Exercise 3.38

```
;; the-grammar
(expression
 ("cond" (arbno expression "==>" expression) "end")
 cond-exp)

;; translation-of
(cond-exp (predicates consequents)
          (cond-exp (map (lambda (p) (translation-of p senv))
                         predicates)
                    (map (lambda (c) (translation-of c senv))
                         consequents)))

;; value-of
(cond-exp (predicates consequents)
          (value-of-cond predicates consequents nameless-env))

;; value-of-cond :
;; Listof(Exp) * Listof(Exp) * Nameless-Env -> ExpVal
(define value-of-cond
  (lambda (predicates consequents nameless-env)
    (cond ((null? predicates)
           (eopl:error 'value-of-cond
                       "no true predicates in cond"))
          ((expval->bool (value-of (car predicates) nameless-env))
           (value-of (car consequents) nameless-env))
          (else (value-of-cond (cdr predicates)
                               (cdr consequents)
                               nameless-env)))))
```

Exercise 3.39

This does not check the condition that the number of variables in an unpack expression matches the number of elements in the corresponding list.

```
;; the-grammar
(expression
 ("cons" "(" expression "," expression ")")
 cons-exp)

(expression
 ("car" "(" expression ")")
 car-exp)

(expression
 ("cdr" "(" expression ")")
 cdr-exp)

(expression
 ("null?" "(" expression ")")
 null?-exp)

(expression
 ("emptylist")
 emptylist-exp)

(expression
 ("list" "(" (separated-list expression ",") ")")
 list-exp)

(expression
 ("unpack" (arbno identifier) "=" expression "in" expression)
 unpack-exp)

(expression
 ("\%unpack" expression "in" expression)
 nameless-unpack-exp)

;; translation-of
(cons-exp (exp1 exp2)
          (cons-exp (translation-of exp1 senv)
                    (translation-of exp2 senv)))
(car-exp (exp1) (car-exp (translation-of exp1 senv)))
(cdr-exp (exp1) (cdr-exp (translation-of exp1 senv)))
(null?-exp (exp1) (null?-exp (translation-of exp1 senv)))
(emptylist-exp () (emptylist-exp))
(list-exp (exps)
          (list-exp (map (lambda (exp)
                           (translation-of exp senv))
                         exps)))
(unpack-exp (vars exp1 body)
            (nameless-unpack-exp
             (translation-of exp1 senv)
             (translation-of body (extend-senv* vars senv))))

;; value-of
(cons-exp (exp1 exp2)
          (pair-val (value-of exp1 nameless-env)
                    (value-of exp2 nameless-env)))
(car-exp (exp1) (car (expval->pair (value-of exp1 nameless-env))))
(cdr-exp (exp1) (cdr (expval->pair (value-of exp1 nameless-env))))
(null?-exp (exp1)
           (let ((val1 (value-of exp1 nameless-env)))
             (cases expval val1
               (emptylist-val () (bool-val #t))
               (else (bool-val #f)))))
(emptylist-exp () (emptylist-val))
(list-exp (exps) (value-of-list exps nameless-env))
```

```
(nameless-unpack-exp (exp1 body)
                     (value-of
                      body
                      (extend-nameless-env*
                       (expval->list (value-of exp1 nameless-env))
                       nameless-env)))

;; value-of-list : Listof(Nameless-exp) * Nameless-env -> ExpVal
(define value-of-list
  (lambda (exps nameless-env)
    (if (null? exps)
        (emptylist-val)
        (pair-val (value-of (car exps) nameless-env)
                  (value-of-list (cdr exps) nameless-env)))))
```