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
 'succesor-generator)
```

3. Auxiliary Functions:

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

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

(Defun is-labelled-solved (n)
  (Eq
   (caddr 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))
    ; Previous two returns either 'SOLVED or 'UNSOLVABLE as value
      of function call
```

Search functions in detail:

```
(Defun  SEARCHE
  (LIST FUNC)
  ; searches list, applies function, returns true element
  (PROG ( )
L1 (cond
    ((Null list) (Return Nil))
    ((FUNCALL FUNC
              (Car LIst)) (Return  (Car List))))
    (Setq LIST  (cdr List))
    (GO  L1)))

(Defun SEARCHV
  (LIST FUNC)
  ; Searches List, applies function, returns true value
  (PROG  (HOLD)
l1
    (Cond
     ((Null List)  (Return Nil))
     ((Setq  HOLD
             (FUNCALL FUNC  (CAR LIST)))
      (Return Hold)))
    (Setq List  (Cdr List))
    (Go  L1)))

(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-NODE  n)
       '(LAMBDA  (S)
       (FUNCALL 'SOLVE-LABEL  S)))
     (MARK-SOLVED  n)) (T NIL)))
((SONS-ARE-AND  n)
 (COND
  ((SEARCHV  (SONS-OF-NODE  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 HHATVALUE  (N1 N2)
  (> (HHAT  N1)  (HHAT N2)))

(Defun  HHAT  (n)
  (CADR N))
; Main guts of the search follows ;
```

```
(Defun  ORDERED-SEARCH
  (START-NODE HVALFUNC SUCOP)
  (PROG (OPEN CLOSED TAUZERO SUCCS  N)
    (SETQ CLOSED NIL)
    (SETQ OPEN (LIST START-NODE))
    (RPLACA (CDR START-NODE)
            (FUNCALL HVALFUNC START-NODE))
    (SETQ N NIL)
    TWO (SETQ TAUZERO
              (POTENTIAL-SOLUTION-TREE START-NODE))
    (SETQ  N (TIP-NODE-ON-OPEN TAUZERO))
    (SETQ OPEN (REMOVE N OPEN))
    (SETQ CLOSED (CONS N CLOSED))
    (COND
     ((TERMINAL N)  (MARK-SOLVED N)
     (SOLVE-LABEL (CAR TAUZERO)); START-NODE
     (COND
      ((IS-LABELLED-SOLVED (CAR TAUZERO))
       (RETURN TAUZERO)))
     (SETQ OPEN (CLEAN-UP OPEN)); REMOVE SOLVED NODES
     (GO TWO)
     NINE
     (SETQ SUCCS (FUNCALL SUCOP N))
     (COND
      (SUCCS
       (GO FOURTEEN)))
     (MARK-UNSOLVABLE  n)
     (UNSOLVABLE-LABEL (car TAUZERO))
     (COND
      ((IS-LABELLED-UNSOLVABLE  (CAR TAUZERO))
       (RETURN NIL)))
     (SETQ OPEN (CLEAN-UP OPEN));  REMOVE UNSOLVABLE NODES
     (GO TWO)
     FOURTEEN
     (MAPC SUCCS
           '(LAMABDA (NODE)
           (RPLACA
            (Cdddddr NODE) N)))
    ; N  is  ANCESTOR OF SUCCESSORS
    (MAPC SUCCS
          '(LAMBDA (NODE)
          (RPLACA (CDR NODE)
                  (APPLY HVALFUNC NODE))))
  (SETQ OPEN
        (DEPTH-FIRST   : (APPEND SUCCS OPEN)
        (BREADTH-FIRST : (APPEND OPEN SUCCS)
        (HILL-CLIMBING : (APPEND
                          (SORT SUCCS 'GREATEST HHATVALUE) OPEN)
        (ORDERED OR BEST-FIRST
                    (SORT
                     (APPEND SUCCS OPEN) 'GREATEST-HATVALUE))
                    ; Additional auxiliary functions

(Defun  HEURISTIC-FUNCTION (NODE)
  ; Problem specific)

(Defun SUCCESSOR-OPERATOR (NODE)
  ; Problem specific)
```

```
(Defun POTENTIAL-SOLUTION-TREE (NODE)
  (COND
        ((IS-LABELLED-SOLVED NODE)
         (LIST NODE))
         ((IS-LABELLED-UNSOLVABLE 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
                  (SEARCHE
                   (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)
   (SEARCHE TAU
            '(LAMBDA
            (NODE)
            (AND
             (MEMBER NODE OPEN)
             (NULL
              (SONS-OF-NODE NODE))))))

(Defun MAPCONC
  (LIS FUNC)
  (COND
   ((NULL LIS)
    NIL)
   (T
    (APPEND
     (APPLY FUNC
            (CAR LIS))
     (MAPCONC  (CDR LIS)
     )))))
```