Massachusetts Institute of Technology

6.034 Artificial Intelligence

Solutions # **1** 6034 Item # **3**

Problem 1

```
(define (count x)
  (if (pair? x)
      (+ (count (car x)) (count (cdr x)))
    1))
```

Problem 2

These are the necessary changes/additions to the original matching code.

```
(define (do-match p d bindings)
  "
  Purpose: Matching
  Arguments: Pattern, datum, optional bindings.
  Returns: A list of bindings (whose car is the symbol 'bindings)
or #f if no match is possible.
  Remarks: Pattern variables are indicated by (? <variable name>)
Prolog's nameless variable is indicated by (? _)
  "
  (cond ((and (null? p) (null? d))
 bindings)
        ;; The placement of this is important.  You need to catch
        ;; the case where d = () as in (match '(a (* x)) '(a)) -> ((x ()))
        ((and (pair? p) (segment-variable? (first p)))
         (match-segment-variable p d bindings))
((or (atom? p) (atom? d)) ; This is not allowed
 #f)
((and (atom? (first p)) (atom? (first d)))
         (match-atoms p d bindings))
        ((simple-variable? (first p))
         (match-simple-variable p d bindings))
        ((and (pair? (first p)) (pair? (first d)))
         (match-pieces p d bindings))
        (else #f)))
```

```scheme
;;; A test for a (* x) type of variable -- segment variable, can match
;;; 0 or more entries in the data list.
(define (segment-variable? x) (and (pair? x) (eq? '* (first x))))

;;; Handle segment variables, analogous to simple-variables.
(define (match-segment-variable p d bindings)
  ;; Check for either consistent match of previously bound variable
  ;; or bind a new value and proceed with the match.
  (let ((binding (find-binding (first p) bindings)))
    ;;Is the pattern variable on the list of bindings:
    (if binding
        ;;If it is, substitute its value and try matching again.
        ;;Note the use of append instead of cons.
        (do-match (append (extract-value binding) (rest p))
                  d
                  bindings)
        ;; Otherwise, match the rest
        (match-segment-variable-aux p d bindings '()))))

;;; This is the tricky bit, we repeatedly try to match longer and longer
;;; tails of the data to the rest of the pattern (minus segment variable).
(define (match-segment-variable-aux p d bindings segment-binding)
  ;; Check to see if the pattern (without var) matches data.
  (let ((result
         (do-match (rest p)
                   d
                   ;; the tentative binding of the var is accumulated
                   ;; in the segment-binding argument.
                   (add-binding (first p) segment-binding bindings))))
    (if result
;; Success
        result
        ;; Failed to match, cut the data and grow the segment binding.
        (if (null? d)
            ;; We ran out of data to match.
            #f
            (match-segment-variable-aux
             p
             ;; the new, shorter, datum to bind against.
             (rest d)
             bindings
             ;; the new, longer, variable binding to try.
             (append segment-binding (list (first d))))))))
```