Exercise 4.50.

Implement a new special form `ramb` that is like `amb` except that it searches alternatives in a random order, rather than from left to right. Show how this can help with Alyssa's problem in exercise 4.49.

Answer.

The syntax procedures to recognize the `ramb` special form are almost identical to `amb`'s:

```
(define (ramb? exp) (tagged-list? exp 'ramb))

(define (ramb-choices exp) (cdr exp))
```

The `amb` clause in `analyze` now must be replaced with `ramb`:

```
((ramb? exp) (analyze-ramb exp))
```

Unlike the execution procedure for `amb`, `analyze-ramb` searches alternative in a random order:

```
(define (analyze-ramb exp)
  (let ((cprocs (map analyze (ramb-choices exp))))
    (define (eliminate item exps)
      (cond ((null? exps) '())
            ((equal? (car exps) item)
             (cdr exps))
            (else
             (cons (car exps) (eliminate item (cdr exps))))))
    (define (list-ref items n)
      (cond ((null? items) '())
            ((= n 0) (car items))
            (else (list-ref (cdr items) (- n 1)))))
    (define (length items)
      (if (null? items)
          0
          (+ (length (cdr items)) 1)))
    (lambda (env succeed fail)
      (define (try-random choices)
        (if (null? choices)
            (fail)
            (let ((random-choice
                   (list-ref choices (random (length choices)))))
              (random-choice env
                             succeed
                             (lambda ()
                               (try-random
                                (eliminate random-choice choices)))))))
      (try-random cprocs))))
```


Email address: informlarry@gmail.com