17. Sequences

17.1 Sequence Concepts

A *sequence* is an ordered collection of *elements*, implemented as either a *vector* or a *list*.

Sequences can be created by the *function* **make-sequence**, as well as other *functions* that create *objects* of *types* that are *subtypes* of **sequence** (e.g., **list**, **make-list**, **mapcar**, and **vector**).

A *sequence function* is a *function* defined by this specification or added as an extension by the *implementation* that operates on one or more *sequences*. Whenever a *sequence function* must construct and return a new *vector*, it always returns a *simple vector*. Similarly, any *strings* constructed will be *simple strings*.

`concatenate`	`length`	`remove`
`copy-seq`	`map`	`remove-duplicates`
`count`	`map-into`	`remove-if`
`count-if`	`merge`	`remove-if-not`
`count-if-not`	`mismatch`	`replace`
`delete`	`notany`	`reverse`
`delete-duplicates`	`notevery`	`search`
`delete-if`	`nreverse`	`some`
`delete-if-not`	`nsubstitute`	`sort`
`elt`	`nsubstitute-if`	`stable-sort`
`every`	`nsubstitute-if-not`	`subseq`
`fill`	`position`	`substitute`
`find`	`position-if`	`substitute-if`
`find-if`	`position-if-not`	`substitute-if-not`
`find-if-not`	`reduce`	

Figure 17-1. Standardized Sequence Functions

17.1.1 General Restrictions on Parameters that must be Sequences

In general, *lists* (including *association lists* and *property lists*) that are treated as *sequences* must be *proper lists*.

17.2 Rules about Test Functions

17.2.1 Satisfying a Two-Argument Test

When an *object* O is being considered iteratively against each *element* Ei of a *sequence* S by an *operator* F listed in the next figure, it is sometimes useful to control the way in which the presence of O is tested in S is tested by F. This control is offered on the basis of a *function* designated with either a `:test` or `:test-not` *argument*.

`adjoin`	`nset-exclusive-or`	`search`
`assoc`	`nsublis`	`set-difference`
`count`	`nsubst`	`set-exclusive-or`
`delete`	`nsubstitute`	`sublis`
`find`	`nunion`	`subsetp`
`intersection`	`position`	`subst`
`member`	`pushnew`	`substitute`
`mismatch`	`rassoc`	`tree-equal`
`nintersection`	`remove`	`union`
`nset-difference`	`remove-duplicates`	

Figure 17-2. Operators that have Two-Argument Tests to be Satisfied

The object O might not be compared directly to Ei. If a `:key` *argument* is provided, it is a *designator* for a *function* of one *argument* to be called with each Ei as an *argument*, and *yielding* an *object* Zi to be used for comparison. (If there is no `:key` *argument*, Zi is Ei.)

The *function* designated by the `:key` *argument* is never called on O itself. However, if the function operates on multiple sequences (e.g., as happens in **set-difference**), O will be the result of calling the `:key` function on an *element* of the other sequence.

A `:test` *argument*, if supplied to F, is a *designator* for a *function* of two *arguments*, O and Zi. An Ei is said (or, sometimes, an O and an Ei are said) to *satisfy the test* if this `:test` *function* returns a *generalized boolean* representing *true*.

A `:test-not` *argument*, if supplied to F, is *designator* for a *function* of two *arguments*, O and Zi. An Ei is said (or, sometimes, an O and an Ei are said) to *satisfy the test* if this `:test-not` *function* returns a *generalized boolean* representing *false*.

If neither a `:test` nor a `:test-not` *argument* is supplied, it is as if a `:test` argument of `#'eql` was supplied.

The consequences are unspecified if both a `:test` and a `:test-not` *argument* are supplied in the same *call* to F.

17.2.1.1 Examples of Satisfying a Two-Argument Test

```
 (remove "FOO" '(foo bar "FOO" "BAR" "foo" "bar") :test #'equal)
=>  (foo bar "BAR" "foo" "bar")
 (remove "FOO" '(foo bar "FOO" "BAR" "foo" "bar") :test #'equalp)
=>  (foo bar "BAR" "bar")
 (remove "FOO" '(foo bar "FOO" "BAR" "foo" "bar") :test #'string-equal)
=>  (bar "BAR" "bar")
 (remove "FOO" '(foo bar "FOO" "BAR" "foo" "bar") :test #'string=)
=>  (BAR "BAR" "foo" "bar")

 (remove 1 '(1 1.0 #C(1.0 0.0) 2 2.0 #C(2.0 0.0)) :test-not #'eql)
=>  (1)
 (remove 1 '(1 1.0 #C(1.0 0.0) 2 2.0 #C(2.0 0.0)) :test-not #'=)
=>  (1 1.0 #C(1.0 0.0))
 (remove 1 '(1 1.0 #C(1.0 0.0) 2 2.0 #C(2.0 0.0)) :test (complement #'=))
=>  (1 1.0 #C(1.0 0.0))

 (count 1 '((one 1) (uno 1) (two 2) (dos 2)) :key #'cadr) =>  2

 (count 2.0 '(1 2 3) :test #'eql :key #'float) =>  1

 (count "FOO" (list (make-pathname :name "FOO" :type "X")
                    (make-pathname :name "FOO" :type "Y"))
        :key #'pathname-name
        :test #'equal)
=>  2
```

17.2.2 Satisfying a One-Argument Test

When using one of the *functions* in the next figure, the elements E of a *sequence* S are filtered not on the basis of the presence or absence of an object O under a two *argument predicate*, as with the *functions* described in Section 17.2.1 (Satisfying a Two-Argument Test), but rather on the basis of a one *argument predicate*.

```
assoc-if       member-if           rassoc-if
assoc-if-not   member-if-not       rassoc-if-not
count-if       nsubst-if           remove-if
count-if-not   nsubst-if-not       remove-if-not
delete-if      nsubstitute-if      subst-if
delete-if-not  nsubstitute-if-not  subst-if-not
find-if        position-if         substitute-if
find-if-not    position-if-not     substitute-if-not
```

Figure 17-3. Operators that have One-Argument Tests to be Satisfied

The element Ei might not be considered directly. If a `:key` *argument* is provided, it is a *designator* for a *function* of one *argument* to be called with each Ei as an *argument*, and *yielding* an *object* Zi to be used for comparison. (If there is no `:key` *argument*, Zi is Ei.)

Functions defined in this specification and having a name that ends in "`-if`" accept a first *argument* that is a *designator* for a *function* of one *argument*, Zi. An Ei is said to *satisfy the test* if this `:test` *function* returns a *generalized boolean* representing *true*.

Functions defined in this specification and having a name that ends in "`-if-not`" accept a first *argument* that is a *designator* for a *function* of one *argument*, Zi. An Ei is said to *satisfy the test* if this `:test` *function* returns a *generalized boolean* representing *false*.

17.2.2.1 Examples of Satisfying a One-Argument Test

```
 (count-if #'zerop '(1 #C(0.0 0.0) 0 0.0d0 0.0s0 3)) =>  4

 (remove-if-not #'symbolp '(0 1 2 3 4 5 6 7 8 9 A B C D E F))
=>  (A B C D E F)
 (remove-if (complement #'symbolp) '(0 1 2 3 4 5 6 7 8 9 A B C D E F))
=>  (A B C D E F)

 (count-if #'zerop '("foo" "" "bar" "" "" "baz" "quux") :key #'length)
=>  3
```