

Exercise 3.9.

In section 1.2.1 we used the substitution model to analyze two procedures for computing factorials, a recursive version

```
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
```

and an iterative version

```
(define (factorial n)
  (fact-iter 1 1 n))

(define (fact-iter product counter max-count)
  (if (> counter max-count)
      product
      (fact-iter (* counter product)
                  (+ counter 1)
                  max-count)))
```

Show the environment structures created by evaluating `(factorial 6)` using each version of the `factorial` procedure.¹

Answer.

Figure 1 shows the procedure objects created by evaluating the definition of `factorial` of recursive

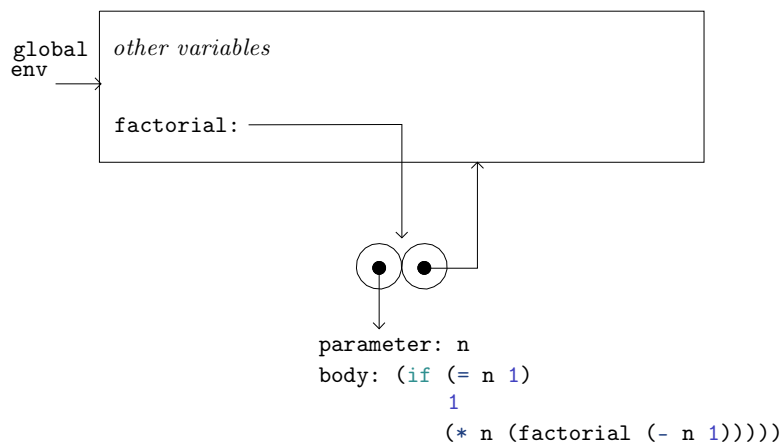



Figure 1. Procedure objects in the global frame

version in the global environment.

In figure 2 we see the environment structure created by evaluating the expression `(factorial 6)` of recursive implementation.

Following the same way, we can obtain the procedure object created by evaluating the definition of `factorial` of iterative version, as shown in figure 3.

Using the procedure objects we attained in figure 3, the environment structure created by evaluating `(factorial 6)` of iterative version now becomes obvious, as in figure 4.

*. Creative Commons  2013, Lawrence X. Amlord (颜世敏, aka 颜序).

1. The environment model will not clarify our claim in section 1.2.1 that the interpreter can execute a procedure such as `fact-iter` in a constant amount of space using tail recursion. We will discuss tail recursion when we deal with the control structure of the interpreter in section 5.4.

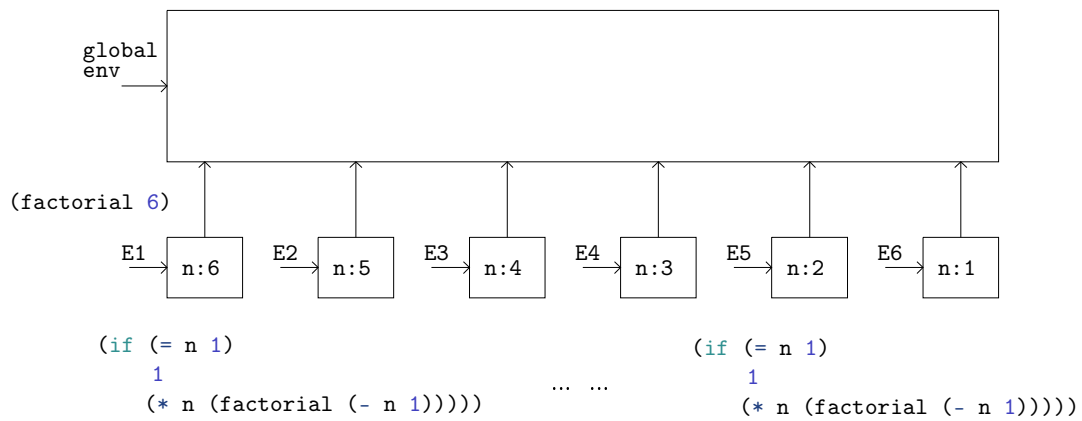


Figure 2. Environments created by evaluating (factorial 6) using the procedures in figure 1.

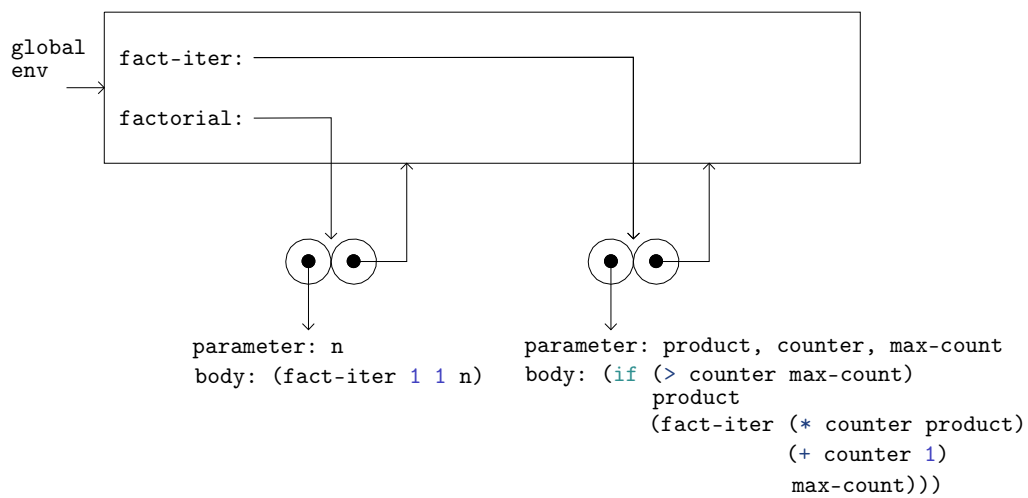


Figure 3. Procedure object in the global frame.

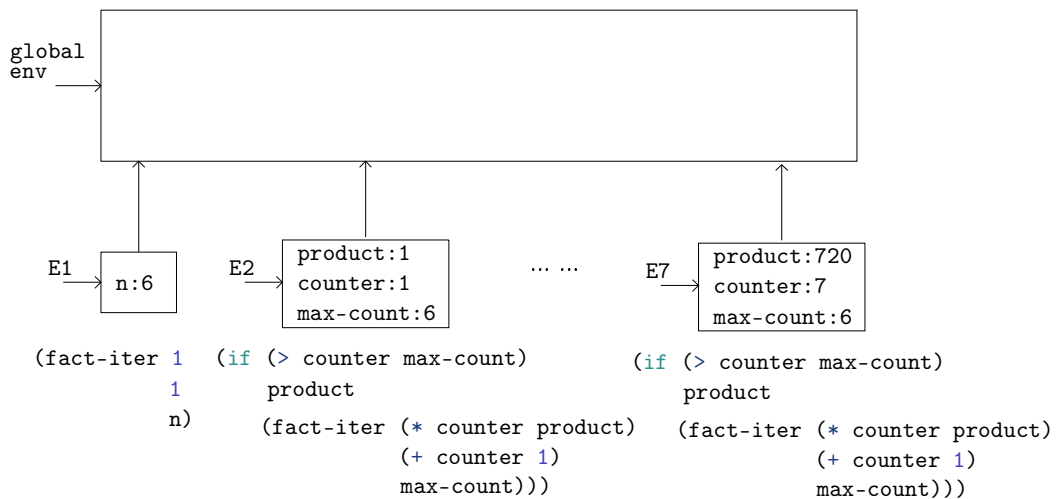


Figure 4. Environments created by evaluating (factorial 6) using the procedures in figure 3.