```
    File created:  18-May-90 01:25:52  {DSK}<usr>local>lde>lispcore>sources>XCLC-ANALYZE.;2

      changes to:  (IL:VARS IL:XCLC-ANALYZECOMS)

   previous date:   7-Oct-87 18:39:59  {DSK}<usr>local>lde>lispcore>sources>XCLC-ANALYZE.;1

     Read Table:  XCL

        Package:  COMPILER

         Format:   XCCS
```

```
(IL:RPAQQ IL:XCLC-ANALYZECOMS
          (;; Analysis of the program tree, prior to and during meta-evaluation

           (IL:FUNCTIONS ANALYZE-TREE)
           (IL:VARIABLES *REDO-FLAG*)

           ;; Environment analysis

           (IL:FUNCTIONS ENV-ANALYZE)
           (IL:FUNCTIONS ENV-ANALYZE-BLOCK ENV-ANALYZE-CALL ENV-ANALYZE-CATCH ENV-ANALYZE-GO ENV-ANALYZE-IF
                  ENV-ANALYZE-LABELS ENV-ANALYZE-LAMBDA ENV-ANALYZE-LITERAL ENV-ANALYZE-MV-CALL
                  ENV-ANALYZE-OPCODES ENV-ANALYZE-MV-PROG1 ENV-ANALYZE-PROGN ENV-ANALYZE-PROGV
                  ENV-ANALYZE-RETURN ENV-ANALYZE-SETQ ENV-ANALYZE-TAGBODY ENV-ANALYZE-THROW
                  ENV-ANALYZE-UNWIND-PROTECT ENV-ANALYZE-VAR-REF)

           ;; Side-effects analysis

           (IL:FUNCTIONS EFFECTS-ANALYZE EFFECTS-UNION REMOVE-EFFECT)
           (IL:FUNCTIONS EFFECTS-ANALYZE-BLOCK EFFECTS-ANALYZE-CALL EFFECTS-ANALYZE-CATCH EFFECTS-ANALYZE-GO
                  EFFECTS-ANALYZE-IF EFFECTS-ANALYZE-LABELS EFFECTS-ANALYZE-LAMBDA EFFECTS-ANALYZE-LITERAL
                  EFFECTS-ANALYZE-MV-CALL EFFECTS-ANALYZE-MV-PROG1 EFFECTS-ANALYZE-OPCODES EFFECTS-ANALYZE-PROGN
                  EFFECTS-ANALYZE-PROGV EFFECTS-ANALYZE-RETURN EFFECTS-ANALYZE-SETQ EFFECTS-ANALYZE-TAGBODY
                  EFFECTS-ANALYZE-THROW EFFECTS-ANALYZE-UNWIND-PROTECT EFFECTS-ANALYZE-VAR-REF)
           (IL:FUNCTIONS EFFECTS-ANALYZE-ANY-CALL EFFECTS-ANALYZE-LIST EFFECTS-REPRESENTATION)

           ;; Testing analysis

           (IL:FUNCTIONS TEST-ANALYSIS)

           ;; Arrange to use the proper compiler.

           (IL:PROP IL:FILETYPE IL:XCLC-ANALYZE)

           ;; Arrange for the proper makefile environment

           (IL:PROP IL:MAKEFILE-ENVIRONMENT IL:XCLC-ANALYZE)))
```

;; Analysis of the program tree, prior to and during meta-evaluation

```
(DEFUN ANALYZE-TREE (TREE &OPTIONAL (*REDO-FLAG* :ONCE))
```

;;; The root of all analysis. The variable *REDO-FLAG* is either :ONCE, meaning that only the given node should be analyzed, or :ALL, meaning that
;;; the whole tree should be analyzed. This latter operation is rarely, if ever, done.

```
   (ENV-ANALYZE TREE)                                                          ; Environment analysis.
   (EFFECTS-ANALYZE TREE)                                                      ; Side-effects analysis.
   TREE)
```

```
(DEFVAR *REDO-FLAG* NIL
```

;;; Used to control the depth of recursion in analysis. It can take on three values:

   ;; :ALL, meaning to recurse all the way down the tree,

   ;; :ONCE, meaning to analyze only the current node, or

   ;; NIL, meaning to do nothing at all.

;;; *REDO-FLAG* is only bound or checked in ANALYZE-TREE and the dispatch functions for each kind of analysis.

```
   )
```

;; Environment analysis

```
(DEFUN ENV-ANALYZE (TREE)
```

;;; Environment analysis only does something other than pass the message down in two cases: SETQ and VARIABLE. These two keep track of the
;;; read- and write-references to lexical variables.

```
   (WHEN (NOT (NULL *REDO-FLAG*))
       (LET ((*REDO-FLAG* (AND (EQ *REDO-FLAG* :ALL)
                               :ALL)))
            (NODE-DISPATCH ENV-ANALYZE TREE))))
```

```
(DEFUN ENV-ANALYZE-BLOCK (NODE)
   (ENV-ANALYZE (BLOCK-STMT NODE)))

(DEFUN ENV-ANALYZE-CALL (NODE)
   (ENV-ANALYZE (CALL-FN NODE))
   (IL:FOR ARG IL:IN (CALL-ARGS NODE) IL:DO (ENV-ANALYZE ARG)))

(DEFUN ENV-ANALYZE-CATCH (NODE)
   (ENV-ANALYZE (CATCH-TAG NODE))
   (ENV-ANALYZE (CATCH-STMT NODE)))

(DEFUN ENV-ANALYZE-GO (NODE)
   NIL)

(DEFUN ENV-ANALYZE-IF (NODE)
   (ENV-ANALYZE (IF-PRED NODE))
   (ENV-ANALYZE (IF-THEN NODE))
   (ENV-ANALYZE (IF-ELSE NODE)))

(DEFUN ENV-ANALYZE-LABELS (NODE)
   (ENV-ANALYZE (LABELS-BODY NODE))
   (IL:FOR FUN IL:IN (LABELS-FUNS NODE) IL:DO (ENV-ANALYZE (CDR FUN))))

(DEFUN ENV-ANALYZE-LAMBDA (NODE)
   (ENV-ANALYZE (LAMBDA-BODY NODE))
   (IL:FOR OPT-VAR IL:IN (LAMBDA-OPTIONAL NODE) IL:DO (ENV-ANALYZE (SECOND OPT-VAR)))
   (IL:FOR KEY-VAR IL:IN (LAMBDA-KEYWORD NODE) IL:DO (ENV-ANALYZE (THIRD KEY-VAR))))

(DEFUN ENV-ANALYZE-LITERAL (NODE)
   NIL)

(DEFUN ENV-ANALYZE-MV-CALL (NODE)
   (ENV-ANALYZE (MV-CALL-FN NODE))
   (IL:FOR ARG IL:IN (MV-CALL-ARG-EXPRS NODE) IL:DO (ENV-ANALYZE ARG)))

(DEFUN ENV-ANALYZE-OPCODES (NODE)
   NIL)

(DEFUN ENV-ANALYZE-MV-PROG1 (NODE)
   (IL:FOR STMT IL:IN (MV-PROG1-STMTS NODE) IL:DO (ENV-ANALYZE STMT)))

(DEFUN ENV-ANALYZE-PROGN (NODE)
   (IL:FOR STMT IL:IN (PROGN-STMTS NODE) IL:DO (ENV-ANALYZE STMT)))

(DEFUN ENV-ANALYZE-PROGV (NODE)
   (ENV-ANALYZE (PROGV-SYMS-EXPR NODE))
   (ENV-ANALYZE (PROGV-VALS-EXPR NODE))
   (ENV-ANALYZE (PROGV-STMT NODE)))

(DEFUN ENV-ANALYZE-RETURN (NODE)
   (ENV-ANALYZE (RETURN-VALUE NODE)))

(DEFUN ENV-ANALYZE-SETQ (NODE)
   ;; This one actually does something: we note the write-ref to the variable being SETQ'd.
   (PUSHNEW NODE (VARIABLE-WRITE-REFS (SETQ-VAR NODE)))
   (ENV-ANALYZE (SETQ-VALUE NODE)))

(DEFUN ENV-ANALYZE-TAGBODY (NODE)
   (IL:FOR SEGMENT IL:IN (TAGBODY-SEGMENTS NODE) IL:DO (IL:FOR STMT IL:IN (SEGMENT-STMTS SEGMENT)
                                                                IL:DO (ENV-ANALYZE STMT))))

(DEFUN ENV-ANALYZE-THROW (NODE)
   (ENV-ANALYZE (THROW-TAG NODE))
   (ENV-ANALYZE (THROW-VALUE NODE)))

(DEFUN ENV-ANALYZE-UNWIND-PROTECT (NODE)
   (ENV-ANALYZE (UNWIND-PROTECT-STMT NODE))
   (ENV-ANALYZE (UNWIND-PROTECT-CLEANUP NODE)))
```

(DEFUN **ENV-ANALYZE-VAR-REF** (NODE)

;; This one actually does something: we note this read-ref to the variable being referenced.

```
   (PUSHNEW NODE (VARIABLE-READ-REFS (VAR-REF-VARIABLE NODE))))
```

;; Side-effects analysis

(DEFUN **EFFECTS-ANALYZE** (TREE)

;;; Side-effects analysis methods store the side-effects data for the subtree they're given in the node at the root of that subtree (in the EFFECTS and
;;; AFFECTED fields).

```
   (WHEN (AND (NOT (NULL *REDO-FLAG*))
              (NOT (NULL TREE)))
       (LET ((*REDO-FLAG* (AND (EQ *REDO-FLAG* :ALL)
                               :ALL)))
            (NODE-DISPATCH EFFECTS-ANALYZE TREE))))
```

(DEFUN **EFFECTS-UNION** (ONE TWO)

;;; Return a side-effects description representing the union of the two descriptions given.

```
   (COND
      ((EQ :NONE ONE)
       TWO)
      ((EQ :NONE TWO)
       ONE)
      ((OR (EQ :ANY ONE)
           (EQ :ANY TWO))
       :ANY)
      (T (UNION (IL:MKLIST ONE)
                (IL:MKLIST TWO)))))
```

(DEFUN **REMOVE-EFFECT** (EFFECT EFFECTS-REP)

```
   (IF (OR (EQ :NONE EFFECTS-REP)
           (EQ :ANY EFFECTS-REP))
       EFFECTS-REP
       (REMOVE EFFECT (IL:MKLIST EFFECTS-REP))))
```

(DEFUN **EFFECTS-ANALYZE-BLOCK** (NODE)

;;; The side-effect of a RETURN is represented by the BLOCK from which it is returning. Thus, we can remove this node from the effects since the
;;; RETURN is invisible outside the BLOCK.

```
   (EFFECTS-ANALYZE (BLOCK-STMT NODE))
   (SETF (NODE-EFFECTS NODE)
         (REMOVE-EFFECT NODE (NODE-EFFECTS (BLOCK-STMT NODE))))
   (SETF (NODE-AFFECTED NODE)
         (NODE-EFFECTS (BLOCK-STMT NODE))))
```

(DEFUN **EFFECTS-ANALYZE-CALL** (NODE)

;;; Much code can be shared between CALL and MV-CALL here.

```
   (EFFECTS-ANALYZE-ANY-CALL NODE (CALL-FN NODE)
          (CALL-ARGS NODE)))
```

(DEFUN **EFFECTS-ANALYZE-CATCH** (NODE)

```
   (EFFECTS-ANALYZE-LIST NODE (LIST (CATCH-TAG NODE)
                                    (CATCH-STMT NODE))))
```

(DEFUN **EFFECTS-ANALYZE-GO** (NODE)

;;; The side-effect of a GO is represented by the TAGBODY to which it is GOing.

```
   (SETF (NODE-EFFECTS NODE)
         (LIST (GO-TAGBODY NODE)))
   (SETF (NODE-AFFECTED NODE)
         :NONE))
```

(DEFUN **EFFECTS-ANALYZE-IF** (NODE)

```
   (EFFECTS-ANALYZE-LIST NODE (LIST (IF-PRED NODE)
                                    (IF-THEN NODE)
                                    (IF-ELSE NODE))))
```

(DEFUN **EFFECTS-ANALYZE-LABELS** (NODE)

;;; The effects of a LABELS are exactly those of the body. The functions have no effects.

```
   (DOLIST (FUN (LABELS-FUNS NODE))
       (EFFECTS-ANALYZE-LAMBDA (CDR FUN)))
   (EFFECTS-ANALYZE (LABELS-BODY NODE))
   (SETF (NODE-EFFECTS NODE)
         (NODE-EFFECTS (LABELS-BODY NODE)))
   (SETF (NODE-AFFECTED NODE)
         (NODE-AFFECTED (LABELS-BODY NODE))))


(DEFUN EFFECTS-ANALYZE-LAMBDA (NODE)
   (LET ((EFFECTS :NONE)
         (AFFECTED :NONE))
        (DOLIST (OPT-VAR (LAMBDA-OPTIONAL NODE))
            (EFFECTS-ANALYZE (SECOND OPT-VAR))
            (SETQ EFFECTS (EFFECTS-UNION EFFECTS (NODE-EFFECTS (SECOND OPT-VAR))))
            (SETQ AFFECTED (EFFECTS-UNION AFFECTED (NODE-AFFECTED (SECOND OPT-VAR)))))
        (DOLIST (KEY-VAR (LAMBDA-KEYWORD NODE))
            (EFFECTS-ANALYZE (THIRD KEY-VAR))
            (SETQ EFFECTS (EFFECTS-UNION EFFECTS (NODE-EFFECTS (THIRD KEY-VAR))))
            (SETQ AFFECTED (EFFECTS-UNION AFFECTED (NODE-AFFECTED (THIRD KEY-VAR)))))
        (EFFECTS-ANALYZE (LAMBDA-BODY NODE))

        ;; Save the information on the lambda as applied; it can be used by EFFECTS-ANALYZE-CALL.

        (SETF (LAMBDA-APPLIED-EFFECTS NODE)
              (EFFECTS-UNION EFFECTS (NODE-EFFECTS (LAMBDA-BODY NODE))))
        (SETF (LAMBDA-APPLIED-AFFECTED NODE)
              (EFFECTS-UNION AFFECTED (NODE-AFFECTED (LAMBDA-BODY NODE))))

        ;; The LAMBDA itself has no effects and cannot be affected.

        (SETF (NODE-EFFECTS NODE)
              :NONE)
        (SETF (NODE-AFFECTED NODE)
              :NONE)))


(DEFUN EFFECTS-ANALYZE-LITERAL (NODE)

;;;

   (IL:IF (EVAL-WHEN-LOAD-P (LITERAL-VALUE NODE))
       IL:THEN

               ;; A load-time form can have any side effects and be affected by anything - in the future we can be smarter about examining the form
               ;; itself.

               (SETF (NODE-EFFECTS NODE)
                     :ANY)
               (SETF (NODE-AFFECTED NODE)
                     :NONE)
     IL:ELSE (SETF (NODE-EFFECTS NODE)
                   :NONE)
             (SETF (NODE-AFFECTED NODE)
                   :NONE)))


(DEFUN EFFECTS-ANALYZE-MV-CALL (NODE)

;;; Much code can be shared between MV-CALL and CALL here.

   (EFFECTS-ANALYZE-ANY-CALL NODE (MV-CALL-FN NODE)
          (MV-CALL-ARG-EXPRS NODE)))


(DEFUN EFFECTS-ANALYZE-MV-PROG1 (NODE)
   (EFFECTS-ANALYZE-LIST NODE (MV-PROG1-STMTS NODE)))


(DEFUN EFFECTS-ANALYZE-OPCODES (NODE)

;;; Remember that OPCODES nodes can only appear in a functional context.  What we're asking for here is not the effect of executing the opcodes but
;;; the effect of computing them in the first place.  Since they're constants, they behave like literals.  See EFFECTS-ANALYZE-CALL for the place where
;;; we decide we know nothing about any opcodes' effects.

   (SETF (NODE-EFFECTS NODE)
         :NONE)
   (SETF (NODE-AFFECTED NODE)
         :NONE))


(DEFUN EFFECTS-ANALYZE-PROGN (NODE)
   (EFFECTS-ANALYZE-LIST NODE (PROGN-STMTS NODE)))


(DEFUN EFFECTS-ANALYZE-PROGV (NODE)
   (EFFECTS-ANALYZE-LIST NODE (LIST (PROGV-SYMS-EXPR NODE)
                                    (PROGV-VALS-EXPR NODE)
                                    (PROGV-STMT NODE))))
```

```
(DEFUN EFFECTS-ANALYZE-RETURN (NODE)

;;; The side effect of a RETURN is represented by the BLOCK from which it is returning.

   (EFFECTS-ANALYZE (RETURN-VALUE NODE))
   (SETF (NODE-EFFECTS NODE)
         (EFFECTS-UNION (LIST (RETURN-BLOCK NODE))
                (NODE-EFFECTS (RETURN-VALUE NODE))))
   (SETF (NODE-AFFECTED NODE)
         (NODE-AFFECTED (RETURN-VALUE NODE))))

(DEFUN EFFECTS-ANALYZE-SETQ (NODE)

;;;

   (EFFECTS-ANALYZE (SETQ-VALUE NODE))
   (SETF (NODE-EFFECTS NODE)
         (EFFECTS-UNION (EFFECTS-REPRESENTATION (SETQ-VAR NODE))
                (NODE-EFFECTS (SETQ-VALUE NODE))))
   (SETF (NODE-AFFECTED NODE)
         (NODE-AFFECTED (SETQ-VALUE NODE))))

(DEFUN EFFECTS-ANALYZE-TAGBODY (NODE)

;;; The side-effect for a GO is represented by the TAGBODY to which it is GOing.  Since the GO is invisible outside the TAGBODY, we can remove the
;;; TAGBODY from the effects.

   (DO ((SEGMENTS (TAGBODY-SEGMENTS NODE)
             (CDR SEGMENTS))
        (EFFECTS :NONE)
        (AFFECTED :NONE))
       ((NULL SEGMENTS)
        (SETF (NODE-EFFECTS NODE)
              (REMOVE-EFFECT NODE EFFECTS))
        (SETF (NODE-AFFECTED NODE)
              AFFECTED))

       ;; For each segment, analyze each statement and accumulate the results.

       (DOLIST (STMT (SEGMENT-STMTS (CAR SEGMENTS)))
           (EFFECTS-ANALYZE STMT)
           (SETQ EFFECTS (EFFECTS-UNION EFFECTS (NODE-EFFECTS STMT)))
           (SETQ AFFECTED (EFFECTS-UNION AFFECTED (NODE-AFFECTED STMT))))))

(DEFUN EFFECTS-ANALYZE-THROW (NODE)
   (EFFECTS-ANALYZE (THROW-TAG NODE))
   (EFFECTS-ANALYZE (THROW-VALUE NODE))
   (SETF (NODE-EFFECTS NODE)
         :ANY)
   (SETF (NODE-AFFECTED NODE)
         (EFFECTS-UNION (NODE-AFFECTED (THROW-TAG NODE))
                (NODE-AFFECTED (THROW-VALUE NODE)))))

(DEFUN EFFECTS-ANALYZE-UNWIND-PROTECT (NODE)

;;;

   (EFFECTS-ANALYZE-LAMBDA (UNWIND-PROTECT-STMT NODE))
   (EFFECTS-ANALYZE-LAMBDA (UNWIND-PROTECT-CLEANUP NODE))
   (SETF (NODE-EFFECTS NODE)
         (EFFECTS-UNION (LAMBDA-APPLIED-EFFECTS (UNWIND-PROTECT-STMT NODE))
                (LAMBDA-APPLIED-EFFECTS (UNWIND-PROTECT-CLEANUP NODE))))
   (SETF (NODE-AFFECTED NODE)
         (EFFECTS-UNION (LAMBDA-APPLIED-AFFECTED (UNWIND-PROTECT-STMT NODE))
                (LAMBDA-APPLIED-AFFECTED (UNWIND-PROTECT-CLEANUP NODE)))))

(DEFUN EFFECTS-ANALYZE-VAR-REF (NODE)

;;;

   (SETF (NODE-EFFECTS NODE)
         :NONE)
   (SETF (NODE-AFFECTED NODE)
         (EFFECTS-REPRESENTATION (VAR-REF-VARIABLE NODE))))

(DEFUN EFFECTS-ANALYZE-ANY-CALL (NODE FN ARGUMENTS)

;;;

   (DO ((ARGS ARGUMENTS (CDR ARGS))
        (EFFECTS :NONE (EFFECTS-UNION EFFECTS (NODE-EFFECTS (CAR ARGS))))
        (AFFECTED :NONE (EFFECTS-UNION AFFECTED (NODE-AFFECTED (CAR ARGS)))))
       ((NULL ARGS)
        ;; Look at the function.  If we don't know anything about it, assume the worst: both EFFECTS and AFFECTED are :ANY.
        (EFFECTS-ANALYZE FN)
        (TYPECASE FN
            (LAMBDA-NODE
               (SETF (NODE-EFFECTS NODE)
                     (EFFECTS-UNION EFFECTS (LAMBDA-APPLIED-EFFECTS FN)))
               (SETF (NODE-AFFECTED NODE)
                     (EFFECTS-UNION AFFECTED (LAMBDA-APPLIED-AFFECTED FN))))
            (VAR-REF-NODE (LET ((VAR (VAR-REF-VARIABLE FN)))
                               (COND
                                  ((CALLER-NOT-INLINE NODE)
                                   ;; If the function is not inline-expandable, we can't assume any knowledge of it.
                                   (SETF (NODE-EFFECTS NODE)
                                         :ANY)
                                   (SETF (NODE-AFFECTED NODE)
                                         :ANY))
                                  ((EQ :FUNCTION (VARIABLE-KIND VAR))
                                   (ECASE (VARIABLE-SCOPE VAR)
                                       (:GLOBAL
                                          ;; Just look in the database. We should be smarter about remembering side-effects of user
                                          ;; functions when we can.
                                          (LET ((DATA (SIDE-EFFECTS (VARIABLE-NAME VAR))))
                                               (SETF (NODE-EFFECTS NODE)
                                                     (EFFECTS-UNION EFFECTS (OR (CAR DATA)
                                                                                :ANY)))
                                               (SETF (NODE-AFFECTED NODE)
                                                     (EFFECTS-UNION AFFECTED (OR (CDR DATA)
                                                                                 :ANY)))))
                                       (:LEXICAL
                                          ;; Local function vars are only bound by LABELS nodes.
                                          (IF (TYPEP (VARIABLE-BINDER VAR)
                                                     'LABELS-NODE)
                                              ;; This is good - we can easily find the function definition and extract its side-effects.
                                              (LET ((FN-DEF (CDR (ASSOC VAR (LABELS-FUNS (VARIABLE-BINDER VAR))
                                                                        :TEST
                                                                        'EQ))))
                                                   (ASSERT (NOT (NULL FN-DEF))
                                                          NIL "BUG: Referenced lexical function not found!")
                                                   (SETF (NODE-EFFECTS NODE)
                                                         (EFFECTS-UNION EFFECTS (LAMBDA-APPLIED-EFFECTS FN-DEF)
                                                                ))
                                                   (SETF (NODE-AFFECTED NODE)
                                                         (EFFECTS-UNION AFFECTED (LAMBDA-APPLIED-AFFECTED
                                                                                  FN-DEF))))
                                              ;; Damn! We can't find the function definition to get at its side-effects. Assume the worst...
                                              (PROGN (SETF (NODE-EFFECTS NODE)
                                                           :ANY)
                                                     (SETF (NODE-AFFECTED NODE)
                                                           :ANY))))))
                                  (T (SETF (NODE-EFFECTS NODE)
                                           :ANY)
                                     (SETF (NODE-AFFECTED NODE)
                                           :ANY)))))
            (OTHERWISE
               (SETF (NODE-EFFECTS NODE)
                     :ANY)
               (SETF (NODE-AFFECTED NODE)
                     :ANY))))
     ;; For each argument, analyze it.
     (EFFECTS-ANALYZE (CAR ARGS))))
```

```
(DEFUN EFFECTS-ANALYZE-LIST (NODE LIST)

;;;

   (DO ((STMTS LIST (CDR STMTS))
        (EFFECTS :NONE (EFFECTS-UNION EFFECTS (NODE-EFFECTS (CAR STMTS))))
        (AFFECTED :NONE (EFFECTS-UNION AFFECTED (NODE-AFFECTED (CAR STMTS)))))
       ((NULL STMTS)
        (SETF (NODE-EFFECTS NODE)
              EFFECTS)
        (SETF (NODE-AFFECTED NODE)
              AFFECTED))
     ;; Analyze each statement.
     (EFFECTS-ANALYZE (CAR STMTS))))
```

```
(DEFUN EFFECTS-REPRESENTATION (VAR)
```

;;; Give a VARIABLE, return the representation of what is effected by a SETQ. Lexical variables are represented by themselves because they're unique
;;; in the tree, but specials and globals must be represented by the name, since they aren't unique.

```
   (ECASE (VARIABLE-KIND VAR)
       (:FUNCTION :NONE)
       (:VARIABLE (ECASE (VARIABLE-SCOPE VAR)
                      ((:SPECIAL :GLOBAL) (LIST (VARIABLE-NAME VAR)))
                      ((:LEXICAL) (LIST VAR))))))
```

;; Testing analysis

```
(DEFUN TEST-ANALYSIS (FN)
   (LET ((TREE (TEST-ALPHA-2 FN)))
        (UNWIND-PROTECT
            (PRINT-TREE (ANALYZE-TREE TREE :ALL))
            (RELEASE-TREE TREE))))
```

;; Arrange to use the proper compiler.

```
(IL:PUTPROPS IL:XCLC-ANALYZE IL:FILETYPE COMPILE-FILE)
```

;; Arrange for the proper makefile environment

```
(IL:PUTPROPS IL:XCLC-ANALYZE IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE (DEFPACKAGE "COMPILER"
                                                                                    (:USE "LISP" "XCL"))))

(IL:PUTPROPS IL:XCLC-ANALYZE IL:COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1990))
```

FUNCTION INDEX

```
ANALYZE-TREE .....................1    EFFECTS-ANALYZE-PROGV ............4    ENV-ANALYZE-LAMBDA ...............2
EFFECTS-ANALYZE ..................3    EFFECTS-ANALYZE-RETURN ...........5    ENV-ANALYZE-LITERAL ..............2
EFFECTS-ANALYZE-ANY-CALL .........5    EFFECTS-ANALYZE-SETQ .............5    ENV-ANALYZE-MV-CALL ..............2
EFFECTS-ANALYZE-BLOCK ............3    EFFECTS-ANALYZE-TAGBODY ..........5    ENV-ANALYZE-MV-PROG1 .............2
EFFECTS-ANALYZE-CALL .............3    EFFECTS-ANALYZE-THROW ............5    ENV-ANALYZE-OPCODES ..............2
EFFECTS-ANALYZE-CATCH ............3    EFFECTS-ANALYZE-UNWIND-PROTECT ...5    ENV-ANALYZE-PROGN ................2
EFFECTS-ANALYZE-GO ...............3    EFFECTS-ANALYZE-VAR-REF ..........5    ENV-ANALYZE-PROGV ................2
EFFECTS-ANALYZE-IF ...............3    EFFECTS-REPRESENTATION ...........7    ENV-ANALYZE-RETURN ...............2
EFFECTS-ANALYZE-LABELS ...........3    EFFECTS-UNION ....................3    ENV-ANALYZE-SETQ .................2
EFFECTS-ANALYZE-LAMBDA ...........4    ENV-ANALYZE ......................1    ENV-ANALYZE-TAGBODY ..............2
EFFECTS-ANALYZE-LIST .............6    ENV-ANALYZE-BLOCK ................1    ENV-ANALYZE-THROW ................2
EFFECTS-ANALYZE-LITERAL ..........4    ENV-ANALYZE-CALL .................2    ENV-ANALYZE-UNWIND-PROTECT .......2
EFFECTS-ANALYZE-MV-CALL ..........4    ENV-ANALYZE-CATCH ................2    ENV-ANALYZE-VAR-REF ..............3
EFFECTS-ANALYZE-MV-PROG1 .........4    ENV-ANALYZE-GO ...................2    REMOVE-EFFECT ....................3
EFFECTS-ANALYZE-OPCODES ..........4    ENV-ANALYZE-IF ...................2    TEST-ANALYSIS ....................7
EFFECTS-ANALYZE-PROGN ............4    ENV-ANALYZE-LABELS ...............2
```

PROPERTY INDEX

```
IL:XCLC-ANALYZE ..................7
```

VARIABLE INDEX

```
*REDO-FLAG* ......................1
```