Exercise 3.20.

Draw environment diagrams to illustrate the evaluation of the sequence of expressions

```
(define x (cons 1 2))
(define z (cons x x))
(set-car! (cdr z) 17)
(car x)
17
```

using the procedure implementation of pairs given above. (Compare exercise 3.11.)

Answer.

Figure 1 shows the environment structure in the evaluation of the following definitions:

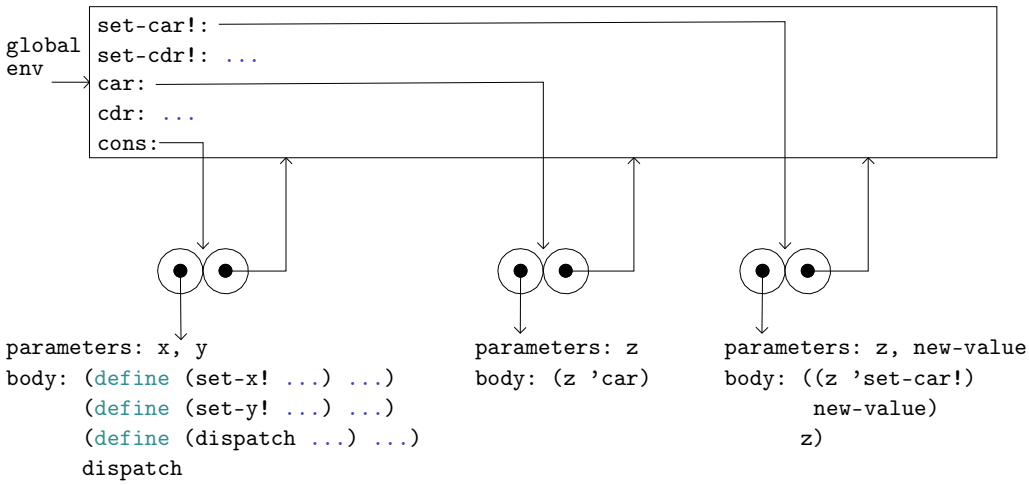

Figure 1. Environment structure produced by evaluting `(define (cons x y) ...)` etc. in the global environment.

```
(define (cons x y)
  (define (set-x! v) (set! x v))
  (define (set-y! v) (set! y v))
  (define (dispatch m)
    (cond ((eq? m 'car) x)
          ((eq? m 'cdr) y)
          ((eq? m 'set-car!) set-x!)
          ((eq? m 'set-cdr!) set-y!)
          (else
            (error "Undifined operations -- CONS" m))))
  dispatch)

(define (car z) (z 'car))

(define (cdr z) (z 'cdr))

(define (set-car! z new-value)
  ((z 'set-car!) new-value)
  z)

(define (set-cdr! z new-value)
  ((z 'set-cdr!) new-value)
  z)
```

As we know, evaluting the expression:

```
(define x (cons 1 2))
```

would set up a new environment, namely E1, subordinate to the global environment, in which the formal parameters of procedure object `cons`: `x` and `y` were bound to `1` and `2` respectively. The body of `cons` was then evaluated in E1. Since the first expression in the body of `cons` is

```
(define (set-x! v) (set! x v))
```

Evaluating this expression defined the procedure object `set-x!` in the environment E1. Similarly, `set-y!` and `dispatch` were defined as procedures in E1. Besides, defining the symbol `x` created a binding in the global environment and associated it with the internal procedure `dispatch` in E1. This process can be illustrated in a more intuitive way, as figure 2 did.

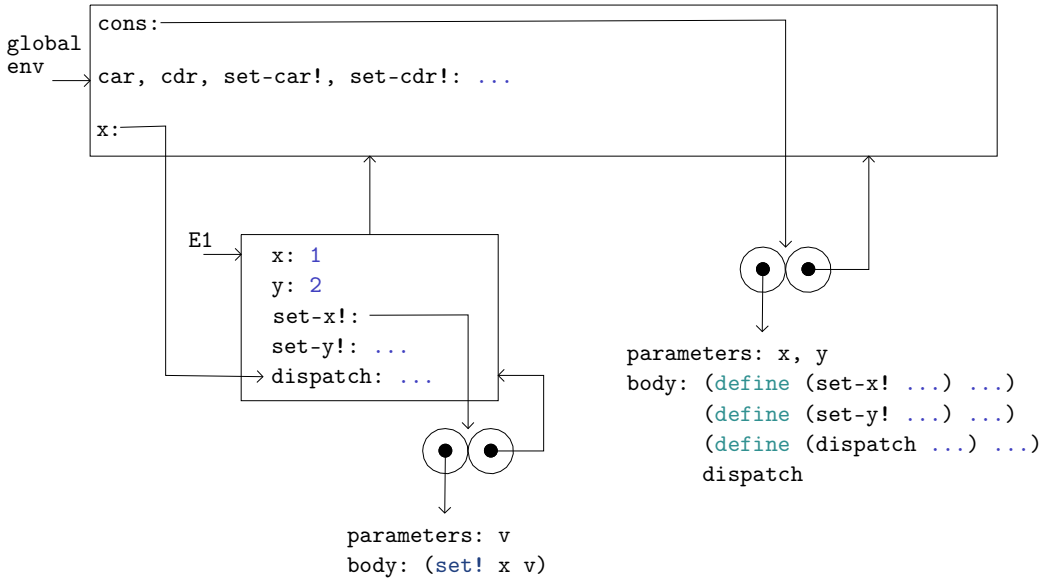

Figure 2. Environments created in evaluating `(define x (cons 1 2))`

Then, the interpreter proceeded to evaluate the subsequent expression

```
(define z (cons x x))
```

This generated almost the same process as the former one and constructs another environment, namely E2. Notice that both of `x` and `y` in E2 are bound to the value of `x` in the global environment, which is associated to the internal procedure `dispatch` of E1. Figure 3 shows the environments established during this evaluation.

When the interpreter tried to evaluate the expression

```
(set-car! (cdr z) 17)
```

It first evaluated the operator subexpression. This established a new environment E3, subordinate to the global environment, in which `z` and `new-value`, the formal parameters of `set-car!` were bound to the value of the operand subexpression, that is, the value of `(cdr z)` and `17` respectively.

To obtain the value of `(cdr z)`, the interpreter then set up a new environment E4, subordinate to the global environment, in which `z`, the formal parameter of the procedure object `cdr`, was bound to another `z` which has been settled in the global environment and associated to the internal procedure object `dispatch` in E2.

This in effect set up another environment subordinate to E2, namely E5 in which, `m` the formal parameter of `dispatch` was bound to `'cdr` and evaluated the body of `dispatch`. All these trivial steps can be illustrated in a more intuitive way, as figure 4 did.

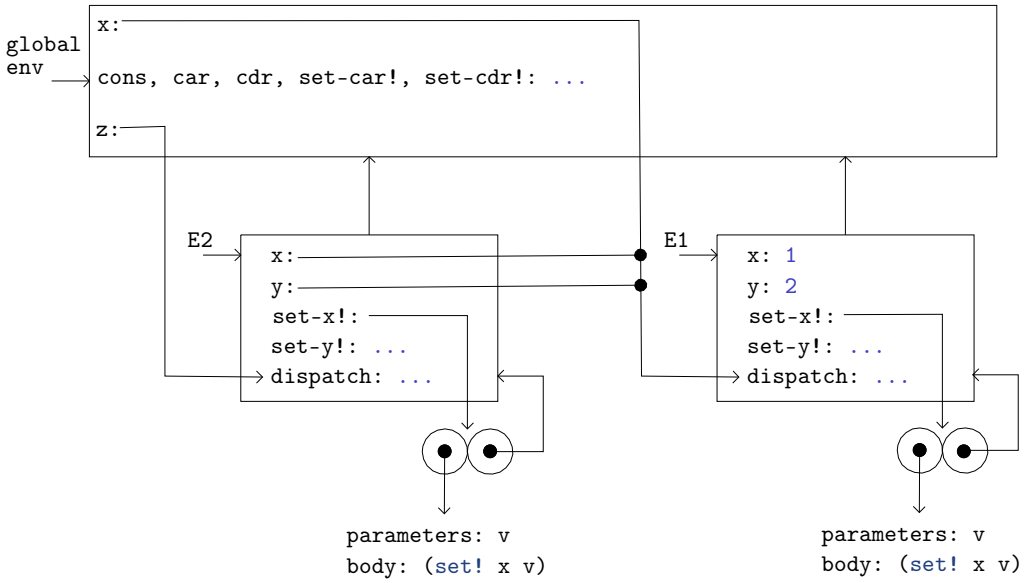

Figure 3. Environments established during the evaluation of `(define z (cons x x))`

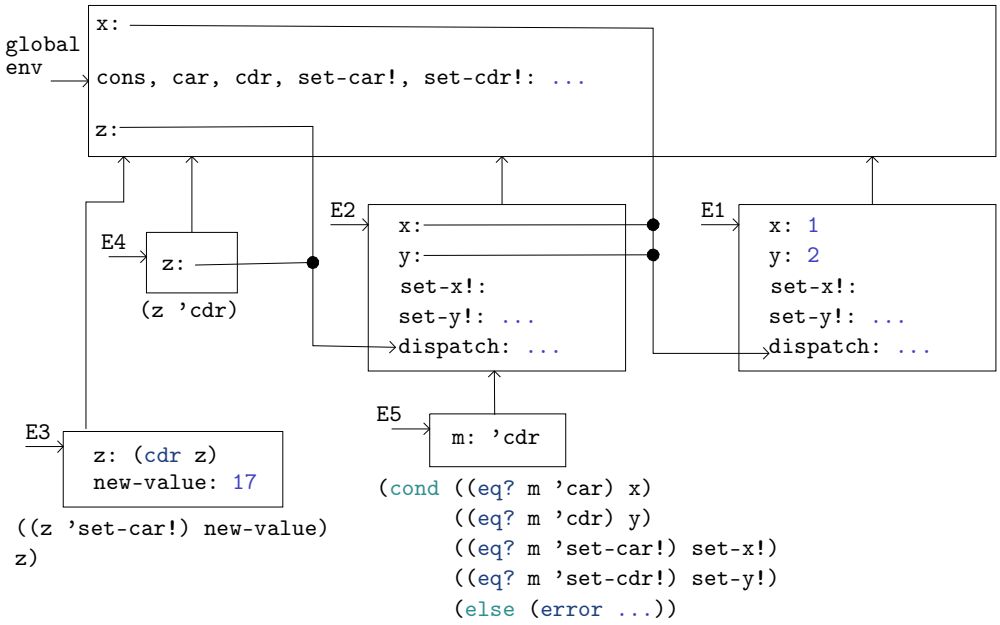

Figure 4. Environments in the evaluation of `(cdr z)` in terms of E3.

By doing these, the interpreter gained the value of `(cdr z)`, one of the operand subexpressions in `(set-car! (car z) 17)`, that is, a binding associated to the internal procedure `dispatch` of E1. Figure 5 shows the result of the evaluation of `(cdr z)` in terms of E3.

With all the value of operand subexpression in `(set-car! (cdr z) 17)` obtained. The interpreter still have to get the value of the operator subexpression, which is the procedure object `set-car!` in E3. So it set up a new environment E6, subordinated to E3 in which `m`, the formal parameter of `dispatch`, was bound to `'set-car!` and evaluated the body of `dispatch`. Hence, this revealed another procedure object `set-x!`. Thus, another environment E7 had to be constructed, in which `v`, the formal parameter of `set-x!` was bound to the value of `new-value`, which is 17. At this point, the interpreter set the value of `x` in `E1` to 17. Figure 6 shows the environments generated in the process of evaluating `(set-car! (cdr z) 17)` in a quite clear way.

When the evaluation of the expression `(set-car! (cdr z) 17)` terminated, environments other than E1 and E2 disappeared, together with `x` in E1 been set to 17. Figure 7 shows the situation after the evaluation of `(set-car! (cdr z) 17)`.

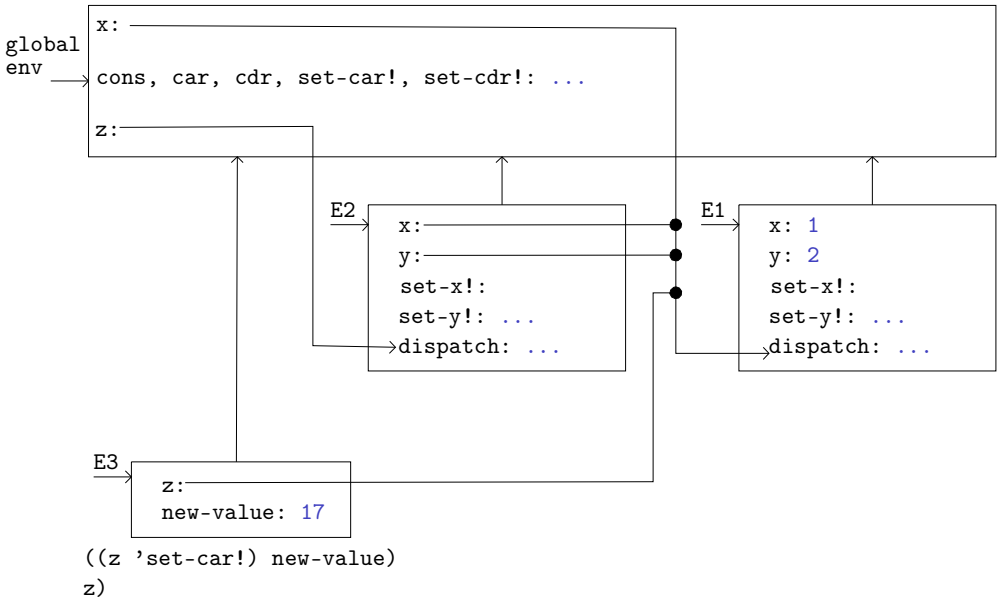

Figure 5. Result of the evaluation of `(cdr z)` in terms of E3.

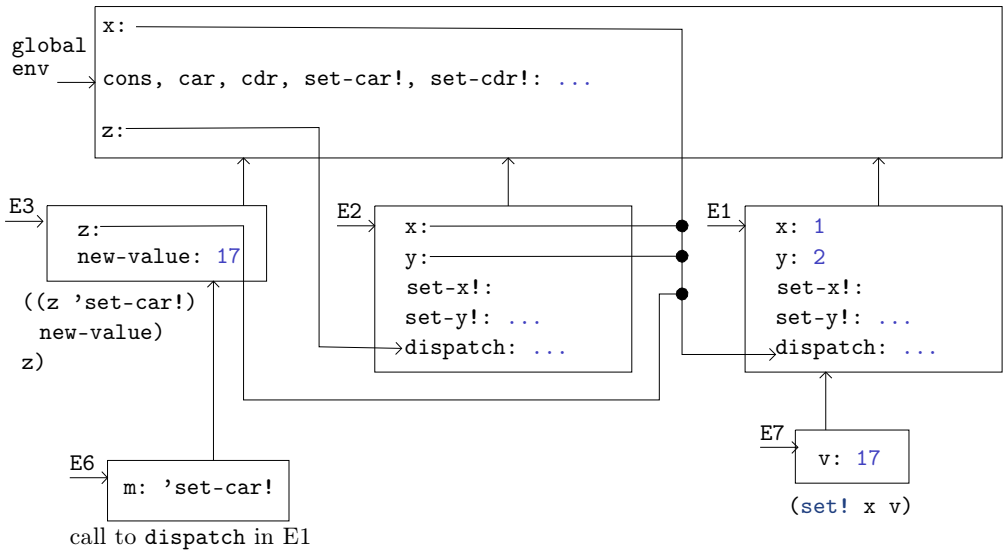

Figure 6. Environments generated in the process of evaluating `(set-car! (cdr z) 17)`.

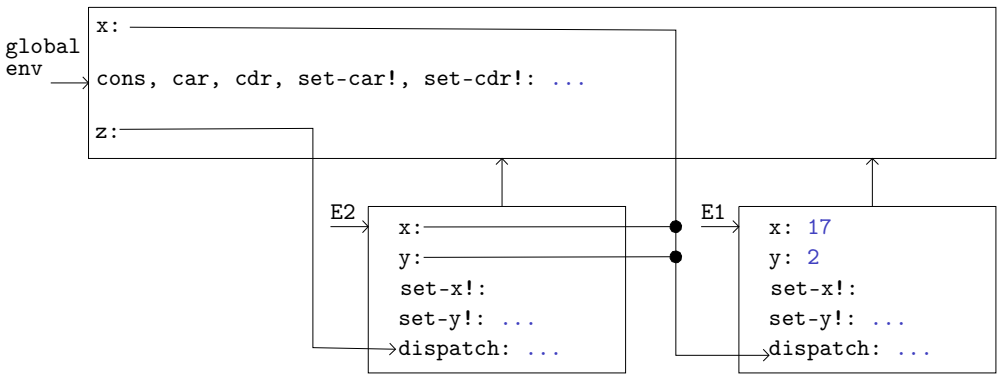

Figure 7. Environments after the evaluation of `(set-car! (cdr z) 17)`.

Finally, when trying to evaluate the expression `(car x)` in the global environment, the interpreter set up another new environment E8 subordinate to the global environment in which `z`, the formal parameter of `car` was bound to `x` of the global environment. We know that the value of `x` in the global environment is the binding of the internal procedure `dispatch` in E1. In other word, evaluting `(car z)` associated the formal parameter of the procedure object `car` to the internal procedure `dispatch` of E1. Hence, this established another new environment E9 subordinated to E1 in which `m`, the formal parameter of `dispatch` was bound to `'car` and evaluated the body of `dispatch`. At this point, it is quite straightforward to obtain the value of `(car x)` in E1, which is 17. Figure 8 shows the environment structure in evaluating `(car x)`.

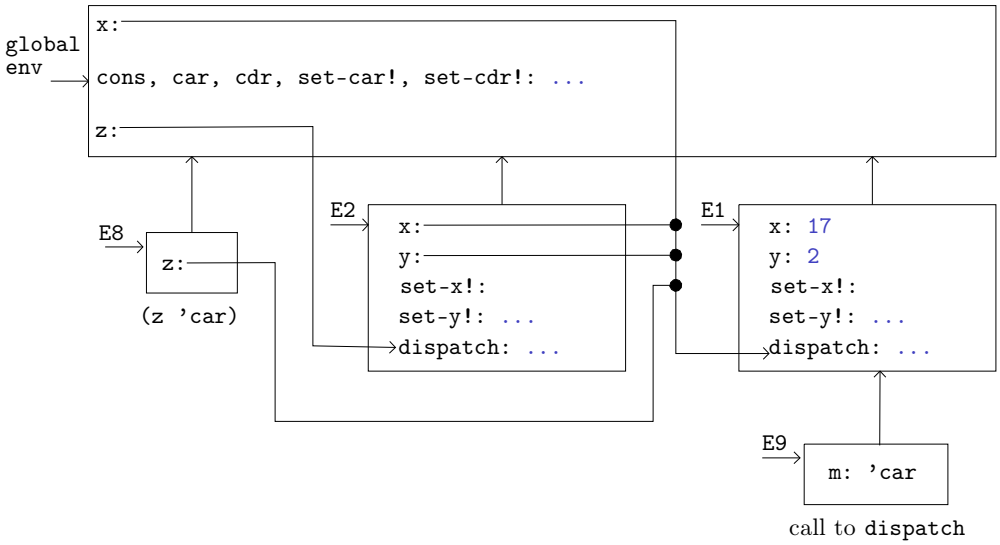

Figure 8. Environment structure in evaluating `(car x)`.