

File created: 16-May-90 14:29:23 {DSK}<usr>local>lde>lispcore>sources>CMLSEQFINDER.;2

changes to: (VARS CMLSEQFINDERCOMS)

previous date: 12-Nov-86 18:41:14 {DSK}<usr>local>lde>lispcore>sources>CMLSEQFINDER.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

::
:: Copyright (c) 1986, 1990 by Venue & Xerox Corporation. All rights reserved.

(RPAQQ **CMLSEQFINDERCOMS**

```
((DECLARE%: EVAL@COMPILE DONTCOPY (FILES CMLSEQCOMMON))
 (FUNCTIONS SIMPLE-FIND-MACRO SIMPLE-FIND SIMPLE-FIND-IF SIMPLE-FIND-IF-NOT COMPLEX-FIND-MACRO
  COMPLEX-FIND COMPLEX-FIND-IF COMPLEX-FIND-IF-NOT CL:FIND CL:FIND-IF CL:FIND-IF-NOT)
 (FUNCTIONS SIMPLE-POSITION-MACRO SIMPLE-POSITION SIMPLE-POSITION-IF SIMPLE-POSITION-IF-NOT
  COMPLEX-POSITION-MACRO COMPLEX-POSITION COMPLEX-POSITION-IF COMPLEX-POSITION-IF-NOT CL:POSITION
  CL:POSITION-IF CL:POSITION-IF-NOT)
 (FUNCTIONS SIMPLE-COUNT-MACRO SIMPLE-COUNT SIMPLE-COUNT-IF SIMPLE-COUNT-IF-NOT COMPLEX-COUNT
  COMPLEX-COUNT-IF COMPLEX-COUNT-IF-NOT CL:COUNT CL:COUNT-IF CL:COUNT-IF-NOT)
 (FUNCTIONS COMPLEX-COMPARE-BACKWARD COMPLEX-COMPARE-FORWARD SIMPLE-COMPARE CL:MISMATCH CL:SEARCH)
 (PROP FILETYPE CMLSEQFINDER)
 (DECLARE%: DONTCOPY DOEVAL@COMPILE DONTEVAL@LOAD (LOCALVARS . T))))
```

(DECLARE%: EVAL@COMPILE DONTCOPY

(FILESLOAD CMLSEQCOMMON)

)

```
(DEFMACRO SIMPLE-FIND-MACRO (ITEM SEQUENCE START END TEST-FORM)
 '[SEQ-DISPATCH ,SEQUENCE (FORWARD-LIST-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
  NIL
  (CL:IF ,TEST-FORM (RETURN CURRENT)))
 (FORWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
  NIL
  (CL:IF ,TEST-FORM (RETURN CURRENT]))
```

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

```
(CL:DEFUN SIMPLE-FIND-IF (TEST SEQUENCE START END)
 (SIMPLE-FIND-MACRO ITEM SEQUENCE START END (CL:FUNCALL TEST CURRENT)))
```

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

```
(DEFMACRO COMPLEX-FIND-MACRO (ITEM SEQUENCE START END FROM-END KEY TEST-FORM)
 '(CL:IF (NULL ,FROM-END)
 [SEQ-DISPATCH ,SEQUENCE (FORWARD-LIST-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
  NIL
  (CL:IF ,TEST-FORM (RETURN CURRENT)))
 (FORWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
  NIL
  (CL:IF ,TEST-FORM (RETURN CURRENT))]
 [SEQ-DISPATCH ,SEQUENCE (FORWARD-LIST-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT LAST-ELEMENT)
  LAST-ELEMENT
  (CL:IF ,TEST-FORM (SETQ LAST-ELEMENT CURRENT)))
 (BACKWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
  NIL
  (CL:IF ,TEST-FORM (RETURN CURRENT]))
```

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

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

```
(CL:DEFUN CL:FIND (ITEM 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))
  "Returns the first element in SEQUENCE satisfying the test (default is EQL) with the given ITEM"
  (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 specified"))
    (CL:IF (OR FROM-END-P KEY-P TEST-P TEST-NOT-P)
      (COMPLEX-FIND ITEM SEQUENCE START END FROM-END KEY (CL:IF TEST-NOT-P
        TEST-NOT
        TEST)
        TEST-NOT-P)
      (SIMPLE-FIND ITEM SEQUENCE START END))))
```

```
(CL:DEFUN CL:FIND-IF (TEST SEQUENCE &KEY (START 0)
  END
  (FROM-END NIL FROM-END-P)
  (KEY 'CL:IDENTITY KEY-P))
  "Returns the zero-origin index of the first element satisfying the test."
  (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)
      (COMPLEX-FIND-IF TEST SEQUENCE START END FROM-END KEY)
      (SIMPLE-FIND-IF TEST SEQUENCE START END))))
```

```
(CL:DEFUN CL:FIND-IF-NOT (TEST SEQUENCE &KEY (START 0)
  END
  (FROM-END NIL FROM-END-P)
  (KEY 'CL:IDENTITY KEY-P))
  "Returns the zero-origin index of the first element not satisfying the test."
  (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)
      (COMPLEX-FIND-IF-NOT TEST SEQUENCE START END FROM-END KEY)
      (SIMPLE-FIND-IF-NOT TEST SEQUENCE START END))))
```

```
(DEFMACRO SIMPLE-POSITION-MACRO (ITEM SEQUENCE START END TEST-FORM)
  `[SEQ-DISPATCH ,SEQUENCE (FORWARD-LIST-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
    NIL
    (CL:IF ,TEST-FORM (RETURN INDEX)))
  (FORWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
    NIL
    (CL:IF ,TEST-FORM (RETURN INDEX]))
```

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

```
(CL:DEFUN SIMPLE-POSITION-IF (TEST SEQUENCE START END)
  (SIMPLE-POSITION-MACRO ITEM SEQUENCE START END (CL:FUNCALL TEST CURRENT)))
```

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

```
(DEFMACRO COMPLEX-POSITION-MACRO (ITEM SEQUENCE START END FROM-END KEY TEST-FORM)
  `(CL:IF (NULL ,FROM-END)
    [SEQ-DISPATCH ,SEQUENCE (FORWARD-LIST-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
      NIL
      (CL:IF ,TEST-FORM (RETURN INDEX)))
    (FORWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
      NIL
      (CL:IF ,TEST-FORM (RETURN INDEX))]
    [SEQ-DISPATCH ,SEQUENCE (FORWARD-LIST-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT LAST-INDEX)
      LAST-INDEX
      (CL:IF ,TEST-FORM (SETQ LAST-INDEX INDEX)))
    (BACKWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT)
      NIL
      (CL:IF ,TEST-FORM (RETURN INDEX]))
```

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

```
(CL:DEFUN COMPLEX-POSITION-IF-NOT (TEST SEQUENCE START END FROM-END KEY)
  [COMPLEX-POSITION-MACRO ITEM SEQUENCE START END FROM-END KEY (NOT (CL:FUNCALL TEST (CL:FUNCALL KEY CURRENT))
  )
```

```
(CL:DEFUN CL:POSITION (ITEM 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))
  "Returns the zero-origin index of the first element in SEQUENCE satisfying the test (default is EQL) with the
  given ITEM"
  (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 specified"))
    (CL:IF (OR FROM-END-P KEY-P TEST-P TEST-NOT-P)
      (COMPLEX-POSITION ITEM SEQUENCE START END FROM-END KEY (CL:IF TEST-NOT-P
        TEST-NOT
        TEST)
        TEST-NOT-P)
      (SIMPLE-POSITION ITEM SEQUENCE START END))))
```

```
(CL:DEFUN CL:POSITION-IF (TEST SEQUENCE &KEY (START 0)
  END
  (FROM-END NIL FROM-END-P)
  (KEY 'CL:IDENTITY KEY-P))
  "Returns the zero-origin index of the first element satisfying test(e1)"
  (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)
      (COMPLEX-POSITION-IF TEST SEQUENCE START END FROM-END KEY)
      (SIMPLE-POSITION-IF TEST SEQUENCE START END))))
```

```
(CL:DEFUN CL:POSITION-IF-NOT (TEST SEQUENCE &KEY (START 0)
  END
  (FROM-END NIL FROM-END-P)
  (KEY 'CL:IDENTITY KEY-P))
  "Returns the zero-origin index of the first element not satisfying test(e1)"
  (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)
      (COMPLEX-POSITION-IF-NOT TEST SEQUENCE START END FROM-END KEY)
      (SIMPLE-POSITION-IF-NOT TEST SEQUENCE START END))))
```

```
(DEFMACRO SIMPLE-COUNT-MACRO (ITEM SEQUENCE START END TEST-FORM)
  `[SEQ-DISPATCH ,SEQUENCE [FORWARD-LIST-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT (CNT 0))
    CNT
    (CL:IF ,TEST-FORM
      (SETQ CNT (CL:1+ CNT)))]
  (FORWARD-VECTOR-LOOP ,SEQUENCE ,START ,END (INDEX CURRENT (CNT 0))
    CNT
    (CL:IF ,TEST-FORM
      (SETQ CNT (CL:1+ CNT)))]])
```

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

```
(CL:DEFUN SIMPLE-COUNT-IF (TEST SEQUENCE START END)
  (SIMPLE-COUNT-MACRO ITEM SEQUENCE START END (CL:FUNCALL TEST CURRENT)))
```

```
(CL:DEFUN SIMPLE-COUNT-IF-NOT (TEST SEQUENCE START END)
```

(SIMPLE-COUNT-MACRO ITEM SEQUENCE START END (NOT (CL:FUNCALL TEST CURRENT))))

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

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

(CL:DEFUN CL:COUNT (ITEM SEQUENCE &KEY (START 0)
END FROM-END (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 specified"))
(CL:IF (OR KEY-P TEST-P TEST-NOT-P)
(COMPLEX-COUNT ITEM SEQUENCE START END KEY (CL:IF TEST-NOT-P
TEST-NOT
TEST)
TEST-NOT-P)
(SIMPLE-COUNT ITEM SEQUENCE START END))))

(CL:DEFUN CL:COUNT-IF (TEST SEQUENCE &KEY (START 0)
END FROM-END (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 KEY-P
(COMPLEX-COUNT-IF TEST SEQUENCE START END KEY)
(SIMPLE-COUNT-IF TEST SEQUENCE START END))))

(CL:DEFUN CL:COUNT-IF-NOT (TEST SEQUENCE &KEY (START 0)
END FROM-END (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 KEY-P
(COMPLEX-COUNT-IF-NOT TEST SEQUENCE START END KEY)
(SIMPLE-COUNT-IF-NOT TEST SEQUENCE START END))))

(CL:DEFUN COMPLEX-COMPARE-BACKWARD (SEQUENCE1 SEQUENCE2 START1 END1 START2 END2 KEY TEST TEST-NOT-P)
[LET ((LEN1 (- END1 START1))
(LEN2 (- END2 START2)))
(CL:IF (> LEN1 LEN2)
(SETQ START1 (- END1 LEN2))
(SETQ START2 (- END2 LEN1)))
(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))
(LAST-MISMATCH (CL:1- START1))
TEST-RESULT)
((EQL INDEX1 END1)
(CL:1+ LAST-MISMATCH))
[SETQ TEST-RESULT (CL:FUNCALL TEST
(CL:FUNCALL KEY
(CAR SUBSEQ1))
(CL:FUNCALL KEY
(CAR SUBSEQ2))
(CL:IF (CL:IF TEST-NOT-P
TEST-RESULT
(NOT TEST-RESULT))
(SETQ LAST-MISMATCH INDEX1))]
(CL:DO ((SUBSEQ1 (CL:NTHCDR START1 SEQUENCE1)
(CDR SUBSEQ1))
(INDEX1 START1 (CL:1+ INDEX1))
(INDEX2 START2 (CL:1+ INDEX2))
(LAST-MISMATCH (CL:1- START1))
TEST-RESULT)

```

      ((EQL INDEX1 END1)
       (CL:1+ LAST-MISMATCH))
[SETQ TEST-RESULT (CL:FUNCALL TEST (CL:FUNCALL KEY (CAR SUBSEQ1))
                                   (CL:FUNCALL KEY (CL:AREF SEQUENCE2 INDEX2])
      (CL:IF (CL:IF TEST-NOT-P
              TEST-RESULT
              (NOT TEST-RESULT))
              (SETQ LAST-MISMATCH INDEX1))))]
(SEQ-DISPATCH SEQUENCE2 (CL:DO ((SUBSEQ2 (CL:NTHCDR START2 SEQUENCE2)
                                       (CDR SUBSEQ2))
                                (INDEX1 START1 (CL:1+ INDEX1))
                                (INDEX2 START2 (CL:1+ INDEX2))
                                (LAST-MISMATCH (CL:1- START1))
                                TEST-RESULT)
      ((EQL INDEX1 END1)
       (CL:1+ LAST-MISMATCH))
[SETQ TEST-RESULT (CL:FUNCALL TEST (CL:FUNCALL KEY (CL:AREF SEQUENCE1
                                                    INDEX1))
                                   (CL:FUNCALL KEY (CAR SUBSEQ2])
      (CL:IF (CL:IF TEST-NOT-P
              TEST-RESULT
              (NOT TEST-RESULT))
              (SETQ LAST-MISMATCH INDEX1))))]
(CL:DO ((INDEX1 (CL:1- END1)
         (CL:1- INDEX1))
        (INDEX2 (CL:1- END2)
         (CL:1- INDEX2))
        TEST-RESULT)
  ([OR (< INDEX1 START1)
        (PROGN [SETQ TEST-RESULT (CL:FUNCALL TEST (CL:FUNCALL KEY (CL:AREF SEQUENCE1
                                                                    INDEX1))
                                                    (CL:FUNCALL KEY (CL:AREF SEQUENCE2 INDEX2])
        (CL:IF TEST-NOT-P
          TEST-RESULT
          (NOT TEST-RESULT))])
    (CL:1+ INDEX1))))])

```

(CL:DEFUN **COMPLEX-COMPARE-FORWARD** (SEQUENCE1 SEQUENCE2 START1 END1 START2 END2 KEY TEST TEST-NOT-P)

```

[LET ((LEN1 (- END1 START1))
      (LEN2 (- END2 START2)))
  (CL:IF (> LEN1 LEN2)
    (SETQ END1 (+ START1 LEN2))
    (SETQ END2 (+ START2 LEN1)))
  (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))
                                                            TEST-RESULT)
          ([OR (EQL INDEX1 END1)
                (PROGN [SETQ TEST-RESULT
                        (CL:FUNCALL TEST (CL:FUNCALL
                                          KEY
                                          (CAR SUBSEQ1))
                                          (CL:FUNCALL KEY (CAR SUBSEQ2])
          (CL:IF TEST-NOT-P
            TEST-RESULT
            (NOT TEST-RESULT))])
            INDEX1))
        (CL:DO ((SUBSEQ1 (CL:NTHCDR START1 SEQUENCE1)
                       (CDR SUBSEQ1))
                 (INDEX1 START1 (CL:1+ INDEX1))
                 (INDEX2 START2 (CL:1+ INDEX2))
                 TEST-RESULT)
          ([OR (EQL INDEX1 END1)
                (PROGN [SETQ TEST-RESULT (CL:FUNCALL TEST (CL:FUNCALL
                                                            KEY
                                                            (CAR SUBSEQ1))
                                                            (CL:FUNCALL KEY
                                                                (CL:AREF SEQUENCE2
                                                                INDEX2])
          (CL:IF TEST-NOT-P
            TEST-RESULT
            (NOT TEST-RESULT))])
            INDEX1)))
        (SEQ-DISPATCH SEQUENCE2 (CL:DO ((SUBSEQ2 (CL:NTHCDR START2 SEQUENCE2)
                                                   (CDR SUBSEQ2))
                                           (INDEX1 START1 (CL:1+ INDEX1))
                                           TEST-RESULT)
          ([OR (EQL INDEX1 END1)
                (PROGN [SETQ TEST-RESULT (CL:FUNCALL TEST
                                          (CL:FUNCALL KEY
                                          (CL:AREF SEQUENCE1
                                          INDEX1))
                                          (CL:FUNCALL KEY (CAR SUBSEQ2])
          (CL:IF TEST-NOT-P
            TEST-RESULT
            (NOT TEST-RESULT))])
            INDEX1))))])

```

```

TEST-RESULT
(NOT TEST-RESULT))]]
      INDEX1))
(CL:DO ((INDEX1 START1 (CL:1+ INDEX1))
        (INDEX2 START2 (CL:1+ INDEX2))
        TEST-RESULT)
      ([OR (EQL INDEX1 END1)
          (PROGN [SETQ TEST-RESULT (CL:FUNCALL TEST (CL:FUNCALL KEY (CL:AREF SEQUENCE1
                                                                INDEX1))
                                                    (CL:FUNCALL KEY (CL:AREF SEQUENCE2 INDEX2]
                                                                TEST-RESULT
                                                                (NOT TEST-RESULT)))]
          INDEX1))]))

(CL:DEFUN SIMPLE-COMPARE (SEQUENCE1 SEQUENCE2 START1 END1 START2 END2)
  [LET ((LEN1 (- END1 START1))
        (LEN2 (- END2 START2)))
    (CL:IF (> LEN1 LEN2)
      (SETQ END1 (+ START1 LEN2))
      (SETQ END2 (+ START2 LEN1)))
    (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)))
                                                                ([OR (EQL INDEX1 END1)
                                                                    (NOT (EQL (CAR SUBSEQ1)
                                                                (CAR SUBSEQ2]
                                                                INDEX1))
                                                                (CL:DO ((SUBSEQ1 (CL:NTHCDR START1 SEQUENCE1)
                                                                    (CDR SUBSEQ1))
                                                                (INDEX1 START1 (CL:1+ INDEX1))
                                                                (INDEX2 START2 (CL:1+ INDEX2)))
                                                                ([OR (EQL INDEX1 END1)
                                                                    (NOT (EQL (CAR SUBSEQ1)
                                                                (CL:AREF SEQUENCE2 INDEX2]
                                                                INDEX1)))
                                                                (SEQ-DISPATCH SEQUENCE2 (CL:DO ((SUBSEQ2 (CL:NTHCDR START2 SEQUENCE2)
                                                                    (CDR SUBSEQ2))
                                                                (INDEX1 START1 (CL:1+ INDEX1)))
                                                                ([OR (EQL INDEX1 END1)
                                                                    (NOT (EQL (CL:AREF SEQUENCE1 INDEX1)
                                                                (CAR SUBSEQ2]
                                                                INDEX1))
                                                                (CL:DO ((INDEX1 START1 (CL:1+ INDEX1))
                                                                (INDEX2 START2 (CL:1+ INDEX2)))
                                                                ([OR (EQL INDEX1 END1)
                                                                    (NOT (EQL (CL:AREF SEQUENCE1 INDEX1)
                                                                (CL:AREF SEQUENCE2 INDEX2]
                                                                INDEX1))]))))

(CL:DEFUN CL:MISMATCH (SEQUENCE1 SEQUENCE2 &KEY (START1 0)
  END1
  (START2 0)
  END2
  (FROM-END NIL FROM-END-P)
  (TEST 'EQL TEST-P)
  (TEST-NOT NIL TEST-NOT-P)
  (KEY 'CL:IDENTITY KEY-P))
  [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)
    (CL:IF (AND TEST-P TEST-NOT-P)
      (CL:ERROR "Both Test and test-not provided")))
  (LET ((SUBLEN1 (- END1 START1))
        (SUBLEN2 (- END2 START2)))
    (CL:IF FROM-END
      (LET ((INDEX (COMPLEX-COMPARE-BACKWARD SEQUENCE1 SEQUENCE2 START1 END1 START2 END2 KEY
        (CL:IF TEST-NOT-P
          TEST-NOT
          TEST)
          TEST-NOT-P)))
        (CL:IF (AND (EQL INDEX START1)
                    (EQL SUBLEN1 SUBLEN2))
          NIL
          INDEX))
      (LET [(INDEX (CL:IF (OR KEY-P TEST-P TEST-NOT-P KEY-P)
        (COMPLEX-COMPARE-FORWARD SEQUENCE1 SEQUENCE2 START1 END1 START2 END2 KEY
          (CL:IF TEST-NOT-P

```

```

TEST-NOT
TEST
TEST-NOT-P
(SIMPLE-COMPARE SEQUENCE1 SEQUENCE2 START1 END1 START2 END2)))
(CL:IF (AND (EQL INDEX END1)
            (EQL SUBLN1 SUBLN2))
      NIL
      INDEX)))]

```

```

(CL:DEFUN CL:SEARCH (SEQUENCE1 SEQUENCE2 &KEY (START1 0)
                   END1
                   (START2 0)
                   END2
                   (FROM-END NIL FROM-END-P)
                   (TEST 'EQL TEST-P)
                   (TEST-NOT NIL TEST-NOT-P)
                   (KEY 'CL:IDENTITY KEY-P))

```

"A search is conducted for the first subsequence of sequence2 which element-wise matches sequence1. If there is such a subsequence in sequence2, the index of the its leftmost element is returned otherwise () is returned."

```

[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)
  (CL:IF (AND TEST-P TEST-NOT-P)
        (CL:ERROR "Both Test and test-not provided"))
  (LET ((SUBLN1 (- END1 START1))
        (SUBLN2 (- END2 START2)))
    (CL:IF (NULL FROM-END)
          (CL:IF (NOT (OR TEST-P TEST-NOT-P KEY-P))
                (CL:DO ((SUBSTART2 START2 (CL:1+ SUBSTART2))
                       (END-SEARCH (- END2 SUBLN1)))
                      ((> SUBSTART2 END-SEARCH))
                    (CL:IF (EQL (SIMPLE-COMPARE SEQUENCE1 SEQUENCE2 START1 END1 SUBSTART2 END2)
                              END1)
                        (RETURN SUBSTART2)))
              (CL:DO ((SUBSTART2 START2 (CL:1+ SUBSTART2))
                     (END-SEARCH (- END2 SUBLN1))
                     (PREDICATE (CL:IF TEST-NOT-P
                                         TEST-NOT
                                         TEST))
                           INDEX)
                   ((> SUBSTART2 END-SEARCH))
                 (SETQ INDEX (COMPLEX-COMPARE-FORWARD SEQUENCE1 SEQUENCE2 START1 END1 SUBSTART2 END2
                                                       KEY PREDICATE TEST-NOT-P))
               (CL:IF (EQL INDEX END1)
                     (RETURN SUBSTART2))))))
    (CL:IF (NOT (OR TEST-P TEST-NOT-P KEY-P))
          (CL:DO ((SUBSTART2 (- END2 SUBLN1)
                    (CL:1- SUBSTART2)))
                ((< SUBSTART2 START2))
              (CL:IF (EQL (SIMPLE-COMPARE SEQUENCE1 SEQUENCE2 START1 END1 SUBSTART2 END2)
                        END1)
                    (RETURN SUBSTART2)))
          (CL:DO ((SUBSTART2 (- END2 SUBLN1)
                    (CL:1- SUBSTART2))
                (PREDICATE (CL:IF TEST-NOT-P
                                    TEST-NOT
                                    TEST)))
                ((< SUBSTART2 START2))
              (CL:IF (EQL (COMPLEX-COMPARE-FORWARD SEQUENCE1 SEQUENCE2 START1 END1 SUBSTART2 END2
                                                    KEY PREDICATE TEST-NOT-P)
                      END1)
                    (RETURN SUBSTART2)))))))]

```

```
(PUTPROPS CMLSEQFINDER FILETYPE CL:COMPILE-FILE)
```

```
(DECLARE%: DONTCOPY DOEVAL@COMPILE DONTEVAL@LOAD
```

```
(DECLARE%: DOEVAL@COMPILE DONTCOPY
```

```
(LOCALVARS . T)
)
)
```

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

FUNCTION INDEX

COMPLEX-COMPARE-BACKWARD	4	COMPLEX-POSITION3	CL:FIND-IF-NOT2	SIMPLE-COUNT-IF3
COMPLEX-COMPARE-FORWARD	.5	COMPLEX-POSITION-IF3	CL:MISMATCH6	SIMPLE-COUNT-IF-NOT3
COMPLEX-COUNT4	COMPLEX-POSITION-IF-NOT	.3	CL:POSITION3	SIMPLE-FIND1
COMPLEX-COUNT-IF4	CL:COUNT4	CL:POSITION-IF3	SIMPLE-FIND-IF1
COMPLEX-COUNT-IF-NOT4	CL:COUNT-IF4	CL:POSITION-IF-NOT3	SIMPLE-FIND-IF-NOT1
COMPLEX-FIND1	CL:COUNT-IF-NOT4	CL:SEARCH7	SIMPLE-POSITION2
COMPLEX-FIND-IF1	CL:FIND2	SIMPLE-COMPARE6	SIMPLE-POSITION-IF2
COMPLEX-FIND-IF-NOT1	CL:FIND-IF2	SIMPLE-COUNT3	SIMPLE-POSITION-IF-NOT	..2

MACRO INDEX

COMPLEX-FIND-MACRO1	SIMPLE-COUNT-MACRO3	SIMPLE-POSITION-MACRO2
COMPLEX-POSITION-MACRO2	SIMPLE-FIND-MACRO1		

PROPERTY INDEX

CMLSEQFINDER7
--------------	--------
