Ordered Search of AND/OR Trees in LISP

1. Each node is a problem with associated tree information:

   (problem-description :anything you wish
   h-hat-value :heuristic rating
   solve-unsolved-label :labels whether this is solved or not
   and-or-tag
   (list of ptrs to all sons)
   (list of ptrs to ancestors))

2. You call the search procedure by:

   (ordered-search
    '(initial-problem 0 nil nil nil nil)
    'heuristic-function
    'successor-generator)

3. Auxiliary Functions:

   (Defun sons-of-node (n)
    (caddddr n))

   (Defun ancestors-of-node (n)
    (caddddr n))

   (Defun is-labelled-solved (n)
    (Eq
     (cadr n) 'solved))

   (Defun is-labeled-unsolvable (n)
    (Eq
     (Caddr n) 'unsolvable))

   (Defun is-unlabelled d(n)
    (null
     (Caddr n)))

   (Defun SONS-ARE-OR (n)
    (Eq
     (Cadddr n) 'OR))

   (Defun SONS-ARE-AND (n)
    (Eq
     (Cadddr n) 'AND))

   (Defun TERMINAL (n)
    (:Problem specific function determining if a problem is trivial))

   (Defun MARK-SOLVED (n)
    (RPLACA
     (Cddr n) 'SOLVED))

   (Defun MARK-UNSOLVABLE (n)
    (RPLACA
     (Cddr n) 'UNSOLVABLE))
Search functions in detail:

(Defun SEARCH-E
  (LIST FUNC)
  ; searches list, applies function, returns true element
  (PROG ( )
  Li (cond
    ((null list) (Return Nil))
    ((FUNCALL FUNC (Car List)) (Return (Car List))))
    ((Setq LIST (cdr List))
     (Go Li))))

(Defun SEARCH-W
  (LIST FUNC)
  ; Searches List, applies function, returns true value
  (PROG (HOLD)
  Li (cond
    ((null List) (Return Nil))
    ((Setq HOLD
     (FUNCALL FUNC (CAR LIST))
     (Return Hold)))))
    ((Setq List (Cdr List))
     (Go Li))))

(Defun MAPC (LIST FUNC)
  (COND ((NULL LIST) NIL)
         ((T (FUNCALL FUNC (CAR LIST)))
          (MAPC (CDR LIST) FUNC))))

(Defun SOLVE-LABEL (n)
  (COND
    (((IS-LABELED-SOLVED n) T)
     ((TERMINAL n) (MARK-SOLVED n))
     ((SONS-ARE-OR n)
      (COND
       (((SEARCHV (SONS-OF-MODE n)
          'LAMBDA (S)
          (FUNCALL 'SOLVE-LABEL S)))
        (MARK-SOLVED n)) (T NIL)))
     ((SONS-ARE-AND n)
      (COND
       (((SEARCHV (SONS-OF-MODE n)
          'LAMBDA (S)
          (FUNCALL 'NOT-SOLVE-LABEL S)))
        NIL)
        (T (MARK-SOLVED n))))))

(Defun NOT-SOLVE-LABEL (n)
  (not (solve-label n)))

(Defun UNSOLVABLE-LABEL (n)
  (; Very similar to the above))

(Defun GREATEST HATVALUE (M1 M2)
  (> (HAT M1) (HAT M2)))

(Defun HAT (n)
  (CADR n))
; Main guts of the search follows ;
(defun successor-node (node)
  (successors node))

(defun heuristic-function (node)
  (successors node))
(Defun POTENTIAL-SOLUTION-TREE (NODE)
    (Cond
      ((IS-LABELLED-SOLVED NODE)
        (List NODE))
      ((IS-LABELLED-UN SOLVABLE NODE)
        NIL)
      ((TERMINAL NODE)
        (List NODE))
      ((SONS-ARE-AND NODE)
        (Cons NODE
          (MapConc
            (sons-of-node NODE)
            '(lambda (s)
              (potential-solution-tree s)))
          )
          )
      ((SONS-ARE-OR NODE)
        (Cons NODE
          (potential-solution-tree
            (Search
              (sons-of-node NODE)
              '(lambda (s)
                (best-hhatvalue s))))
          )
          ))
    )))

(Defun BEST-HHATVALUE
  ; finds the best son according to H-HAT)

(Defun TIP-NODE-ON-OPEN
  (TAU)
  (Search TAU
    (List (lambda (NODE) (and (member NODE OPEN) (null (sons-of-node NODE))))))

(Defun MAPCONC
  (LIST FUNC)
  (Cond
    ((null LIST) NIL)
    (T
      (append (apply FUNC (CAR LIST))
              (mapcon (CDR LIST))))))
