Exercise 4.23.

Alyssa P. Hacker doesn't understand why `analyze-sequence` needs to be so complicated. All the other analysis procedures are straightforward transformations of the corresponding evaluation procedures (or `eval` clauses) in section 4.1.1. She expected `analyze-sequence` to look like this:

```
(define (analyze-sequence exps)
  (define (execute-sequence procs env)
    (cond ((null? (cdr procs)) ((car procs) env))
          (else ((car procs) env)
                (execute-sequence (cdr procs) env))))
  (let ((procs (map analyze exps)))
    (if (null? procs)
        (error "Empty sequence -- ANALYZE"))
    (lambda (env) (execute-sequence procs env))))
```

Eva Lu Ator explains to Alyssa that the version in the text does more of the work of evaluating a sequence at analysis time. Alyssa's sequence-execution procedure, rather than having the calls to the individual execution procedures built in, loops through the procedures in order to call them: In effect, although the individual expressions in the sequence have been analyzed, the sequence itself has not been.

Compare the two versions of `analyze-sequence`. For example, consider the common case (typical of procedure bodies) where the sequence has just one expression. What work will the execution procedure produced by Alyssa's program do? What about the execution procedure produced by the program in the text above? How do the two versions compare for a sequence with two expressions?

Answer.

We compare the two version of `analyze-sequence` by investigating the result they produce while evaluating the expression:

```
(analyze (<procedure> <args>))
```

Consider a common procedure of single body expression:

```
(define (<procedure> <parameters>)
  <body-exp>)
```

Using the rule of substitution, we see that both Alyssa's program and the one in the text above generate exactly the same expression:

```
(lambda (env)
  (<body-exp>
   (extend-environment <parameters>
                       <args>
                       env)))
```

However, the fact no longer holds when they are compared for a sequence with two expressions:

```
(define (<procedure> <parameters>)
  (begin <body-exp1>
         <body-exp2>))
```

Alyssa's implementation, analyzes the individual execution procedures through procedure call, terminates with all the subexpressions except the first one of a sequence unanalyzed:

```
(lambda (env)
  (<body-exp1>
   (extend-environment <parameters>
                         <args>
                         env))
 (execute-sequence (cdr (begin <body-exp1> <body-exp2>)) env))
```

By comparision, our original `analyze-sequence` procedure analyzes the sequence thoroughly:

```
(lambda (env)
  (<body-exp1>
   (extend-environment <parameters>
                       <args>
                       env))
  (<body-exp2>
   (extend-environment <parameters>
                       <args>
                       env)))
```