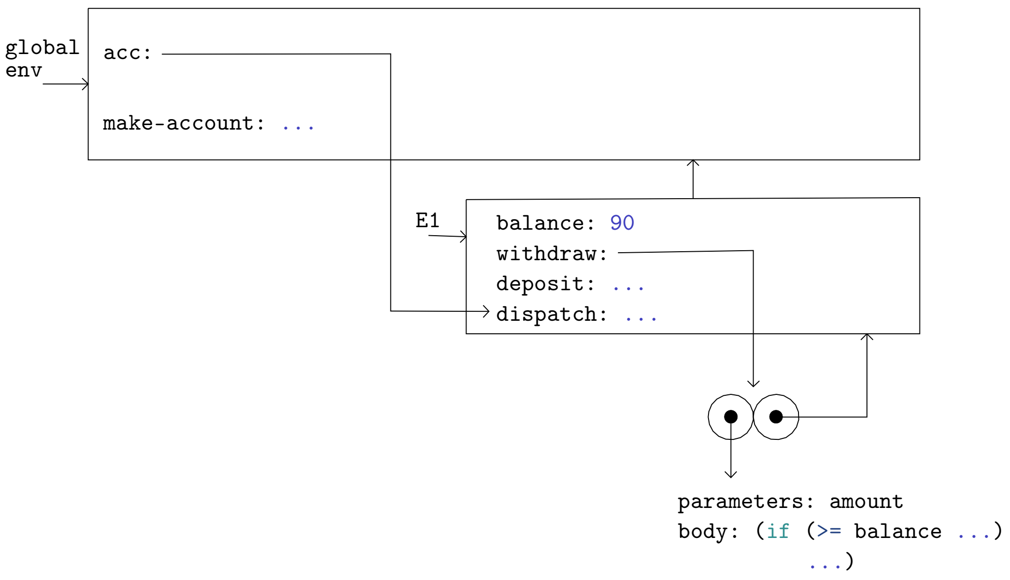

Figure 3. Environments after the call to `(acc 'deposit)`.

Similarly, we can depict the environment structure at the completion of evaluating

```
((acc 'withdraw) 60)
30
```

As shown in figure 4

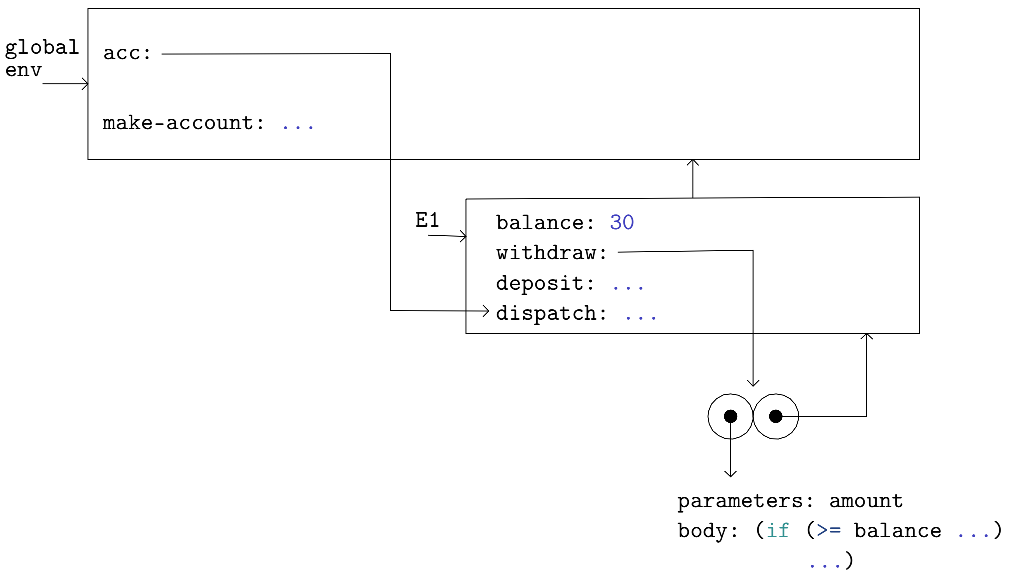

Figure 4. Environment after the evaluation of `((acc 'withdraw) 60)`.

When we create a second call “account” object by making another call to `make-account`:

```
(define acc2 (make-account 100))
```

This produces the environment structure of figure 5, which shows that `acc2` is another procedure object, that is, a name associates the procedure object `dispatch`. The environment E2 for `acc2` was created by the call to `make-account`. It contains a frame with its own local binding for `balance`. We can also see that the internal definition of the environment structure are shared between `acc` and `acc2`.

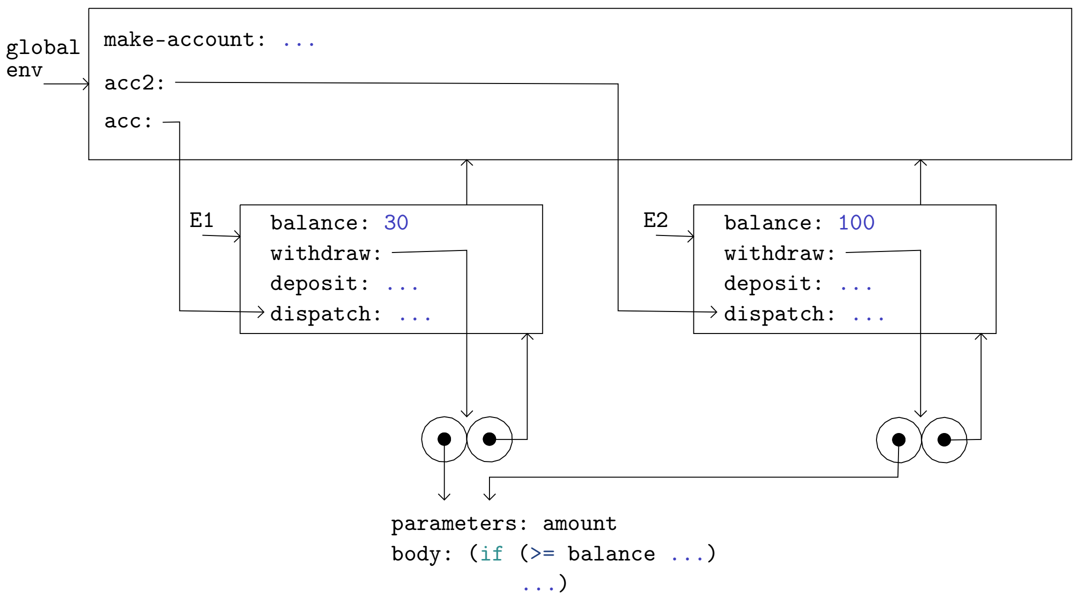

Figure 5. Using `(define acc2 (make-account 100))` to create a second account.