Chapter 3 Exercise 3.20

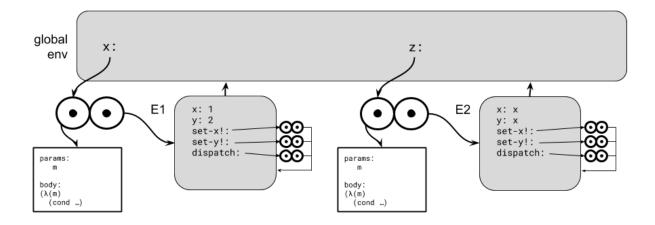


Figure 1: Frame structure created after defining x and y

Exercise 3.20

Problem statement

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 procedural implementation of pairs given above. (Compare Exercise 3.11.)

Solution

For simplification I will omit the five following functions that point to the global environment: cons, car, cdr, set-car! and set-cdr!.

Let's start with the first line, (define x (cons 1 2)). This will generate an environment E1 that points to the global environment, containing the bindings for variables x and y of the cons, as well as function definitions for dispatch, set-x! and set-y!. The second call (define z (cons x x)) will create a similar function pointing to an environment E2 — the Excercise 3.11 equivalent of (define acc2 (make-account 100)). There is one key difference though, and that is that the variables x and y from E2 are bound to variable x from the global environment now. See Figure 1 for the environment structure at this point.

The next call, (set-car! (cdr z) 17), needs a bit of attention. At first it will search the procedure set-car! in the global environment, where luckily we had defined it as

Chapter 3 Exercise 3.20

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

This will generate a temporary frame E3 pointing to the global environment where z takes the value (car z) — the value z from (car z) will be taken from the global environment, otherwise we'd be stuck in a loop! car itself is defined in the global environment as

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

This again will create a frame E4 where z takes the value of z from the global environment, which essentially aliases dispatch from E2. Now that dispatch will return the value of y from frame E2, which was our friend x from the global environment all along! So our function becomes $(set-car! x 17) \rightarrow (x 'set-car! 17)$ — remember x was actually dispatch from E1, so the value of x in E1 will now take the value 17. Careful, x in the global environment and frame E2 will continue their happy lives as variables that were never changed.

What is the state of affairs now though? Making the final call to (car x) will call the dispatch method in E1 with 'car, which returns the value x from E2 is bound to, which is 17. Some temp frames are created when calling car, but they don't change our overall environment structure.