

File created: 16-May-90 13:26:57 {DSK}<usr>local>lde>lispcore>sources>CMLLIST.;2

changes to: (IL:VARS IL:CMLLISTCOMS)

previous date: 23-May-88 20:23:20 {DSK}<usr>local>lde>lispcore>sources>CMLLIST.;1

Read Table: XCL

Package: LISP

Format: XCCS

; Copyright (c) 1985, 1986, 1987, 1988, 1990 by Venue & Xerox Corporation. All rights reserved.

(IL:RPAQQ IL:CMLLISTCOMS
(

::: CMLLIST. Common Lisp Lists Covers all of chapter 15

(IL:COMS

:: Section 15.1 Conses.

:: CAR, CDR, ..., CDDDDR (all functions on pages 262-263) are shared with Interlisp.

:: CONS is shared with Interlisp.

(IL:COMS (IL:FUNCTIONS %SIMPLE-TREE-EQUAL %COMPLEX-TREE-EQUAL)
(IL:FUNCTIONS TREE-EQUAL))

(IL:COMS

:: Section 15.2 Lists.

(IL:FUNCTIONS ENDP LIST-LENGTH)

(IL:COMS (IL:FUNCTIONS NTH %SET-NTH)

(IL:SETFS NTH)

(XCL:OPTIMIZERS NTH)

; To be compatible with old compiled code

(IL:DECLARE\ : IL:DOCOPY IL:DONTEVAL@LOAD IL:DONTEVAL@COMPILE (IL:P (IL:MOVD

'%SET-NTH

'IL:%SETNTH)))

(IL:FUNCTIONS FIRST SECOND THIRD FOURTH FIFTH SIXTH SEVENTH EIGHTH NINTH TENTH REST)

(IL:COMS (IL:FUNCTIONS NTHCDR)

(XCL:OPTIMIZERS NTHCDR))

:: LAST, LIST, and LIST* are shared with Interlisp.

(IL:FUNCTIONS MAKE-LIST)

:: Common Lisp APPEND is different from Interlisp APPEND because Interlisp APPEND copies its last arg while Common Lisp

:: APPEND does not. See page 268 of the silver book.

(IL:COMS (IL:FUNCTIONS %APPEND)

(IL:FNS APPEND))

(IL:FUNCTIONS COPY-LIST COPY-ALIST COPY-TREE REVAPPEND)

:: NCONC is shared with Interlisp.

(IL:FUNCTIONS NRECONC)

:: CL:PUSH and CL:PUSHNEW are macros defined elsewhere. POP is shared with Interlisp.

(IL:FUNCTIONS BUTLAST NBTLAST LDIFF))

(IL:COMS

:: Section 15.3 Alteration of List Structure.

:: RPLACA, and RPLACD are shared with Interlisp.

)

(IL:COMS

:: Section 15.4 Substitution of Expressions.

(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE (IL:FUNCTIONS %SUBST-MACRO %NSUBST-MACRO))

(IL:COMS (IL:FUNCTIONS %SIMPLE-SUBST %COMPLEX-SUBST %SUBST-IF %SUBST-IF-NOT)

(IL:FUNCTIONS SUBST SUBST-IF SUBST-IF-NOT))

(IL:COMS (IL:FUNCTIONS %SIMPLE-NSUBST %COMPLEX-NSUBST %NSUBST-IF %NSUBST-IF-NOT)

(IL:FUNCTIONS NSUBST NSUBST-IF NSUBST-IF-NOT))

(IL:COMS (IL:FUNCTIONS %SIMPLE-SUBLIS %COMPLEX-SUBLIS)

(IL:FUNCTIONS SUBLIS))

(IL:COMS (IL:FUNCTIONS %SIMPLE-NSUBLIS %COMPLEX-NSUBLIS)

(IL:FUNCTIONS NSUBLIS))

(IL:COMS

:: Section 15.5 Usng Lists as Sets.

:: Utilities

(IL:COMS (IL:FUNCTIONS %EQCODEP)

:: used in various optimizers

(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE (IL:FUNCTIONS %CONSTANT-FUNCTION
%CONSTANT-EXPRESSION))

(IL:COMS (IL:FUNCTIONS %SIMPLE-MEMBER %COMPLEX-MEMBER)

(IL:FUNCTIONS MEMBER MEMBER-IF MEMBER-IF-NOT)

(XCL:OPTIMIZERS MEMBER)

(IL:PROP IL:DOPVAL %SIMPLE-MEMBER))

:: TAILP is shared with Interlisp.

```
(IL:FUNCTIONS ADJOIN)
(XCL:OPTIMIZERS ADJOIN)
(IL:FUNCTIONS UNION NUNION)
(IL:FUNCTIONS INTERSECTION NINTERSECTION)
(IL:FUNCTIONS SET-DIFFERENCE NSET-DIFFERENCE)
(IL:FUNCTIONS SET-EXCLUSIVE-OR NSET-EXCLUSIVE-OR)
(IL:FUNCTIONS SUBSETP))
(IL:COMS
  ;; Section 15.6 Association Lists.
  (IL:FUNCTIONS ACONS)
  (IL:FUNCTIONS PAIRLIS)
  (IL:COMS (IL:FUNCTIONS %SIMPLE-ASSOC %COMPLEX-ASSOC)
    (IL:FUNCTIONS ASSOC ASSOC-IF ASSOC-IF-NOT)
    (XCL:OPTIMIZERS ASSOC)
    (IL:PROP IL:DOPVAL %SIMPLE-ASSOC))
  (IL:COMS (IL:FUNCTIONS %SIMPLE-RASSOC %COMPLEX-RASSOC)
    (IL:FUNCTIONS RASSOC RASSOC-IF RASSOC-IF-NOT)))
(IL:COMS
  ;; Section 7.8.4 Mapping
  (IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE (IL:FUNCTIONS %MIN-LIST-LENGTH
    %FILL-SLICE-FROM-LISTS))
  (IL:COMS
    ;; Utilities
    (IL:FUNCTIONS %LIST-MAP-OPTIMIZER %LIST-COLLECT))
    (IL:COMS (IL:FUNCTIONS %MAPCAR-SINGLE %MAPCAR-MULTIPLE)
      (IL:FUNCTIONS MAPCAR)
      (XCL:OPTIMIZERS MAPCAR))
    (IL:COMS (IL:FUNCTIONS %MAPLIST-SINGLE %MAPLIST-MULTIPLE)
      (IL:FUNCTIONS MAPLIST)
      (XCL:OPTIMIZERS MAPLIST))
    (IL:COMS (IL:FUNCTIONS %MAPC-SINGLE %MAPC-MULTIPLE)
      (IL:FUNCTIONS MAPC)
      (XCL:OPTIMIZERS MAPC))
    (IL:COMS (IL:FUNCTIONS %MAPL-SINGLE %MAPL-MULTIPLE)
      (IL:FUNCTIONS MAPL)
      (XCL:OPTIMIZERS MAPL))
    (IL:COMS (IL:FUNCTIONS %MAPCAN-SINGLE %MAPCAN-MULTIPLE)
      (IL:FUNCTIONS MAPCAN)
      (XCL:OPTIMIZERS MAPCAN))
    (IL:COMS (IL:FUNCTIONS %MAPCON-SINGLE %MAPCON-MULTIPLE)
      (IL:FUNCTIONS MAPCON)
      (XCL:OPTIMIZERS MAPCON))
    (IL:COMS
      ;; optimizers for Interlisp mapping functions whose bytemacros are not visible to the pav-compiler
      ; Utility
      (IL:FUNCTIONS %EVERY-MAP-OPTIMIZER)
      (XCL:OPTIMIZERS IL:MAP IL:MAPC IL:MAPLIST IL:MAPCAR IL:MAPCON IL:MAPCONC)
      (XCL:OPTIMIZERS IL:SOME IL:EVERY IL:NOTANY IL:NOTEVERY IL:SUBSET)))
    (IL:FUNCTIONS XCL:WITH-COLLECTION)
    (IL:COMS
      ;; some people apparently still use memq
      (IL:DECLARE\ : IL:DOCOPY IL:DONTEVAL@LOAD (IL:P (IL:MOVD 'IL:FMEMB 'IL:MEMQ))))
    ;; Arrange to use the correct compiler.
    (IL:PROP IL:FILETYPE IL:CMLLIST)
    (IL:PROP IL:MAKEFILE-ENVIRONMENT IL:CMLLIST)
    (IL:DECLARE\ : IL:DONTEVAL@LOAD IL:DOEVAL@COMPILE IL:DONTCOPY (IL:LOCALVARS . T))
    (IL:DECLARE\ : IL:DONTEVAL@LOAD IL:DOEVAL@COMPILE IL:DONTCOPY IL:COMPILERVERS (IL:ADDVARS (IL:NLAMA)
      (IL:NLAML)
      (IL:LAMA APPEND))
    )))
  )))
```

;;; CMLLIST. Common Lisp Lists Covers all of chapter 15

;; Section 15.1 Conses.

;; CAR, CDR, ..., CDDDDR (all functions on pages 262-263) are shared with Interlisp.

;; CONS is shared with Interlisp.

```
(DEFUN %SIMPLE-TREE-EQUAL (X Y)
  (IF (AND (CONSP X)
    (CONSP Y))
    (AND (%SIMPLE-TREE-EQUAL (CAR X)
      (CAR Y))
      (%SIMPLE-TREE-EQUAL (CDR X)
        (CDR Y)))
    (EQL X Y)))
```

```
(DEFUN %COMPLEX-TREE-EQUAL (X Y TEST TEST-NOT-P)
```

```
(COND
  ((CONSP X)
   (AND (CONSP Y)
        (%COMPLEX-TREE-EQUAL (CAR X)
                              (CAR Y)
                              TEST TEST-NOT-P)
         (%COMPLEX-TREE-EQUAL (CDR X)
                              (CDR Y)
                              TEST TEST-NOT-P)))
  ((CONSP Y)
   NIL)
  (T (IF TEST-NOT-P
         (NOT (FUNCALL TEST X Y))
         (FUNCALL TEST X Y))))
```

```
(DEFUN TREE-EQUAL (X Y &KEY TEST TEST-NOT)
  (IF (AND TEST TEST-NOT)
      (ERROR "Both test and test-not supplied")
      (IF (OR TEST TEST-NOT)
          (%COMPLEX-TREE-EQUAL X Y (OR TEST TEST-NOT)
                                TEST-NOT)
          (%SIMPLE-TREE-EQUAL X Y))))
```

:: Section 15.2 Lists.

```
(DEFUN ENDP (OBJECT)
  (COND
   ((CONSP OBJECT)
    NIL)
   ((NULL OBJECT)
    T)
   ((ERROR "Not a list: ~S" OBJECT))))
```

```
(DEFUN LIST-LENGTH (LIST)
  ;; Returns the length of the given LIST or NIL if the LIST is circular.
```

```
(LET ((N 0)
      (FAST-POINTER LIST)
      (SLOW-POINTER LIST))
  (LOOP (COND
        ((NULL FAST-POINTER)
         (RETURN N))
        ((NULL (CDR FAST-POINTER))
         (RETURN (+ N 1)))
        ((AND (EQ FAST-POINTER SLOW-POINTER)
              (> N 0))
         (RETURN NIL)))
        (SETQ N (+ N 2))
        (SETQ FAST-POINTER (CDDR FAST-POINTER))
        (SETQ SLOW-POINTER (CDR SLOW-POINTER))))))
```

```
(DEFUN NTH (N LIST)
  (CAR (NTHCDR N LIST)))
```

```
(DEFUN %SET-NTH (N LIST NEW-VALUE)
  (IF (< N 0)
      (ERROR "Illegal index: ~S" N)
      (DO ((CNT N (1- CNT))
          (TAIL LIST))
          ((EQL CNT 0)
           (RPLACA TAIL NEW-VALUE)
           NEW-VALUE)
        (SETQ TAIL (CDR TAIL))
        (IF (NULL TAIL)
            (ERROR "Index out of bounds: ~S" N))))))
```

```
(DEFSETF NTH %SET-NTH)
```

```
(XCL:DEFOPTIMIZER NTH (N-ARG LIST-ARG)
  (IF (AND (TYPEP N-ARG 'FIXNUM)
           (<= 0 N-ARG 10))
      ;; The optimizer for NTHCDR will take care of the rest of this...
      `(CAR (NTHCDR ,N-ARG ,LIST-ARG))
      'COMPILER:PASS))
```

:: To be compatible with old compiled code

```
(IL:DECLARE\ : IL:DOCOPY IL:DONTEVAL@LOAD IL:DONTEVAL@COMPILE
```

```
(IL:MOVD '%SET-NTH 'IL:%SETNTH)
)
```

```
(XCL:DEFINLINE FIRST (LIST)
 (CAR LIST))
```

```
(XCL:DEFINLINE SECOND (LIST)
 (CADR LIST))
```

```
(XCL:DEFINLINE THIRD (LIST)
 (CADDR LIST))
```

```
(XCL:DEFINLINE FOURTH (LIST)
 (CADDR LIST))
```

```
(XCL:DEFINLINE FIFTH (LIST)
 (CAR (CDDDDR LIST)))
```

```
(XCL:DEFINLINE SIXTH (LIST)
 (CADR (CDDDDR LIST)))
```

```
(XCL:DEFINLINE SEVENTH (LIST)
 (CADDR (CDDDDR LIST)))
```

```
(XCL:DEFINLINE EIGHTH (LIST)
 (CADDR (CDDDDR LIST)))
```

```
(XCL:DEFINLINE NINTH (LIST)
 (CAR (CDDDDR (CDDDDR LIST))))
```

```
(XCL:DEFINLINE TENTH (LIST)
 (CADR (CDDDDR (CDDDDR LIST))))
```

```
(XCL:DEFINLINE REST (LIST)
 (CDR LIST))
```

```
(DEFUN NTHCDR (N LIST)
 (IF (< N 0)
 (ERROR "Illegal index: ~S" N)
 (LET ((TAIL LIST)
 (DOTIMES (I N TAIL)
 (SETQ TAIL (CDR TAIL))))))
```

```
(XCL:DEFOPTIMIZER NTHCDR (N-ARG LIST-ARG)
 (IF (AND (TYPEP N-ARG 'FIXNUM)
 (<= 0 N-ARG 10))
 (LET ((CDR-FORM LIST-ARG)
 (DOTIMES (I N-ARG CDR-FORM)
 (SETQ CDR-FORM (LIST 'CDR CDR-FORM))))
 'COMPILER:PASS))
```

:: LAST, LIST, and LIST* are shared with Interlisp.

```
(DEFUN MAKE-LIST (SIZE &KEY INITIAL-ELEMENT)
 (IF (< SIZE 0)
 (ERROR "Illegal size: ~S" SIZE)
 (LET ((RESULT NIL)
 (DOTIMES (I SIZE RESULT)
 (SETQ RESULT (CONS INITIAL-ELEMENT RESULT))))))
```

:: Common Lisp APPEND is different from Interlisp APPEND because Interlisp APPEND copies its last arg while Common Lisp APPEND does not. See page 268 of the silver book.

```
(DEFUN %APPEND (LIST1 LIST2)
 (IF (ATOM LIST1)
 LIST2
 (DO* ((RESULT (LIST (CAR LIST1)))
 (LIST1-TAIL (CDR LIST1)
 (CDR LIST1-TAIL))
 (RESULT-TAIL RESULT (CDR RESULT-TAIL)))
```

```

      ((ATOM LIST1-TAIL)
       (RPLACD RESULT-TAIL LIST2)
       RESULT)
      (RPLACD RESULT-TAIL (LIST (CAR LIST1-TAIL))))))

```

(IL:DEFINEQ

(APPEND

(IL:LAMBDA ARGS

; Edited 12-Jan-87 12:22 by jop

:: The result is a list that is the concatenation of the arguments. The arguments are not destroyed. Note that APPEND copies the top-level list structure of each of its arguments except the last.

```

(CASE ARGS
 (0 NIL)
 (1 (IL:ARG ARGS 1))
 (OTHERWISE (DO ((RESULT (IL:ARG ARGS ARGS))
                 (I (1- ARGS)
                   (1- I)))
                 ((EQL I 0)
                  RESULT)
                (SETQ RESULT (%APPEND (IL:ARG ARGS I)
                                       RESULT))))))

```

)

(DEFUN COPY-LIST (LIST)

```

(IF (CONSP LIST)
    (DO* ((RESULT (LIST (CAR LIST)))
         (RESULT-TAIL RESULT (CDR RESULT-TAIL))
         (LIST-TAIL (CDR LIST)
                   (CDR LIST-TAIL)))
        ((NOT (CONSP LIST-TAIL))
         (IF LIST-TAIL (RPLACD RESULT-TAIL LIST-TAIL)
                      RESULT)
         (RPLACD RESULT-TAIL (LIST (CAR LIST-TAIL))))
      LIST))

```

(DEFUN COPY-ALIST (LIST)

```

(IF (CONSP LIST)
    (DO* ((RESULT (LIST (IF (CONSP (CAR LIST))
                           (CONS (CAAR LIST)
                                  (CDAR LIST))
                           (CAR LIST))))
         (LIST-TAIL (CDR LIST)
                   (CDR LIST-TAIL))
         (RESULT-TAIL RESULT (CDR RESULT-TAIL))
         LIST-ELEMENT)
        ((NOT (CONSP LIST-TAIL))
         (IF LIST-TAIL (RPLACD RESULT-TAIL LIST-TAIL)
                      RESULT)
         (SETQ LIST-ELEMENT (CAR LIST-TAIL))
         (RPLACD RESULT-TAIL (LIST (IF (CONSP LIST-ELEMENT)
                                       (CONS (CAR LIST-ELEMENT)
                                             (CDR LIST-ELEMENT))
                                       LIST-ELEMENT))))
      LIST))

```

; Non-null terminated alist done here.

(DEFUN COPY-TREE (OBJECT)

```

(IF (CONSP OBJECT)
    (CONS (COPY-TREE (CAR OBJECT))
          (COPY-TREE (CDR OBJECT)))
    OBJECT))

```

(DEFUN REVAPPEND (X Y)

:: Returns (APPEND (REVERSE X) Y)

```

(IF (CONSP X)
    (DO ((TAIL X (CDR TAIL))
        (RESULT Y (CONS (CAR TAIL)
                        RESULT)))
        ((NULL TAIL)
         RESULT))
    Y))

```

:: NCONC is shared with Interlisp.

(DEFUN NRECONC (X Y)

```

(IF (CONSP X)
    (LET ((TAIL X)
          (RESULT Y)
          NEXT-CELL)
        (LOOP (IF (NULL TAIL)

```

```

      (RETURN RESULT))
      (SETQ TAIL (CDR (SETQ NEXT-CELL TAIL)))
      (SETQ RESULT (RPLACD NEXT-CELL RESULT))))
  Y))

```

:: CL:PUSH and CL:PUSHNEW are macros defined elsewhere. POP is shared with Interlisp.

```

(DEFUN BUTLAST (LIST &OPTIONAL (N 1))
  (IF (< N 0)
    (ERROR "Illegal n: ~s" N) ; Use IL:length because cmlist is in the init but cmlseq isn't
    (LET ((LENGTH (IL:LENGTH LIST))
          (IF (<= LENGTH N)
              NIL
              (DO* ((RESULT (LIST (CAR LIST)))
                    (LIST-TAIL (CDR LIST)
                                (CDR LIST-TAIL))
                    (RESULT-TAIL RESULT (CDR RESULT-TAIL))
                    (CNT (1- LENGTH)
                        (1- CNT)))
                  ((EQL CNT N)
                   RESULT)
                  (RPLACD RESULT-TAIL (LIST (CAR LIST-TAIL))))))))

```

```

(DEFUN NBUTLAST (LIST &OPTIONAL (N 1))
  (IF (< N 0)
    (ERROR "Illegal n: ~s" N) ; Use IL:length because cmlist is in the init but cmlseq isn't
    (LET* ((LENGTH (IL:LENGTH LIST))
           (INDEX (- LENGTH N 1)))
      (COND
        ((< INDEX 0)
         NIL)
        (T (RPLACD (NTHCDR INDEX LIST)
                   NIL)
           LIST))))

```

```

(DEFUN LDIFF (LIST SUBLIST)
  (IF (EQ LIST SUBLIST)
      NIL
      (DO* ((RESULT (LIST (CAR LIST)))
            (LIST-TAIL (CDR LIST)
                        (CDR LIST-TAIL))
            (RESULT-TAIL RESULT (CDR RESULT-TAIL))
            ((OR (NULL LIST-TAIL)
                 (EQ LIST-TAIL SUBLIST))
             RESULT)
            (RPLACD RESULT-TAIL (LIST (CAR LIST-TAIL))))))

```

:: Section 15.3 Alteration of List Structure.
 :: RPLACA, and RPLACD are shared with Interlisp.
 :: Section 15.4 Substitution of Expressions.

```
(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE
```

```

(DEFMACRO %SUBST-MACRO (TEST-FORM CAR-RECURSION-FORM CDR-RECURSION-FORM)
  `(COND
    (,TEST-FORM NEW)
    ((ATOM TREE)
     TREE)
    (T (LET ((NEW-CAR ,CAR-RECURSION-FORM)
             (NEW-CDR ,CDR-RECURSION-FORM))
         (IF (AND (EQ (CAR TREE)
                     NEW-CAR)
                 (EQ (CDR TREE)
                     NEW-CDR))
             TREE
             (CONS NEW-CAR NEW-CDR))))))

```

```

(DEFMACRO %NSUBST-MACRO (TEST-FORM RECURSION-FORM)
  `(IF , (SUBST 'TREE 'TREE-FORM TEST-FORM)
      NEW
      (LET ((TAIL TREE))
        (LOOP (IF (ATOM TAIL)
                  (RETURN TREE))
              (IF , (SUBST ' (CAR TAIL)
                            'TREE-FORM TEST-FORM)
                  (RPLACA TAIL NEW)
                  , (SUBST ' (CAR TAIL)
                            'TREE-FORM RECURSION-FORM))
              (WHEN , (SUBST ' (CDR TAIL)
                              'TREE-FORM TEST-FORM)

```

; If we replace the cdr, then we need not recurse any further

```

      (RPLACD TAIL NEW)
      (RETURN TREE))
      (SETQ TAIL (CDR TAIL))))))
)

```

```

(DEFUN %SIMPLE-SUBST (NEW OLD TREE)
  (%SUBST-MACRO (EQL OLD TREE)
    (%SIMPLE-SUBST NEW OLD (CAR TREE))
    (%SIMPLE-SUBST NEW OLD (CDR TREE))))

```

```

(DEFUN %COMPLEX-SUBST (NEW OLD TREE TEST TEST-NOT-P KEY)
  (%SUBST-MACRO (LET ((TEST-RESULT (FUNCCALL TEST OLD (IF KEY
    (FUNCCALL KEY TREE)
    TREE))))
    (IF TEST-NOT-P
      (NOT TEST-RESULT)
      TEST-RESULT))
    (%COMPLEX-SUBST NEW OLD (CAR TREE)
      TEST TEST-NOT-P KEY)
    (%COMPLEX-SUBST NEW OLD (CDR TREE)
      TEST TEST-NOT-P KEY)))

```

```

(DEFUN %SUBST-IF (NEW TEST TREE KEY)
  (%SUBST-MACRO (FUNCCALL TEST (IF KEY
    (FUNCCALL KEY TREE)
    TREE))
    (%SUBST-IF NEW TEST (CAR TREE)
      KEY)
    (%SUBST-IF NEW TEST (CDR TREE)
      KEY)))

```

```

(DEFUN %SUBST-IF-NOT (NEW TEST TREE KEY)
  (%SUBST-MACRO (NOT (FUNCCALL TEST (IF KEY
    (FUNCCALL KEY TREE)
    TREE)))
    (%SUBST-IF-NOT NEW TEST (CAR TREE)
      KEY)
    (%SUBST-IF-NOT NEW TEST (CDR TREE)
      KEY)))

```

```

(DEFUN SUBST (NEW OLD TREE &KEY (TEST 'EQL TEST-P)
  (TEST-NOT-NIL TEST-NOT-P)
  (KEY-NIL KEY-P))
  (IF (AND TEST-P TEST-NOT-P)
    (ERROR "Both test and test-not supplied"))
  (IF (OR TEST-P TEST-NOT-P KEY-P)
    (%COMPLEX-SUBST NEW OLD TREE (IF TEST-NOT-P
      TEST-NOT-TEST)
      TEST-NOT-P KEY)
    (%SIMPLE-SUBST NEW OLD TREE)))

```

```

(DEFUN SUBST-IF (NEW TEST TREE &KEY KEY)
  (%SUBST-IF NEW TEST TREE KEY))

```

```

(DEFUN SUBST-IF-NOT (NEW TEST TREE &KEY KEY)
  (%SUBST-IF-NOT NEW TEST TREE KEY))

```

```

(DEFUN %SIMPLE-NSUBST (NEW OLD TREE)
  (%NSUBST-MACRO (EQL OLD TREE-FORM)
    (%SIMPLE-NSUBST NEW OLD TREE-FORM)))

```

```

(DEFUN %COMPLEX-NSUBST (NEW OLD TREE TEST TEST-NOT-P KEY)
  (LET (TEST-RESULT)
    (%NSUBST-MACRO (PROGN (SETQ TEST-RESULT (FUNCCALL TEST OLD (IF KEY
      (FUNCCALL KEY TREE-FORM)
      TREE-FORM))))
      (IF TEST-NOT-P
        (NOT TEST-RESULT)
        TEST-RESULT))
    (%COMPLEX-NSUBST NEW OLD TREE-FORM TEST TEST-NOT-P KEY))))

```

```

(DEFUN %NSUBST-IF (NEW TEST TREE KEY)
  (%NSUBST-MACRO (FUNCCALL TEST (IF KEY
    (FUNCCALL KEY TREE-FORM)
    TREE-FORM))
    (%NSUBST-IF NEW TEST TREE-FORM KEY)))

```

```
(DEFUN %NSUBST-IF-NOT (NEW TEST TREE KEY)
  (%NSUBST-MACRO (NOT (FUNCALL TEST (IF KEY
                                     (FUNCALL KEY TREE-FORM)
                                     TREE-FORM)))
    (%NSUBST-IF-NOT NEW TEST TREE-FORM KEY)))
```

```
(DEFUN NSUBST (NEW OLD TREE &KEY (TEST 'EQL TEST-P)
                                     (TEST-NOT NIL TEST-NOT-P)
                                     (KEY NIL KEY-P))
  (IF (AND TEST-P TEST-NOT-P)
    (ERROR "Both test and test-not supplied"))
  (IF (OR TEST-P TEST-NOT-P KEY-P)
    (%COMPLEX-NSUBST NEW OLD TREE (IF TEST-NOT-P
                                       TEST-NOT
                                       TEST)
      TEST-NOT-P KEY)
    (%SIMPLE-NSUBST NEW OLD TREE)))
```

```
(DEFUN NSUBST-IF (NEW TEST TREE &KEY KEY)
  (%NSUBST-IF NEW TEST TREE KEY))
```

```
(DEFUN NSUBST-IF-NOT (NEW TEST TREE &KEY KEY)
  (%NSUBST-IF-NOT NEW TEST TREE KEY))
```

```
(DEFUN %SIMPLE-SUBLIS (A-LIST TREE)
  (LET ((PAIR (%SIMPLE-ASSOC TREE A-LIST)))
    (COND
      (PAIR (CDR PAIR))
      ((ATOM TREE)
       TREE)
      (T (LET ((NEW-CAR (%SIMPLE-SUBLIS A-LIST (CAR TREE)))
                (NEW-CDR (%SIMPLE-SUBLIS A-LIST (CDR TREE))))
          (IF (AND (EQ (CAR TREE)
                       NEW-CAR)
                  (EQ (CDR TREE)
                       NEW-CDR))
              TREE
              (CONS NEW-CAR NEW-CDR)))))))
```

```
(DEFUN %COMPLEX-SUBLIS (A-LIST TREE TEST TEST-NOT-P KEY)
  (LET ((PAIR (%COMPLEX-ASSOC (IF KEY
                                (FUNCALL KEY TREE)
                                TREE)
    A-LIST TEST TEST-NOT-P NIL)))
    (COND
      (PAIR (CDR PAIR))
      ((ATOM TREE)
       TREE)
      (T (LET ((NEW-CAR (%COMPLEX-SUBLIS A-LIST (CAR TREE)
                                         TEST TEST-NOT-P KEY))
                (NEW-CDR (%COMPLEX-SUBLIS A-LIST (CDR TREE)
                                         TEST TEST-NOT-P KEY)))
          (IF (AND (EQ (CAR TREE)
                       NEW-CAR)
                  (EQ (CDR TREE)
                       NEW-CDR))
              TREE
              (CONS NEW-CAR NEW-CDR)))))))
```

```
(DEFUN SUBLIS (A-LIST TREE &KEY (TEST 'EQL TEST-P)
                                     (TEST-NOT NIL TEST-NOT-P)
                                     (KEY NIL KEY-P))
  (IF (AND TEST-P TEST-NOT-P)
    (ERROR "Both test and test-not supplied"))
  (IF (OR TEST-P TEST-NOT-P KEY-P)
    (%COMPLEX-SUBLIS A-LIST TREE (IF TEST-NOT-P
                                       TEST-NOT
                                       TEST)
      TEST-NOT-P KEY)
    (%SIMPLE-SUBLIS A-LIST TREE)))
```

```
(DEFUN %SIMPLE-NSUBLIS (A-LIST TREE)
  (LET ((PAIR NIL))
    (IF (SETQ PAIR (%SIMPLE-ASSOC TREE A-LIST))
        (CDR PAIR)
        (LET ((TAIL TREE))
          (LOOP (IF (ATOM TAIL)
                    (RETURN TREE)))))))
```



```
(IF (SETQ PAIR (%SIMPLE-ASSOC (CAR TAIL)
                             A-LIST))
    (RPLACA TAIL (CDR PAIR))
    (%SIMPLE-NSUBLIS A-LIST (CAR TAIL)))
(WHEN (SETQ PAIR (%SIMPLE-ASSOC (CDR TAIL)
                             A-LIST))
      (RPLACD TAIL (CDR PAIR))
      (RETURN TREE))
(SETQ TAIL (CDR TAIL))))))
```

```
(DEFUN %COMPLEX-NSUBLIS (A-LIST TREE TEST TEST-NOT-P KEY)
  (LET ((PAIR NIL))
    (IF (SETQ PAIR (%COMPLEX-ASSOC (IF KEY
                                    (FUNCALL KEY TREE)
                                    TREE)
                                  A-LIST TEST TEST-NOT-P NIL))
        (CDR PAIR)
        (LET ((TAIL TREE))
          (LOOP (IF (ATOM TAIL)
                   (RETURN TREE))
                (IF (SETQ PAIR (%COMPLEX-ASSOC (IF KEY
                                                (FUNCALL KEY (CAR TAIL))
                                                (CAR TAIL))
                                              A-LIST TEST TEST-NOT-P NIL))
                    (RPLACA TAIL (CDR PAIR))
                    (%COMPLEX-NSUBLIS A-LIST (CAR TAIL)
                                       TEST TEST-NOT-P KEY))
                (WHEN (SETQ PAIR (%COMPLEX-ASSOC (IF KEY
                                                  (FUNCALL KEY (CDR TAIL))
                                                  (CDR TAIL))
                                                A-LIST TEST TEST-NOT-P NIL))
                    (RPLACD TAIL (CDR PAIR))
                    (RETURN TREE))
                (SETQ TAIL (CDR TAIL))))))
```

```
(DEFUN NSUBLIS (A-LIST TREE &KEY (TEST 'EQL TEST-P)
              (TEST-NOT NIL TEST-NOT-P)
              (KEY NIL KEY-P))
  (IF (AND TEST-P TEST-NOT-P)
      (ERROR "Both test and test-not supplied"))
  (IF (OR TEST-P TEST-NOT-P KEY-P)
      (%COMPLEX-NSUBLIS A-LIST TREE (IF TEST-NOT-P
                                         TEST-NOT
                                         TEST)
                        TEST-NOT-P KEY)
      (%SIMPLE-NSUBLIS A-LIST TREE)))
```

:: Section 15.5 Usng Lists as Sets.

:: Utilities

```
(DEFUN %EQCODEP (TESTFN KNOWNFN)
  ;; KNOWNFN is a symbol (like 'eq), and TESTFN is either a symbol or a compiled closure object. Tests if TESTFN represents the "same" function as
  ;; KNOWNFN.
  (OR (EQ TESTFN KNOWNFN)
      (AND (TYPEP TESTFN 'IL:COMPILED-CLOSURE)
           (TYPEP KNOWNFN 'SYMBOL)
           (EQ (IL:|fetch| (IL:COMPILED-CLOSURE IL:FNHEADER) IL:|of| TESTFN)
              (IL:|fetch| (SYMBOL IL:DEFPONTER) IL:|of| KNOWNFN)))))
```

:: used in various optimizers

```
(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE
```

```
(DEFMACRO %CONSTANT-FUNCTION (FN-ARG)
  `(OR (CAR (IL:CONSTANTEXPRESSIONP ,FN-ARG))
      (AND (CONSP ,FN-ARG)
           (EQ (CAR ,FN-ARG)
               'FUNCTION)
           (CADR ,FN-ARG))))
```

```
(DEFMACRO %CONSTANT-EXPRESSION (EXPR)
  `(CAR (IL:CONSTANTEXPRESSIONP ,EXPR))
)
```

```
(DEFUN %SIMPLE-MEMBER (ITEM LIST)
  (IF (OR (SYMBOLP ITEM)
          (TYPEP ITEM 'FIXNUM)
          (CHARACTERP ITEM))
      (IL:FMEMB ITEM LIST)
```

; Can use the eq opcode

```
(DO ((TAIL LIST (CDR TAIL)))
  ((OR (NULL TAIL)
    (EQL ITEM (CAR TAIL)))
  TAIL)))
```

```
(DEFUN %COMPLEX-MEMBER (ITEM LIST TEST TEST-NOT-P KEY)
  (IF TEST-NOT-P
    (IF KEY
      (DO ((TAIL LIST (CDR TAIL))
        TEST-RESULT)
        ((OR (NULL TAIL)
          (NOT (FUNCALL TEST ITEM (FUNCALL KEY (CAR TAIL))))))
        TAIL))
      (DO ((TAIL LIST (CDR TAIL))
        TEST-RESULT)
        ((OR (NULL TAIL)
          (NOT (FUNCALL TEST ITEM (CAR TAIL))))
        TAIL)))
    (IF KEY
      (COND
        ((%EQCODEP TEST 'EQL)
          (DO ((TAIL LIST (CDR TAIL))
            TEST-RESULT)
            ((OR (NULL TAIL)
              (EQL ITEM (FUNCALL KEY (CAR TAIL))))))
            TAIL)))
        ((%EQCODEP TEST 'EQ)
          (DO ((TAIL LIST (CDR TAIL))
            TEST-RESULT)
            ((OR (NULL TAIL)
              (EQ ITEM (FUNCALL KEY (CAR TAIL))))))
            TAIL)))
        ((%EQCODEP TEST 'EQUAL)
          (DO ((TAIL LIST (CDR TAIL))
            TEST-RESULT)
            ((OR (NULL TAIL)
              (EQUAL ITEM (FUNCALL KEY (CAR TAIL))))))
            TAIL)))
        (T (DO ((TAIL LIST (CDR TAIL))
          TEST-RESULT)
          ((OR (NULL TAIL)
            (FUNCALL TEST ITEM (FUNCALL KEY (CAR TAIL))))))
          TAIL))))
      (COND
        ((%EQCODEP TEST 'EQUAL)
          (DO ((TAIL LIST (CDR TAIL))
            TEST-RESULT)
            ((OR (NULL TAIL)
              (EQUAL ITEM (CAR TAIL))))))
            TAIL)))
        ((%EQCODEP TEST 'EQ)
          (IL:FMEMB ITEM LIST))
        ((%EQCODEP TEST 'EQL)
          (%SIMPLE-MEMBER ITEM LIST))
        (T (DO ((TAIL LIST (CDR TAIL))
          TEST-RESULT)
          ((OR (NULL TAIL)
            (FUNCALL TEST ITEM (CAR TAIL))))
          TAIL))))))
```

```
(DEFUN MEMBER (ITEM LIST &KEY (TEST 'EQL TEST-P)
  (TEST-NOT NIL TEST-NOT-P)
  (KEY NIL KEY-P))
  (IF (AND TEST-P TEST-NOT-P)
    (ERROR "Both test and test-not supplied"))
  (IF (OR TEST-P TEST-NOT-P KEY-P)
    (%COMPLEX-MEMBER ITEM LIST (IF TEST-NOT-P
      TEST-NOT
      TEST)
      TEST-NOT-P KEY)
    (%SIMPLE-MEMBER ITEM LIST)))
```

```
(DEFUN MEMBER-IF (PREDICATE LIST &KEY KEY)
  (DO ((TAIL LIST (CDR TAIL))
    ITEM)
    ((OR (NULL TAIL)
      (PROGN (SETQ ITEM (IF KEY
        (FUNCALL KEY (CAR TAIL))
        (CAR TAIL))))
      (FUNCALL PREDICATE ITEM)))
    TAIL)))
```

```
(DEFUN MEMBER-IF-NOT (PREDICATE LIST &KEY KEY)
```

```
(DO ((TAIL LIST (CDR TAIL))
    ITEM)
  ((OR (NULL TAIL)
       (PROGN (SETQ ITEM (IF KEY
                             (FUNCALL KEY (CAR TAIL))
                             (CAR TAIL)))
              (NOT (FUNCALL PREDICATE ITEM))))
   TAIL)))
```

```
(XCL:DEFOPTIMIZER MEMBER (ITEM LIST &KEY (TEST 'EQL TEST-P)
                                     (TEST-NOT NIL TEST-NOT-P)
                                     (KEY NIL KEY-P))
  ;; optimize the simple cases
  (LET ((CONSTANT-ITEM (%CONSTANT-EXPRESSION ITEM))
        (CONSTANT-LIST (%CONSTANT-EXPRESSION LIST))
        (CONSTANT-TEST (%CONSTANT-FUNCTION TEST)))
    (COND
      ((OR (AND (EQ CONSTANT-TEST 'EQ)
                (NULL TEST-NOT-P)
                (NULL KEY-P))
           (AND (EQ CONSTANT-TEST 'EQL)
                (NULL TEST-NOT-P)
                (NULL KEY-P)
                (OR (AND CONSTANT-ITEM (TYPEP CONSTANT-ITEM 'SYMBOL))
                    (AND CONSTANT-LIST (CONSP CONSTANT-LIST)
                        (EVERY #'(LAMBDA (X)
                                (TYPEP X 'SYMBOL))
                               CONSTANT-LIST))))))
        ;; Use the eq opcode
        `(IL:FMEMB ,ITEM ,LIST))
      ((AND (EQ CONSTANT-TEST 'EQL)
            (NULL TEST-NOT-P)
            (NULL KEY-P))
        `(%SIMPLE-MEMBER ,ITEM ,LIST))
      (T 'COMPILER:PASS)))
```

```
(IL:PUTPROPS %SIMPLE-MEMBER IL:DOPVAL (2 IL:CMLMEMBER))
```

;; TAILP is shared with Interlisp.

```
(DEFUN ADJOIN (ITEM LIST &KEY (TEST 'EQL TEST-P)
                                 (TEST-NOT NIL TEST-NOT-P)
                                 (KEY NIL KEY-P))
  ;; Add item to list unless it is already a member
  (IF (AND TEST-P TEST-NOT-P)
      (ERROR "Both test and test-not supplied"))
  (IF (NOT (IF (OR TEST-P TEST-NOT-P KEY-P)
               (%COMPLEX-MEMBER (IF KEY
                                   (FUNCALL KEY ITEM)
                                   ITEM)
                                LIST
                                (IF TEST-NOT-P
                                    TEST-NOT-TEST)
                                TEST-NOT-P KEY)
               (%SIMPLE-MEMBER ITEM LIST)))
      (CONS ITEM LIST)
      LIST))
```

; Adjoin applies key to item (pg. 276)

```
(XCL:DEFOPTIMIZER ADJOIN (ITEM LIST &KEY (TEST 'EQL TEST-P)
                                     (TEST-NOT NIL TEST-NOT-P)
                                     (KEY NIL KEY-P))
  (LET ((CONSTANT-TEST (%CONSTANT-FUNCTION TEST)))
    ;; take advantage of microcode support for list membership tests
    ;; note: PUSHNEW expands to ADJOIN & is the main reason we care that ADJOIN is fast.
    (IF (AND (OR (EQ CONSTANT-TEST 'EQ)
                 (EQ CONSTANT-TEST 'EQL))
            (NOT TEST-NOT-P)
            (NOT KEY-P))
        (LET ((ITEMVAR (GENTEMP))
              (LISTVAR (GENTEMP)))
          ` (LET ((,ITEMVAR ,ITEM)
                  (,LISTVAR ,LIST))
              (IF (, (CASE CONSTANT-TEST
                      (EQ 'IL:FMEMB)
                      (EQL '%SIMPLE-MEMBER))
                  ,ITEMVAR
                  ,LISTVAR
                  ,LISTVAR
```

(CONS ,ITEMVAR ,LISTVAR)))
'COMPILER:PASS))

(DEFUN UNION (LIST1 LIST2 &KEY (TEST 'EQL TEST-P)
(TEST-NOT NIL TEST-NOT-P)
(KEY NIL KEY-P))

:: Returns the union of LIST1 and LIST2.

(IF (AND TEST-P TEST-NOT-P)
(ERROR "Both test and test-not supplied"))
(LET ((LIST1-EXTRAS NIL)
(LIST1-EXTRAS-TAIL NIL))
(IF (OR TEST-P TEST-NOT-P KEY-P)
(LET ((LOOP-TEST (IF TEST-NOT-P
TEST-NOT
TEST)))
(DOLIST (ELEMENT LIST1)
(IF (NOT (%COMPLEX-MEMBER (IF KEY
(FUNCALL KEY ELEMENT)
ELEMENT)
LIST2 LOOP-TEST TEST-NOT-P KEY))
(%LIST-COLLECT LIST1-EXTRAS LIST1-EXTRAS-TAIL (LIST ELEMENT))))))
(DOLIST (ELEMENT LIST1)
(IF (NOT (%SIMPLE-MEMBER ELEMENT LIST2))
(%LIST-COLLECT LIST1-EXTRAS LIST1-EXTRAS-TAIL (LIST ELEMENT))))))
(COND
(LIST1-EXTRAS (RPLACD LIST1-EXTRAS-TAIL LIST2)
LIST1-EXTRAS)
(T LIST2))))

(DEFUN NUNION (LIST1 LIST2 &KEY (TEST 'EQL TEST-P)
(TEST-NOT NIL TEST-NOT-P)
(KEY NIL KEY-P))

(IF (AND TEST-P TEST-NOT-P)
(ERROR "Both test and test-not supplied"))
(LET ((LIST1-EXTRAS NIL)
(LIST1-EXTRAS-TAIL NIL))
(IF (OR TEST-P TEST-NOT-P KEY-P)
(DO ((LIST1-TAIL LIST1 (CDR LIST1-TAIL))
(LOOP-TEST (IF TEST-NOT-P
TEST-NOT
TEST)))
(NULL LIST1-TAIL))
(IF (NOT (%COMPLEX-MEMBER (IF KEY
(FUNCALL KEY (CAR LIST1-TAIL))
(CAR LIST1-TAIL))
LIST2 LOOP-TEST TEST-NOT-P KEY))
(%LIST-COLLECT LIST1-EXTRAS LIST1-EXTRAS-TAIL LIST1-TAIL)))
(DO ((LIST1-TAIL LIST1 (CDR LIST1-TAIL))
(NULL LIST1-TAIL))
(IF (NOT (%SIMPLE-MEMBER (CAR LIST1-TAIL)
LIST2))
(%LIST-COLLECT LIST1-EXTRAS LIST1-EXTRAS-TAIL LIST1-TAIL))))))
(COND
(LIST1-EXTRAS (RPLACD LIST1-EXTRAS-TAIL LIST2)
LIST1-EXTRAS)
(T LIST2))))

(DEFUN INTERSECTION (LIST1 LIST2 &KEY (TEST 'EQL TEST-P)
(TEST-NOT NIL TEST-NOT-P)
(KEY NIL KEY-P))

(IF (AND TEST-P TEST-NOT-P)
(ERROR "Both test and test-not supplied"))
(LET ((RESULT NIL)
(RESULT-TAIL NIL))
(IF (OR TEST-P TEST-NOT-P KEY-P)
(LET ((LOOP-TEST (IF TEST-NOT-P
TEST-NOT
TEST)))
(DOLIST (ELEMENT LIST1)
(IF (%COMPLEX-MEMBER (IF KEY
(FUNCALL KEY ELEMENT)
ELEMENT)
LIST2 LOOP-TEST TEST-NOT-P KEY)
(%LIST-COLLECT RESULT RESULT-TAIL (LIST ELEMENT))))))
(DOLIST (ELEMENT LIST1)
(IF (%SIMPLE-MEMBER ELEMENT LIST2)
(%LIST-COLLECT RESULT RESULT-TAIL (LIST ELEMENT))))))
RESULT))

(DEFUN NINTERSECTION (LIST1 LIST2 &KEY (TEST 'EQL TEST-P)
(TEST-NOT NIL TEST-NOT-P)
(KEY NIL KEY-P))

```
(IF (AND TEST-P TEST-NOT-P)
  (ERROR "Both test and test-not supplied"))
(LET ((RESULT NIL)
      (RESULT-TAIL NIL))
  (IF (OR TEST-P TEST-NOT-P KEY-P)
    (DO ((LIST1-TAIL LIST1 (CDR LIST1-TAIL))
        (LOOP-TEST (IF TEST-NOT-P
                       TEST-NOT
                       TEST)))
      (TEMP)
      ((NULL LIST1-TAIL))
      (IF (%COMPLEX-MEMBER (IF KEY
                              (FUNCCALL KEY (CAR LIST1-TAIL))
                              (CAR LIST1-TAIL))
          LIST2 LOOP-TEST TEST-NOT-P KEY)
        (%LIST-COLLECT RESULT RESULT-TAIL LIST1-TAIL)))
    (DO ((LIST1-TAIL LIST1 (CDR LIST1-TAIL))
        (TEMP)
        ((NULL LIST1-TAIL))
        (IF (%SIMPLE-MEMBER (CAR LIST1-TAIL)
                              LIST2)
          (%LIST-COLLECT RESULT RESULT-TAIL LIST1-TAIL))))
      (COND
        (RESULT (RPLACD RESULT-TAIL NIL)
                RESULT)
        (T RESULT))))
```

```
(DEFUN SET-DIFFERENCE (LIST1 LIST2 &KEY (TEST 'EQL TEST-P)
                                     (TEST-NOT NIL TEST-NOT-P)
                                     (KEY NIL KEY-P))
  (IF (AND TEST-P TEST-NOT-P)
    (ERROR "Both test and test-not supplied"))
  (LET ((RESULT NIL)
        (RESULT-TAIL NIL))
    (IF (OR TEST-P TEST-NOT-P KEY-P)
      (LET ((LOOP-TEST (IF TEST-NOT-P
                           TEST-NOT
                           TEST)))
        (DOLIST (ELEMENT LIST1)
          (IF (NOT (%COMPLEX-MEMBER (IF KEY
                                      (FUNCCALL KEY ELEMENT)
                                      ELEMENT)
                                    LIST2 LOOP-TEST TEST-NOT-P KEY))
            (%LIST-COLLECT RESULT RESULT-TAIL (LIST ELEMENT))))))
      (DOLIST (ELEMENT LIST1)
        (IF (NOT (%SIMPLE-MEMBER ELEMENT LIST2))
          (%LIST-COLLECT RESULT RESULT-TAIL (LIST ELEMENT))))))
    RESULT))
```

```
(DEFUN NSET-DIFFERENCE (LIST1 LIST2 &KEY (TEST 'EQL TEST-P)
                                         (TEST-NOT NIL TEST-NOT-P)
                                         (KEY NIL KEY-P))
  (IF (AND TEST-P TEST-NOT-P)
    (ERROR "Both test and test-not supplied"))
  (LET ((RESULT NIL)
        (RESULT-TAIL NIL))
    (IF (OR TEST-P TEST-NOT-P KEY-P)
      (DO ((LIST1-TAIL LIST1 (CDR LIST1-TAIL))
          (LOOP-TEST (IF TEST-NOT-P
                        TEST-NOT
                        TEST)))
        ((NULL LIST1-TAIL))
        (IF (NOT (%COMPLEX-MEMBER (IF KEY
                                      (FUNCCALL KEY (CAR LIST1-TAIL))
                                      (CAR LIST1-TAIL))
                                    LIST2 LOOP-TEST TEST-NOT-P KEY))
          (%LIST-COLLECT RESULT RESULT-TAIL LIST1-TAIL)))
      (DO ((LIST1-TAIL LIST1 (CDR LIST1-TAIL))
          ((NULL LIST1-TAIL))
          (IF (NOT (%SIMPLE-MEMBER (CAR LIST1-TAIL)
                                    LIST2))
            (%LIST-COLLECT RESULT RESULT-TAIL LIST1-TAIL))))
        (COND
          (RESULT (RPLACD RESULT-TAIL NIL)
                  RESULT)
          (T RESULT))))
```

```
(DEFUN SET-EXCLUSIVE-OR (LIST1 LIST2 &KEY (TEST 'EQL TEST-P)
                                           (TEST-NOT NIL TEST-NOT-P)
                                           (KEY NIL KEY-P))
  (IF (AND TEST-P TEST-NOT-P)
    (ERROR "Both test and test-not supplied"))
  (LET ((RESULT NIL)
        (RESULT-TAIL NIL))
```

```

(COND
  ((OR TEST-P TEST-NOT-P KEY-P)
   (LET ((LOOP-TEST (IF TEST-NOT-P
                        TEST-NOT
                        TEST)))
     (DOLIST (ELEMENT LIST1)
      (IF (NOT (%COMPLEX-MEMBER (IF KEY
                                  (FUNCCALL KEY ELEMENT)
                                  ELEMENT)
                                LIST2 LOOP-TEST TEST-NOT-P KEY))
          (%LIST-COLLECT RESULT RESULT-TAIL (LIST ELEMENT))))
     (DOLIST (ELEMENT LIST2)
      (IF (NOT (%COMPLEX-MEMBER (IF KEY
                                  (FUNCCALL KEY ELEMENT)
                                  ELEMENT)
                                LIST1 LOOP-TEST TEST-NOT-P KEY))
          (%LIST-COLLECT RESULT RESULT-TAIL (LIST ELEMENT))))
     RESULT))
  (T (DOLIST (ELEMENT LIST1)
   (IF (NOT (%SIMPLE-MEMBER ELEMENT LIST2))
       (%LIST-COLLECT RESULT RESULT-TAIL (LIST ELEMENT))))
   (DOLIST (ELEMENT LIST2)
   (IF (NOT (%SIMPLE-MEMBER ELEMENT LIST1))
       (%LIST-COLLECT RESULT RESULT-TAIL (LIST ELEMENT))))))
  RESULT))

```

```

(DEFUN NSET-EXCLUSIVE-OR (LIST1 LIST2 &KEY (TEST 'EQL TEST-P)
                        (TEST-NOT NIL TEST-NOT-P)
                        (KEY NIL KEY-P))
  (IF (AND TEST-P TEST-NOT-P)
      (ERROR "Both test and test-not supplied"))
  (LET ((RESULT NIL)
        (RESULT-TAIL NIL))
    (IF (OR TEST-P TEST-NOT-P KEY-P)
        (LET ((LIST1-HANDLE NIL)
              (LIST1-PREVIOUS NIL)
              (LOOP-TEST (IF TEST-NOT-P
                            TEST-NOT
                            TEST)))
          (DO ((LIST1-TAIL LIST1 (CDR LIST1-TAIL))
              (NULL LIST1-TAIL))
              (COND
                ((NOT (%COMPLEX-MEMBER (IF KEY
                                          (FUNCCALL KEY (CAR LIST1-TAIL))
                                          (CAR LIST1-TAIL))
                                        LIST2 LOOP-TEST TEST-NOT-P KEY))
                 (%LIST-COLLECT RESULT RESULT-TAIL LIST1-TAIL)
                 ; splice cell out of list1
                 (IF LIST1-PREVIOUS
                     (RPLACD LIST1-PREVIOUS (CDR LIST1-TAIL))))
                (T (IF (NULL LIST1-HANDLE)
                      (SETQ LIST1-HANDLE (SETQ LIST1-PREVIOUS LIST1-TAIL))
                      (SETQ LIST1-PREVIOUS (CDR LIST1-PREVIOUS))))))
          (DO ((LIST2-TAIL LIST2 (CDR LIST2-TAIL))
              (NULL LIST2-TAIL))
              (IF (NOT (%COMPLEX-MEMBER (IF KEY
                                          (FUNCCALL KEY (CAR LIST2-TAIL))
                                          (CAR LIST2-TAIL))
                                        LIST1-HANDLE LOOP-TEST TEST-NOT-P KEY))
                  (%LIST-COLLECT RESULT RESULT-TAIL LIST2-TAIL))))
        (LET ((LIST1-HANDLE NIL)
              (LIST1-PREVIOUS NIL))
          (DO ((LIST1-TAIL LIST1 (CDR LIST1-TAIL))
              (NULL LIST1-TAIL))
              (COND
                ((NOT (%SIMPLE-MEMBER (CAR LIST1-TAIL)
                                      LIST2))
                 (%LIST-COLLECT RESULT RESULT-TAIL LIST1-TAIL)
                 ; splice cell out of list1
                 (IF LIST1-PREVIOUS
                     (RPLACD LIST1-PREVIOUS (CDR LIST1-TAIL))))
                (T (IF (NULL LIST1-HANDLE)
                      (SETQ LIST1-HANDLE (SETQ LIST1-PREVIOUS LIST1-TAIL))
                      (SETQ LIST1-PREVIOUS (CDR LIST1-PREVIOUS))))))
          (DO ((LIST2-TAIL LIST2 (CDR LIST2-TAIL))
              (NULL LIST2-TAIL))
              (IF (NOT (%SIMPLE-MEMBER (CAR LIST2-TAIL)
                                      LIST1-HANDLE))
                  (%LIST-COLLECT RESULT RESULT-TAIL LIST2-TAIL))))
        (COND
          (RESULT (RPLACD RESULT-TAIL NIL)
                 RESULT)
          (T RESULT))))

```

```

(DEFUN SUBSETP (LIST1 LIST2 &KEY (TEST 'EQL TEST-P)

```

```

                (TEST-NOT NIL TEST-NOT-P)
                (KEY NIL KEY-P))
(IF (AND TEST-P TEST-NOT-P)
  (ERROR "Both test and test-not supplied"))
(IF (OR TEST-P TEST-NOT-P KEY-P)
  (LET ((LOOP-TEST (IF TEST-NOT-P
                        TEST-NOT
                        TEST)))
    (DOLIST (ELEMENT LIST1 T)
      (IF (NOT (%COMPLEX-MEMBER (IF KEY
                                   (FUNCALL KEY ELEMENT)
                                   ELEMENT)
                                LIST2 LOOP-TEST TEST-NOT-P KEY))
          (RETURN NIL))))))
(DOLIST (ELEMENT LIST1 T)
  (IF (NOT (%SIMPLE-MEMBER ELEMENT LIST2))
      (RETURN NIL))))

```

:: Section 15.6 Association Lists.

```

(XCL:DEFININE ACONS (KEY DATUM A-LIST)
  (CONS (CONS KEY DATUM)
        A-LIST))

```

```

(DEFUN PAIRLIS (KEYS DATA &OPTIONAL A-LIST)
  ;; Construct an association list from KEYS and DATA (adding to ALIST)

```

```

(COND
  ((AND (ENDP KEYS)
        (ENDP DATA))
   A-LIST)
  ((NOT (EQL (IL:LENGTH KEYS)
             (IL:LENGTH DATA)))
   (ERROR "Lists of unequal length: ~S and ~S" KEYS DATA))
  (T (DO* ((RESULT (LIST (CONS (CAR KEYS)
                              (CAR DATA))))
          (LAST-CONS RESULT)
          (KEYS-TAIL (CDR KEYS)
                    (CDR KEYS-TAIL))
          (DATA-TAIL (CDR DATA)
                    (CDR DATA-TAIL)))
    ((NULL KEYS-TAIL)
     (RPLACD LAST-CONS A-LIST)
     RESULT)
    (SETQ LAST-CONS (CDR (RPLACD LAST-CONS (LIST (CONS (CAR KEYS-TAIL)
                                                         (CAR DATA-TAIL))))))))))

```

; Use IL:Length since cmlist is in the init but cmlseq is not

```

(DEFUN %SIMPLE-ASSOC (ITEM A-LIST)
  (IF (OR (SYMBOLP ITEM)
          (TYPEP ITEM 'FIXNUM)
          (CHARACTERP ITEM))
      (IL:ASSOC ITEM A-LIST)
      (DO ((TAIL A-LIST (CDR TAIL))
          (PAIR)
          ((NULL TAIL)
           NIL)
          (SETQ PAIR (CAR TAIL))
          (IF (AND (CONSP PAIR)
                  (EQL ITEM (CAR PAIR))))
              (RETURN PAIR))))))

```

; Can use the eq opcode

```

(DEFUN %COMPLEX-ASSOC (ITEM A-LIST TEST TEST-NOT-P KEY)
  (IF TEST-NOT-P
      (IF KEY
          (DO ((TAIL A-LIST (CDR TAIL))
              (PAIR TEST-RESULT)
              ((NULL TAIL)
               NIL)
              (SETQ PAIR (CAR TAIL))
              (IF (AND (CONSP PAIR)
                      (NOT (FUNCALL TEST ITEM (FUNCALL KEY (CAR PAIR))))
                  (RETURN PAIR)))
              (DO ((TAIL A-LIST (CDR TAIL))
                  (PAIR TEST-RESULT)
                  ((NULL TAIL)
                   NIL)
                  (SETQ PAIR (CAR TAIL))
                  (IF (AND (CONSP PAIR)
                          (NOT (FUNCALL TEST ITEM (CAR PAIR))))
                      (RETURN PAIR))))))
          (IF KEY
              (COND
                ((%EQCODEP TEST 'EQL)

```

```

(DO ((TAIL A-LIST (CDR TAIL))
    PAIR TEST-RESULT)
  ((NULL TAIL)
   NIL)
  (SETQ PAIR (CAR TAIL))
  (IF (AND (CONSP PAIR)
           (EQL ITEM (FUNCALL KEY (CAR PAIR))))
      (RETURN PAIR)))
(%EQCODEP TEST 'EQ)
(DO ((TAIL A-LIST (CDR TAIL))
    PAIR TEST-RESULT)
  ((NULL TAIL)
   NIL)
  (SETQ PAIR (CAR TAIL))
  (IF (AND (CONSP PAIR)
           (EQ ITEM (FUNCALL KEY (CAR PAIR))))
      (RETURN PAIR)))
(%EQCODEP TEST 'EQUAL)
(DO ((TAIL A-LIST (CDR TAIL))
    PAIR TEST-RESULT)
  ((NULL TAIL)
   NIL)
  (SETQ PAIR (CAR TAIL))
  (IF (AND (CONSP PAIR)
           (EQUAL ITEM (FUNCALL KEY (CAR PAIR))))
      (RETURN PAIR)))
(T (DO ((TAIL A-LIST (CDR TAIL))
    PAIR TEST-RESULT)
  ((NULL TAIL)
   NIL)
  (SETQ PAIR (CAR TAIL))
  (IF (AND (CONSP PAIR)
           (FUNCALL TEST ITEM (FUNCALL KEY (CAR PAIR))))
      (RETURN PAIR))))))
(COND
  (%EQCODEP TEST 'EQUAL)
  (DO ((TAIL A-LIST (CDR TAIL))
    PAIR TEST-RESULT)
    ((NULL TAIL)
     NIL)
    (SETQ PAIR (CAR TAIL))
    (IF (AND (CONSP PAIR)
             (EQUAL ITEM (CAR PAIR)))
        (RETURN PAIR)))
  (%EQCODEP TEST 'EQ)
  (IL:ASSOC ITEM A-LIST))
(%EQCODEP TEST 'EQL)
(%SIMPLE-ASSOC ITEM A-LIST))
(T (DO ((TAIL A-LIST (CDR TAIL))
    PAIR TEST-RESULT)
  ((NULL TAIL)
   NIL)
  (SETQ PAIR (CAR TAIL))
  (IF (AND (CONSP PAIR)
           (FUNCALL TEST ITEM (CAR PAIR)))
      (RETURN PAIR))))))

```

```

(DEFUN ASSOC (ITEM A-LIST &KEY (TEST 'EQL TEST-P)
             (TEST-NOT-NIL TEST-NOT-P)
             (KEY-NIL KEY-P))
  (IF (AND TEST-P TEST-NOT-P)
      (ERROR "Both test and test-not supplied"))
  (IF (OR TEST-P TEST-NOT-P KEY-P)
      (%COMPLEX-ASSOC ITEM A-LIST (IF TEST-NOT-P
                                       TEST-NOT
                                       TEST)
                       TEST-NOT-P KEY)
      (%SIMPLE-ASSOC ITEM A-LIST)))

```

```

(DEFUN ASSOC-IF (PREDICATE A-LIST &KEY (KEY-NIL KEY-P))
  (IF KEY-P
      (DO ((TAIL A-LIST (CDR TAIL))
    PAIR)
        ((NULL TAIL)
         NIL)
        (SETQ PAIR (CAR TAIL))
        (IF (AND (CONSP PAIR)
                 (FUNCALL PREDICATE (FUNCALL KEY (CAR PAIR))))
            (RETURN PAIR)))
      (DO ((TAIL A-LIST (CDR TAIL))
    PAIR)
        ((NULL TAIL)
         NIL)
        (SETQ PAIR (CAR TAIL))
        (IF (AND (CONSP PAIR)
                 (FUNCALL PREDICATE (FUNCALL KEY (CAR PAIR))))
            (RETURN PAIR))))))

```



```

                (KEY NIL KEY-P))
(IF (AND TEST-P TEST-NOT-P)
  (ERROR "Both test and test-not supplied"))
(IF (OR TEST-P TEST-NOT-P KEY-P)
  (%COMPLEX-RASSOC ITEM A-LIST (IF TEST-NOT-P
                                   TEST-NOT
                                   TEST)
                    TEST-NOT-P KEY)
  (%SIMPLE-RASSOC ITEM A-LIST)))

```

```

(DEFUN RASSOC-IF (PREDICATE A-LIST &KEY (KEY NIL KEY-P))
  (IF KEY-P
    (DO ((TAIL A-LIST (CDR TAIL))
        (PAIR)
        ((NULL TAIL)
         NIL)
        (SETQ PAIR (CAR TAIL))
        (IF (AND (CONSP PAIR)
                 (FUNCALL PREDICATE (FUNCALL KEY (CDR PAIR))))
            (RETURN PAIR)))
      (DO ((TAIL A-LIST (CDR TAIL))
          (PAIR)
          ((NULL TAIL)
           NIL)
          (SETQ PAIR (CAR TAIL))
          (IF (AND (CONSP PAIR)
                   (FUNCALL PREDICATE (CDR PAIR)))
              (RETURN PAIR))))))

```

```

(DEFUN RASSOC-IF-NOT (PREDICATE A-LIST &KEY (KEY NIL KEY-P))
  (IF KEY-P
    (DO ((TAIL A-LIST (CDR TAIL))
        (PAIR)
        ((NULL TAIL)
         NIL)
        (SETQ PAIR (CAR TAIL))
        (IF (AND (CONSP PAIR)
                 (NOT (FUNCALL PREDICATE (FUNCALL KEY (CDR PAIR))))
            (RETURN PAIR)))
      (DO ((TAIL A-LIST (CDR TAIL))
          (PAIR)
          ((NULL TAIL)
           NIL)
          (SETQ PAIR (CAR TAIL))
          (IF (AND (CONSP PAIR)
                   (NOT (FUNCALL PREDICATE (CDR PAIR))))
              (RETURN PAIR))))))

```

:: Section 7.8.4 Mapping

```
(IL:DECLARE\ : IL:DONTCOPY IL:DOEVAL@COMPILE
```

```

(DEFMACRO %MIN-LIST-LENGTH (LISTS)
  `(LET ((MIN (IL:LENGTH (CAR ,LISTS)))
        (NEXT-LENGTH)
        (DOLIST (LIST (CDR ,LISTS)
                      (MIN)
                      (SETQ NEXT-LENGTH (IL:LENGTH LIST))
                      (IF (< NEXT-LENGTH MIN)
                          (SETQ MIN NEXT-LENGTH))))))

```

```

(DEFMACRO %FILL-SLICE-FROM-LISTS (LISTS ARG-SLICE ARG-TAIL-FORM)
  `(DO ((SUBSLICE ,ARG-SLICE (CDR SUBSLICE))
      (SUBLIST ,LISTS (CDR SUBLIST))
      (SOME-LIST-EMPTY NIL)
      LIST)
    ((NULL SUBLIST)
     (COND
      (SOME-LIST-EMPTY ; Ran out of entries in a list.
       NIL)
      (T ; still work to do; return it.
       ,ARG-SLICE)))
    (SETQ LIST (CAR SUBLIST))
    (SETQ SOME-LIST-EMPTY (OR SOME-LIST-EMPTY (NULL LIST)))
    (RPLACA SUBSLICE (PROG1 ,(SUBST 'LIST 'ARG-TAIL ARG-TAIL-FORM)
                                   (RPLACA SUBLIST (CDR LIST))))))
)

```

:: Utilities

```
(DEFUN %LIST-MAP-OPTIMIZER (FN LISTS &KEY TAIL-P COLLECT-P NCONC-P INC-FN NIL-RESULT-P)
```

;; Keywords INC-FN and NIL-RESULT-P are for Interlisp mapping functions

```

(LET
  ((CONSTANT-FN (COND
    ((CONSTANTP FN)
     (EVAL FN))
    ((AND (CONSP FN)
          (OR (EQ (CAR FN)
                  'FUNCTION)
              (EQ (CAR FN)
                  'IL:FUNCTION))))
     (CADR FN))))
   (CONSTANT-INC-FN (IF INC-FN
    (COND
      ((CONSTANTP INC-FN)
       (EVAL INC-FN))
      ((AND (CONSP INC-FN)
            (OR (EQ (CAR INC-FN)
                    'FUNCTION)
                (EQ (CAR INC-FN)
                    'IL:FUNCTION))))
       (EQ (CAR INC-FN)
           'IL:FUNCTION)))
      (CADR INC-FN)))
    'CDR))
  (RESULT-P (OR COLLECT-P NCONC-P)))
  (IF (AND CONSTANT-FN CONSTANT-INC-FN)
    (LET*
      ((FIRST-LIST-RETURNED-P (NOT (OR RESULT-P NIL-RESULT-P)))
        (FIRST-LIST (CAR LISTS))
        (OTHER-LISTS (CDR LISTS))
        (OTHER-SUBLISTS (DO ((LST NIL)
                            (SI-PACKAGE (FIND-PACKAGE "SI")))
                            (I 1 (1+ I))
                            (MAP-LIST OTHER-LISTS (CDR MAP-LIST)))
                          ((NULL MAP-LIST)
                           (NREVERSE LST))
                          (PUSH (INTERN (CONCATENATE 'STRING "%$$MAP-SUBLIST" (PRIN1-TO-STRING I))
                                      SI-PACKAGE)
                                LST))))
      ` (DO* (,@ (IF FIRST-LIST-RETURNED-P
                    `((SI::%$$MAP-FIRST-LIST ,FIRST-LIST)))
              (SI::%$$MAP-FIRST-SUBLIST , (IF FIRST-LIST-RETURNED-P
                                               'SI::%$$MAP-FIRST-LIST
                                               FIRST-LIST)
              (,CONSTANT-INC-FN SI::%$$MAP-FIRST-SUBLIST)
      ,@ (IF OTHER-SUBLISTS
            (MAPCAR #'(LAMBDA (SYMBOL VAR)
                      ` (,SYMBOL ,VAR (,CONSTANT-INC-FN ,SYMBOL)))
                  OTHER-SUBLISTS OTHER-LISTS))
      ,@ (IF RESULT-P
            `((SI::%$$MAP-RESULT NIL)
              (SI::%$$MAP-RESULT-TAIL NIL)
              SI::%$$MAP-ELEMENT)))
        (, (IF OTHER-SUBLISTS
              ` (OR (NULL SI::%$$MAP-FIRST-SUBLIST)
                    ,@ (MAPCAR #'(LAMBDA (SYMBOL)
                                ` (NULL ,SYMBOL))
                              OTHER-SUBLISTS))
              ` (NULL SI::%$$MAP-FIRST-SUBLIST))
          , (IF RESULT-P
                'SI::%$$MAP-RESULT
                (IF NIL-RESULT-P
                    NIL
                    'SI::%$$MAP-FIRST-LIST)))
        , (LET ((FORM ` (,CONSTANT-FN , (IF TAIL-P
                                             'SI::%$$MAP-FIRST-SUBLIST
                                             ' (CAR SI::%$$MAP-FIRST-SUBLIST))
                    ,@ (MAPCAR #'(LAMBDA (SYMBOL)
                                (IF TAIL-P
                                    SYMBOL
                                    ` (CAR ,SYMBOL)))
                              OTHER-SUBLISTS))))
              (IF RESULT-P
                ` (SETQ SI::%$$MAP-ELEMENT ,FORM)
                FORM))
        ,@ (COND
            (COLLECT-P ' ((IF SI::%$$MAP-RESULT
                              (REPLACD SI::%$$MAP-RESULT-TAIL (SETQ SI::%$$MAP-RESULT-TAIL (LIST
                                                                                               SI::%$$MAP-ELEMENT
                                                                                               )))
                          (SETQ SI::%$$MAP-RESULT (SETQ SI::%$$MAP-RESULT-TAIL (LIST SI::%$$MAP-ELEMENT
                                                                                               ))))))
            (NCONC-P ' ((IF SI::%$$MAP-RESULT-TAIL
                              (REPLACD SI::%$$MAP-RESULT-TAIL SI::%$$MAP-ELEMENT)
                              (SETQ SI::%$$MAP-RESULT SI::%$$MAP-ELEMENT))
                      (IF (CONSP SI::%$$MAP-ELEMENT)
                          (SETQ SI::%$$MAP-RESULT-TAIL (LAST SI::%$$MAP-ELEMENT))))))))
    'COMPILER:PASS)))

```

```

(DEFMACRO %LIST-COLLECT (RESULT RESULT-TAIL ITEM-FORM)
  `(IF ,RESULT
    (RPLACD ,RESULT-TAIL (SETQ ,RESULT-TAIL ,ITEM-FORM))
    (SETQ ,RESULT (SETQ ,RESULT-TAIL ,ITEM-FORM))))

(DEFUN %MAPCAR-SINGLE (FN LIST)
  (DO ((SUBLIST LIST (CDR SUBLIST))
      (RESULT NIL)
      (RESULT-TAIL NIL))
    ((NULL SUBLIST)
     RESULT)
    (%LIST-COLLECT RESULT RESULT-TAIL (LIST (FUNCALL FN (CAR SUBLIST))))))

(DEFUN %MAPCAR-MULTIPLE (FN LISTS)
  (LET ((ARG-SLICE (MAKE-LIST (IL:LENGTH LISTS))))
    (DO ((RESULT NIL)
        (RESULT-TAIL NIL)
        (CURRENT-SLICE ARG-SLICE)
        ELEMENT)
      ((NULL CURRENT-SLICE)
       RESULT)
      (SETQ CURRENT-SLICE (%FILL-SLICE-FROM-LISTS LISTS ARG-SLICE (CAR ARG-TAIL)))
      (COND
        (CURRENT-SLICE ; There is really more work to do.
         (SETQ ELEMENT (APPLY FN CURRENT-SLICE))
         (%LIST-COLLECT RESULT RESULT-TAIL (LIST ELEMENT)))))))

(DEFUN MAPCAR (FUNCTION LIST &REST MORE-LISTS)
  ;; FUNCTION must take as many arguments as there are lists provided. The result is a list such that element i is the result of applying FUNCTION to
  ;; element i of each of the argument lists.
  (IF (NULL MORE-LISTS)
      (%MAPCAR-SINGLE FUNCTION LIST)
      (%MAPCAR-MULTIPLE FUNCTION (CONS LIST MORE-LISTS))))

(XCL:DEFOPTIMIZER MAPCAR (FN &REST LISTS)
  (%LIST-MAP-OPTIMIZER FN LISTS :COLLECT-P T))

(DEFUN %MAPLIST-SINGLE (FN LIST)
  (DO ((SUBLIST LIST (CDR SUBLIST))
      (RESULT NIL)
      (RESULT-TAIL NIL))
    ((NULL SUBLIST)
     RESULT)
    (%LIST-COLLECT RESULT RESULT-TAIL (LIST (FUNCALL FN SUBLIST))))))

(DEFUN %MAPLIST-MULTIPLE (FN LISTS)
  (LET ((ARG-SLICE (MAKE-LIST (IL:LENGTH LISTS))))
    (DO ((RESULT NIL)
        (RESULT-TAIL NIL)
        (CURRENT-SLICE ARG-SLICE)
        ELEMENT)
      ((NULL CURRENT-SLICE)
       RESULT)
      (SETQ CURRENT-SLICE (%FILL-SLICE-FROM-LISTS LISTS ARG-SLICE ARG-TAIL))
      (COND
        (CURRENT-SLICE ; There is really more work to do.
         (SETQ ELEMENT (APPLY FN CURRENT-SLICE))
         (%LIST-COLLECT RESULT RESULT-TAIL (LIST ELEMENT)))))))

(DEFUN MAPLIST (FUNCTION LIST &REST MORE-LISTS)
  ;; FUNCTION must take as many arguments as there are lists provided. The result is a list such that element i is the result of applying FUNCTION to
  ;; element i of each of the argument lists.
  (IF (NULL MORE-LISTS)
      (%MAPLIST-SINGLE FUNCTION LIST)
      (%MAPLIST-MULTIPLE FUNCTION (CONS LIST MORE-LISTS))))

(XCL:DEFOPTIMIZER MAPLIST (FN &REST LISTS)
  (%LIST-MAP-OPTIMIZER FN LISTS :TAIL-P T :COLLECT-P T))

(DEFUN %MAPC-SINGLE (FN LIST)
  (DOLIST (ELEMENT LIST)
    (FUNCALL FN ELEMENT)))

```

```

(DEFUN %MAPC-MULTIPLE (FN LISTS)
  ;; MAPC for multiple lists
  (LET ((ARG-SLICE (MAKE-LIST (IL:LENGTH LISTS))))
    (DO ((SLICE ARG-SLICE)
        ((NULL SLICE))
        ;; %FILL-SLICE-FROM-LISTS returns NIL if one of the lists we're slicing thru bottoms out.
        (SETQ SLICE (%FILL-SLICE-FROM-LISTS LISTS ARG-SLICE (CAR ARG-TAIL)))
        (COND
          (SLICE (APPLY FN SLICE))))))

(DEFUN MAPC (FUNCTION LIST &REST MORE-LISTS)
  ;; FUNCTION must take as many arguments as there are lists provided.
  (IF (NULL MORE-LISTS)
      (%MAPC-SINGLE FUNCTION LIST)
      (%MAPC-MULTIPLE FUNCTION (CONS LIST MORE-LISTS)))
  LIST ; Always return the first list argument
)

```

```

(XCL:DEFOPTIMIZER MAPC (FN &REST LISTS)
  (%LIST-MAP-OPTIMIZER FN LISTS))

```

```

(DEFUN %MAPL-SINGLE (FN LIST)
  (DO ((TAIL LIST (CDR TAIL))
      ((NULL TAIL))
      (FUNCALL FN TAIL)))

```

```

(DEFUN %MAPL-MULTIPLE (FN LISTS)
  (LET ((ARG-SLICE (MAKE-LIST (IL:LENGTH LISTS))))
    (DO ((SLICE ARG-SLICE)
        ((NULL SLICE))
        ;; %FILL-SLICE-FROM-LISTS returns NIL if one of the lists we're slicing thru bottoms out.
        (SETQ SLICE (%FILL-SLICE-FROM-LISTS LISTS ARG-SLICE ARG-TAIL))
        (COND
          (SLICE (APPLY FN SLICE))))))

```

```

(DEFUN MAPL (FUNCTION LIST &REST MORE-LISTS)
  ;; FUNCTION must take as many arguments as there are lists provided.
  (IF (NULL MORE-LISTS)
      (%MAPL-SINGLE FUNCTION LIST)
      (%MAPL-MULTIPLE FUNCTION (CONS LIST MORE-LISTS)))
  LIST ; always return the first list argument
)

```

```

(XCL:DEFOPTIMIZER MAPL (FN &REST LISTS)
  (%LIST-MAP-OPTIMIZER FN LISTS :TAIL-P T))

```

```

(DEFUN %MAPCAN-SINGLE (FN LIST)
  (DO ((SUBLIST LIST (CDR SUBLIST))
      (RESULT NIL)
      (RESULT-TAIL NIL)
      ELEMENT)
      ((NULL SUBLIST)
       RESULT)
      (SETQ ELEMENT (FUNCALL FN (CAR SUBLIST))) ; accumulate the results by nconc
      (IF RESULT-TAIL
          (RPLACD RESULT-TAIL ELEMENT)
          (SETQ RESULT-TAIL ELEMENT))
      (IF (CONSP ELEMENT)
          (SETQ RESULT-TAIL (LAST ELEMENT))))
)

```

```

(DEFUN %MAPCAN-MULTIPLE (FN LISTS)
  (LET ((ARG-SLICE (MAKE-LIST (IL:LENGTH LISTS))))
    (DO ((RESULT NIL)
        (RESULT-TAIL NIL)
        (CURRENT-SLICE ARG-SLICE)
        ELEMENT)
        ((NULL CURRENT-SLICE)
         RESULT)
        (SETQ CURRENT-SLICE (%FILL-SLICE-FROM-LISTS LISTS ARG-SLICE (CAR ARG-TAIL)))
        (COND
          (CURRENT-SLICE ; There is really more work to do.
           (SETQ ELEMENT (APPLY FN CURRENT-SLICE))
           (IF RESULT-TAIL
               (RPLACD RESULT-TAIL ELEMENT)

```

```

      (SETQ RESULT ELEMENT))
    (IF (CONSP ELEMENT)
        (SETQ RESULT-TAIL (LAST ELEMENT)))))))))

```

```

(DEFUN MAPCAN (FUNCTION LIST &REST MORE-LISTS)
  ;; FUNCTION must take as many arguments as there are lists provided.

```

```

  (IF (NULL MORE-LISTS)
      (%MAPCAN-SINGLE FUNCTION LIST)
      (%MAPCAN-MULTIPLE FUNCTION (CONS LIST MORE-LISTS))))

```

```

(XCL:DEFOPTIMIZER MAPCAN (FN &REST LISTS)
  (%LIST-MAP-OPTIMIZER FN LISTS :NCONC-P T))

```

```

(DEFUN %MAPCON-SINGLE (FN LIST)
  (DO ((SUBLIST LIST (CDR SUBLIST))
      (RESULT NIL)
      (RESULT-TAIL NIL)
      ELEMENT)
      ((NULL SUBLIST)
       RESULT)
    (SETQ ELEMENT (FUNCALL FN SUBLIST))
    (IF RESULT-TAIL
        (RPLACD RESULT-TAIL ELEMENT)
        (SETQ RESULT-TAIL ELEMENT))
    (IF (CONSP ELEMENT)
        (SETQ RESULT-TAIL (LAST ELEMENT))))))

```

```

(DEFUN %MAPCON-MULTIPLE (FN LISTS)
  (LET ((ARG-SLICE (MAKE-LIST (IL:LENGTH LISTS))))
    (DO ((RESULT NIL)
        (RESULT-TAIL NIL)
        (CURRENT-SLICE ARG-SLICE)
        ELEMENT)
        ((NULL CURRENT-SLICE)
         RESULT)
      (SETQ CURRENT-SLICE (%FILL-SLICE-FROM-LISTS LISTS ARG-SLICE ARG-TAIL))
      (COND
        (CURRENT-SLICE
         (SETQ ELEMENT (APPLY FN CURRENT-SLICE))
         (IF RESULT-TAIL
             (RPLACD RESULT-TAIL ELEMENT)
             (SETQ RESULT-TAIL ELEMENT))
         (IF (CONSP ELEMENT)
             (SETQ RESULT-TAIL (LAST ELEMENT))))))))))
; There is really more work to do.

```

```

(DEFUN MAPCON (FUNCTION LIST &REST MORE-LISTS)
  ;; FUNCTION must take as many arguments as there are lists provided.
  (IF (NULL MORE-LISTS)
      (%MAPCON-SINGLE FUNCTION LIST)
      (%MAPCON-MULTIPLE FUNCTION (CONS LIST MORE-LISTS))))

```

```

(XCL:DEFOPTIMIZER MAPCON (FN &REST LISTS)
  (%LIST-MAP-OPTIMIZER FN LISTS :TAIL-P T :NCONC-P T))

```

;; optimizers for Interlisp mapping functions whose bytemacros are not visible to the pav-compiler
 ;; Utility

```

(DEFUN %EVERY-MAP-OPTIMIZER (LIST FN &OPTIONAL INC-FN &KEY SOME-P NEGATE-P)
  (LET ((CONSTANT-FN (COND
    ((CONSTANTP FN)
     (EVAL FN))
    ((AND (CONSP FN)
         (OR (EQ (CAR FN)
                 'FUNCTION)
             (EQ (CAR FN)
                 'IL:FUNCTION))))
     (CADR FN))))
      (CONSTANT-INC-FN (IF INC-FN
                          (COND
        ((CONSTANTP INC-FN)
         (EVAL INC-FN))
        ((AND (CONSP INC-FN)
             (OR (EQ (CAR INC-FN)
                     'FUNCTION)
                 (EQ (CAR INC-FN)
                     'IL:FUNCTION))))
         (CADR INC-FN))))

```

```

      'CDR))
  (IF (AND CONSTANT-FN CONSTANT-INC-FN)
    ` (DO ((SI::%$MAP-SUBLIST ,LIST (,CONSTANT-INC-FN SI::%$MAP-SUBLIST)))
      ((NULL SI::%$MAP-SUBLIST)
        , (IF SOME-P
            NEGATE-P
            (NOT NEGATE-P)))
        , (IF SOME-P
            ` (IF (,CONSTANT-FN (CAR SI::%$MAP-SUBLIST)
                          SI::%$MAP-SUBLIST)
                (RETURN , (IF NEGATE-P
                              NIL
                              'SI::%$MAP-SUBLIST)))
              ` (IF (NULL (,CONSTANT-FN (CAR SI::%$MAP-SUBLIST)
                                         SI::%$MAP-SUBLIST))
                    (RETURN ,NEGATE-P))))
      'COMPILER:PASS)))

```

```

(XCL:DEFOPTIMIZER IL:MAP (LIST IL:MAPFN1 &OPTIONAL IL:MAPFN2)
  (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
    (%LIST-MAP-OPTIMIZER IL:MAPFN1 (LIST LIST)
      :TAIL-P T :INC-FN IL:MAPFN2 :NIL-RESULT-P T)
    'COMPILER:PASS))

```

```

(XCL:DEFOPTIMIZER IL:MAPC (LIST IL:MAPFN1 &OPTIONAL IL:MAPFN2)
  (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
    (%LIST-MAP-OPTIMIZER IL:MAPFN1 (LIST LIST)
      :INC-FN IL:MAPFN2 :NIL-RESULT-P T)
    'COMPILER:PASS))

```

```

(XCL:DEFOPTIMIZER IL:MAPLIST (LIST IL:MAPFN1 &OPTIONAL IL:MAPFN2)
  (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
    (%LIST-MAP-OPTIMIZER IL:MAPFN1 (LIST LIST)
      :TAIL-P T :COLLECT-P T :INC-FN IL:MAPFN2)
    'COMPILER:PASS))

```

```

(XCL:DEFOPTIMIZER IL:MAPCAR (LIST IL:MAPFN1 &OPTIONAL IL:MAPFN2)
  (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
    (%LIST-MAP-OPTIMIZER IL:MAPFN1 (LIST LIST)
      :COLLECT-P T :INC-FN IL:MAPFN2)
    'COMPILER:PASS))

```

```

(XCL:DEFOPTIMIZER IL:MAPCON (LIST IL:MAPFN1 &OPTIONAL IL:MAPFN2)
  (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
    (%LIST-MAP-OPTIMIZER IL:MAPFN1 (LIST LIST)
      :TAIL-P T :NCONC-P T :INC-FN IL:MAPFN2)
    'COMPILER:PASS))

```

```

(XCL:DEFOPTIMIZER IL:MAPCONC (LIST IL:MAPFN1 &OPTIONAL IL:MAPFN2)
  (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
    (%LIST-MAP-OPTIMIZER IL:MAPFN1 (LIST LIST)
      :NCONC-P T :INC-FN IL:MAPFN2)
    'COMPILER:PASS))

```

```

(XCL:DEFOPTIMIZER IL:SOME (LIST IL:MAPFN1 &OPTIONAL IL:MAPFN2)
  (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
    (%EVERY-MAP-OPTIMIZER LIST IL:MAPFN1 IL:MAPFN2 :SOME-P T)
    'COMPILER:PASS))

```

```

(XCL:DEFOPTIMIZER IL:EVERY (LIST IL:MAPFN1 &OPTIONAL IL:MAPFN2)
  (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
    (%EVERY-MAP-OPTIMIZER LIST IL:MAPFN1 IL:MAPFN2)
    'COMPILER:PASS))

```

```

(XCL:DEFOPTIMIZER IL:NOTANY (LIST IL:MAPFN1 &OPTIONAL IL:MAPFN2)
  (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
    (%EVERY-MAP-OPTIMIZER LIST IL:MAPFN1 IL:MAPFN2 :SOME-P T :NEGATE-P T)
    'COMPILER:PASS))

```

```

(XCL:DEFOPTIMIZER IL:NOTEVERY (LIST IL:MAPFN1 &OPTIONAL IL:MAPFN2)
  (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*
    (%EVERY-MAP-OPTIMIZER LIST IL:MAPFN1 IL:MAPFN2 :NEGATE-P T)
    'COMPILER:PASS))

```

```

(XCL:DEFOPTIMIZER IL:SUBSET (LIST IL:MAPFN1 &OPTIONAL IL:MAPFN2)
  (IF COMPILER::*NEW-COMPILER-IS-EXPANDING*

```

```

      (LET ((IL:CONSTANT-FN (COND
                            ((CONSTANTP IL:MAPFN1)
                             (EVAL IL:MAPFN1))
                            ((AND (CONSP IL:MAPFN1)
                                   (OR (EQ (CAR IL:MAPFN1)
                                           'IL:FUNCTION)
                                       (EQ (CAR IL:MAPFN1)
                                           'FUNCTION)))
                                   (CADR IL:MAPFN1))))))
          (IF IL:CONSTANT-FN
              `(IL:MAPCONC ,LIST (IL:FUNCTION (IL:LAMBDA (IL:X)
                                                (IL:IF (,IL:CONSTANT-FN IL:X)
                                                       IL:THEN (LIST IL:X))))
                            ,IL:MAPFN2)
              'COMPILER:PASS))
          'COMPILER:PASS))

```

```

(DEFMACRO XCL:WITH-COLLECTION (&BODY XCL::BODY)
  `(LET ((SI::$WITH-COLLECTION-RESULT$ NIL)
         SI::$WITH-COLLECTION-TAIL$)
      (MACROLET ((XCL:COLLECT (XCL::FORM)
                    ;; written in this way to take advantage of RPLCONS. The FORM is evaluated first so that COLLECT nests properly,
                    ;; i.e., The test to determine if this is the first value collected should be done after the value itself is generated in
                    ;; case it does collection as well.
                    `(LET ((SI::$WITH-COLLECTION-VALUE$ ,XCL::FORM)
                          (IF SI::$WITH-COLLECTION-RESULT$
                              (RPLACD SI::$WITH-COLLECTION-TAIL$ (SETQ SI::$WITH-COLLECTION-TAIL$
                                                                            (LIST SI::$WITH-COLLECTION-VALUE$)
                                                                           )))
                              (SETQ SI::$WITH-COLLECTION-RESULT$ (SETQ SI::$WITH-COLLECTION-TAIL$
                                                                            (LIST SI::$WITH-COLLECTION-VALUE$)
                                                                           )))
                            SI::$WITH-COLLECTION-VALUE$)))
          ,@XCL::BODY SI::$WITH-COLLECTION-RESULT$)))

```

;; some people apparantly still use memq

```

(IL:DECLARE\ : IL:DOCOPY IL:DONTEVAL@LOAD
(IL:MOVD 'IL:FMEMB 'IL:MEMQ)
)

```

;; Arrange to use the correct compiler.

```

(IL:PUTPROPS IL:CMLLIST IL:FILETYPE COMPILE-FILE)
(IL:PUTPROPS IL:CMLLIST IL:MAKEFILE-ENVIRONMENT (:READTABLE "XCL" :PACKAGE "LISP"))
(IL:DECLARE\ : IL:DONTEVAL@LOAD IL:DOEVAL@COMPILE IL:DONTCOPY
(IL:DECLARE\ : IL:DOEVAL@COMPILE IL:DONTCOPY
(IL:LOCALVARS . T)
)
)
(IL:DECLARE\ : IL:DONTEVAL@LOAD IL:DOEVAL@COMPILE IL:DONTCOPY IL:COMPILERVERS
(IL:ADDTOVAR IL:NLAMA )
(IL:ADDTOVAR IL:NLAML )
(IL:ADDTOVAR IL:LAMA APPEND)
)
(IL:PUTPROPS IL:CMLLIST IL:COPYRIGHT ("Venue & Xerox Corporation" 1985 1986 1987 1988 1990))

```


FUNCTION INDEX

%APPEND	4	%NSUBST-IF-NOT	8	FIRST	4	NTH	3
%COMPLEX-ASSOC	15	%SET-NTH	3	FOURTH	4	NTHCDR	4
%COMPLEX-MEMBER	10	%SIMPLE-ASSOC	15	INTERSECTION	12	NUNION	12
%COMPLEX-NSUBLIS	9	%SIMPLE-MEMBER	9	LDIFF	6	PAIRLIS	15
%COMPLEX-NSUBST	7	%SIMPLE-NSUBLIS	8	LIST-LENGTH	3	RASSOC	17
%COMPLEX-RASSOC	17	%SIMPLE-NSUBST	7	MAKE-LIST	4	RASSOC-IF	18
%COMPLEX-SUBLIS	8	%SIMPLE-RASSOC	17	MAPC	21	RASSOC-IF-NOT	18
%COMPLEX-SUBST	7	%SIMPLE-SUBLIS	8	MAPCAN	22	REST	4
%COMPLEX-TREE-EQUAL	2	%SIMPLE-SUBST	7	MAPCAR	20	REVAPPEND	5
%EQCODEP	9	%SIMPLE-TREE-EQUAL	2	MAPCON	22	SECOND	4
%EVERY-MAP-OPTIMIZER	22	%SUBST-IF	7	MAPL	21	SET-DIFFERENCE	13
%LIST-MAP-OPTIMIZER	18	%SUBST-IF-NOT	7	MAPLIST	20	SET-EXCLUSIVE-OR	13
%MAPC-MULTIPLE	21	ACONS	15	MEMBER	10	SEVENTH	4
%MAPC-SINGLE	20	ADJOIN	11	MEMBER-IF	10	SIXTH	4
%MAPCAN-MULTIPLE	21	APPEND	5	MEMBER-IF-NOT	10	SUBLIS	8
%MAPCAN-SINGLE	21	ASSOC	16	NBUTLAST	6	SUBSETP	14
%MAPCAR-MULTIPLE	20	ASSOC-IF	16	NINTERSECTION	12	SUBST	7
%MAPCAR-SINGLE	20	ASSOC-IF-NOT	17	NINTH	4	SUBST-IF	7
%MAPCON-MULTIPLE	22	BUTLAST	6	NRECONC	5	SUBST-IF-NOT	7
%MAPCON-SINGLE	22	COPY-ALIST	5	NSET-DIFFERENCE	13	TENTH	4
%MAPL-MULTIPLE	21	COPY-LIST	5	NSET-EXCLUSIVE-OR	14	THIRD	4
%MAPL-SINGLE	21	COPY-TREE	5	NSUBLIS	9	TREE-EQUAL	3
%MAPLIST-MULTIPLE	20	EIGHTH	4	NSUBST	8	UNION	12
%MAPLIST-SINGLE	20	ENDP	3	NSUBST-IF	8		
%NSUBST-IF	7	FIFTH	4	NSUBST-IF-NOT	8		

OPTIMIZER INDEX

ADJOIN	11	MAPC	21	IL:MAPCAR	23	MAPL	21	IL:NOTANY	23	IL:SOME	23
ASSOC	17	IL:MAPC	23	MAPCON	22	MAPLIST	20	IL:NOTEVERY	23	IL:SUBSET	23
IL:EVERY	23	MAPCAN	22	IL:MAPCON	23	IL:MAPLIST	23	NTH	3		
IL:MAP	23	MAPCAR	20	IL:MAPCONC	23	MEMBER	11	NTHCDR	4		

MACRO INDEX

%CONSTANT-EXPRESSION	9	%FILL-SLICE-FROM-LISTS	18	%MIN-LIST-LENGTH	18	%SUBST-MACRO	6
%CONSTANT-FUNCTION	9	%LIST-COLLECT	20	%NSUBST-MACRO	6	XCL:WITH-COLLECTION	24

PROPERTY INDEX

%SIMPLE-ASSOC	17	%SIMPLE-MEMBER	11	IL:CMLLIST	24
---------------------	----	----------------------	----	------------------	----

SETF INDEX

NTH	3
-----------	---