File created: 16-May-90 14:33:28 **{DSK}<usr>local>lde>lispcore>sources>CMLSEQMODIFY.;2**

changes to: (VARS CMLSEQMODIFYCOMS)

previous date: 15-Mar-87 15:52:22 {DSK}<usr>local>lde>lispcore>sources>CMLSEQMODIFY.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS


```
(RPAQQ CMLSEQMODIFYCOMS
       ((DECLARE%: EVAL@COMPILE DONTCOPY (FILES CMLSEQCOMMON))
        (FUNCTIONS CL:FILL CL:REPLACE)
        (FUNCTIONS %%DESTRUCTIVE-RESULT-VECTOR)
        (FUNCTIONS SIMPLE-REMOVE-MACRO SIMPLE-REMOVE SIMPLE-REMOVE-IF SIMPLE-REMOVE-IF-NOT COMPLEX-REMOVE-MACRO
               COMPLEX-REMOVE COMPLEX-REMOVE-IF COMPLEX-REMOVE-IF-NOT CL:REMOVE CL:REMOVE-IF CL:REMOVE-IF-NOT)
        (FUNCTIONS SIMPLE-DELETE-MACRO SIMPLE-DELETE SIMPLE-DELETE-IF SIMPLE-DELETE-IF-NOT COMPLEX-DELETE-MACRO
               COMPLEX-DELETE COMPLEX-DELETE-IF COMPLEX-DELETE-IF-NOT CL:DELETE CL:DELETE-IF CL:DELETE-IF-NOT)
        (FUNCTIONS SIMPLE-REMOVE-DUPLICATES COMPLEX-REMOVE-DUPLICATES CL:REMOVE-DUPLICATES)
        (FUNCTIONS SIMPLE-DELETE-DUPLICATES COMPLEX-DELETE-DUPLICATES CL:DELETE-DUPLICATES)
        (FUNCTIONS SIMPLE-SUBSTITUTE-MACRO SIMPLE-SUBSTITUTE SIMPLE-SUBSTITUTE-IF SIMPLE-SUBSTITUTE-IF-NOT
               COMPLEX-SUBSTITUTE-MACRO COMPLEX-SUBSTITUTE COMPLEX-SUBSTITUTE-IF COMPLEX-SUBSTITUTE-IF-NOT
               CL:SUBSTITUTE CL:SUBSTITUTE-IF CL:SUBSTITUTE-IF-NOT)
        (FUNCTIONS SIMPLE-NSUBSTITUTE-MACRO SIMPLE-NSUBSTITUTE SIMPLE-NSUBSTITUTE-IF SIMPLE-NSUBSTITUTE-IF-NOT
               COMPLEX-NSUBSTITUTE-MACRO COMPLEX-NSUBSTITUTE COMPLEX-NSUBSTITUTE-IF COMPLEX-NSUBSTITUTE-IF-NOT
               CL:NSUBSTITUTE CL:NSUBSTITUTE-IF CL:NSUBSTITUTE-IF-NOT)
        (PROP FILETYPE CMLSEQMODIFY)
        (DECLARE%: DONTCOPY DONTEVAL@LOAD DOEVAL@COMPILE (LOCALVARS . T))))

(DECLARE%: EVAL@COMPILE DONTCOPY

(FILESLOAD CMLSEQCOMMON)
)


(CL:DEFUN CL:FILL (SEQUENCE ITEM &KEY (START 0)
                            END)
   "Replace the specified elements of SEQUENCE with ITEM."
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (SEQ-DISPATCH SEQUENCE (FORWARD-LIST-LOOP SEQUENCE START END (INDEX CURRENT)
                                       SEQUENCE
                                       (RPLACA %%SUBSEQ ITEM))
               (FILL-VECTOR-SUBSEQ SEQUENCE START END ITEM))))


(CL:DEFUN CL:REPLACE (SEQUENCE1 SEQUENCE2 &KEY (START1 0)
                                END1
                                (START2 0)
                                END2)
   (LET ((LENGTH1 (CL:LENGTH SEQUENCE1))
         (LENGTH2 (CL:LENGTH SEQUENCE2)))
        (CL:IF (NULL END1)
               (SETQ END1 LENGTH1))
        (CL:IF (NULL END2)
               (SETQ END2 LENGTH2))
        (CHECK-SUBSEQ SEQUENCE1 START1 END1 LENGTH1)
        (CHECK-SUBSEQ SEQUENCE2 START2 END2 LENGTH2)
        (LET ((SUBLEN1 (- END1 START1))
              (SUBLEN2 (- END2 START2)))                          ; Make equal length
             (CL:IF (< SUBLEN1 SUBLEN2)
                 (SETQ END2 (+ START2 SUBLEN1))
                 (SETQ END1 (+ START1 SUBLEN2)))                  ; Check for overlap
             (CL:WHEN (AND (EQ SEQUENCE1 SEQUENCE2)
                           (> START1 START2)
                           (< START1 END2))
                 (SETQ SEQUENCE2 (CL:SUBSEQ SEQUENCE2 START2 END2))
                 (SETQ START2 0)
                 (SETQ END2 (- END2 START2)))
             [SEQ-DISPATCH SEQUENCE1 [SEQ-DISPATCH SEQUENCE2 (CL:DO ((SUBSEQ1 (CL:NTHCDR START1 SEQUENCE1)
                                                                             (CDR SUBSEQ1))
                                                                     (SUBSEQ2 (CL:NTHCDR START2 SEQUENCE2)
                                                                             (CDR SUBSEQ2))
                                                                     (INDEX1 START1 (CL:1+ INDEX1)))
                                                                    ((EQL INDEX1 END1))
                                                                  (RPLACA SUBSEQ1 (CAR SUBSEQ2)))
                                                (CL:DO ((SUBSEQ1 (CL:NTHCDR START1 SEQUENCE1)
                                                                 (CDR SUBSEQ1))
                                                        (INDEX1 START1 (CL:1+ INDEX1))
```

```
                                                (INDEX2 START2 (CL:1+ INDEX2)))
                                               ((EQL INDEX1 END1))
                                             (RPLACA SUBSEQ1 (CL:AREF SEQUENCE2 INDEX2)))]
                    (SEQ-DISPATCH SEQUENCE2 (CL:DO ((SUBSEQ2 (CL:NTHCDR START2 SEQUENCE2)
                                                             (CDR SUBSEQ2))
                                                (INDEX1 START1 (CL:1+ INDEX1)))
                                               ((EQL INDEX1 END1))
                                             (CL:SETF (CL:AREF SEQUENCE1 INDEX1)
                                                   (CAR SUBSEQ2)))
                                 (CL:DO ((INDEX1 START1 (CL:1+ INDEX1))
                                         (INDEX2 START2 (CL:1+ INDEX2)))
                                        ((EQL INDEX1 END1))
                                    (CL:SETF (CL:AREF SEQUENCE1 INDEX1)
                                           (CL:AREF SEQUENCE2 INDEX2)))]
             SEQUENCE1)))

(CL:DEFUN %%DESTRUCTIVE-RESULT-VECTOR (VECTOR START)
   (CL:IF (CL:ARRAY-HAS-FILL-POINTER-P VECTOR)
       VECTOR
       (LET ((RESULT (CL:MAKE-ARRAY (VECTOR-LENGTH VECTOR)
                          :ELEMENT-TYPE
                          (CL:ARRAY-ELEMENT-TYPE VECTOR)
                          :FILL-POINTER T)))
            (COPY-VECTOR VECTOR RESULT :END1 START))))

(DEFMACRO SIMPLE-REMOVE-MACRO (SEQUENCE START END TEST-FORM)
   `(SEQ-DISPATCH ,SEQUENCE (LET [(RESULT-HEAD (CL:SUBSEQ ,SEQUENCE 0 ,START))
                                  (RESULT-TAIL (CL:NTHCDR ,END ,SEQUENCE))
                                  (RESULT-MIDDLE (FORWARD-LIST-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT
                                                                                                NEW-LIST TAIL)
                                                          NEW-LIST
                                                          (CL:IF (NOT ,TEST-FORM)
                                                                 (COLLECT-ITEM CURRENT NEW-LIST TAIL]
                              (NCONC RESULT-HEAD RESULT-MIDDLE RESULT-TAIL))
          (LET* ((LENGTH (VECTOR-LENGTH ,SEQUENCE))
                 (RESULT (CL:MAKE-ARRAY LENGTH :ELEMENT-TYPE (CL:ARRAY-ELEMENT-TYPE ,SEQUENCE)
                                :FILL-POINTER T))
                 (NUMBER-OF-MATCHES 0))
                (COPY-VECTOR-SUBSEQ ,SEQUENCE 0 ,START RESULT 0)
                [FORWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT (SLOW-INDEX ,START))
                       NIL
                       (COND
                          ((NOT ,TEST-FORM)
                           (CL:SETF (CL:AREF RESULT SLOW-INDEX)
                                  CURRENT)
                           (CL:INCF SLOW-INDEX))
                          (T (CL:INCF NUMBER-OF-MATCHES]
                (COPY-VECTOR-SUBSEQ ,SEQUENCE ,END LENGTH RESULT (- ,END NUMBER-OF-MATCHES))
                (CL:SETF (CL:FILL-POINTER RESULT)
                       (- LENGTH NUMBER-OF-MATCHES))
                RESULT)))

(CL:DEFUN SIMPLE-REMOVE (ITEM SEQUENCE START END)
   (SIMPLE-REMOVE-MACRO SEQUENCE START END (EQL ITEM CURRENT)))

(CL:DEFUN SIMPLE-REMOVE-IF (TEST SEQUENCE START END)
   (SIMPLE-REMOVE-MACRO SEQUENCE START END (CL:FUNCALL TEST CURRENT)))

(CL:DEFUN SIMPLE-REMOVE-IF-NOT (TEST SEQUENCE START END)
   (SIMPLE-REMOVE-MACRO SEQUENCE START END (NOT (CL:FUNCALL TEST CURRENT))))

(DEFMACRO COMPLEX-REMOVE-MACRO (SEQUENCE START END FROM-END KEY COUNT TEST-FORM)
   `(LET ((NUMBER-OF-MATCHES 0))
         (SEQ-DISPATCH ,SEQUENCE (LET [(RESULT-HEAD (CL:SUBSEQ ,SEQUENCE 0 ,START))
                                       (RESULT-TAIL (CL:NTHCDR ,END ,SEQUENCE))
                                       (RESULT-MIDDLE (CL:IF (NULL (AND ,FROM-END ,COUNT))
                                                             [FORWARD-LIST-LOOP
                                                              ,SEQUENCE
                                                              ,START
                                                              ,END
                                                              (INDEX CURRENT NEW-LIST TAIL)
                                                              NEW-LIST
                                                              (COND
                                                                 ((OR (AND ,COUNT (>= NUMBER-OF-MATCHES
                                                                                      ,COUNT))
                                                                      (NOT ,TEST-FORM))
                                                                  (COLLECT-ITEM CURRENT NEW-LIST TAIL))
                                                                 (T (CL:INCF NUMBER-OF-MATCHES]
                                                             [BACKWARD-LIST-LOOP
                                                              ,SEQUENCE
                                                              ,START
```

```
                                                    ,END
                                                    (INDEX CURRENT NEW-LIST)
                                                    NEW-LIST
                                                    (COND
                                                       ((OR (AND ,COUNT (>= NUMBER-OF-MATCHES
                                                                         ,COUNT))
                                                                 (NOT ,TEST-FORM))
                                                        (CL:PUSH CURRENT NEW-LIST))
                                                       (T (CL:INCF NUMBER-OF-MATCHES])]
                                    (NCONC RESULT-HEAD RESULT-MIDDLE RESULT-TAIL))
                 (LET* ((LENGTH (VECTOR-LENGTH ,SEQUENCE))
                        (RESULT (CL:MAKE-ARRAY LENGTH :ELEMENT-TYPE (CL:ARRAY-ELEMENT-TYPE ,SEQUENCE)
                                       :FILL-POINTER T)))
                       (COPY-VECTOR-SUBSEQ ,SEQUENCE 0 ,START RESULT 0)
                       (CL:IF (NULL (AND ,FROM-END ,COUNT))
                           [FORWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT (RESULT-INDEX ,START))
                                  NIL
                                  (COND
                                     ((OR (AND ,COUNT (>= NUMBER-OF-MATCHES ,COUNT))
                                          (NOT ,TEST-FORM))
                                      (CL:SETF (CL:AREF RESULT RESULT-INDEX)
                                             CURRENT)
                                      (CL:INCF RESULT-INDEX))
                                     (T (CL:INCF NUMBER-OF-MATCHES]
                           [BACKWARD-VECTOR-LOOP ,SEQUENCE ,START ,END [INDEX CURRENT (RESULT-INDEX
                                                                                (CL:1- ,END]
                                  (AND (> NUMBER-OF-MATCHES 0)
                                       (COPY-VECTOR-SUBSEQ RESULT (+ ,START NUMBER-OF-MATCHES)
                                              ,END RESULT ,START (- ,END NUMBER-OF-MATCHES)))
                                  (COND
                                     ((OR (AND ,COUNT (>= NUMBER-OF-MATCHES ,COUNT))
                                          (NOT ,TEST-FORM))
                                      (CL:SETF (CL:AREF RESULT RESULT-INDEX)
                                             CURRENT)
                                      (CL:DECF RESULT-INDEX))
                                     (T (CL:INCF NUMBER-OF-MATCHES])
                       (COPY-VECTOR-SUBSEQ ,SEQUENCE ,END LENGTH RESULT (- ,END NUMBER-OF-MATCHES))
                       (CL:SETF (CL:FILL-POINTER RESULT)
                              (- LENGTH NUMBER-OF-MATCHES))
                       RESULT))))

(CL:DEFUN COMPLEX-REMOVE (ITEM SEQUENCE START END FROM-END KEY COUNT TEST TEST-NOT-P)
   [COMPLEX-REMOVE-MACRO SEQUENCE START END FROM-END KEY COUNT (CL:IF TEST-NOT-P
                                                                     (NOT (CL:FUNCALL TEST ITEM
                                                                                (CL:FUNCALL KEY CURRENT)))
                                                                     (CL:FUNCALL TEST ITEM (CL:FUNCALL KEY
                                                                                                 CURRENT)))])


(CL:DEFUN COMPLEX-REMOVE-IF (TEST SEQUENCE START END FROM-END KEY COUNT)
   (COMPLEX-REMOVE-MACRO SEQUENCE START END FROM-END KEY COUNT (CL:FUNCALL TEST (CL:FUNCALL KEY CURRENT))))


(CL:DEFUN COMPLEX-REMOVE-IF-NOT (TEST SEQUENCE START END FROM-END KEY COUNT)
   [COMPLEX-REMOVE-MACRO SEQUENCE START END FROM-END KEY COUNT (NOT (CL:FUNCALL TEST (CL:FUNCALL KEY CURRENT]
)


(CL:DEFUN CL:REMOVE (ITEM SEQUENCE &KEY (START 0)
                           END
                           (FROM-END NIL FROM-END-P)
                           COUNT
                           (KEY 'CL:IDENTITY KEY-P)
                           (TEST 'EQL TEST-P)
                           (TEST-NOT NIL TEST-NOT-P))
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (AND TEST-P TEST-NOT-P)
               (CL:ERROR "Both test and test-not provided"))
        (CL:IF (OR FROM-END-P KEY-P COUNT TEST-P TEST-NOT-P)
            (COMPLEX-REMOVE ITEM SEQUENCE START END FROM-END KEY COUNT (CL:IF TEST-NOT-P
                                                                              TEST-NOT
                                                                              TEST)
                  TEST-NOT-P)
            (SIMPLE-REMOVE ITEM SEQUENCE START END))))


(CL:DEFUN CL:REMOVE-IF (TEST SEQUENCE &KEY (START 0)
                              END
                              (FROM-END NIL FROM-END-P)
                              COUNT
                              (KEY 'CL:IDENTITY KEY-P))
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
```

```
            (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (OR FROM-END-P KEY-P COUNT)
            (COMPLEX-REMOVE-IF TEST SEQUENCE START END FROM-END KEY COUNT)
            (SIMPLE-REMOVE-IF TEST SEQUENCE START END))))


(CL:DEFUN CL:REMOVE-IF-NOT (TEST SEQUENCE &KEY (START 0)
                                  END
                                  (FROM-END NIL FROM-END-P)
                                  COUNT
                                  (KEY 'CL:IDENTITY KEY-P))
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
            (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (OR FROM-END-P KEY-P COUNT)
            (COMPLEX-REMOVE-IF-NOT TEST SEQUENCE START END FROM-END KEY COUNT)
            (SIMPLE-REMOVE-IF-NOT TEST SEQUENCE START END))))


(DEFMACRO SIMPLE-DELETE-MACRO (SEQUENCE START END TEST-FORM)
   `(SEQ-DISPATCH ,SEQUENCE [LET [(HANDLE (CONS NIL ,SEQUENCE]
                                  (FORWARD-LIST-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT
                                                                                  (PREVIOUS (CL:NTHCDR
                                                                                            ,START HANDLE)))
                                      (CDR HANDLE)
                                      (CL:IF (NOT ,TEST-FORM)
                                          (SETQ PREVIOUS (CDR PREVIOUS))
                                          (RPLACD PREVIOUS (CDR %%SUBSEQ)))]
          (LET [(LENGTH (VECTOR-LENGTH ,SEQUENCE))
                (NUMBER-OF-MATCHES 0)
                (RESULT (%%DESTRUCTIVE-RESULT-VECTOR ,SEQUENCE ,START]
               [FORWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT (SLOW-INDEX ,START))
                      NIL
                      (COND
                         ((NOT ,TEST-FORM)
                          (CL:SETF (CL:AREF RESULT SLOW-INDEX)
                                   CURRENT)
                          (CL:INCF SLOW-INDEX))
                         (T (CL:INCF NUMBER-OF-MATCHES]
               (COPY-VECTOR-SUBSEQ ,SEQUENCE ,END LENGTH RESULT (- ,END NUMBER-OF-MATCHES))
               (CL:SETF (CL:FILL-POINTER RESULT)
                        (- LENGTH NUMBER-OF-MATCHES))
               RESULT)))


(CL:DEFUN SIMPLE-DELETE (ITEM SEQUENCE START END)
   (SIMPLE-DELETE-MACRO SEQUENCE START END (EQL ITEM CURRENT)))


(CL:DEFUN SIMPLE-DELETE-IF (TEST SEQUENCE START END)
   (SIMPLE-DELETE-MACRO SEQUENCE START END (CL:FUNCALL TEST CURRENT)))


(CL:DEFUN SIMPLE-DELETE-IF-NOT (TEST SEQUENCE START END)
   (SIMPLE-DELETE-MACRO SEQUENCE START END (NOT (CL:FUNCALL TEST CURRENT))))


(DEFMACRO COMPLEX-DELETE-MACRO (SEQUENCE START END FROM-END KEY COUNT TEST-FORM)
   `(LET ((NUMBER-OF-MATCHES 0))
         (SEQ-DISPATCH ,SEQUENCE [LET [(HANDLE (CONS NIL ,SEQUENCE]
                                      (CL:IF (NULL (AND ,FROM-END ,COUNT))
                                          (CL:DO ((PREVIOUS (CL:NTHCDR ,START HANDLE))
                                                  (%%SUBSEQ (CL:NTHCDR ,START ,SEQUENCE)
                                                            (CDR %%SUBSEQ))
                                                  (INDEX ,START (CL:1+ INDEX))
                                                  CURRENT)
                                                 ([OR (EQL INDEX ,END)
                                                      (AND ,COUNT (>= NUMBER-OF-MATCHES ,COUNT]
                                                  (CDR HANDLE))
                                              (SETQ CURRENT (CAR %%SUBSEQ))
                                              (COND
                                                 ((NOT ,TEST-FORM)
                                                  (SETQ PREVIOUS (CDR PREVIOUS)))
                                                 (T (RPLACD PREVIOUS (CDR %%SUBSEQ))
                                                    (CL:INCF NUMBER-OF-MATCHES))))
                                          (CL:DO ((INDEX (CL:1- ,END)
                                                         (CL:1- INDEX))
                                                  (LAST (CL:NTHCDR ,END ,SEQUENCE))
                                                  PREVIOUS CURRENT)
                                                 ([OR (< INDEX ,START)
                                                      (AND ,COUNT (>= NUMBER-OF-MATCHES ,COUNT]
                                                  (CDR HANDLE))
                                              (SETQ PREVIOUS (CL:NTHCDR INDEX HANDLE))
                                              (SETQ CURRENT (CADR PREVIOUS))
                                              (COND
```

```
                                          ((NOT ,TEST-FORM)
                                           (SETQ LAST (CDR PREVIOUS)))
                                          (T (RPLACD PREVIOUS LAST)
                                             (CL:INCF NUMBER-OF-MATCHES))))))]
                (LET [(LENGTH (VECTOR-LENGTH ,SEQUENCE))
                      (RESULT (%%DESTRUCTIVE-RESULT-VECTOR ,SEQUENCE ,START]
                     (CL:IF (NULL (AND ,FROM-END ,COUNT))
                         [FORWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT (SLOW-INDEX ,START))
                                NIL
                                (COND
                                   ((OR (AND ,COUNT (>= NUMBER-OF-MATCHES ,COUNT))
                                        (NOT ,TEST-FORM))
                                    (CL:SETF (CL:AREF RESULT SLOW-INDEX)
                                           CURRENT)
                                    (CL:INCF SLOW-INDEX))
                                   (T (CL:INCF NUMBER-OF-MATCHES]
                         [BACKWARD-VECTOR-LOOP ,SEQUENCE ,START ,END [INDEX CURRENT (SLOW-INDEX
                                                                                    (CL:1- ,END]
                                (AND (> NUMBER-OF-MATCHES 0)
                                     (COPY-VECTOR-SUBSEQ RESULT (+ ,START NUMBER-OF-MATCHES)
                                            ,END RESULT ,START (- ,END NUMBER-OF-MATCHES)))
                                (COND
                                   ((OR (AND ,COUNT (>= NUMBER-OF-MATCHES ,COUNT))
                                        (NOT ,TEST-FORM))
                                    (CL:SETF (CL:AREF RESULT SLOW-INDEX)
                                           CURRENT)
                                    (CL:DECF SLOW-INDEX))
                                   (T (CL:INCF NUMBER-OF-MATCHES])
                         (COPY-VECTOR-SUBSEQ ,SEQUENCE ,END LENGTH RESULT (- ,END NUMBER-OF-MATCHES))
                         (CL:SETF (CL:FILL-POINTER RESULT)
                                (- LENGTH NUMBER-OF-MATCHES))
                         RESULT))))

(CL:DEFUN COMPLEX-DELETE (ITEM SEQUENCE START END FROM-END KEY COUNT TEST TEST-NOT-P)
   [COMPLEX-DELETE-MACRO SEQUENCE START END FROM-END KEY COUNT (CL:IF TEST-NOT-P
                                                                     (NOT (CL:FUNCALL TEST ITEM (CL:FUNCALL
                                                                                                  KEY CURRENT)))
                                                                     (CL:FUNCALL TEST ITEM (CL:FUNCALL KEY
                                                                                                  CURRENT)))])


(CL:DEFUN COMPLEX-DELETE-IF (TEST SEQUENCE START END FROM-END KEY COUNT)
   (COMPLEX-DELETE-MACRO SEQUENCE START END FROM-END KEY COUNT (CL:FUNCALL TEST (CL:FUNCALL KEY CURRENT))))


(CL:DEFUN COMPLEX-DELETE-IF-NOT (TEST SEQUENCE START END FROM-END KEY COUNT)
   [COMPLEX-DELETE-MACRO SEQUENCE START END FROM-END KEY COUNT (NOT (CL:FUNCALL TEST (CL:FUNCALL KEY CURRENT])


(CL:DEFUN CL:DELETE (ITEM SEQUENCE &KEY (START 0)
                          END
                          (FROM-END NIL FROM-END-P)
                          COUNT
                          (KEY 'CL:IDENTITY KEY-P)
                          (TEST 'EQL TEST-P)
                          (TEST-NOT NIL TEST-NOT-P))
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (AND TEST-P TEST-NOT-P)
               (CL:ERROR "Both test and test-not provided"))
        (CL:IF (OR FROM-END-P KEY-P COUNT TEST-P TEST-NOT-P)
            (COMPLEX-DELETE ITEM SEQUENCE START END FROM-END KEY COUNT (CL:IF TEST-NOT-P
                                                                              TEST-NOT
                                                                              TEST)
                  TEST-NOT-P)
            (SIMPLE-DELETE ITEM SEQUENCE START END))))


(CL:DEFUN CL:DELETE-IF (TEST SEQUENCE &KEY (START 0)
                             END
                             (FROM-END NIL FROM-END-P)
                             COUNT
                             (KEY 'CL:IDENTITY KEY-P))
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (OR FROM-END-P KEY-P COUNT)
            (COMPLEX-DELETE-IF TEST SEQUENCE START END FROM-END KEY COUNT)
            (SIMPLE-DELETE-IF TEST SEQUENCE START END))))


(CL:DEFUN CL:DELETE-IF-NOT (TEST SEQUENCE &KEY (START 0)
                                  END
```

```
                                    (FROM-END NIL FROM-END-P)
                                    COUNT
                                    (KEY 'CL:IDENTITY KEY-P))
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (OR FROM-END-P KEY-P COUNT)
            (COMPLEX-DELETE-IF-NOT TEST SEQUENCE START END FROM-END KEY COUNT)
            (SIMPLE-DELETE-IF-NOT TEST SEQUENCE START END))))


(CL:DEFUN SIMPLE-REMOVE-DUPLICATES (SEQUENCE START END)
   (SIMPLE-REMOVE-MACRO SEQUENCE START END (SIMPLE-POSITION CURRENT SEQUENCE (CL:1+ INDEX)
                                                  END)))


(CL:DEFUN COMPLEX-REMOVE-DUPLICATES (SEQUENCE START END FROM-END KEY TEST TEST-NOT-P)
   (SEQ-DISPATCH SEQUENCE (LET [(RESULT-HEAD (CL:SUBSEQ SEQUENCE 0 START))
                                (RESULT-TAIL (CL:NTHCDR END SEQUENCE))
                                (RESULT-MIDDLE (CL:IF (NULL FROM-END)
                                                      (FORWARD-LIST-LOOP SEQUENCE START END (INDEX CURRENT NEW-LIST
                                                                                                   TAIL)
                                                             NEW-LIST
                                                             (CL:IF (NOT (COMPLEX-POSITION (CL:FUNCALL KEY CURRENT)
                                                                                           SEQUENCE
                                                                                           (CL:1+ INDEX)
                                                                                           END NIL KEY TEST TEST-NOT-P))
                                                                    (COLLECT-ITEM CURRENT NEW-LIST TAIL)))
                                                      (FORWARD-LIST-LOOP SEQUENCE START END (INDEX CURRENT NEW-LIST
                                                                                                   TAIL)
                                                             NEW-LIST
                                                             (CL:IF (NOT (COMPLEX-POSITION (CL:FUNCALL KEY CURRENT)
                                                                                           SEQUENCE START INDEX NIL KEY TEST
                                                                                           TEST-NOT-P))
                                                                    (COLLECT-ITEM CURRENT NEW-LIST TAIL)))))]
                           (NCONC RESULT-HEAD RESULT-MIDDLE RESULT-TAIL))
          (LET* ((LENGTH (VECTOR-LENGTH SEQUENCE))
                 (RESULT (CL:MAKE-ARRAY LENGTH :ELEMENT-TYPE (CL:ARRAY-ELEMENT-TYPE SEQUENCE)
                                :FILL-POINTER T))
                 (NUMBER-OF-MATCHES 0))
                (COPY-VECTOR-SUBSEQ SEQUENCE 0 START RESULT 0)
                (CL:IF (NULL FROM-END)
                    [FORWARD-VECTOR-LOOP SEQUENCE START END (INDEX CURRENT (RESULT-INDEX START)
                                                                  TEST-RESULT)
                           NIL
                           (COND
                              ((NOT (COMPLEX-POSITION (CL:FUNCALL KEY CURRENT)
                                                      SEQUENCE
                                                      (CL:1+ INDEX)
                                                      END NIL KEY TEST TEST-NOT-P))
                               (CL:SETF (CL:AREF RESULT RESULT-INDEX)
                                      CURRENT)
                               (CL:INCF RESULT-INDEX))
                              (T (CL:INCF NUMBER-OF-MATCHES]
                    [FORWARD-VECTOR-LOOP SEQUENCE START END (INDEX CURRENT (RESULT-INDEX START)
                                                                  TEST-RESULT)
                           NIL
                           (COND
                              ((NOT (COMPLEX-POSITION (CL:FUNCALL KEY CURRENT)
                                                      SEQUENCE START INDEX NIL KEY TEST TEST-NOT-P))
                               (CL:SETF (CL:AREF RESULT RESULT-INDEX)
                                      CURRENT)
                               (CL:INCF RESULT-INDEX))
                              (T (CL:INCF NUMBER-OF-MATCHES])
                (COPY-VECTOR-SUBSEQ SEQUENCE END LENGTH RESULT (- END NUMBER-OF-MATCHES))
                (CL:SETF (CL:FILL-POINTER RESULT)
                       (- LENGTH NUMBER-OF-MATCHES))
                RESULT)))


(CL:DEFUN CL:REMOVE-DUPLICATES (SEQUENCE &KEY (START 0)
                                           END
                                           (FROM-END NIL FROM-END-P)
                                           (KEY 'CL:IDENTITY KEY-P)
                                           (TEST 'EQL TEST-P)
                                           (TEST-NOT NIL TEST-NOT-P))
   "The elements of Sequence are examined, and if any two match, one is discarded.  The resulting sequence is
   returned."
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (AND TEST-P TEST-NOT-P)
               (CL:ERROR "Both test and test-not provided"))
        (CL:IF (OR FROM-END-P KEY-P TEST-P TEST-NOT-P)
            (COMPLEX-REMOVE-DUPLICATES SEQUENCE START END FROM-END KEY (CL:IF TEST-NOT-P
```

```
                                                                    TEST-NOT
                                                                    TEST)
                TEST-NOT-P)
          (SIMPLE-REMOVE-DUPLICATES SEQUENCE START END))))

(CL:DEFUN SIMPLE-DELETE-DUPLICATES (SEQUENCE START END)
   (SEQ-DISPATCH SEQUENCE [LET ((HANDLE (CONS NIL SEQUENCE)))
                               (FORWARD-LIST-LOOP SEQUENCE START END (INDEX CURRENT (PREVIOUS (CL:NTHCDR START
                                                                                                HANDLE)))
                                      (CDR HANDLE)
                                      (CL:IF (NOT (SIMPLE-POSITION CURRENT (CDR %%SUBSEQ)
                                                                   0
                                                                   (- END INDEX 1)))
                                          (SETQ PREVIOUS (CDR PREVIOUS))
                                          (RPLACD PREVIOUS (CDR %%SUBSEQ)))]
          (LET ((LENGTH (VECTOR-LENGTH SEQUENCE))
                (NUMBER-OF-MATCHES 0)
                (RESULT (%%DESTRUCTIVE-RESULT-VECTOR SEQUENCE START)))
               [FORWARD-VECTOR-LOOP SEQUENCE START END (INDEX CURRENT (SLOW-INDEX START))
                      NIL
                      (COND
                         ((NOT (SIMPLE-POSITION CURRENT SEQUENCE (CL:1+ INDEX)
                                                END))
                          (CL:SETF (CL:AREF RESULT SLOW-INDEX)
                                 CURRENT)
                          (CL:INCF SLOW-INDEX))
                         (T (CL:INCF NUMBER-OF-MATCHES]
               (COPY-VECTOR-SUBSEQ SEQUENCE END LENGTH RESULT (- END NUMBER-OF-MATCHES))
               (CL:SETF (CL:FILL-POINTER RESULT)
                      (- LENGTH NUMBER-OF-MATCHES))
               RESULT)))

(CL:DEFUN COMPLEX-DELETE-DUPLICATES (SEQUENCE START END FROM-END KEY TEST TEST-NOT-P)
   (SEQ-DISPATCH SEQUENCE [LET ((HANDLE (CONS NIL SEQUENCE)))
                               (CL:IF (NULL FROM-END)
                                   (CL:DO ((PREVIOUS (CL:NTHCDR START HANDLE))
                                           (%%SUBSEQ (CL:NTHCDR START SEQUENCE)
                                                  (CDR %%SUBSEQ))
                                           (INDEX START (CL:1+ INDEX)))
                                          ((EQL INDEX END)
                                           (CDR HANDLE))
                                      (CL:IF (NOT (COMPLEX-POSITION (CL:FUNCALL KEY (CAR %%SUBSEQ))
                                                                    (CDR %%SUBSEQ)
                                                                    0
                                                                    (- END INDEX 1)
                                                                    NIL KEY TEST TEST-NOT-P))
                                          (SETQ PREVIOUS (CDR PREVIOUS))
                                          (RPLACD PREVIOUS (CDR %%SUBSEQ))))
                                   (CL:DO ((NUMBER-OF-MATCHES 0)
                                           (PREVIOUS (CL:NTHCDR START HANDLE))
                                           (%%SUBSEQ (CL:NTHCDR START SEQUENCE)
                                                  (CDR %%SUBSEQ))
                                           (INDEX START (CL:1+ INDEX)))
                                          ((EQL INDEX END)
                                           (CDR HANDLE))
                                      (COND
                                         ((NOT (COMPLEX-POSITION (CL:FUNCALL KEY (CAR %%SUBSEQ))
                                                                 SEQUENCE START (- INDEX NUMBER-OF-MATCHES)
                                                                 NIL KEY TEST TEST-NOT-P))
                                          (SETQ PREVIOUS (CDR PREVIOUS)))
                                         (T (RPLACD PREVIOUS (CDR %%SUBSEQ))
                                            (CL:INCF NUMBER-OF-MATCHES)))))]
          (LET ((LENGTH (VECTOR-LENGTH SEQUENCE))
                (NUMBER-OF-MATCHES 0)
                (RESULT (%%DESTRUCTIVE-RESULT-VECTOR SEQUENCE START)))
               (CL:IF (NULL FROM-END)
                   [FORWARD-VECTOR-LOOP SEQUENCE START END (INDEX CURRENT (RESULT-INDEX START)
                                                                          TEST-RESULT)
                          NIL
                          (COND
                             ((NOT (COMPLEX-POSITION (CL:FUNCALL KEY CURRENT)
                                          SEQUENCE
                                          (CL:1+ INDEX)
                                          END NIL KEY TEST TEST-NOT-P))
                              (CL:SETF (CL:AREF RESULT RESULT-INDEX)
                                     CURRENT)
                              (CL:INCF RESULT-INDEX))
                             (T (CL:INCF NUMBER-OF-MATCHES]
                   [FORWARD-VECTOR-LOOP SEQUENCE START END (INDEX CURRENT (RESULT-INDEX START)
                                                                          TEST-RESULT)
                          NIL
                          (COND
                             ((NOT (COMPLEX-POSITION (CL:FUNCALL KEY CURRENT)
                                          SEQUENCE START INDEX NIL KEY TEST TEST-NOT-P))
                              (CL:SETF (CL:AREF RESULT RESULT-INDEX)
```

```
                                    CURRENT)
                                (CL:INCF RESULT-INDEX))
                               (T (CL:INCF NUMBER-OF-MATCHES])
                    (COPY-VECTOR-SUBSEQ SEQUENCE END LENGTH RESULT (- END NUMBER-OF-MATCHES))
                    (CL:SETF (CL:FILL-POINTER RESULT)
                             (- LENGTH NUMBER-OF-MATCHES))
                    RESULT)))

(CL:DEFUN CL:DELETE-DUPLICATES (SEQUENCE &KEY (START 0)
                                              END
                                              (FROM-END NIL FROM-END-P)
                                              (KEY 'CL:IDENTITY KEY-P)
                                              (TEST 'EQL TEST-P)
                                              (TEST-NOT NIL TEST-NOT-P))
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (AND TEST-P TEST-NOT-P)
               (CL:ERROR "Both test and test-not provided"))
        (CL:IF (OR FROM-END-P KEY-P TEST-P TEST-NOT-P)
            (COMPLEX-DELETE-DUPLICATES SEQUENCE START END FROM-END KEY (CL:IF TEST-NOT-P
                                                                              TEST-NOT
                                                                              TEST)
                  TEST-NOT-P)
            (SIMPLE-DELETE-DUPLICATES SEQUENCE START END))))

(DEFMACRO SIMPLE-SUBSTITUTE-MACRO (NEWITEM SEQUENCE START END TEST-FORM)
   `(SEQ-DISPATCH ,SEQUENCE (LET [(RESULT-HEAD (CL:SUBSEQ ,SEQUENCE 0 ,START))
                                  (RESULT-TAIL (CL:NTHCDR ,END ,SEQUENCE))
                                  (RESULT-MIDDLE (FORWARD-LIST-LOOP ,SEQUENCE ,START ,END
                                                                    (INDEX CURRENT NEW-LIST TAIL NEW-ELEMENT)
                                                                    NEW-LIST
                                                                    (SETQ NEW-ELEMENT (CL:IF ,TEST-FORM
                                                                                             ,NEWITEM
                                                                                             CURRENT))
                                                                    (COLLECT-ITEM NEW-ELEMENT NEW-LIST TAIL]
                                 (NCONC RESULT-HEAD RESULT-MIDDLE RESULT-TAIL))
          (LET* [(LENGTH (VECTOR-LENGTH ,SEQUENCE))
                 (RESULT (MAKE-VECTOR LENGTH :ELEMENT-TYPE (CL:ARRAY-ELEMENT-TYPE ,SEQUENCE]
                (COPY-VECTOR-SUBSEQ ,SEQUENCE 0 ,START RESULT 0)
                (FORWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
                       NIL
                       (CL:SETF (CL:AREF RESULT INDEX)
                              (CL:IF ,TEST-FORM
                                  ,NEWITEM
                                  CURRENT)))
                (COPY-VECTOR-SUBSEQ ,SEQUENCE ,END LENGTH RESULT ,END)
                RESULT)))

(CL:DEFUN SIMPLE-SUBSTITUTE (NEWITEM OLDITEM SEQUENCE START END)
   (SIMPLE-SUBSTITUTE-MACRO NEWITEM SEQUENCE START END (EQL OLDITEM CURRENT)))

(CL:DEFUN SIMPLE-SUBSTITUTE-IF (NEWITEM TEST SEQUENCE START END)
   (SIMPLE-SUBSTITUTE-MACRO NEWITEM SEQUENCE START END (CL:FUNCALL TEST CURRENT)))

(CL:DEFUN SIMPLE-SUBSTITUTE-IF-NOT (NEWITEM TEST SEQUENCE START END)
   (SIMPLE-SUBSTITUTE-MACRO NEWITEM SEQUENCE START END (NOT (CL:FUNCALL TEST CURRENT))))

(DEFMACRO COMPLEX-SUBSTITUTE-MACRO (NEWITEM SEQUENCE START END FROM-END KEY COUNT TEST-FORM)
   `(LET ((NUMBER-OF-MATCHES 0))
         (SEQ-DISPATCH ,SEQUENCE (LET [(RESULT-HEAD (CL:SUBSEQ ,SEQUENCE 0 ,START))
                                       (RESULT-TAIL (CL:NTHCDR ,END ,SEQUENCE))
                                       (RESULT-MIDDLE (CL:IF (NULL (AND ,FROM-END ,COUNT))
                                                             (FORWARD-LIST-LOOP
                                                              ,SEQUENCE
                                                              ,START
                                                              ,END
                                                              (INDEX CURRENT NEW-LIST TAIL NEW-ELEMENT)
                                                              NEW-LIST
                                                              [SETQ NEW-ELEMENT
                                                               (COND
                                                                   ((OR (AND ,COUNT (>= NUMBER-OF-MATCHES
                                                                                        ,COUNT))
                                                                        (NOT ,TEST-FORM))
                                                                    CURRENT)
                                                                   (T (CL:INCF NUMBER-OF-MATCHES)
                                                                      ,NEWITEM]
                                                              (COLLECT-ITEM NEW-ELEMENT NEW-LIST TAIL))
                                                             (BACKWARD-LIST-LOOP
                                                              ,SEQUENCE
```

```
                                                    ,START
                                                    ,END
                                                    (INDEX CURRENT NEW-LIST NEW-ELEMENT)
                                                    NEW-LIST
                                                    [SETQ NEW-ELEMENT
                                                     (COND
                                                        ((OR (AND ,COUNT (>= NUMBER-OF-MATCHES
                                                                             ,COUNT))
                                                             (NOT ,TEST-FORM))
                                                         CURRENT)
                                                        (T (CL:INCF NUMBER-OF-MATCHES)
                                                           ,NEWITEM]
                                                    (CL:PUSH NEW-ELEMENT NEW-LIST)))]
                                (NCONC RESULT-HEAD RESULT-MIDDLE RESULT-TAIL))
                  (LET* [(LENGTH (VECTOR-LENGTH ,SEQUENCE))
                         (RESULT (MAKE-VECTOR LENGTH :ELEMENT-TYPE (CL:ARRAY-ELEMENT-TYPE ,SEQUENCE]
                        (COPY-VECTOR-SUBSEQ ,SEQUENCE 0 ,START RESULT 0)
                        (CL:IF (NULL ,FROM-END)
                            [FORWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
                                   NIL
                                   (CL:SETF (CL:AREF RESULT INDEX)
                                          (COND
                                             ((OR (AND ,COUNT (>= NUMBER-OF-MATCHES ,COUNT))
                                                  (NOT ,TEST-FORM))
                                              CURRENT)
                                             (T (CL:INCF NUMBER-OF-MATCHES)
                                                ,NEWITEM]
                            [BACKWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
                                   NIL
                                   (CL:SETF (CL:AREF RESULT INDEX)
                                          (COND
                                             ((OR (AND ,COUNT (>= NUMBER-OF-MATCHES ,COUNT))
                                                  (NOT ,TEST-FORM))
                                              CURRENT)
                                             (T (CL:INCF NUMBER-OF-MATCHES)
                                                ,NEWITEM])
                        (COPY-VECTOR-SUBSEQ ,SEQUENCE ,END LENGTH RESULT ,END)
                        RESULT))))

(CL:DEFUN COMPLEX-SUBSTITUTE (NEWITEM OLDITEM SEQUENCE START END FROM-END KEY COUNT TEST TEST-NOT-P)
   [COMPLEX-SUBSTITUTE-MACRO NEWITEM SEQUENCE START END FROM-END KEY COUNT
          (CL:IF TEST-NOT-P
              (NOT (CL:FUNCALL TEST OLDITEM (CL:FUNCALL KEY CURRENT)))
              (CL:FUNCALL TEST OLDITEM (CL:FUNCALL KEY CURRENT)))])

(CL:DEFUN COMPLEX-SUBSTITUTE-IF (NEWITEM TEST SEQUENCE START END FROM-END KEY COUNT)
   (COMPLEX-SUBSTITUTE-MACRO NEWITEM SEQUENCE START END FROM-END KEY COUNT (CL:FUNCALL TEST (CL:FUNCALL KEY
                                                                                                CURRENT))))

(CL:DEFUN COMPLEX-SUBSTITUTE-IF-NOT (NEWITEM TEST SEQUENCE START END FROM-END KEY COUNT)
   [COMPLEX-SUBSTITUTE-MACRO NEWITEM SEQUENCE START END FROM-END KEY COUNT (NOT (CL:FUNCALL TEST
                                                                                   (CL:FUNCALL KEY CURRENT
                                                                                          ])

(CL:DEFUN CL:SUBSTITUTE (NEWITEM OLDITEM SEQUENCE &KEY (START 0)
                                 END
                                 (FROM-END NIL FROM-END-P)
                                 COUNT
                                 (KEY 'CL:IDENTITY KEY-P)
                                 (TEST 'EQL TEST-P)
                                 (TEST-NOT NIL TEST-NOT-P))
   "Returns a sequence of the same kind as Sequence with the same elements except that all elements that match
   Old are replaced with New."
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (AND TEST-P TEST-NOT-P)
               (CL:ERROR "Both test and test-not provided"))
        (CL:IF (OR FROM-END-P KEY-P COUNT TEST-P TEST-NOT-P)
            (COMPLEX-SUBSTITUTE NEWITEM OLDITEM SEQUENCE START END FROM-END KEY COUNT (CL:IF TEST-NOT-P
                                                                                             TEST-NOT
                                                                                             TEST)
                  TEST-NOT-P)
            (SIMPLE-SUBSTITUTE NEWITEM OLDITEM SEQUENCE START END))))

(CL:DEFUN CL:SUBSTITUTE-IF (NEWITEM TEST SEQUENCE &KEY (START 0)
                                   END
                                   (FROM-END NIL FROM-END-P)
                                   COUNT
                                   (KEY 'CL:IDENTITY KEY-P))
   "Returns a sequence of the same kind as Sequence with the same elements except that all elements that match
```

```
   Old are replaced with New."
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (OR FROM-END-P KEY-P COUNT)
            (COMPLEX-SUBSTITUTE-IF NEWITEM TEST SEQUENCE START END FROM-END KEY COUNT)
            (SIMPLE-SUBSTITUTE-IF NEWITEM TEST SEQUENCE START END))))


(CL:DEFUN CL:SUBSTITUTE-IF-NOT (NEWITEM TEST SEQUENCE &KEY (START 0)
                                        END
                                        (FROM-END NIL FROM-END-P)
                                        COUNT
                                        (KEY 'CL:IDENTITY KEY-P))
   "Returns a sequence of the same kind as Sequence with the same elements except that all elements that match
   Old are replaced with New."
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (OR FROM-END-P KEY-P COUNT)
            (COMPLEX-SUBSTITUTE-IF-NOT NEWITEM TEST SEQUENCE START END FROM-END KEY COUNT)
            (SIMPLE-SUBSTITUTE-IF-NOT NEWITEM TEST SEQUENCE START END))))


(DEFMACRO SIMPLE-NSUBSTITUTE-MACRO (NEWITEM SEQUENCE START END TEST-FORM)
   `[SEQ-DISPATCH ,SEQUENCE [FORWARD-LIST-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT NEW-LIST TAIL NEW-ELEMENT)
                                    ,SEQUENCE
                                    (CL:IF ,TEST-FORM
                                           (RPLACA %%SUBSEQ ,NEWITEM))]
          (FORWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
                 ,SEQUENCE
                 (CL:IF ,TEST-FORM
                     (CL:SETF (CL:AREF ,SEQUENCE INDEX)
                            ,NEWITEM))])


(CL:DEFUN SIMPLE-NSUBSTITUTE (NEWITEM OLDITEM SEQUENCE START END)
   (SIMPLE-NSUBSTITUTE-MACRO NEWITEM SEQUENCE START END (EQL OLDITEM CURRENT)))


(CL:DEFUN SIMPLE-NSUBSTITUTE-IF (NEWITEM TEST SEQUENCE START END)
   (SIMPLE-NSUBSTITUTE-MACRO NEWITEM SEQUENCE START END (CL:FUNCALL TEST CURRENT)))


(CL:DEFUN SIMPLE-NSUBSTITUTE-IF-NOT (NEWITEM TEST SEQUENCE START END)
   (SIMPLE-NSUBSTITUTE-MACRO NEWITEM SEQUENCE START END (NOT (CL:FUNCALL TEST CURRENT))))


(DEFMACRO COMPLEX-NSUBSTITUTE-MACRO (NEWITEM SEQUENCE START END FROM-END KEY COUNT TEST-FORM)
   `[LET ((NUMBER-OF-MATCHES 0))
         (SEQ-DISPATCH ,SEQUENCE (CL:IF (NULL (AND ,FROM-END ,COUNT))
                                        (CL:DO ((%%SUBSEQ (CL:NTHCDR ,START ,SEQUENCE)
                                                          (CDR %%SUBSEQ))
                                                (INDEX ,START (CL:1+ INDEX))
                                                CURRENT)
                                               ([OR (EQL INDEX ,END)
                                                    (AND ,COUNT (>= NUMBER-OF-MATCHES ,COUNT]
                                                ,SEQUENCE)
                                            (SETQ CURRENT (CAR %%SUBSEQ))
                                            (CL:IF (AND ,TEST-FORM (CL:INCF NUMBER-OF-MATCHES))
                                                (RPLACA %%SUBSEQ ,NEWITEM)))
                                        (CL:DO ((INDEX (CL:1- ,END)
                                                       (CL:1- INDEX))
                                                %%SUBSEQ CURRENT)
                                               ([OR (< INDEX ,START)
                                                    (AND ,COUNT (>= NUMBER-OF-MATCHES ,COUNT]
                                                ,SEQUENCE)
                                            (SETQ %%SUBSEQ (CL:NTHCDR INDEX ,SEQUENCE))
                                            (SETQ CURRENT (CAR %%SUBSEQ))
                                            (CL:IF (AND ,TEST-FORM (CL:INCF NUMBER-OF-MATCHES))
                                                (RPLACA %%SUBSEQ ,NEWITEM))))
                  (LET [(LENGTH (VECTOR-LENGTH ,SEQUENCE]
                       (CL:IF (NULL ,FROM-END)
                           (CL:DO ((INDEX ,START (CL:1+ INDEX))
                                   CURRENT)
                                  ([OR (EQL INDEX ,END)
                                       (AND ,COUNT (>= NUMBER-OF-MATCHES ,COUNT]
                                   ,SEQUENCE)
                               (SETQ CURRENT (CL:AREF ,SEQUENCE INDEX))
                               (CL:IF (AND ,TEST-FORM (CL:INCF NUMBER-OF-MATCHES))
                                   (CL:SETF (CL:AREF ,SEQUENCE INDEX)
                                            ,NEWITEM)))
                           (CL:DO ((INDEX (CL:1- ,END)
                                          (CL:1- INDEX))
                                   CURRENT)
```

```
                                   ([OR (< INDEX ,START)
                                        (AND ,COUNT (>= NUMBER-OF-MATCHES ,COUNT]
                                    ,SEQUENCE)
                                  (SETQ CURRENT (CL:AREF ,SEQUENCE INDEX))
                                  (CL:IF (AND ,TEST-FORM (CL:INCF NUMBER-OF-MATCHES))
                                      (CL:SETF (CL:AREF ,SEQUENCE INDEX)
                                             ,NEWITEM))))])


(CL:DEFUN COMPLEX-NSUBSTITUTE (NEWITEM OLDITEM SEQUENCE START END FROM-END KEY COUNT TEST TEST-NOT-P)
   [COMPLEX-NSUBSTITUTE-MACRO NEWITEM SEQUENCE START END FROM-END KEY COUNT
          (CL:IF TEST-NOT-P
              (NOT (CL:FUNCALL TEST OLDITEM (CL:FUNCALL KEY CURRENT)))
              (CL:FUNCALL TEST OLDITEM (CL:FUNCALL KEY CURRENT)))])


(CL:DEFUN COMPLEX-NSUBSTITUTE-IF (NEWITEM TEST SEQUENCE START END FROM-END KEY COUNT)
   (COMPLEX-NSUBSTITUTE-MACRO NEWITEM SEQUENCE START END FROM-END KEY COUNT (CL:FUNCALL TEST (CL:FUNCALL KEY
                                                                                                CURRENT)))
)


(CL:DEFUN COMPLEX-NSUBSTITUTE-IF-NOT (NEWITEM TEST SEQUENCE START END FROM-END KEY COUNT)
   [COMPLEX-NSUBSTITUTE-MACRO NEWITEM SEQUENCE START END FROM-END KEY COUNT (NOT (CL:FUNCALL TEST
                                                                                    (CL:FUNCALL KEY
                                                                                           CURRENT])


(CL:DEFUN CL:NSUBSTITUTE (NEWITEM OLDITEM SEQUENCE &KEY (START 0)
                                  END
                                  (FROM-END NIL FROM-END-P)
                                  COUNT
                                  (KEY 'CL:IDENTITY KEY-P)
                                  (TEST 'EQL TEST-P)
                                  (TEST-NOT NIL TEST-NOT-P))
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (AND TEST-P TEST-NOT-P)
               (CL:ERROR "Both test and test-not provided"))
        (CL:IF (OR FROM-END-P KEY-P COUNT TEST-P TEST-NOT-P)
            (COMPLEX-NSUBSTITUTE NEWITEM OLDITEM SEQUENCE START END FROM-END KEY COUNT (CL:IF TEST-NOT-P
                                                                                              TEST-NOT
                                                                                              TEST)
                  TEST-NOT-P)
            (SIMPLE-NSUBSTITUTE NEWITEM OLDITEM SEQUENCE START END))))


(CL:DEFUN CL:NSUBSTITUTE-IF (NEWITEM TEST SEQUENCE &KEY (START 0)
                                    END
                                    (FROM-END NIL FROM-END-P)
                                    COUNT
                                    (KEY 'CL:IDENTITY KEY-P))
   "Returns a sequence of the same kind as Sequence with the same elements except that all elements that match
   Old are replaced with New."
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (OR FROM-END-P KEY-P COUNT)
            (COMPLEX-NSUBSTITUTE-IF NEWITEM TEST SEQUENCE START END FROM-END KEY COUNT)
            (SIMPLE-NSUBSTITUTE-IF NEWITEM TEST SEQUENCE START END))))


(CL:DEFUN CL:NSUBSTITUTE-IF-NOT (NEWITEM TEST SEQUENCE &KEY (START 0)
                                        END
                                        (FROM-END NIL FROM-END-P)
                                        COUNT
                                        (KEY 'CL:IDENTITY KEY-P))
   "Returns a sequence of the same kind as Sequence with the same elements except that all elements that match
   Old are replaced with New."
   (LET ((LENGTH (CL:LENGTH SEQUENCE)))
        (CL:IF (NULL END)
               (SETQ END LENGTH))
        (CHECK-SUBSEQ SEQUENCE START END LENGTH)
        (CL:IF (OR FROM-END-P KEY-P COUNT)
            (COMPLEX-NSUBSTITUTE-IF-NOT NEWITEM TEST SEQUENCE START END FROM-END KEY COUNT)
            (SIMPLE-NSUBSTITUTE-IF-NOT NEWITEM TEST SEQUENCE START END))))

(PUTPROPS CMLSEQMODIFY FILETYPE CL:COMPILE-FILE)

(DECLARE%: DONTCOPY DONTEVAL@LOAD DOEVAL@COMPILE

(DECLARE%: DOEVAL@COMPILE DONTCOPY

(LOCALVARS . T)
```

```
)
)

(PUTPROPS CMLSEQMODIFY COPYRIGHT ("Venue & Xerox Corporation" 1986 1987 1990))
```

FUNCTION INDEX

MACRO INDEX

PROPERTY INDEX