Quick Reference

cl

Common lisp

Bert Burgemeister

Contents

1 Numbers 3
1.1 Predicates 3
1.2 Numeric Functions . . 3
1.3 Logic Functions 5
1.4 Integer Functions . . . 6
1.5 Implementation-Dependent 6

2 Characters 7

3 Strings 8

4 Conses 8
4.1 Predicates 8
4.2 Lists 9
4.3 Association Lists 10
4.4 Trees 10
4.5 Sets 11

5 Arrays 11
5.1 Predicates 11
5.2 Array Functions 11
5.3 Vector Functions . . . 12

6 Sequences 12
6.1 Sequence Predicates . . 12
6.2 Sequence Functions . . 13

7 Hash Tables 15

8 Structures 16

9 Control Structure 16
9.1 Predicates 16
9.2 Variables 17
9.3 Functions 18
9.4 Macros 19
9.5 Control Flow 21
9.6 Iteration 22
9.7 Loop Facility 22

10 CLOS 25
10.1 Classes 25
10.2 Generic Functions . . . 26
10.3 Method Combination Types 28

11 Conditions and Errors 29

12 Types and Classes 31

13 Input/Output 33
13.1 Predicates 33
13.2 Reader 34
13.3 Character Syntax . . . 35
13.4 Printer 36
13.5 Format 38
13.6 Streams 41
13.7 Pathnames and Files . 42

14 Packages and Symbols 44
14.1 Predicates 44
14.2 Packages 44
14.3 Symbols 45
14.4 Standard Packages . . 46

15 Compiler 46
15.1 Predicates 46
15.2 Compilation 46
15.3 REPL and Debugging . 48
15.4 Declarations 49

16 External Environment 49

Typographic Conventions

name; $_f$**name**; $_g$**name**; $_m$**name**; $_s$**name**; $_v$***name***; $_c$**name**
▷ Symbol defined in Common Lisp; esp. function, generic function, macro, special operator, variable, constant.

them ▷ Placeholder for actual code.

`me` ▷ Literal text.

$[foo_{\boxed{\texttt{bar}}}]$ ▷ Either one *foo* or nothing; defaults to `bar`.

foo^*; $\{foo\}^*$ ▷ Zero or more *foo*s.

foo^+; $\{foo\}^+$ ▷ One or more *foo*s.

foos ▷ English plural denotes a list argument.

$\{foo|bar|baz\}$; $\begin{cases}foo\\bar\\baz\end{cases}$ ▷ Either *foo*, or *bar*, or *baz*.

$\left\{\begin{array}{|l}foo\\bar\\baz\end{array}\right.$ ▷ Anything from none to each of *foo*, *bar*, and *baz*.

$\widehat{foo}$ ▷ Argument *foo* is not evaluated.

$\widetilde{bar}$ ▷ Argument *bar* is possibly modified.

$foo^{\mathrm{P}*}$ ▷ foo^* is evaluated as in $_s$**progn**; see page 21.

$\underline{foo}$; $\underset{2}{\underline{bar}}$; $\underset{n}{\underline{baz}}$ ▷ Primary, secondary, and *n*th return value.

`T`; `NIL` ▷ **t**, or truth in general; and **nil** or **()**.

1 Numbers

1.1 Predicates

($_f$**=** *number*$^+$)
($_f$**/=** *number*$^+$)
▷ $\underline{\mathtt{T}}$ if all *number*s, or none, respectively, are equal in value.

($_f$**>** *number*$^+$)
($_f$**>=** *number*$^+$)
($_f$**<** *number*$^+$)
($_f$**<=** *number*$^+$)
▷ Return $\underline{\mathtt{T}}$ if *number*s are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively.

($_f$**minusp** a)
($_f$**zerop** a)
($_f$**plusp** a)
▷ $\underline{\mathtt{T}}$ if $a < 0$, $a = 0$, or $a > 0$, respectively.

($_f$**evenp** *int*)
($_f$**oddp** *int*)
▷ $\underline{\mathtt{T}}$ if *int* is even or odd, respectively.

($_f$**numberp** *foo*)
($_f$**realp** *foo*)
($_f$**rationalp** *foo*)
($_f$**floatp** *foo*)
($_f$**integerp** *foo*)
($_f$**complexp** *foo*)
($_f$**random-state-p** *foo*)
▷ $\underline{\mathtt{T}}$ if *foo* is of indicated type.

1.2 Numeric Functions

($_f$**+** $a_{\boxed{0}}{}^*$)
($_f$***** $a_{\boxed{1}}{}^*$)
▷ Return $\underline{\sum a}$ or $\underline{\prod a}$, respectively.

($_f$**−** a b^*)
($_f$**/** a b^*)
▷ Return $\underline{a - \sum b}$ or $\underline{a/\prod b}$, respectively. Without any bs, return $\underline{-a}$ or $\underline{1/a}$, respectively.

($_f$**1+** a)
($_f$**1−** a)
▷ Return $\underline{a+1}$ or $\underline{a-1}$, respectively.

($\left\{\begin{matrix} _m\textbf{incf} \\ _m\textbf{decf} \end{matrix}\right\}$ $\widetilde{place}$ [$delta_{\boxed{1}}$])
▷ Increment or decrement the value of *place* by *delta*. Return $\underline{\text{new value}}$.

($_f$**exp** p)
($_f$**expt** b p)
▷ Return $\underline{e^p}$ or $\underline{b^p}$, respectively.

($_f$**log** a [$b_{\boxed{e}}$]) ▷ Return $\underline{\log_b a}$ or, without b, $\underline{\ln a}$.

($_f$**sqrt** n)
($_f$**isqrt** n)
▷ $\underline{\sqrt{n}}$ in complex numbers/natural numbers.

($_f$**lcm** *integer*$^*_{\boxed{1}}$)
($_f$**gcd** *integer**)
▷ $\underline{\text{Least common multiple}}$ or $\underline{\text{greatest common denominator}}$, respectively, of *integer*s. (**gcd**) returns $\underline{0}$.

$_c$**pi** ▷ **long-float** approximation of π, Ludolph's number.

($_f$**sin** a)
($_f$**cos** a)
($_f$**tan** a)
▷ $\underline{\sin a}$, $\underline{\cos a}$, or $\underline{\tan a}$, respectively. (a in radians.)

($_f$**asin** a)
($_f$**acos** a)
▷ $\underline{\arcsin a}$ or $\underline{\arccos a}$, respectively, in radians.

(${}_f$**atan** a [$b_{\boxed{1}}$]) ▷ $\underline{\arctan \frac{a}{b}}$ in radians.

(${}_f$**sinh** a)
(${}_f$**cosh** a) ▷ $\underline{\sinh a}$, $\underline{\cosh a}$, or $\underline{\tanh a}$, respectively.
(${}_f$**tanh** a)

(${}_f$**asinh** a)
(${}_f$**acosh** a) ▷ $\underline{\operatorname{asinh} a}$, $\underline{\operatorname{acosh} a}$, or $\underline{\operatorname{atanh} a}$, respectively.
(${}_f$**atanh** a)

(${}_f$**cis** a) ▷ Return $\underline{e^{i\,a}} = \underline{\cos a + i \sin a}$.

(${}_f$**conjugate** a) ▷ Return complex $\underline{\text{conjugate of } a}$.

(${}_f$**max** num^+)
(${}_f$**min** num^+) ▷ $\underline{\text{Greatest}}$ or $\underline{\text{least}}$, respectively, of *nums*.

$$\left(\left\{\begin{array}{l}\{{}_f\textbf{round}|{}_f\textbf{fround}\}\\ \{{}_f\textbf{floor}|{}_f\textbf{ffloor}\}\\ \{{}_f\textbf{ceiling}|{}_f\textbf{fceiling}\}\\ \{{}_f\textbf{truncate}|{}_f\textbf{ftruncate}\}\end{array}\right\} n\ [d_{\boxed{1}}]\right)$$

▷ Return as **integer** or **float**, respectively, $\underline{n/d}$ rounded, or rounded towards $-\infty$, $+\infty$, or 0, respectively; and $\underline{\text{remainder}}_2$.

$$\left(\left\{\begin{array}{l}{}_f\textbf{mod}\\ {}_f\textbf{rem}\end{array}\right\} n\ d\right)$$

▷ Same as ${}_f$**floor** or ${}_f$**truncate**, respectively, but return $\underline{\text{remainder}}$ only.

(${}_f$**random** *limit* [$\widetilde{state}_{\boxed{{}_v\textbf{*random-state*}}}$])

▷ Return non-negative $\underline{\text{random number}}$ less than *limit*, and of the same type.

(${}_f$**make-random-state** $\left[\{state|\texttt{NIL}|\texttt{T}\}_{\boxed{\texttt{NIL}}}\right]$)

▷ $\underline{\text{Copy}}$ of **random-state** object *state* or of the current random state; or a randomly initialized fresh $\underline{\text{random state}}$.

${}_v$***random-state*** ▷ Current random state.

(${}_f$**float-sign** *num-a* [$num\text{-}b_{\boxed{1}}$]) ▷ $\underline{num\text{-}b}$ with *num-a*'s sign.

(${}_f$**signum** n)

▷ $\underline{\text{Number}}$ of magnitude 1 representing sign or phase of n.

(${}_f$**numerator** *rational*)
(${}_f$**denominator** *rational*)

▷ $\underline{\text{Numerator}}$ or $\underline{\text{denominator}}$, respectively, of *rational*'s canonical form.

(${}_f$**realpart** *number*)
(${}_f$**imagpart** *number*)

▷ $\underline{\text{Real part}}$ or $\underline{\text{imaginary part}}$, respectively, of *number*.

(${}_f$**complex** *real* [$imag_{\boxed{0}}$]) ▷ Make a $\underline{\text{complex number}}$.

(${}_f$**phase** *num*) ▷ $\underline{\text{Angle}}$ of *num*'s polar representation.

(${}_f$**abs** n) ▷ Return $\underline{|n|}$.

(${}_f$**rational** *real*)
(${}_f$**rationalize** *real*)

▷ Convert *real* to $\underline{\text{rational}}$. Assume complete/limited accuracy for *real*.

(${}_f$**float** *real* [$prototype_{\boxed{\texttt{0.0F0}}}$])

▷ Convert *real* into $\underline{\text{float}}$ with type of *prototype*.

1.3 Logic Functions

Negative integers are used in two's complement representation.

($_f$**boole** *operation int-a int-b*)
▷ Return value of bitwise logical *operation*. *operation*s are

$_c$**boole-1**	▷ *int-a*.
$_c$**boole-2**	▷ *int-b*.
$_c$**boole-c1**	▷ $\neg$*int-a*.
$_c$**boole-c2**	▷ $\neg$*int-b*.
$_c$**boole-set**	▷ All bits set.
$_c$**boole-clr**	▷ All bits zero.
$_c$**boole-eqv**	▷ *int-a* $\equiv$ *int-b*.
$_c$**boole-and**	▷ *int-a* $\wedge$ *int-b*.
$_c$**boole-andc1**	▷ $\neg$*int-a* $\wedge$ *int-b*.
$_c$**boole-andc2**	▷ *int-a* $\wedge$ $\neg$*int-b*.
$_c$**boole-nand**	▷ $\neg$(*int-a* $\wedge$ *int-b*).
$_c$**boole-ior**	▷ *int-a* $\vee$ *int-b*.
$_c$**boole-orc1**	▷ $\neg$*int-a* $\vee$ *int-b*.
$_c$**boole-orc2**	▷ *int-a* $\vee$ $\neg$*int-b*.
$_c$**boole-xor**	▷ $\neg$(*int-a* $\equiv$ *int-b*).
$_c$**boole-nor**	▷ $\neg$(*int-a* $\vee$ *int-b*).

($_f$**lognot** *integer*) ▷ $\neg$*integer*.

($_f$**logeqv** *integer**)
($_f$**logand** *integer**)
▷ Return value of exclusive-nored or anded *integer*s, respectively. Without any *integer*, return -1.

($_f$**logandc1** *int-a int-b*) ▷ $\neg$*int-a* $\wedge$ *int-b*.

($_f$**logandc2** *int-a int-b*) ▷ *int-a* $\wedge$ $\neg$*int-b*.

($_f$**lognand** *int-a int-b*) ▷ $\neg$(*int-a* $\wedge$ *int-b*).

($_f$**logxor** *integer**)
($_f$**logior** *integer**)
▷ Return value of exclusive-ored or ored *integer*s, respectively. Without any *integer*, return 0.

($_f$**logorc1** *int-a int-b*) ▷ $\neg$*int-a* $\vee$ *int-b*.

($_f$**logorc2** *int-a int-b*) ▷ *int-a* $\vee$ $\neg$*int-b*.

($_f$**lognor** *int-a int-b*) ▷ $\neg$(*int-a* $\vee$ *int-b*).

($_f$**logbitp** *i int*) ▷ `T` if zero-indexed *i*th bit of *int* is set.

($_f$**logtest** *int-a int-b*)
▷ Return `T` if there is any bit set in *int-a* which is set in *int-b* as well.

($_f$**logcount** *int*)
▷ Number of 1 bits in *int* ≥ 0, number of 0 bits in *int* < 0.

1.4 Integer Functions

(${}_f$**integer-length** *integer*)

▷ $\underline{\text{Number of bits}}$ necessary to represent *integer*.

(${}_f$**ldb-test** *byte-spec integer*)

▷ Return $\underline{\texttt{T}}$ if any bit specified by *byte-spec* in *integer* is set.

(${}_f$**ash** *integer count*)

▷ Return copy of $\underline{\textit{integer}}$ arithmetically shifted left by *count* adding zeros at the right, or, for *count* < 0, shifted right discarding bits.

(${}_f$**ldb** *byte-spec integer*)

▷ Extract $\underline{\text{byte}}$ denoted by *byte-spec* from *integer*. **setf**able.

$$\left(\left\{\begin{array}{l}{}_f\textbf{deposit-field}\\ {}_f\textbf{dpb}\end{array}\right\}\ \textit{int-a byte-spec int-b}\right)$$

▷ Return $\underline{\textit{int-b}}$ with bits denoted by *byte-spec* replaced by corresponding bits of *int-a*, or by the low (${}_f$**byte-size** *byte-spec*) bits of *int-a*, respectively.

(${}_f$**mask-field** *byte-spec integer*)

▷ Return copy of $\underline{\textit{integer}}$ with all bits unset but those denoted by *byte-spec*. **setf**able.

(${}_f$**byte** *size position*)

▷ $\underline{\text{Byte specifier}}$ for a byte of *size* bits starting at a weight of $2^{position}$.

(${}_f$**byte-size** *byte-spec*)
(${}_f$**byte-position** *byte-spec*)

▷ $\underline{\text{Size}}$ or $\underline{\text{position}}$, respectively, of *byte-spec*.

1.5 Implementation-Dependent

$$\left.\begin{array}{l}{}_c\textbf{short-float}\\ {}_c\textbf{single-float}\\ {}_c\textbf{double-float}\\ {}_c\textbf{long-float}\end{array}\right\}\text{-}\left\{\begin{array}{l}\textbf{epsilon}\\ \textbf{negative-epsilon}\end{array}\right.$$

▷ Smallest possible number making a difference when added or subtracted, respectively.

$$\left.\begin{array}{l}{}_c\textbf{least-negative}\\ {}_c\textbf{least-negative-normalized}\\ {}_c\textbf{least-positive}\\ {}_c\textbf{least-positive-normalized}\end{array}\right\}\text{-}\left\{\begin{array}{l}\textbf{short-float}\\ \textbf{single-float}\\ \textbf{double-float}\\ \textbf{long-float}\end{array}\right.$$

▷ Available numbers closest to -0 or $+0$, respectively.

$$\left.\begin{array}{l}{}_c\textbf{most-negative}\\ {}_c\textbf{most-positive}\end{array}\right\}\text{-}\left\{\begin{array}{l}\textbf{short-float}\\ \textbf{single-float}\\ \textbf{double-float}\\ \textbf{long-float}\\ \textbf{fixnum}\end{array}\right.$$

▷ Available numbers closest to $-\infty$ or $+\infty$, respectively.

(${}_f$**decode-float** *n*)
(${}_f$**integer-decode-float** *n*)

▷ Return $\underline{\text{significand}}$, $\underline{\text{exponent}}_2$, and $\underline{\text{sign}}_3$ of **float** *n*.

(${}_f$**scale-float** *n i*) ▷ With *n*'s radix *b*, return nb^i.

(${}_f$**float-radix** *n*)
(${}_f$**float-digits** *n*)
(${}_f$**float-precision** *n*)

▷ $\underline{\text{Radix}}$, $\underline{\text{number of digits}}$ in that radix, or $\underline{\text{precision}}$ in that radix, respectively, of float *n*.

(${}_f$**upgraded-complex-part-type** *foo* [*environment*$_{\boxed{\texttt{NIL}}}$])

▷ $\underline{\text{Type}}$ of most specialized **complex** number able to hold parts of type *foo*.

2 Characters

The **standard-char** type comprises `a-z`, `A-Z`, `0-9`, `Newline`, `Space`, and `` !?$"'`.:,;*+-/|\~_^<=>#%@&()[]{} ``.

(${}_f$**characterp** *foo*)
(${}_f$**standard-char-p** *char*)
▷ T if argument is of indicated type.

(${}_f$**graphic-char-p** *character*)
(${}_f$**alpha-char-p** *character*)
(${}_f$**alphanumericp** *character*)
▷ T if *character* is visible, alphabetic, or alphanumeric, respectively.

(${}_f$**upper-case-p** *character*)
(${}_f$**lower-case-p** *character*)
(${}_f$**both-case-p** *character*)
▷ Return T if *character* is uppercase, lowercase, or able to be in another case, respectively.

(${}_f$**digit-char-p** *character* [$radix_{\boxed{10}}$])
▷ Return its weight if *character* is a digit, or `NIL` otherwise.

(${}_f$**char=** *character*$^+$)
(${}_f$**char/=** *character*$^+$)
▷ Return T if all *character*s, or none, respectively, are equal.

(${}_f$**char-equal** *character*$^+$)
(${}_f$**char-not-equal** *character*$^+$)
▷ Return T if all *character*s, or none, respectively, are equal ignoring case.

(${}_f$**char>** *character*$^+$)
(${}_f$**char>=** *character*$^+$)
(${}_f$**char<** *character*$^+$)
(${}_f$**char<=** *character*$^+$)
▷ Return T if *character*s are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively.

(${}_f$**char-greaterp** *character*$^+$)
(${}_f$**char-not-lessp** *character*$^+$)
(${}_f$**char-lessp** *character*$^+$)
(${}_f$**char-not-greaterp** *character*$^+$)
▷ Return T if *character*s are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively, ignoring case.

(${}_f$**char-upcase** *character*)
(${}_f$**char-downcase** *character*)
▷ Return corresponding uppercase/lowercase character, respectively.

(${}_f$**digit-char** *i* [$radix_{\boxed{10}}$]) ▷ Character representing digit *i*.

(${}_f$**char-name** *char*) ▷ *char*'s name if any, or `NIL`.

(${}_f$**name-char** *foo*) ▷ Character named *foo* if any, or `NIL`.

(${}_f$**char-int** *character*)
(${}_f$**char-code** *character*)
▷ Code of *character*.

(${}_f$**code-char** *code*) ▷ Character with *code*.

${}_c$**char-code-limit** ▷ Upper bound of (${}_f$**char-code** *char*); $\geq$ 96.

(${}_f$**character** *c*) ▷ Return #\c.

3 Strings

Strings can as well be manipulated by array and sequence functions; see pages 11 and 12.

$({}_f\textbf{stringp}\ \textit{foo})$
$({}_f\textbf{simple-string-p}\ \textit{foo})$
▷ $\underline{\texttt{T}}$ if *foo* is of indicated type.

$$\left(\begin{Bmatrix} {}_f\textbf{string=} \\ {}_f\textbf{string-equal} \end{Bmatrix} \textit{foo}\ \textit{bar} \left\{\left|\begin{array}{l} \textbf{:start1}\ \textit{start-foo}_{\boxed{0}} \\ \textbf{:start2}\ \textit{start-bar}_{\boxed{0}} \\ \textbf{:end1}\ \textit{end-foo}_{\boxed{\texttt{NIL}}} \\ \textbf{:end2}\ \textit{end-bar}_{\boxed{\texttt{NIL}}} \end{array}\right.\right\}\right)$$

▷ Return $\underline{\texttt{T}}$ if subsequences of *foo* and *bar* are equal. Obey/ignore, respectively, case.

$$\left(\left\{\begin{array}{l} {}_f\textbf{string}\{/=\ |\textbf{-not-equal}\} \\ {}_f\textbf{string}\{>\ |\textbf{-greaterp}\} \\ {}_f\textbf{string}\{>=\ |\textbf{-not-lessp}\} \\ {}_f\textbf{string}\{<\ |\textbf{-lessp}\} \\ {}_f\textbf{string}\{<=\ |\textbf{-not-greaterp}\} \end{array}\right\} \textit{foo}\ \textit{bar} \left\{\left|\begin{array}{l} \textbf{:start1}\ \textit{start-foo}_{\boxed{0}} \\ \textbf{:start2}\ \textit{start-bar}_{\boxed{0}} \\ \textbf{:end1}\ \textit{end-foo}_{\boxed{\texttt{NIL}}} \\ \textbf{:end2}\ \textit{end-bar}_{\boxed{\texttt{NIL}}} \end{array}\right.\right\}\right)$$

▷ If *foo* is lexicographically not equal, greater, not less, less, or not greater, respectively, then return $\underline{\text{position}}$ of first mismatching character in *foo*. Otherwise return $\underline{\texttt{NIL}}$. Obey/ignore, respectively, case.

$$\left({}_f\textbf{make-string}\ \textit{size} \left\{\left|\begin{array}{l} \textbf{:initial-element}\ \textit{char} \\ \textbf{:element-type}\ \textit{type}_{\boxed{\textbf{character}}} \end{array}\right.\right\}\right)$$

▷ Return $\underline{\text{string}}$ of length *size*.

$({}_f\textbf{string}\ x)$

$$\left(\left\{\begin{array}{l} {}_f\textbf{string-capitalize} \\ {}_f\textbf{string-upcase} \\ {}_f\textbf{string-downcase} \end{array}\right\} x \left\{\left|\begin{array}{l} \textbf{:start}\ \textit{start}_{\boxed{0}} \\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}} \end{array}\right.\right\}\right)$$

▷ Convert x (**symbol**, **string**, or **character**) into a $\underline{\text{string}}$, a $\underline{\text{string with capitalized words}}$, an $\underline{\text{all-uppercase string}}$, or an $\underline{\text{all-lowercase string}}$, respectively.

$$\left(\left\{\begin{array}{l} {}_f\textbf{nstring-capitalize} \\ {}_f\textbf{nstring-upcase} \\ {}_f\textbf{nstring-downcase} \end{array}\right\} \widetilde{\textit{string}} \left\{\left|\begin{array}{l} \textbf{:start}\ \textit{start}_{\boxed{0}} \\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}} \end{array}\right.\right\}\right)$$

▷ Convert *string* into a $\underline{\text{string with capitalized words}}$, an $\underline{\text{all-uppercase string}}$, or an $\underline{\text{all-lowercase string}}$, respectively.

$$\left(\left\{\begin{array}{l} {}_f\textbf{string-trim} \\ {}_f\textbf{string-left-trim} \\ {}_f\textbf{string-right-trim} \end{array}\right\} \textit{char-bag}\ \textit{string}\right)$$

▷ Return $\underline{\textit{string}}$ with all characters in sequence *char-bag* removed from both ends, from the beginning, or from the end, respectively.

$({}_f\textbf{char}\ \textit{string}\ i)$
$({}_f\textbf{schar}\ \textit{string}\ i)$
▷ Return zero-indexed $\underline{i\text{th character}}$ of string ignoring/obeying, respectively, fill pointer. **setf**able.

$$\left({}_f\textbf{parse-integer}\ \textit{string} \left\{\left|\begin{array}{l} \textbf{:start}\ \textit{start}_{\boxed{0}} \\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}} \\ \textbf{:radix}\ \textit{int}_{\boxed{10}} \\ \textbf{:junk-allowed}\ \textit{bool}_{\boxed{\texttt{NIL}}} \end{array}\right.\right\}\right)$$

▷ Return $\underline{\text{integer}}$ parsed from *string* and $\underline{\text{index}}_2$ of parse end.

4 Conses

4.1 Predicates

$({}_f\textbf{consp}\ \textit{foo})$
$({}_f\textbf{listp}\ \textit{foo})$
▷ Return $\underline{\texttt{T}}$ if *foo* is of indicated type.

$({}_f\textbf{endp}\ \textit{list})$
$({}_f\textbf{null}\ \textit{foo})$
▷ Return $\underline{\texttt{T}}$ if *list*/*foo* is `NIL`.

($_f$**atom** *foo*) ▷ Return T if *foo* is not a **cons**.

($_f$**tailp** *foo list*) ▷ Return T if *foo* is a tail of *list*.

($_f$**member** *foo list* $\left\{\begin{array}{l}\left|\left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\text{\#'eql}}}\\ \textbf{:test-not}\ \textit{function}\end{array}\right\}\right.\\ \left|\textbf{:key}\ \textit{function}\right.\end{array}\right\}$)

▷ Return tail of *list* starting with its first element matching *foo*. Return NIL if there is no such element.

($\left\{\begin{array}{l}{}_f\textbf{member-if}\\ {}_f\textbf{member-if-not}\end{array}\right\}$ *test list* [**:key** *function*])

▷ Return tail of *list* starting with its first element satisfying *test*. Return NIL if there is no such element.

($_f$**subsetp** *list-a list-b* $\left\{\begin{array}{l}\left|\left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\text{\#'eql}}}\\ \textbf{:test-not}\ \textit{function}\end{array}\right\}\right.\\ \left|\textbf{:key}\ \textit{function}\right.\end{array}\right\}$)

▷ Return T if *list-a* is a subset of *list-b*.

4.2 Lists

($_f$**cons** *foo bar*) ▷ Return new cons (*foo* . *bar*).

($_f$**list** *foo**) ▷ Return list of *foo*s.

($_f$**list*** *foo*$^+$)

▷ Return list of *foo*s with last *foo* becoming cdr of last cons. Return *foo* if only one *foo* given.

($_f$**make-list** *num* [**:initial-element** *foo*$_{\boxed{\text{NIL}}}$])

▷ New list with *num* elements set to *foo*.

($_f$**list-length** *list*) ▷ Length of *list*; NIL for circular *list*.

($_f$**car** *list*) ▷ Car of *list* or NIL if *list* is NIL. **setf**able.

($_f$**cdr** *list*)
($_f$**rest** *list*) ▷ Cdr of *list* or NIL if *list* is NIL. **setf**able.

($_f$**nthcdr** *n list*) ▷ Return tail of *list* after calling $_f$**cdr** *n* times.

({$_f$**first**|$_f$**second**|$_f$**third**|$_f$**fourth**|$_f$**fifth**|$_f$**sixth**|. . . |$_f$**ninth**|$_f$**tenth**} *list*)

▷ Return nth element of *list* if any, or NIL otherwise. **setf**able.

($_f$**nth** *n list*) ▷ Zero-indexed nth element of *list*. **setf**able.

($_f$**c***X***r** *list*)

▷ With *X* being one to four **a**s and **d**s representing $_f$**car**s and $_f$**cdr**s, e.g. ($_f$**cadr** *bar*) is equivalent to ($_f$**car** ($_f$**cdr** *bar*)). **setf**able.

($_f$**last** *list* [*num*$_{\boxed{1}}$]) ▷ Return list of last *num* conses of *list*.

($\left\{\begin{array}{l}{}_f\textbf{butlast}\ \textit{list}\\ {}_f\textbf{nbutlast}\ \widetilde{\textit{list}}\end{array}\right\}$ [*num*$_{\boxed{1}}$]) ▷ *list* excluding last *num* conses.

($\left\{\begin{array}{l}{}_f\textbf{rplaca}\\ {}_f\textbf{rplacd}\end{array}\right\}$ $\widetilde{\textit{cons}}$ *object*)

▷ Replace car, or cdr, respectively, of *cons* with *object*.

($_f$**ldiff** *list foo*)

▷ If *foo* is a tail of *list*, return preceding part of *list*. Otherwise return *list*.

($_f$**adjoin** *foo list* $\left\{\begin{array}{l}\left|\left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\text{\#'eql}}}\\ \textbf{:test-not}\ \textit{function}\end{array}\right\}\right.\\ \left|\textbf{:key}\ \textit{function}\right.\end{array}\right\}$)

▷ Return *list* if *foo* is already member of *list*. If not, return ($_f$**cons** *foo list*).

($_m$**pop** $\widetilde{\textit{place}}$) ▷ Set *place* to ($_f$**cdr** *place*), return ($_f$**car** *place*).

($_m$**push** *foo* $\widetilde{place}$) ▷ Set *place* to ($_f$**cons** *foo place*).

$$(_m\textbf{pushnew}\ foo\ \widetilde{place}\ \left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:test}\ function_{\boxed{\#'\text{eql}}}\\ \textbf{:test-not}\ function\end{array}\right.\\ \textbf{:key}\ function\end{array}\right.\right\})$$

▷ Set *place* to ($_f$**adjoin** *foo place*).

($_f$**append** [*proper-list** $foo_{\boxed{\text{NIL}}}$])

($_f$**nconc** [$\widetilde{non\text{-}circular\text{-}list}$\* $foo_{\boxed{\text{NIL}}}$])

▷ Return concatenated list or, with only one argument, *foo*. *foo* can be of any type.

($_f$**revappend** *list foo*)

($_f$**nreconc** $\widetilde{list}$ *foo*)

▷ Return concatenated list after reversing order in *list*.

$$(\left\{\begin{array}{l}_f\textbf{mapcar}\\ _f\textbf{maplist}\end{array}\right\}\ function\ list^+)$$

▷ Return list of return values of *function* successively invoked with corresponding arguments, either cars or cdrs, respectively, from each *list*.

$$(\left\{\begin{array}{l}_f\textbf{mapcan}\\ _f\textbf{mapcon}\end{array}\right\}\ function\ \widetilde{list}^+)$$

▷ Return list of concatenated return values of *function* successively invoked with corresponding arguments, either cars or cdrs, respectively, from each *list*. *function* should return a list.

$$(\left\{\begin{array}{l}_f\textbf{mapc}\\ _f\textbf{mapl}\end{array}\right\}\ function\ list^+)$$

▷ Return first *list* after successively applying *function* to corresponding arguments, either cars or cdrs, respectively, from each *list*. *function* should have some side effects.

($_f$**copy-list** *list*) ▷ Return copy of *list* with shared elements.

4.3 Association Lists

($_f$**pairlis** *keys values* [$alist_{\boxed{\text{NIL}}}$])

▷ Prepend to *alist* an association list made from lists *keys* and *values*.

($_f$**acons** *key value alist*)

▷ Return *alist* with a (*key* . *value*) pair added.

$$(\left\{\begin{array}{l}_f\textbf{assoc}\\ _f\textbf{rassoc}\end{array}\right\}\ foo\ alist\ \left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:test}\ test_{\boxed{\#'\text{eql}}}\\ \textbf{:test-not}\ test\end{array}\right.\\ \textbf{:key}\ function\end{array}\right.\right\})$$

$$(\left\{\begin{array}{l}_f\textbf{assoc-if}[\textbf{-not}]\\ _f\textbf{rassoc-if}[\textbf{-not}]\end{array}\right\}\ test\ alist\ [\textbf{:key}\ function])$$

▷ First cons whose car, or cdr, respectively, satisfies *test*.

($_f$**copy-alist** *alist*) ▷ Return copy of *alist*.

4.4 Trees

$$(_f\textbf{tree-equal}\ foo\ bar\ \left\{\begin{array}{l}\textbf{:test}\ test_{\boxed{\#'\text{eql}}}\\ \textbf{:test-not}\ test\end{array}\right\})$$

▷ Return `T` if trees *foo* and *bar* have same shape and leaves satisfying *test*.

$$(\left\{\begin{array}{l}_f\textbf{subst}\ new\ old\ tree\\ _f\textbf{nsubst}\ new\ old\ \widetilde{tree}\end{array}\right\}\ \left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:test}\ function_{\boxed{\#'\text{eql}}}\\ \textbf{:test-not}\ function\end{array}\right.\\ \textbf{:key}\ function\end{array}\right.\right\})$$

▷ Make copy of *tree* with each subtree or leaf matching *old* replaced by *new*.

$$(\left\{\begin{array}{l}_f\textbf{subst-if}[\textbf{-not}]\ new\ test\ tree\\ _f\textbf{nsubst-if}[\textbf{-not}]\ new\ test\ \widetilde{tree}\end{array}\right\}\ [\textbf{:key}\ function])$$

▷ Make copy of *tree* with each subtree or leaf satisfying *test* replaced by *new*.

$$\left(\left\{\begin{array}{l}{}_f\textbf{sublis}\ \textit{association-list tree}\\ {}_f\textbf{nsublis}\ \textit{association-list}\ \widetilde{\textit{tree}}\end{array}\right\}\left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\texttt{\#'eql}}}\\ \textbf{:test-not}\ \textit{function}\end{array}\right.\\ \textbf{:key}\ \textit{function}\end{array}\right.\right\}\right)$$

▷ Make copy of *tree* with each subtree or leaf matching a key in *association-list* replaced by that key's value.

($_f$**copy-tree** *tree*) ▷ Copy of *tree* with same shape and leaves.

4.5 Sets

$$\left(\left\{\begin{array}{l}\left.\begin{array}{l}{}_f\textbf{intersection}\\ {}_f\textbf{set-difference}\\ {}_f\textbf{union}\\ {}_f\textbf{set-exclusive-or}\end{array}\right\}\ a\ \ b\\ \left.\begin{array}{l}{}_f\textbf{nintersection}\\ {}_f\textbf{nset-difference}\end{array}\right\}\ \widetilde{a}\ \ b\\ \left.\begin{array}{l}{}_f\textbf{nunion}\\ {}_f\textbf{nset-exclusive-or}\end{array}\right\}\ \widetilde{a}\ \ \widetilde{b}\end{array}\right\}\left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\texttt{\#'eql}}}\\ \textbf{:test-not}\ \textit{function}\end{array}\right.\\ \textbf{:key}\ \textit{function}\end{array}\right.\right\}\right)$$

▷ Return $a \cap b$, $a \setminus b$, $a \cup b$, or $a \triangle b$, respectively, of lists a and b.

5 Arrays

5.1 Predicates

($_f$**arrayp** *foo*)
($_f$**vectorp** *foo*)
($_f$**simple-vector-p** *foo*) ▷ T if *foo* is of indicated type.
($_f$**bit-vector-p** *foo*)
($_f$**simple-bit-vector-p** *foo*)

($_f$**adjustable-array-p** *array*)
($_f$**array-has-fill-pointer-p** *array*)

▷ T if *array* is adjustable/has a fill pointer, respectively.

($_f$**array-in-bounds-p** *array* [*subscripts*])

▷ Return T if *subscripts* are in *array*'s bounds.

5.2 Array Functions

$$\left(\left\{\begin{array}{l}{}_f\textbf{make-array}\ \textit{dimension-sizes}\ \left[\textbf{:adjustable}\ \textit{bool}_{\boxed{\texttt{NIL}}}\right]\\ {}_f\textbf{adjust-array}\ \widetilde{\textit{array}}\ \textit{dimension-sizes}\end{array}\right\}\right.$$
$$\left.\left\{\left|\begin{array}{l}\textbf{:element-type}\ \textit{type}_{\boxed{\texttt{T}}}\\ \textbf{:fill-pointer}\ \{\textit{num}\,|\,\textit{bool}\}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:initial-element}\ \textit{obj}\\ \textbf{:initial-contents}\ \textit{tree-or-array}\\ \textbf{:displaced-to}\ \textit{array}_{\boxed{\texttt{NIL}}}\ [\textbf{:displaced-index-offset}\ i_{\boxed{\texttt{0}}}]\end{array}\right.\end{array}\right.\right\}\right)$$

▷ Return fresh, or readjust, respectively, vector or array.

($_f$**aref** *array* [*subscripts*])

▷ Return array element pointed to by *subscripts*. **setf**able.

($_f$**row-major-aref** *array* *i*)

▷ Return *i*th element of *array* in row-major order. **setf**able.

($_f$**array-row-major-index** *array* [*subscripts*])

▷ Index in row-major order of the element denoted by *subscripts*.

($_f$**array-dimensions** *array*)

▷ List containing the lengths of *array*'s dimensions.

($_f$**array-dimension** *array* *i*) ▷ Length of *i*th dimension of *array*.

($_f$**array-total-size** *array*) ▷ Number of elements in *array*.

($_f$**array-rank** *array*) ▷ Number of dimensions of *array*.

($_f$**array-displacement** *array*) ▷ $\underline{\text{Target array}}$ and $\underline{\text{offset}}_2$.

($_f$**bit** *bit-array* [*subscripts*])
($_f$**sbit** *simple-bit-array* [*subscripts*])
▷ Return $\underline{\text{element}}$ of *bit-array* or of *simple-bit-array*. **setf**able.

($_f$**bit-not** $\widetilde{\textit{bit-array}}$ [$\widetilde{\textit{result-bit-array}}_{\boxed{\texttt{NIL}}}$])
▷ Return $\underline{\text{result}}$ of bitwise negation of *bit-array*. If *result-bit-array* is `T`, put result in *bit-array*; if it is `NIL`, make a new array for result.

$$\left(\left\{\begin{array}{l}{}_f\textbf{bit-eqv}\\ {}_f\textbf{bit-and}\\ {}_f\textbf{bit-andc1}\\ {}_f\textbf{bit-andc2}\\ {}_f\textbf{bit-nand}\\ {}_f\textbf{bit-ior}\\ {}_f\textbf{bit-orc1}\\ {}_f\textbf{bit-orc2}\\ {}_f\textbf{bit-xor}\\ {}_f\textbf{bit-nor}\end{array}\right\} \widetilde{\textit{bit-array-a}}\ \textit{bit-array-b}\ [\widetilde{\textit{result-bit-array}}_{\boxed{\texttt{NIL}}}]\right)$$

▷ Return $\underline{\text{result}}$ of bitwise logical operations (cf. operations of $_f$**boole**, page 5) on *bit-array-a* and *bit-array-b*. If *result-bit-array* is `T`, put result in *bit-array-a*; if it is `NIL`, make a new array for result.

$_c$**array-rank-limit** ▷ Upper bound of array rank; ≥ 8.

$_c$**array-dimension-limit**
▷ Upper bound of an array dimension; ≥ 1024.

$_c$**array-total-size-limit** ▷ Upper bound of array size; ≥ 1024.

5.3 Vector Functions

Vectors can as well be manipulated by sequence functions; see section 6.

($_f$**vector** *foo**) ▷ Return fresh $\underline{\text{simple vector of } \textit{foo}\text{s}}$.

($_f$**svref** *vector* *i*) ▷ $\underline{\text{Element } i}$ of simple *vector*. **setf**able.

($_f$**vector-push** *foo* $\widetilde{\textit{vector}}$)
▷ Return $\underline{\texttt{NIL}}$ if *vector*'s fill pointer equals size of *vector*. Otherwise replace element of *vector* pointed to by $\underline{\text{fill pointer}}$ with *foo*; then increment fill pointer.

($_f$**vector-push-extend** *foo* $\widetilde{\textit{vector}}$ [*num*])
▷ Replace element of *vector* pointed to by $\underline{\text{fill pointer}}$ with *foo*, then increment fill pointer. Extend *vector*'s size by $\geq$ *num* if necessary.

($_f$**vector-pop** $\widetilde{\textit{vector}}$)
▷ Return $\underline{\text{element of } \textit{vector}}$ its fillpointer points to after decrementation.

($_f$**fill-pointer** *vector*) ▷ $\underline{\text{Fill pointer}}$ of *vector*. **setf**able.

6 Sequences

6.1 Sequence Predicates

$$\left(\left\{\begin{array}{l}{}_f\textbf{every}\\ {}_f\textbf{notevery}\end{array}\right\} \textit{test}\ \textit{sequence}^{+}\right)$$

▷ Return $\underline{\texttt{NIL}}$ or $\underline{\texttt{T}}$, respectively, as soon as *test* on any set of corresponding elements of *sequence*s returns `NIL`.

$\left(\left\{\begin{array}{l}{}_f\textbf{some}\\{}_f\textbf{notany}\end{array}\right\}\ \textit{test sequence}^{+}\right)$

▷ Return value of *test* or `NIL`, respectively, as soon as *test* on any set of corresponding elements of *sequence*s returns non-`NIL`.

$\left({}_f\textbf{mismatch}\ \textit{sequence-a sequence-b}\ \left\{\left|\begin{array}{l}\textbf{:from-end}\ \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\texttt{\#'eql}}}\\ \textbf{:test-not}\ \textit{function}\end{array}\right.\\ \textbf{:start1}\ \textit{start-a}_{\boxed{0}}\\ \textbf{:start2}\ \textit{start-b}_{\boxed{0}}\\ \textbf{:end1}\ \textit{end-a}_{\boxed{\texttt{NIL}}}\\ \textbf{:end2}\ \textit{end-b}_{\boxed{\texttt{NIL}}}\\ \textbf{:key}\ \textit{function}\end{array}\right.\right\}\right)$

▷ Return position in *sequence-a* where *sequence-a* and *sequence-b* begin to mismatch. Return `NIL` if they match entirely.

6.2 Sequence Functions

$({}_f\textbf{make-sequence}\ \textit{sequence-type size}\ [\textbf{:initial-element}\ \textit{foo}])$

▷ Make sequence of *sequence-type* with *size* elements.

$({}_f\textbf{concatenate}\ \textit{type sequence}^{*})$

▷ Return concatenated sequence of *type*.

$({}_f\textbf{merge}\ \textit{type}\ \widetilde{\textit{sequence}}\textit{-a}\ \widetilde{\textit{sequence}}\textit{-b}\ \textit{test}\ [\textbf{:key}\ \textit{function}_{\boxed{\texttt{NIL}}}])$

▷ Return interleaved sequence of *type*. Merged sequence will be sorted if both *sequence-a* and *sequence-b* are sorted.

$\left({}_f\textbf{fill}\ \widetilde{\textit{sequence}}\ \textit{foo}\ \left\{\left|\begin{array}{l}\textbf{:start}\ \textit{start}_{\boxed{0}}\\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}\end{array}\right.\right\}\right)$

▷ Return *sequence* after setting elements between *start* and *end* to *foo*.

$({}_f\textbf{length}\ \textit{sequence})$

▷ Return length of *sequence* (being value of fill pointer if applicable).

$\left({}_f\textbf{count}\ \textit{foo sequence}\ \left\{\left|\begin{array}{l}\textbf{:from-end}\ \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\texttt{\#'eql}}}\\ \textbf{:test-not}\ \textit{function}\end{array}\right.\\ \textbf{:start}\ \textit{start}_{\boxed{0}}\\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key}\ \textit{function}\end{array}\right.\right\}\right)$

▷ Return number of elements in *sequence* which match *foo*.

$\left(\left\{\begin{array}{l}{}_f\textbf{count-if}\\{}_f\textbf{count-if-not}\end{array}\right\}\ \textit{test sequence}\ \left\{\left|\begin{array}{l}\textbf{:from-end}\ \textit{bool}_{\boxed{\texttt{NIL}}}\\ \textbf{:start}\ \textit{start}_{\boxed{0}}\\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key}\ \textit{function}\end{array}\right.\right\}\right)$

▷ Return number of elements in *sequence* which satisfy *test*.

$({}_f\textbf{elt}\ \textit{sequence index})$

▷ Return element of *sequence* pointed to by zero-indexed *index*. **setf**able.

$({}_f\textbf{subseq}\ \textit{sequence start}\ [\textit{end}_{\boxed{\texttt{NIL}}}])$

▷ Return subsequence of *sequence* between *start* and *end*. **setf**able.

$\left(\left\{\begin{array}{l}{}_f\textbf{sort}\\{}_f\textbf{stable-sort}\end{array}\right\}\ \widetilde{\textit{sequence}}\ \textit{test}\ [\textbf{:key}\ \textit{function}]\right)$

▷ Return *sequence* sorted. Order of elements considered equal is not guaranteed/retained, respectively.

$({}_f\textbf{reverse}\ \textit{sequence})$
$({}_f\textbf{nreverse}\ \widetilde{\textit{sequence}})$

▷ Return *sequence* in reverse order.

$$\left(\left\{\begin{array}{l}{}_f\textbf{find}\\ {}_f\textbf{position}\end{array}\right\} \textit{foo sequence} \left\{\left|\begin{array}{l}\textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\textbf{\#'eql}}}\\ \textbf{:test-not } \textit{test}\end{array}\right.\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\end{array}\right.\right\}\right)$$

▷ Return first element in *sequence* which matches *foo*, or its position relative to the begin of *sequence*, respectively.

$$\left(\left\{\begin{array}{l}{}_f\textbf{find-if}\\ {}_f\textbf{find-if-not}\\ {}_f\textbf{position-if}\\ {}_f\textbf{position-if-not}\end{array}\right\} \textit{test sequence} \left\{\left|\begin{array}{l}\textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\end{array}\right.\right\}\right)$$

▷ Return first element in *sequence* which satisfies *test*, or its position relative to the begin of *sequence*, respectively.

$$\left({}_f\textbf{search } \textit{sequence-a sequence-b} \left\{\left|\begin{array}{l}\textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\textbf{\#'eql}}}\\ \textbf{:test-not } \textit{function}\end{array}\right.\\ \textbf{:start1 } \textit{start-a}_{\boxed{0}}\\ \textbf{:start2 } \textit{start-b}_{\boxed{0}}\\ \textbf{:end1 } \textit{end-a}_{\boxed{\texttt{NIL}}}\\ \textbf{:end2 } \textit{end-b}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\end{array}\right.\right\}\right)$$

▷ Search *sequence-b* for a subsequence matching *sequence-a*. Return position in *sequence-b*, or `NIL`.

$$\left(\left\{\begin{array}{l}{}_f\textbf{remove } \textit{foo sequence}\\ {}_f\textbf{delete } \textit{foo } \widetilde{\textit{sequence}}\end{array}\right\} \left\{\left|\begin{array}{l}\textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\textbf{\#'eql}}}\\ \textbf{:test-not } \textit{function}\end{array}\right.\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\\ \textbf{:count } \textit{count}_{\boxed{\texttt{NIL}}}\end{array}\right.\right\}\right)$$

▷ Make copy of *sequence* without elements matching *foo*.

$$\left(\left\{\begin{array}{l}\left.\begin{array}{l}{}_f\textbf{remove-if}\\ {}_f\textbf{remove-if-not}\end{array}\right\} \textit{test sequence}\\ \left.\begin{array}{l}{}_f\textbf{delete-if}\\ {}_f\textbf{delete-if-not}\end{array}\right\} \textit{test } \widetilde{\textit{sequence}}\end{array}\right\} \left\{\left|\begin{array}{l}\textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\\ \textbf{:count } \textit{count}_{\boxed{\texttt{NIL}}}\end{array}\right.\right\}\right)$$

▷ Make copy of *sequence* with all (or *count*) elements satisfying *test* removed.

$$\left(\left\{\begin{array}{l}{}_f\textbf{remove-duplicates } \textit{sequence}\\ {}_f\textbf{delete-duplicates } \widetilde{\textit{sequence}}\end{array}\right\} \left\{\left|\begin{array}{l}\textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\textbf{\#'eql}}}\\ \textbf{:test-not } \textit{function}\end{array}\right.\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\end{array}\right.\right\}\right)$$

▷ Make copy of *sequence* without duplicates.

$$\left(\left\{\begin{array}{l}{}_f\textbf{substitute } \textit{new old sequence}\\ {}_f\textbf{nsubstitute } \textit{new old } \widetilde{\textit{sequence}}\end{array}\right\} \left\{\left|\begin{array}{l}\textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\textbf{\#'eql}}}\\ \textbf{:test-not } \textit{function}\end{array}\right.\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\\ \textbf{:count } \textit{count}_{\boxed{\texttt{NIL}}}\end{array}\right.\right\}\right)$$

▷ Make copy of *sequence* with all (or *count*) *old*s replaced by *new*.

$$\left(\left\{\begin{array}{l}\left.\begin{array}{l}{}_f\textbf{substitute-if}\\ {}_f\textbf{substitute-if-not}\end{array}\right\} \textit{new test sequence}\\ \left.\begin{array}{l}{}_f\textbf{nsubstitute-if}\\ {}_f\textbf{nsubstitute-if-not}\end{array}\right\} \textit{new test } \widetilde{\textit{sequence}}\end{array}\right\} \left\{\left|\begin{array}{l}\textbf{:from-end } \textit{bool}_{\boxed{\texttt{NIL}}}\\ \textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key } \textit{function}\\ \textbf{:count } \textit{count}_{\boxed{\texttt{NIL}}}\end{array}\right.\right\}\right)$$

▷ Make copy of *sequence* with all (or *count*) elements satisfying *test* replaced by *new*.

($_f$**replace** $\widetilde{\textit{sequence-a}}$ *sequence-b* $\left\{\begin{array}{|l}\textbf{:start1}\ \textit{start-a}_{\boxed{0}}\\ \textbf{:start2}\ \textit{start-b}_{\boxed{0}}\\ \textbf{:end1}\ \textit{end-a}_{\boxed{\texttt{NIL}}}\\ \textbf{:end2}\ \textit{end-b}_{\boxed{\texttt{NIL}}}\end{array}\right\}$)

▷ Replace elements of *sequence-a* with elements of *sequence-b*.

($_f$**map** *type function sequence*$^+$)

▷ Apply *function* successively to corresponding elements of the *sequences*. Return values as a sequence of *type*. If *type* is `NIL`, return `NIL`.

($_f$**map-into** $\widetilde{\textit{result-sequence}}$ *function sequence**)

▷ Store into *result-sequence* successively values of *function* applied to corresponding elements of the *sequences*.

($_f$**reduce** *function sequence* $\left\{\begin{array}{|l}\textbf{:initial-value}\ \textit{foo}_{\boxed{\texttt{NIL}}}\\ \textbf{:from-end}\ \textit{bool}_{\boxed{\texttt{NIL}}}\\ \textbf{:start}\ \textit{start}_{\boxed{0}}\\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key}\ \textit{function}\end{array}\right\}$)

▷ Starting with the first two elements of *sequence*, apply *function* successively to its last return value together with the next element of *sequence*. Return last value of function.

($_f$**copy-seq** *sequence*)

▷ Copy of *sequence* with shared elements.

7 Hash Tables

The Loop Facility provides additional hash table-related functionality; see **loop**, page 22.

Key-value storage similar to hash tables can as well be achieved using association lists and property lists; see pages 10 and 17.

($_f$**hash-table-p** *foo*) ▷ Return `T` if *foo* is of type **hash-table**.

($_f$**make-hash-table** $\left\{\begin{array}{|l}\textbf{:test}\ \{{}_f\textbf{eq}|{}_f\textbf{eql}|{}_f\textbf{equal}|{}_f\textbf{equalp}\}_{\boxed{\texttt{\#'eql}}}\\ \textbf{:size}\ \textit{int}\\ \textbf{:rehash-size}\ \textit{num}\\ \textbf{:rehash-threshold}\ \textit{num}\end{array}\right\}$)

▷ Make a hash table.

($_f$**gethash** *key hash-table* [*default*$_{\boxed{\texttt{NIL}}}$])

▷ Return object with *key* if any or *default* otherwise; and $\underset{2}{\texttt{T}}$ if found, $\underset{2}{\texttt{NIL}}$ otherwise. **setf**able.

($_f$**hash-table-count** *hash-table*)

▷ Number of entries in *hash-table*.

($_f$**remhash** *key* $\widetilde{\textit{hash-table}}$)

▷ Remove from *hash-table* entry with *key* and return `T` if it existed. Return `NIL` otherwise.

($_f$**clrhash** $\widetilde{\textit{hash-table}}$) ▷ Empty *hash-table*.

($_f$**maphash** *function hash-table*)

▷ Iterate over *hash-table* calling *function* on key and value. Return `NIL`.

($_m$**with-hash-table-iterator** (*foo hash-table*) (**declare** $\widehat{\textit{decl}^*}$)* $\textit{form}^{P*}$)

▷ Return values of *forms*. In *forms*, invocations of (*foo*) return: `T` if an entry is returned; its key; its value.

($_f$**hash-table-test** *hash-table*)

▷ Test function used in *hash-table*.

($_f$**hash-table-size** *hash-table*)
($_f$**hash-table-rehash-size** *hash-table*)
($_f$**hash-table-rehash-threshold** *hash-table*)

▷ Current size, rehash-size, or rehash-threshold, respectively, as used in $_f$**make-hash-table**.

($_f$**sxhash** *foo*) ▷ Hash code unique for any argument $_f$**equal** *foo*.

8 Structures

($_m$**defstruct**

$$\left\{\begin{array}{l} foo \\ (foo \left\{\left|\begin{array}{l} \left\{\begin{array}{l}\textbf{:conc-name}\\ (\textbf{:conc-name}\ [\widehat{slot\text{-}prefix}_{\boxed{foo\text{-}}}])\end{array}\right. \\ \left\{\begin{array}{l}\textbf{:constructor}\\ (\textbf{:constructor}\ [\widehat{maker}_{\boxed{\texttt{MAKE-}foo}}\ [(\widehat{ord\text{-}\lambda^*})]])\end{array}\right\}^* \\ \left\{\begin{array}{l}\textbf{:copier}\\ (\textbf{:copier}\ [\widehat{copier}_{\boxed{\texttt{COPY-}foo}}])\end{array}\right. \\ (\textbf{:include}\ \widehat{struct} \left\{\begin{array}{l}\widehat{slot}\\ (\widehat{slot}\ [init \left\{\left|\begin{array}{l}\textbf{:type}\ \widehat{sl\text{-}type}\\ \textbf{:read-only}\ \widehat{b}\end{array}\right.\right\}])\end{array}\right\}^*) \\ \left\{\begin{array}{l}(\textbf{:type} \left\{\begin{array}{l}\textbf{list}\\ \textbf{vector}\\ (\textbf{vector}\ \widehat{type})\end{array}\right\})\ [(\textbf{:initial-offset}\ \widehat{n})] \\ \left\{\begin{array}{l}(\textbf{:print-object}\ [\widehat{o\text{-}printer}])\\ (\textbf{:print-function}\ [\widehat{f\text{-}printer}])\end{array}\right.\end{array}\right. \\ \textbf{:named} \\ \left\{\begin{array}{l}\textbf{:predicate}\\ (\textbf{:predicate}\ [\widehat{p\text{-}name}_{\boxed{foo\texttt{-P}}}])\end{array}\right.\end{array}\right.\right\})\end{array}\right\}$$

$$[\widehat{doc}] \left\{\begin{array}{l} slot \\ (slot\ [init \left\{\left|\begin{array}{l}\textbf{:type}\ \widehat{slot\text{-}type}\\ \textbf{:read-only}\ \widehat{bool}\end{array}\right.\right\}])\end{array}\right\}^*)$$

▷ Define structure foo together with functions `MAKE-`*foo*, `COPY-`*foo* and *foo*`-P`; and **setf**able accessors *foo*`-`*slot*. Instances are of class *foo* or, if **defstruct** option **:type** is given, of the specified type. They can be created by (`MAKE-`*foo* {**:***slot value*}*) or, if *ord-λ* (see page 18) is given, by (*maker arg** {**:***key value*}*). In the latter case, *arg*s and **:***key*s correspond to the positional and keyword parameters defined in *ord-λ* whose *var*s in turn correspond to *slot*s. **:print-object**/**:print-function** generate a $_g$**print-object** method for an instance *bar* of *foo* calling (*o-printer bar stream*) or (*f-printer bar stream print-level*), respectively. If **:type** without **:named** is given, no *foo*`-P` is created.

($_f$**copy-structure** *structure*)
▷ Return copy of *structure* with shared slot values.

9 Control Structure

9.1 Predicates

($_f$**eq** *foo bar*) ▷ `T` if *foo* and *bar* are identical.

($_f$**eql** *foo bar*)
▷ `T` if *foo* and *bar* are identical, or the same **character**, or **number**s of the same type and value.

($_f$**equal** *foo bar*)
▷ `T` if *foo* and *bar* are $_f$**eql**, or are equivalent **pathname**s, or are **cons**es with $_f$**equal** cars and cdrs, or are **string**s or **bit-vector**s with $_f$**eql** elements below their fill pointers.

($_f$**equalp** *foo bar*)
▷ `T` if *foo* and *bar* are identical; or are the same **character** ignoring case; or are **number**s of the same value ignoring type; or are equivalent **pathname**s; or are **cons**es or **array**s of the same shape with $_f$**equalp** elements; or are structures of the same type with $_f$**equalp** elements; or are **hash-table**s of the same size with the same **:test** function, the same keys in terms of **:test** function, and $_f$**equalp** elements.

($_f$**not** *foo*) ▷ $\underline{\texttt{T}}$ if *foo* is NIL; $\underline{\texttt{NIL}}$ otherwise.

($_f$**boundp** *symbol*) ▷ $\underline{\texttt{T}}$ if *symbol* is a special variable.

($_f$**constantp** *foo* [*environment*$_{\boxed{\texttt{NIL}}}$])
▷ $\underline{\texttt{T}}$ if *foo* is a constant form.

($_f$**functionp** *foo*) ▷ $\underline{\texttt{T}}$ if *foo* is of type **function**.

($_f$**fboundp** $\left\{\begin{array}{l} foo \\ (\textbf{setf}\ foo) \end{array}\right\}$) ▷ $\underline{\texttt{T}}$ if *foo* is a global function or macro.

9.2 Variables

($\left\{\begin{array}{l} {}_m\textbf{defconstant} \\ {}_m\textbf{defparameter} \end{array}\right\}$ $\widehat{foo}$ *form* [$\widehat{doc}$])
▷ Assign value of *form* to global constant/dynamic variable $\underline{foo}$.

($_m$**defvar** $\widehat{foo}$ [*form* [$\widehat{doc}$]])
▷ Unless bound already, assign value of *form* to dynamic variable $\underline{foo}$.

($\left\{\begin{array}{l} {}_m\textbf{setf} \\ {}_m\textbf{psetf} \end{array}\right\}$ {*place form*}*)
▷ Set *place*s to primary values of *form*s. Return $\underline{\text{values of last } form}$/$\underline{\texttt{NIL}}$; work sequentially/in parallel, respectively.

($\left\{\begin{array}{l} {}_s\textbf{setq} \\ {}_m\textbf{psetq} \end{array}\right\}$ {*symbol form*}*)
▷ Set *symbol*s to primary values of *form*s. Return $\underline{\text{value of last } form}$/$\underline{\texttt{NIL}}$; work sequentially/in parallel, respectively.

($_f$**set** $\widetilde{symbol}$ *foo*) ▷ Set *symbol*'s value cell to $\underline{foo}$. Deprecated.

($_m$**multiple-value-setq** *vars form*)
▷ Set elements of *vars* to the values of *form*. Return $\underline{form\text{'s primary value}}$.

($_m$**shiftf** $\widetilde{place}^+$ *foo*)
▷ Store value of *foo* in rightmost *place* shifting values of *place*s left, returning $\underline{\text{first } place}$.

($_m$**rotatef** $\widetilde{place}^*$)
▷ Rotate values of *place*s left, old first becoming new last *place*'s value. Return $\underline{\texttt{NIL}}$.

($_f$**makunbound** $\widetilde{foo}$) ▷ Delete special variable $\underline{foo}$ if any.

($_f$**get** *symbol key* [*default*$_{\boxed{\texttt{NIL}}}$])
($_f$**getf** *place key* [*default*$_{\boxed{\texttt{NIL}}}$])
▷ $\underline{\text{First entry } key}$ from property list stored in *symbol*/in *place*, respectively, or $\underline{default}$ if there is no *key*. **setf**able.

($_f$**get-properties** *property-list keys*)
▷ Return $\underline{\text{key}}$ and $\underset{2}{\underline{\text{value}}}$ of first entry from *property-list* matching a key from *keys*, and $\underset{3}{\underline{\text{tail of } property\text{-}list}}$ starting with that key. Return $\underline{\texttt{NIL}}$, $\underset{2}{\underline{\texttt{NIL}}}$, and $\underset{3}{\underline{\texttt{NIL}}}$ if there was no matching key in *property-list*.

($_f$**remprop** $\widetilde{symbol}$ *key*)
($_m$**remf** $\widetilde{place}$ *key*)
▷ Remove first entry *key* from property list stored in *symbol*/in *place*, respectively. Return $\underline{\texttt{T}}$ if *key* was there, or $\underline{\texttt{NIL}}$ otherwise.

($_s$**progv** *symbols values form*$^{\text{P}*}$)
▷ Evaluate *form*s with locally established dynamic bindings of *symbols* to *values* or NIL. Return $\underline{\text{values of } forms}$.

$$\left(\begin{Bmatrix}{}_s\textbf{let}\\{}_s\textbf{let*}\end{Bmatrix}\ \left(\left\{\left|\begin{matrix}name\\(name\ [value_{\boxed{\texttt{NIL}}}])\end{matrix}\right.\right\}^{*}\right)\ (\textbf{declare}\ \widehat{decl^*})^*\ form^{P*}\right)$$

▷ Evaluate *forms* with *names* lexically bound (in parallel or sequentially, respectively) to *values*. Return values of *forms*.

$$({}_m\textbf{multiple-value-bind}\ (\widehat{var^*})\ values\text{-}form\ (\textbf{declare}\ \widehat{decl^*})^*\ body\text{-}form^{P*})$$

▷ Evaluate *body-forms* with *vars* lexically bound to the return values of *values-form*. Return values of *body-forms*.

$$({}_m\textbf{destructuring-bind}\ destruct\text{-}\lambda\ bar\ (\textbf{declare}\ \widehat{decl^*})^*\ form^{P*})$$

▷ Evaluate *forms* with variables from tree *destruct-λ* bound to corresponding elements of tree *bar*, and return their values. *destruct-λ* resembles *macro-λ* (section 9.4), but without any **&environment** clause.

9.3 Functions

Below, ordinary lambda list (*ord-λ**) has the form

$$(var^*\ \left[\textbf{\&optional}\ \begin{Bmatrix}var\\(var\ [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]])\end{Bmatrix}^*\right]\ [\textbf{\&rest}\ var]$$

$$\left[\textbf{\&key}\ \begin{Bmatrix}var\\ \left(\begin{Bmatrix}var\\(\text{:}key\ var)\end{Bmatrix}\ [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]]\right)\end{Bmatrix}^*\ [\textbf{\&allow-other-keys}]\right]$$

$$\left[\textbf{\&aux}\ \begin{Bmatrix}var\\(var\ [init_{\boxed{\texttt{NIL}}}])\end{Bmatrix}^*\right]).$$

supplied-p is `T` if there is a corresponding argument. *init* forms can refer to any *init* and *supplied-p* to their left.

$$\left(\begin{Bmatrix}{}_m\textbf{defun}\ \begin{Bmatrix}foo\ (ord\text{-}\lambda^*)\\(\textbf{setf}\ foo)\ (new\text{-}value\ ord\text{-}\lambda^*)\end{Bmatrix}\\{}_m\textbf{lambda}\ (ord\text{-}\lambda^*)\end{Bmatrix}\ \left\{\left|\begin{matrix}(\textbf{declare}\ \widehat{decl^*})^*\\ \widehat{doc}\end{matrix}\right.\right\}\ form^{P*}\right)$$

▷ Define a function named *foo* or (**setf** *foo*), or an anonymous function, respectively, which applies *forms* to *ord-λ*s. For $_m$**defun**, *forms* are enclosed in an implicit $_s$**block** named *foo*.

$$\left(\begin{Bmatrix}{}_s\textbf{flet}\\{}_s\textbf{labels}\end{Bmatrix}\ \left(\left(\begin{Bmatrix}foo\ (ord\text{-}\lambda^*)\\(\textbf{setf}\ foo)\ (new\text{-}value\ ord\text{-}\lambda^*)\end{Bmatrix}\ \left\{\left|\begin{matrix}(\textbf{declare}\ \widehat{local\text{-}decl^*})^*\\ \widehat{doc}\end{matrix}\right.\right\}\ local\text{-}form^{P*}\right)^*\right)\ (\textbf{declare}\ \widehat{decl^*})^*\ form^{P*}\right)$$

▷ Evaluate *forms* with locally defined functions *foo*. Globally defined functions of the same name are shadowed. Each *foo* is also the name of an implicit $_s$**block** around its corresponding *local-form**. Only for $_s$**labels**, functions *foo* are visible inside *local-forms*. Return values of *forms*.

$$\left({}_s\textbf{function}\ \begin{Bmatrix}foo\\({}_m\textbf{lambda}\ form^*)\end{Bmatrix}\right)$$

▷ Return lexically innermost function named *foo* or a lexical closure of the $_m$**lambda** expression.

$$\left({}_f\textbf{apply}\ \begin{Bmatrix}function\\(\textbf{setf}\ function)\end{Bmatrix}\ arg^*\ args\right)$$

▷ Values of *function* called with *args* and the list elements of *args*. **setf**able if *function* is one of $_f$**aref**, $_f$**bit**, and $_f$**sbit**.

($_f$**funcall** *function* arg*) ▷ Values of *function* called with *args*.

($_s$**multiple-value-call** *function form**)

▷ Call *function* with all the values of each *form* as its arguments. Return values returned by *function*.

($_f$**values-list** *list*) ▷ Return elements of *list*.

($_f$**values** *foo**)

▷ Return as multiple values the primary values of the *foo*s. **setf**able.

($_f$**multiple-value-list** *form*) ▷ List of the values of *form*.

(${}_{m}$**nth-value** *n form*)

▷ Zero-indexed $\underline{n\text{th return value}}$ of *form*.

(${}_{f}$**complement** *function*)

▷ Return $\underline{\text{new function}}$ with same arguments and same side effects as *function*, but with complementary truth value.

(${}_{f}$**constantly** *foo*)

▷ $\underline{\text{Function}}$ of any number of arguments returning *foo*.

(${}_{f}$**identity** *foo*) ▷ Return $\underline{foo}$.

(${}_{f}$**function-lambda-expression** *function*)

▷ If available, return $\underline{\text{lambda expression}}$ of *function*, $\underset{2}{\underline{\texttt{NIL}}}$ if *function* was defined in an environment without bindings, and $\underset{3}{\underline{\text{name}}}$ of *function*.

$$({}_{f}\textbf{fdefinition} \begin{Bmatrix} foo \\ (\textbf{setf}\ foo) \end{Bmatrix})$$

▷ $\underline{\text{Definition}}$ of global function *foo*. **setf**able.

(${}_{f}$**fmakunbound** *foo*)

▷ Remove global function or macro definition $\underline{foo}$.

${}_{c}$**call-arguments-limit**
${}_{c}$**lambda-parameters-limit**

▷ Upper bound of the number of function arguments or lambda list parameters, respectively; ≥ 50.

${}_{c}$**multiple-values-limit**

▷ Upper bound of the number of values a multiple value can have; ≥ 20.

9.4 Macros

Below, macro lambda list (*macro-λ**) has the form of either

$$([\textbf{\&whole}\ var]\ [E]\ \begin{Bmatrix} var \\ (macro\text{-}\lambda^*) \end{Bmatrix}^*\ [E]$$

$$[\textbf{\&optional} \begin{Bmatrix} var \\ (\begin{Bmatrix} var \\ (macro\text{-}\lambda^*) \end{Bmatrix}\ [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]]) \end{Bmatrix}^*]\ [E]$$

$$[\begin{Bmatrix} \textbf{\&rest} \\ \textbf{\&body} \end{Bmatrix} \begin{Bmatrix} rest\text{-}var \\ (macro\text{-}\lambda^*) \end{Bmatrix}]\ [E]$$

$$[\textbf{\&key} \begin{Bmatrix} var \\ (\begin{Bmatrix} var \\ (\textbf{:}key \begin{Bmatrix} var \\ (macro\text{-}\lambda^*) \end{Bmatrix}) \end{Bmatrix}\ [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]]) \end{Bmatrix}^*\ [E]$$

$$[\textbf{\&allow-other-keys}]]\ [\textbf{\&aux} \begin{Bmatrix} var \\ (var\ [init_{\boxed{\texttt{NIL}}}]) \end{Bmatrix}^*]\ [E])$$

or

$$([\textbf{\&whole}\ var]\ [E]\ \begin{Bmatrix} var \\ (macro\text{-}\lambda^*) \end{Bmatrix}^*\ [E]$$

$$[\textbf{\&optional} \begin{Bmatrix} var \\ (\begin{Bmatrix} var \\ (macro\text{-}\lambda^*) \end{Bmatrix}\ [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]]) \end{Bmatrix}^*]\ [E]\ \textbf{.}\ rest\text{-}var).$$

One toplevel [*E*] may be replaced by **&environment** *var*. *supplied-p* is `T` if there is a corresponding argument. *init* forms can refer to any *init* and *supplied-p* to their left.

$$(\begin{Bmatrix} {}_{m}\textbf{defmacro} \\ {}_{m}\textbf{define-compiler-macro} \end{Bmatrix} \begin{Bmatrix} foo \\ (\textbf{setf}\ foo) \end{Bmatrix}\ (macro\text{-}\lambda^*)$$

$$\begin{Bmatrix} \left| \begin{matrix} (\textbf{declare}\ \widehat{decl^*})^* \\ \widehat{doc} \end{matrix} \right. \end{Bmatrix}\ form^{\overset{\text{P}}{*}})$$

▷ Define macro $\underline{foo}$ which on evaluation as (*foo tree*) applies expanded *form*s to arguments from *tree*, which corresponds to *tree*-shaped *macro-λ*s. *form*s are enclosed in an implicit ${}_{s}$**block** named *foo*.

($_m$**define-symbol-macro** *foo form*)

▷ Define symbol macro $\underline{foo}$ which on evaluation evaluates expanded *form*.

($_s$**macrolet** ((*foo* (*macro-λ**) $\left\{\begin{array}{|l}(\textbf{declare}\ \widehat{local\text{-}decl}^*)^* \\ \widehat{doc}\end{array}\right\}$ *macro-form*^{P_*})*) (**declare** $\widehat{decl}^*$)* *form*^{P_*})

▷ Evaluate $\underline{forms}$ with locally defined mutually invisible macros *foo* which are enclosed in implicit $_s$**block**s of the same name.

($_s$**symbol-macrolet** ((*foo expansion-form*)*) (**declare** $\widehat{decl}^*$)* *form*^{P_*})

▷ Evaluate $\underline{forms}$ with locally defined symbol macros *foo*.

($_m$**defsetf** $\widehat{function}$ $\left\{\begin{array}{l}\widehat{updater}\ [\widehat{doc}] \\ (setf\text{-}\lambda^*)\ (s\text{-}var^*)\ \left\{\begin{array}{|l}(\textbf{declare}\ \widehat{decl}^*)^* \\ \widehat{doc}\end{array}\right\}\ form^{P_*}\end{array}\right\}$)

where defsetf lambda list (*setf-λ**) has the form

(*var** $\Bigl[$**&optional** $\left\{\begin{array}{l}var \\ (var\ \bigl[init_{\boxed{\text{NIL}}}\ [supplied\text{-}p]\bigr])\end{array}\right\}^*\Bigr]$ [**&rest** *var*]

$\left[\textbf{\&key}\ \left\{\begin{array}{l}var \\ \left(\left\{\begin{array}{l}var \\ (\textbf{:}key\ var)\end{array}\right\}\ \bigl[init_{\boxed{\text{NIL}}}\ [supplied\text{-}p]\bigr]\right)\end{array}\right\}^*\right.$

[**&allow-other-keys**]] $\bigl[$**&environment** *var*$\bigr]$)

▷ Specify how to **setf** a place accessed by $\underline{function}$. **Short form:** (**setf** (*function arg**) *value-form*) is replaced by (*updater arg** *value-form*); the latter must return *value-form*. **Long form:** on invocation of (**setf** (*function arg**) *value-form*), *form*s must expand into code that sets the place accessed where *setf-λ* and *s-var** describe the arguments of *function* and the value(s) to be stored, respectively; and that returns the value(s) of *s-var**. *form*s are enclosed in an implicit $_s$**block** named *function*.

($_m$**define-setf-expander** *function* (*macro-λ**) $\left\{\begin{array}{|l}(\textbf{declare}\ \widehat{decl}^*)^* \\ \widehat{doc}\end{array}\right\}$ *form*^{P_*})

▷ Specify how to **setf** a place accessed by $\underline{function}$. On invocation of (**setf** (*function arg**) *value-form*), *form** must expand into code returning *arg-vars*, *args*, *newval-vars*, *set-form*, and *get-form* as described with $_f$**get-setf-expansion** where the elements of macro lambda list *macro-λ** are bound to corresponding *args*. *form*s are enclosed in an implicit $_s$**block** named *function*.

($_f$**get-setf-expansion** *place* [*environment*$_{\boxed{\text{NIL}}}$])

▷ Return lists of temporary variables $\underline{arg\text{-}vars}$ and of corresponding $\underset{2}{\underline{args}}$ as given with *place*, list $\underset{3}{\underline{newval\text{-}vars}}$ with temporary variables corresponding to the new values, and $\underset{4}{\underline{set\text{-}form}}$ and $\underset{5}{\underline{get\text{-}form}}$ specifying in terms of *arg-vars* and *newval-vars* how to **setf** and how to read *place*.

($_m$**define-modify-macro** *foo* ($\Bigl[$**&optional** $\left\{\begin{array}{l}var \\ (var\ \bigl[init_{\boxed{\text{NIL}}}\ [supplied\text{-}p]\bigr])\end{array}\right\}^*\Bigr]$ [**&rest** *var*]) *function* [$\widehat{doc}$])

▷ Define macro $\underline{foo}$ able to modify a place. On invocation of (*foo place arg**), the value of *function* applied to *place* and *args* will be stored into *place* and returned.

$_c$**lambda-list-keywords**

▷ List of macro lambda list keywords. These are at least:

&whole *var* ▷ Bind *var* to the entire macro call form.

&optional *var**

▷ Bind *var*s to corresponding arguments if any.

{**&rest**|**&body**} *var*

▷ Bind *var* to a list of remaining arguments.

&key *var**

▷ Bind *var*s to corresponding keyword arguments.

&allow-other-keys

▷ Suppress keyword argument checking. Callers can do so using **:allow-other-keys** `T`.

&environment *var*

▷ Bind *var* to the lexical compilation environment.

&aux var^* ▷ Bind *var*s as in ${}_s$**let***.

9.5 Control Flow

$({}_s\textbf{if}\ test\ then\ [else_{\boxed{\texttt{NIL}}}])$

▷ Return values of *then* if *test* returns `T`; return values of *else* otherwise.

$({}_m\textbf{cond}\ (test\ then^{\overset{P}{*}}{}_{\boxed{test}})^*)$

▷ Return the values of the first $then^*$ whose *test* returns `T`; return `NIL` if all *test*s return `NIL`.

$\left(\left\{\begin{array}{l}{}_m\textbf{when}\\ {}_m\textbf{unless}\end{array}\right\}\ test\ foo^{\overset{P}{*}}\right)$

▷ Evaluate *foo*s and return their values if *test* returns `T` or `NIL`, respectively. Return `NIL` otherwise.

$\left({}_m\textbf{case}\ test\ \left(\left\{\begin{array}{l}(\widehat{key^*})\\ \widehat{key}\end{array}\right\}\ foo^{\overset{P}{*}}\right)^*\ \left[\left(\left\{\begin{array}{l}\textbf{otherwise}\\ \texttt{T}\end{array}\right\}\ bar^{\overset{P}{*}}\right)_{\boxed{\texttt{NIL}}}\right]\right)$

▷ Return the values of the first foo^* one of whose *key*s is **eql** *test*. Return values of *bar*s if there is no matching *key*.

$\left(\left\{\begin{array}{l}{}_m\textbf{ecase}\\ {}_m\textbf{ccase}\end{array}\right\}\ test\ \left(\left\{\begin{array}{l}(\widehat{key^*})\\ \widehat{key}\end{array}\right\}\ foo^{\overset{P}{*}}\right)^*\right)$

▷ Return the values of the first foo^* one of whose *key*s is **eql** *test*. Signal non-correctable/correctable **type-error** if there is no matching *key*.

$({}_m\textbf{and}\ form^*{}_{\boxed{\texttt{T}}})$

▷ Evaluate *form*s from left to right. Immediately return `NIL` if one *form*'s value is `NIL`. Return values of last *form* otherwise.

$({}_m\textbf{or}\ form^*{}_{\boxed{\texttt{NIL}}})$

▷ Evaluate *form*s from left to right. Immediately return primary value of first non-`NIL`-evaluating form, or all values if last *form* is reached. Return `NIL` if no *form* returns `T`.

$({}_s\textbf{progn}\ form^*{}_{\boxed{\texttt{NIL}}})$

▷ Evaluate *form*s sequentially. Return values of last *form*.

$({}_s\textbf{multiple-value-prog1}\ form\text{-}r\ form^*)$
$({}_m\textbf{prog1}\ form\text{-}r\ form^*)$
$({}_m\textbf{prog2}\ form\text{-}a\ form\text{-}r\ form^*)$

▷ Evaluate forms in order. Return values/primary value, respectively, of *form-r*.

$\left(\left\{\begin{array}{l}{}_m\textbf{prog}\\ {}_m\textbf{prog*}\end{array}\right\}\ \left(\left\{\begin{array}{|l}name\\ (name\ [value_{\boxed{\texttt{NIL}}}])\end{array}\right\}^*\right)\ (\textbf{declare}\ \widehat{decl}^*)^*\ \left\{\begin{array}{l}\widehat{tag}\\ form\end{array}\right\}^*\right)$

▷ Evaluate ${}_s$**tagbody**-like body with *name*s lexically bound (in parallel or sequentially, respectively) to *value*s. Return `NIL` or explicitly ${}_m$**return**ed values. Implicitly, the whole form is a ${}_s$**block** named `NIL`.

$({}_s\textbf{unwind-protect}\ protected\ cleanup^*)$

▷ Evaluate *protected* and then, no matter how control leaves *protected*, *cleanup*s. Return values of *protected*.

$({}_s\textbf{block}\ name\ form^{\overset{P}{*}})$

▷ Evaluate *form*s in a lexical environment, and return their values unless interrupted by ${}_s$**return-from**.

$({}_s\textbf{return-from}\ foo\ [result_{\boxed{\texttt{NIL}}}])$
$({}_m\textbf{return}\ [result_{\boxed{\texttt{NIL}}}])$

▷ Have nearest enclosing ${}_s$**block** named *foo*/named `NIL`, respectively, return with values of *result*.

$(_s\textbf{tagbody}\ \{\widehat{tag}|\mathit{form}\}^*)$

▷ Evaluate *forms* in a lexical environment. *tags* (symbols or integers) have lexical scope and dynamic extent, and are targets for $_s$**go**. Return NIL.

$(_s\textbf{go}\ \widehat{tag})$

▷ Within the innermost possible enclosing $_s$**tagbody**, jump to a tag $_f$**eql** *tag*.

$(_s\textbf{catch}\ \mathit{tag}\ \mathit{form}^{\text{P}*})$

▷ Evaluate *forms* and return their values unless interrupted by $_s$**throw**.

$(_s\textbf{throw}\ \mathit{tag}\ \mathit{form})$

▷ Have the nearest dynamically enclosing $_s$**catch** with a tag $_f$**eq** *tag* return with the values of *form*.

$(_f\textbf{sleep}\ n)$ ▷ Wait *n* seconds; return NIL.

9.6 Iteration

$$(\begin{Bmatrix} _m\textbf{do} \\ _m\textbf{do*} \end{Bmatrix}\ (\begin{Bmatrix} \mathit{var} \\ (\mathit{var}\ [\mathit{start}\ [\mathit{step}]]) \end{Bmatrix}^*)\ (\mathit{stop}\ \mathit{result}^{\text{P}*})\ (\textbf{declare}\ \widehat{\mathit{decl}^*})^*\ \begin{Bmatrix} \widehat{\mathit{tag}} \\ \mathit{form} \end{Bmatrix}^*)$$

▷ Evaluate $_s$**tagbody**-like body with *vars* successively bound according to the values of the corresponding *start* and *step* forms. *vars* are bound in parallel/sequentially, respectively. Stop iteration when *stop* is T. Return values of *result**. Implicitly, the whole form is a $_s$**block** named NIL.

$(_m\textbf{dotimes}\ (\mathit{var}\ i\ [\mathit{result}_{\boxed{\texttt{NIL}}}])\ (\textbf{declare}\ \widehat{\mathit{decl}^*})^*\ \{\widehat{\mathit{tag}}|\mathit{form}\}^*)$

▷ Evaluate $_s$**tagbody**-like body with *var* successively bound to integers from 0 to $i - 1$. Upon evaluation of *result*, *var* is *i*. Implicitly, the whole form is a $_s$**block** named NIL.

$(_m\textbf{dolist}\ (\mathit{var}\ \mathit{list}\ [\mathit{result}_{\boxed{\texttt{NIL}}}])\ (\textbf{declare}\ \widehat{\mathit{decl}^*})^*\ \{\widehat{\mathit{tag}}|\mathit{form}\}^*)$

▷ Evaluate $_s$**tagbody**-like body with *var* successively bound to the elements of *list*. Upon evaluation of *result*, *var* is NIL. Implicitly, the whole form is a $_s$**block** named NIL.

9.7 Loop Facility

$(_m\textbf{loop}\ \mathit{form}^*)$

▷ **Simple Loop.** If *forms* do not contain any atomic Loop Facility keywords, evaluate them forever in an implicit $_s$**block** named NIL.

$(_m\textbf{loop}\ \mathit{clause}^*)$

▷ **Loop Facility.** For Loop Facility keywords see below and Figure 1.

$\textbf{named}\ n_{\boxed{\texttt{NIL}}}$ ▷ Give $_m$**loop**'s implicit $_s$**block** a name.

$$\{\textbf{with}\ \begin{Bmatrix} \mathit{var\text{-}s} \\ (\mathit{var\text{-}s}^*) \end{Bmatrix}\ [\mathit{d\text{-}type}]\ [=\mathit{foo}]\}^+$$

$$\{\textbf{and}\ \begin{Bmatrix} \mathit{var\text{-}p} \\ (\mathit{var\text{-}p}^*) \end{Bmatrix}\ [\mathit{d\text{-}type}]\ [=\mathit{bar}]\}^*$$

where destructuring type specifier *d-type* has the form

$$\{\textbf{fixnum}|\textbf{float}|\texttt{T}|\texttt{NIL}|\{\textbf{of-type}\ \begin{Bmatrix} \mathit{type} \\ (\mathit{type}^*) \end{Bmatrix}\}\}$$

▷ Initialize (possibly trees of) local variables *var-s* sequentially and *var-p* in parallel.

$$\{\{\textbf{for}|\textbf{as}\}\ \begin{Bmatrix} \mathit{var\text{-}s} \\ (\mathit{var\text{-}s}^*) \end{Bmatrix}\ [\mathit{d\text{-}type}]\}^+\ \{\textbf{and}\ \begin{Bmatrix} \mathit{var\text{-}p} \\ (\mathit{var\text{-}p}^*) \end{Bmatrix}\ [\mathit{d\text{-}type}]\}^*$$

▷ Begin of iteration control clauses. Initialize and step (possibly trees of) local variables *var-s* sequentially and *var-p* in parallel. Destructuring type specifier *d-type* as with **with**.

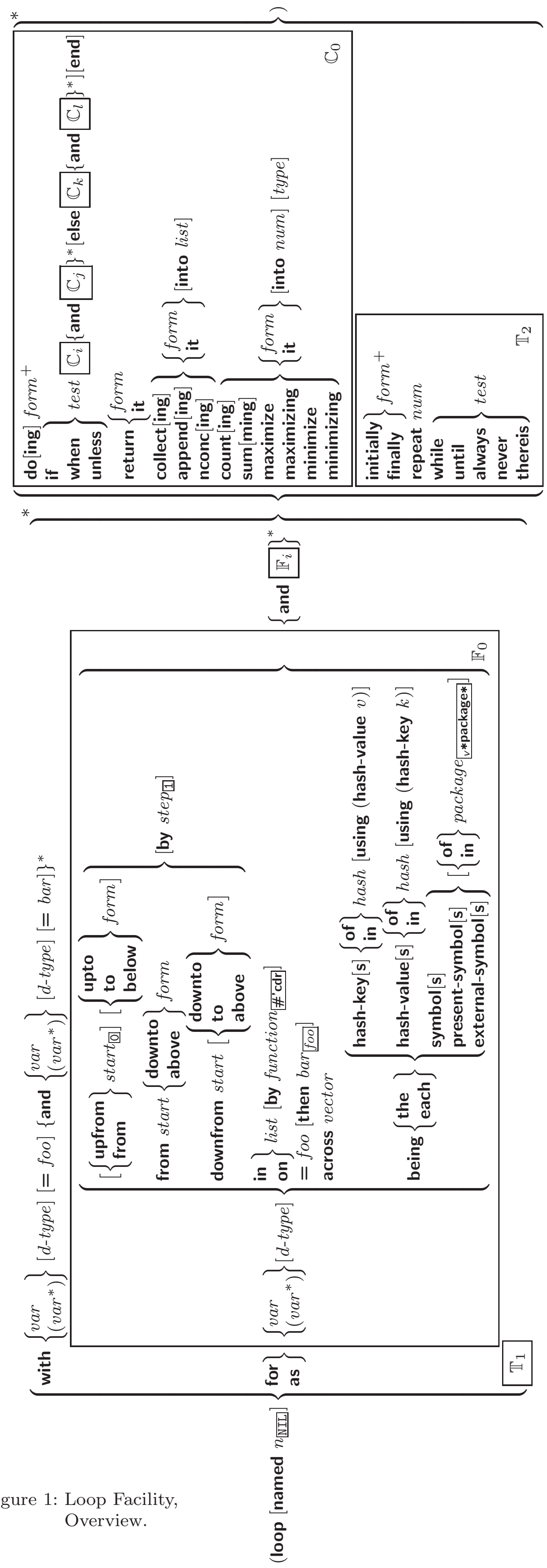
(loop [named n_NIL]
with {var | (var*)} [d-type] [= foo] {and {var | (var*)} [d-type] [= bar]}*
for | as {var | (var*)} [d-type]
[{upfrom | from} start_0] [{upto | to | below} form]
from start {downto | above} form
downfrom start [{downto | to | above} form]
[by step_1]
in | on list [by function_#'cdr]
= foo [then bar_foo]
across vector
being {the | each}
hash-key[s] {of | in} hash [using (hash-value v)]
hash-value[s] {of | in} hash [using (hash-key k)]
symbol[s] | present-symbol[s] | external-symbol[s] [{of | in} package_v*package*]
F_0
{and F_i}*
T_1
do[ing] form+
if | when | unless test C_i {and C_j}* [else C_k {and C_l}*] [end]
return {form | it}
collect[ing] | append[ing] | nconc[ing] {form | it} [into list]
count[ing] | sum[ming] | maximize | maximizing | minimize | minimizing {form | it} [into num] [type]
C_0
initially | finally form+
repeat num
while | until | always | never | thereis test
T_2
)*

Figure 1: Loop Facility, Overview.

{**upfrom**|**from**|**downfrom**} *start*
▷ Start stepping with *start*

{**upto**|**downto**|**to**|**below**|**above**} *form*
▷ Specify *form* as the end value for stepping.

{**in**|**on**} *list*
▷ Bind *var* to successive elements/tails, respectively, of *list*.

by {$step_{\boxed{1}}$|$function_{\boxed{\texttt{\#'cdr}}}$}
▷ Specify the (positive) decrement or increment or the *function* of one argument returning the next part of the list.

= *foo* [**then** $bar_{\boxed{foo}}$]
▷ Bind *var* initially to *foo* and later to *bar*.

across *vector*
▷ Bind *var* to successive elements of *vector*.

being {**the**|**each**}
▷ Iterate over a hash table or a package.

{**hash-key**|**hash-keys**} {**of**|**in**} *hash-table* [**using** (**hash-value** *value*)]
▷ Bind *var* successively to the keys of *hash-table*; bind *value* to corresponding values.

{**hash-value**|**hash-values**} {**of**|**in**} *hash-table* [**using** (**hash-key** *key*)]
▷ Bind *var* successively to the values of *hash-table*; bind *key* to corresponding keys.

{**symbol**|**symbols**|**present-symbol**|**present-symbols**| **external-symbol**|**external-symbols**} [{**of**|**in**} $package_{\boxed{{}_v\textbf{*package*}}}$]
▷ Bind *var* successively to the accessible symbols, or the present symbols, or the external symbols respectively, of *package*.

{**do**|**doing**} $form^{+}$ ▷ Evaluate *form*s in every iteration.

{**if**|**when**|**unless**} *test i-clause* {**and** *j-clause*}* [**else** *k-clause* {**and** *l-clause*}*] [**end**]
▷ If *test* returns `T`, `T`, or `NIL`, respectively, evaluate *i-clause* and *j-clause*s; otherwise, evaluate *k-clause* and *l-clause*s.

it ▷ Inside *i-clause* or *k-clause*: value of *test*.

return {*form*|**it**}
▷ Return immediately, skipping any **finally** parts, with values of *form* or **it**.

{**collect**|**collecting**} {*form*|**it**} [**into** *list*]
▷ Collect values of *form* or **it** into *list*. If no *list* is given, collect into an anonymous list which is returned after termination.

{**append**|**appending**|**nconc**|**nconcing**} {*form*|**it**} [**into** *list*]
▷ Concatenate values of *form* or **it**, which should be lists, into *list* by the means of $_f$**append** or $_f$**nconc**, respectively. If no *list* is given, collect into an anonymous list which is returned after termination.

{**count**|**counting**} {*form*|**it**} [**into** *n*] [*type*]
▷ Count the number of times the value of *form* or of **it** is `T`. If no *n* is given, count into an anonymous variable which is returned after termination.

{**sum**|**summing**} {*form*|**it**} [**into** *sum*] [*type*]
▷ Calculate the sum of the primary values of *form* or of **it**. If no *sum* is given, sum into an anonymous variable which is returned after termination.

{**maximize**|**maximizing**|**minimize**|**minimizing**} {*form*|**it**} [**into** *max-min*] [*type*]
▷ Determine the maximum or minimum, respectively, of the primary values of *form* or of **it**. If no *max-min* is given, use an anonymous variable which is returned after termination.

{**initially**|**finally**} *form*$^+$

▷ Evaluate *form*s before begin, or after end, respectively, of iterations.

repeat *num*

▷ Terminate ${}_m$**loop** after *num* iterations; *num* is evaluated once.

{**while**|**until**} *test*

▷ Continue iteration until *test* returns `NIL` or `T`, respectively.

{**always**|**never**} *test*

▷ Terminate ${}_m$**loop** returning `NIL` and skipping any **finally** parts as soon as *test* is `NIL` or `T`, respectively. Otherwise continue ${}_m$**loop** with its default return value set to `T`.

thereis *test*

▷ Terminate ${}_m$**loop** when *test* is `T` and return value of *test*, skipping any **finally** parts. Otherwise continue ${}_m$**loop** with its default return value set to `NIL`.

(${}_m$**loop-finish**)

▷ Terminate ${}_m$**loop** immediately executing any **finally** clauses and returning any accumulated results.

10 CLOS

10.1 Classes

(${}_f$**slot-exists-p** *foo bar*) ▷ $\underline{\texttt{T}}$ if *foo* has a slot *bar*.

(${}_f$**slot-boundp** *instance slot*) ▷ $\underline{\texttt{T}}$ if *slot* in *instance* is bound.

$$({}_m\textbf{defclass}\ foo\ (superclass^*_{\boxed{\text{standard-object}}})$$

$$\left(\left\{\begin{array}{l} slot \\ (slot \left\{\begin{array}{|l} \{\textbf{:reader}\ reader\}^* \\ \{\textbf{:writer} \left\{\begin{array}{l} writer \\ (\textbf{setf}\ writer)\end{array}\right\}\}^* \\ \{\textbf{:accessor}\ accessor\}^* \\ \textbf{:allocation} \left\{\begin{array}{l} \textbf{:instance} \\ \textbf{:class}\end{array}\right\}_{\boxed{\text{:instance}}} \\ \{\textbf{:initarg}\ [\textbf{:}]initarg\text{-}name\}^* \\ \textbf{:initform}\ form \\ \textbf{:type}\ type \\ \textbf{:documentation}\ slot\text{-}doc \end{array}\right\}) \end{array}\right\}^*\right)$$

$$\left\{\begin{array}{|l} (\textbf{:default-initargs}\ \{name\ value\}^*) \\ (\textbf{:documentation}\ class\text{-}doc) \\ (\textbf{:metaclass}\ name_{\boxed{\text{standard-class}}}) \end{array}\right\})$$

▷ Define or modify $\underline{\text{class } foo}$ as a subclass of *superclass*es. Transform existing instances, if any, by ${}_g$**make-instances-obsolete**. In a new instance *i* of *foo*, a *slot*'s value defaults to *form* unless set via [**:**]*initarg-name*; it is readable via (*reader i*) or (*accessor i*), and writable via (*writer value i*) or (**setf** (*accessor i*) *value*). *slot*s with **:allocation :class** are shared by all instances of class *foo*.

(${}_f$**find-class** *symbol* $\left[errorp_{\boxed{\texttt{T}}}\ [environment]\right]$)

▷ Return $\underline{\text{class}}$ named *symbol*. **setf**able.

(${}_g$**make-instance** *class* {[**:**]*initarg value*}* *other-keyarg**)

▷ Make new $\underline{\text{instance of } class}$.

(${}_g$**reinitialize-instance** *instance* {[**:**]*initarg value*}* *other-keyarg**)

▷ Change local slots of $\underline{instance}$ according to *initarg*s by means of ${}_g$**shared-initialize**.

(${}_f$**slot-value** *foo slot*) ▷ Return $\underline{\text{value of } slot \text{ in } foo}$. **setf**able.

(${}_f$**slot-makunbound** *instance slot*)

▷ Make *slot* in $\underline{instance}$ unbound.

$$\left(\begin{cases}{}_m\textbf{with-slots}\ (\{\widehat{slot}|(\widehat{var}\ \widehat{slot})\}^*)\\ {}_m\textbf{with-accessors}\ ((\widehat{var}\ \widehat{accessor})^*)\end{cases}\right\}\ instance\ (\textbf{declare}\ \widehat{decl}^*)^*\ form^{\mathrm{P}*})$$

▷ Return values of *forms* after evaluating them in a lexical environment with slots of *instance* visible as **setf**able *slots* or *vars*/with *accessors* of *instance* visible as **setf**able *vars*.

(${}_g$**class-name** *class*)
((**setf** ${}_g$**class-name**) *new-name class*) ▷ Get/set name of *class*.

(${}_f$**class-of** *foo*) ▷ Class *foo* is a direct instance of.

(${}_g$**change-class** $\widetilde{instance}$ *new-class* {[**:**]*initarg value*}* *other-keyarg**)

▷ Change class of *instance* to *new-class*. Retain the status of any slots that are common between *instance*'s original class and *new-class*. Initialize any newly added slots with the *values* of the corresponding *initargs* if any, or with the values of their **:initform** forms if not.

(${}_g$**make-instances-obsolete** *class*)

▷ Update all existing instances of *class* using ${}_g$**update-instance-for-redefined-class**.

$$\left(\begin{Bmatrix}{}_g\textbf{initialize-instance}\ instance\\ {}_g\textbf{update-instance-for-different-class}\ previous\ current\end{Bmatrix}\right.$$
{[**:**]*initarg value*}* *other-keyarg**)

▷ Set slots on behalf of ${}_g$**make-instance**/of ${}_g$**change-class** by means of ${}_g$**shared-initialize**.

(${}_g$**update-instance-for-redefined-class** *new-instance added-slots discarded-slots discarded-slots-property-list* {[**:**]*initarg value*}* *other-keyarg**)

▷ On behalf of ${}_g$**make-instances-obsolete** and by means of ${}_g$**shared-initialize**, set any *initarg* slots to their corresponding *values*; set any remaining *added-slots* to the values of their **:initform** forms. Not to be called by user.

(${}_g$**allocate-instance** *class* {[**:**]*initarg value*}* *other-keyarg**)

▷ Return uninitialized instance of *class*. Called by ${}_g$**make-instance**.

$$({}_g\textbf{shared-initialize}\ instance\ \begin{Bmatrix}initform\text{-}slots\\ \mathtt{T}\end{Bmatrix}\ \{[\textbf{:}]initarg\text{-}slot\ value\}^*$$
*other-keyarg**)

▷ Fill the *initarg-slots* of *instance* with the corresponding *values*, and fill those *initform-slots* that are not *initarg-slots* with the values of their **:initform** forms.

$$({}_g\textbf{slot-missing}\ class\ instance\ slot\ \begin{Bmatrix}\textbf{setf}\\ \textbf{slot-boundp}\\ \textbf{slot-makunbound}\\ \textbf{slot-value}\end{Bmatrix}[value])$$

(${}_g$**slot-unbound** *class instance slot*)

▷ Called on attempted access to non-existing or unbound *slot*. Default methods signal **error**/**unbound-slot**, respectively. Not to be called by user.

10.2 Generic Functions

(${}_f$**next-method-p**) ▷ T if enclosing method has a next method.

$$({}_m\textbf{defgeneric}\ \begin{Bmatrix}foo\\ (\textbf{setf}\ foo)\end{Bmatrix}\ (required\text{-}var^*\ \Big[\textbf{\&optional}\ \begin{Bmatrix}var\\ (var)\end{Bmatrix}^*\Big]\ \Big[\textbf{\&rest}$$

$$var\Big]\ \Big[\textbf{\&key}\ \begin{Bmatrix}var\\ (var|(\textbf{:}key\ var))\end{Bmatrix}^*\ [\textbf{\&allow-other-keys}]\Big])$$

$$\left\{\begin{array}{|l}(\textbf{:argument-precedence-order}\ required\text{-}var^+)\\ (\textbf{declare}\ (\textbf{optimize}\ method\text{-}selection\text{-}optimization)^+)\\ (\textbf{:documentation}\ \widehat{string})\\ (\textbf{:generic-function-class}\ gf\text{-}class_{\boxed{\textbf{standard-generic-function}}})\\ (\textbf{:method-class}\ method\text{-}class_{\boxed{\textbf{standard-method}}})\\ (\textbf{:method-combination}\ c\text{-}type_{\boxed{\textbf{standard}}}\ c\text{-}arg^*)\\ (\textbf{:method}\ defmethod\text{-}args)^*\end{array}\right\})$$

▷ Define or modify generic function *foo*. Remove any methods previously defined by defgeneric. *gf-class* and the lambda paramters *required-var** and *var** must be compatible with existing methods. *defmethod-args* resemble those of $_m$**defmethod**. For *c-type* see section 10.3.

$(_f\textbf{ensure-generic-function}\ \begin{Bmatrix} \textit{foo} \\ (\textbf{setf}\ \textit{foo}) \end{Bmatrix}$
$\left\{\begin{array}{|l} \textbf{:argument-precedence-order}\ \textit{required-var}^+ \\ \textbf{:declare}\ (\textbf{optimize}\ \textit{method-selection-optimization}) \\ \textbf{:documentation}\ \textit{string} \\ \textbf{:generic-function-class}\ \textit{gf-class} \\ \textbf{:method-class}\ \textit{method-class} \\ \textbf{:method-combination}\ \textit{c-type}\ \textit{c-arg}^* \\ \textbf{:lambda-list}\ \textit{lambda-list} \\ \textbf{:environment}\ \textit{environment} \end{array}\right\})$

▷ Define or modify generic function *foo*. *gf-class* and *lambda-list* must be compatible with a pre-existing generic function or with existing methods, respectively. Changes to *method-class* do not propagate to existing methods. For *c-type* see section 10.3.

$(_m\textbf{defmethod}\ \begin{Bmatrix} \textit{foo} \\ (\textbf{setf}\ \textit{foo}) \end{Bmatrix}\ \left[\begin{Bmatrix} \textbf{:before} \\ \textbf{:after} \\ \textbf{:around} \\ \textit{qualifier}^* \end{Bmatrix}_{\boxed{\text{primary method}}}\right]$
$(\begin{Bmatrix} \textit{var} \\ (\textit{spec-var}\ \begin{Bmatrix} \textit{class} \\ (\textbf{eql}\ \textit{bar}) \end{Bmatrix}) \end{Bmatrix}^*\ \big[\textbf{\&optional}$
$\begin{Bmatrix} \textit{var} \\ (\textit{var}\ \big[\textit{init}\ [\textit{supplied-p}]\big]) \end{Bmatrix}^*\big]\ \big[\textbf{\&rest}\ \textit{var}\big]\ \big[\textbf{\&key}$
$\begin{Bmatrix} \textit{var} \\ (\begin{Bmatrix} \textit{var} \\ (\textit{:key}\ \textit{var}) \end{Bmatrix}\big[\textit{init}\ [\textit{supplied-p}]\big]) \end{Bmatrix}^*\ \big[\textbf{\&allow-other-keys}\big]\big]$
$\big[\textbf{\&aux}\ \begin{Bmatrix} \textit{var} \\ (\textit{var}\ [\textit{init}]) \end{Bmatrix}^*\big])\ \left\{\begin{array}{|l} (\textbf{declare}\ \widehat{\textit{decl}}^*)^* \\ \widehat{\textit{doc}} \end{array}\right\}\ \textit{form}^{P*})$

▷ Define new method for generic function *foo*. *spec-var*s specialize to either being of *class* or being **eql** *bar*, respectively. On invocation, *var*s and *spec-var*s of the new method act like parameters of a function with body *form**. *form*s are enclosed in an implicit $_s$**block** *foo*. Applicable *qualifier*s depend on the **method-combination** type; see section 10.3.

$(\begin{Bmatrix} _g\textbf{add-method} \\ _g\textbf{remove-method} \end{Bmatrix}\ \textit{generic-function method})$

▷ Add (if necessary) or remove (if any) *method* to/from *generic-function*.

$(_g\textbf{find-method}\ \textit{generic-function qualifiers specializers}\ [\textit{error}_{\boxed{T}}])$

▷ Return suitable method, or signal **error**.

$(_g\textbf{compute-applicable-methods}\ \textit{generic-function args})$

▷ List of methods suitable for *args*, most specific first.

$(_f\textbf{call-next-method}\ \textit{arg}^*_{\boxed{\text{current args}}})$

▷ From within a method, call next method with *args*; return its values.

$(_g\textbf{no-applicable-method}\ \textit{generic-function arg}^*)$

▷ Called on invocation of *generic-function* on *args* if there is no applicable method. Default method signals **error**. Not to be called by user.

$(\begin{Bmatrix} _f\textbf{invalid-method-error}\ \textit{method} \\ _f\textbf{method-combination-error} \end{Bmatrix}\ \textit{control arg}^*)$

▷ Signal **error** on applicable method with invalid qualifiers, or on method combination. For *control* and *args* see **format**, page 38.

$(_g\textbf{no-next-method}\ \textit{generic-function method arg}^*)$

▷ Called on invocation of **call-next-method** when there is no next method. Default method signals **error**. Not to be called by user.

($_g$**function-keywords** *method*)

▷ Return list of keyword parameters of *method* and $\underline{\mathtt{T}}_{2}$ if other keys are allowed.

($_g$**method-qualifiers** *method*) ▷ List of qualifiers of *method*.

10.3 Method Combination Types

standard

▷ Evaluate most specific **:around** method supplying the values of the generic function. From within this method, $_f$**call-next-method** can call less specific **:around** methods if there are any. If not, or if there are no **:around** methods at all, call all **:before** methods, most specific first, and the most specific primary method which supplies the values of the calling $_f$**call-next-method** if any, or of the generic function; and which can call less specific primary methods via $_f$**call-next-method**. After its return, call all **:after** methods, least specific first.

and|**or**|**append**|**list**|**nconc**|**progn**|**max**|**min**|**+**

▷ Simple built-in **method-combination** types; have the same usage as the *c-type*s defined by the short form of $_m$**define-method-combination**.

$$(_m\textbf{define-method-combination}\ \textit{c-type}\ \left\{\left|\begin{array}{l}\textbf{:documentation}\ \widehat{\textit{string}}\\ \textbf{:identity-with-one-argument}\ \textit{bool}_{\boxed{\text{NIL}}}\\ \textbf{:operator}\ \textit{operator}_{\boxed{\textit{c-type}}}\end{array}\right.\right\})$$

▷ **Short Form.** Define new **method-combination** *c-type*. In a generic function using *c-type*, evaluate most specific **:around** method supplying the values of the generic function. From within this method, $_f$**call-next-method** can call less specific **:around** methods if there are any. If not, or if there are no **:around** methods at all, return from the calling **call-next-method** or from the generic function, respectively, the values of (*operator* (*primary-method gen-arg**)*), *gen-arg** being the arguments of the generic function. The *primary-method*s are ordered $\left[\left\{\begin{array}{l}\textbf{:most-specific-first}\\ \textbf{:most-specific-last}\end{array}\right\}_{\boxed{\textbf{:most-specific-first}}}\right]$ (specified as *c-arg* in $_m$**defgeneric**). Using *c-type* as the *qualifier* in $_m$**defmethod** makes the method primary.

$$(_m\textbf{define-method-combination}\ \textit{c-type}\ (\textit{ord-}\lambda^*)\ ((\textit{group}\ \left\{\begin{array}{l}\boldsymbol{*}\\ (\textit{qualifier}^*\ [\boldsymbol{*}])\\ \textit{predicate}\end{array}\right\}\ \left\{\left|\begin{array}{l}\textbf{:description}\ \textit{control}\\ \textbf{:order}\ \left\{\begin{array}{l}\textbf{:most-specific-first}\\ \textbf{:most-specific-last}\end{array}\right\}_{\boxed{\textbf{:most-specific-first}}}\\ \textbf{:required}\ \textit{bool}\end{array}\right.\right\})^*)\ \left\{\left|\begin{array}{l}(\textbf{:arguments}\ \textit{method-combination-}\lambda^*)\\ (\textbf{:generic-function}\ \textit{symbol})\\ \left\{\left|\begin{array}{l}(\textbf{declare}\ \widehat{\textit{decl}^*})^*\\ \widehat{\textit{doc}}\end{array}\right.\right\}\end{array}\right.\right\}\ \textit{body}^{\text{P}*})$$

▷ **Long Form.** Define new **method-combination** *c-type*. A call to a generic function using *c-type* will be equivalent to a call to the forms returned by *body** with *ord-λ** bound to *c-arg** (cf. $_m$**defgeneric**), with *symbol* bound to the generic function, with *method-combination-λ** bound to the arguments of the generic function, and with *group*s bound to lists of methods. An applicable method becomes a member of the leftmost *group* whose *predicate* or *qualifier*s match. Methods can be called via $_m$**call-method**. Lambda lists (*ord-λ**) and (*method-combination-λ**) according to *ord-λ* on page 18, the latter enhanced by an optional **&whole** argument.

$$(_m\textbf{call-method}\ \left\{\begin{array}{l}\widehat{\textit{method}}\\ (_m\textbf{make-method}\ \widehat{\textit{form}})\end{array}\right\}\ \left[(\left\{\begin{array}{l}\widehat{\textit{next-method}}\\ (_m\textbf{make-method}\ \widehat{\textit{form}})\end{array}\right\}^*)\right])$$

▷ From within an effective method form, call *method* with the arguments of the generic function and with information about its *next-methods*; return its values.

11 Conditions and Errors

For standardized condition types cf. Figure 2 on page 32.

$$({}_m\textbf{define-condition}\ \textit{foo}\ (\textit{parent-type}^*_{\boxed{\text{condition}}})$$

$$\left(\left\{\begin{array}{l} \textit{slot}\\ (\textit{slot}\ \left\{\begin{array}{|l} \{\textbf{:reader}\ \textit{reader}\}^*\\ \{\textbf{:writer}\ \left\{\begin{array}{l}\textit{writer}\\ (\textbf{setf}\ \textit{writer})\end{array}\right\}\}^*\\ \{\textbf{:accessor}\ \textit{accessor}\}^*\\ \textbf{:allocation}\ \left\{\begin{array}{l}\textbf{:instance}\\ \textbf{:class}\end{array}\right\}_{\boxed{\text{:instance}}}\\ \{\textbf{:initarg}\ [\textbf{:}]\textit{initarg-name}\}^*\\ \textbf{:initform}\ \textit{form}\\ \textbf{:type}\ \textit{type}\\ \textbf{:documentation}\ \textit{slot-doc}\end{array}\right\})\end{array}\right\}^*\right)$$

$$\left\{\begin{array}{|l}(\textbf{:default-initargs}\ \{\textit{name}\ \textit{value}\}^*)\\ (\textbf{:documentation}\ \textit{condition-doc})\\ (\textbf{:report}\ \left\{\begin{array}{l}\textit{string}\\ \textit{report-function}\end{array}\right\})\end{array}\right\})$$

▷ Define, as a subtype of *parent-type*s, condition type *foo*. In a new condition, a *slot*'s value defaults to *form* unless set via [**:**]*initarg-name*; it is readable via (*reader i*) or (*accessor i*), and writable via (*writer value i*) or (**setf** (*accessor i*) *value*). With **:allocation :class**, *slot* is shared by all conditions of type *foo*. A condition is reported by *string* or by *report-function* of arguments condition and stream.

$$({}_f\textbf{make-condition}\ \textit{condition-type}\ \{[\textbf{:}]\textit{initarg-name}\ \textit{value}\}^*)$$

▷ Return new instance of *condition-type*.

$$\left(\left\{\begin{array}{l}{}_f\textbf{signal}\\ {}_f\textbf{warn}\\ {}_f\textbf{error}\end{array}\right\}\ \left\{\begin{array}{l}\textit{condition}\\ \textit{condition-type}\ \{[\textbf{:}]\textit{initarg-name}\ \textit{value}\}^*\\ \textit{control}\ \textit{arg}^*\end{array}\right\}\right)$$

▷ Unless handled, signal as **condition**, **warning** or **error**, respectively, *condition* or a new instance of *condition-type* or, with $_f$**format** *control* and *args* (see page 38), **simple-condition**, **simple-warning**, or **simple-error**, respectively. From $_f$**signal** and $_f$**warn**, return `NIL`.

$$({}_f\textbf{cerror}\ \textit{continue-control}\ \left\{\begin{array}{l}\textit{condition}\ \textit{continue-arg}^*\\ \textit{condition-type}\ \{[\textbf{:}]\textit{initarg-name}\ \textit{value}\}^*\\ \textit{control}\ \textit{arg}^*\end{array}\right\})$$

▷ Unless handled, signal as correctable **error** *condition* or a new instance of *condition-type* or, with $_f$**format** *control* and *args* (see page 38), **simple-error**. In the debugger, use $_f$**format** arguments *continue-control* and *continue-arg*s to tag the continue option. Return `NIL`.

$$({}_m\textbf{ignore-errors}\ \textit{form}^{P*})$$

▷ Return values of *forms* or, in case of **error**s, `NIL` and the condition.$_2$

$$({}_f\textbf{invoke-debugger}\ \textit{condition})$$

▷ Invoke debugger with *condition*.

$$({}_m\textbf{assert}\ \textit{test}\ \big[(\textit{place}^*)\ [\left\{\begin{array}{l}\textit{condition}\ \textit{continue-arg}^*\\ \textit{condition-type}\ \{[\textbf{:}]\textit{initarg-name}\ \textit{value}\}^*\\ \textit{control}\ \textit{arg}^*\end{array}\right\}]\big])$$

▷ If *test*, which may depend on *place*s, returns `NIL`, signal as correctable **error** *condition* or a new instance of *condition-type* or, with $_f$**format** *control* and *args* (see page 38), **error**. When using the debugger's continue option, *place*s can be altered before re-evaluation of *test*. Return `NIL`.

(${}_m$**handler-case** *foo* (*type* ([*var*]) (**declare** $\widehat{decl}^*$)* *condition-form*^{P*})*
[(**:no-error** (*ord-λ**) (**declare** $\widehat{decl}^*$)* *form*^{P*})])

▷ If, on evaluation of *foo*, a condition of *type* is signalled, evaluate matching *condition-form*s with *var* bound to the condition, and return their values. Without a condition, bind *ord-λ*s to values of *foo* and return values of *form*s or, without a **:no-error** clause, return values of *foo*. See page 18 for (*ord-λ**).

(${}_m$**handler-bind** ((*condition-type handler-function*)*) *form*^{P*})

▷ Return values of *forms* after evaluating them with *condition-type*s dynamically bound to their respective *handler-function*s of argument condition.

(${}_m$**with-simple-restart** ($\left\{\begin{array}{l}\textit{restart}\\ \texttt{NIL}\end{array}\right\}$ *control arg**) *form*^{P*})

▷ Return values of *forms* unless *restart* is called during their evaluation. In this case, describe *restart* using ${}_f$**format** *control* and *arg*s (see page 38) and return `NIL` and $\underset{2}{\texttt{T}}$.

(${}_m$**restart-case** *form* (*restart* (*ord-λ**) $\left\{\begin{array}{|l}\textbf{:interactive}\ \textit{arg-function}\\ \textbf{:report}\ \left\{\begin{array}{l}\textit{report-function}\\ \textit{string}_{\boxed{\texttt{"}\textit{restart}\texttt{"}}}\end{array}\right.\\ \textbf{:test}\ \textit{test-function}_{\boxed{\texttt{T}}}\end{array}\right\}$
(**declare** $\widehat{decl}^*$)* *restart-form*^{P*})*)

▷ Return values of *form* or, if during evaluation of *form* one of the dynamically established *restart*s is called, the values of its *restart-form*s. A *restart* is visible under *condition* if (**funcall #'***test-function condition*) returns `T`. If presented in the debugger, *restart*s are described by *string* or by **#'***report-function* (of a stream). A *restart* can be called by (**invoke-restart** *restart arg**), where *arg*s match *ord-λ**, or by (**invoke-restart-interactively** *restart*) where a list of the respective *arg*s is supplied by **#'***arg-function*. See page 18 for *ord-λ**.

(${}_m$**restart-bind** (($\left\{\begin{array}{l}\widehat{\textit{restart}}\\ \texttt{NIL}\end{array}\right\}$ *restart-function*
$\left\{\begin{array}{|l}\textbf{:interactive-function}\ \textit{arg-function}\\ \textbf{:report-function}\ \textit{report-function}\\ \textbf{:test-function}\ \textit{test-function}\end{array}\right\}$)*) *form*^{P*})

▷ Return values of *forms* evaluated with dynamically established *restart*s whose *restart-function*s should perform a non-local transfer of control. A restart is visible under *condition* if (*test-function condition*) returns `T`. If presented in the debugger, *restart*s are described by *restart-function* (of a stream). A *restart* can be called by (**invoke-restart** *restart arg**), where *arg*s must be suitable for the corresponding *restart-function*, or by (**invoke-restart-interactively** *restart*) where a list of the respective *arg*s is supplied by *arg-function*.

(${}_f$**invoke-restart** *restart arg**)
(${}_f$**invoke-restart-interactively** *restart*)

▷ Call function associated with *restart* with arguments given or prompted for, respectively. If *restart* function returns, return its values.

($\left\{\begin{array}{l}{}_f\textbf{find-restart}\\ {}_f\textbf{compute-restarts}\ \textit{name}\end{array}\right\}$ [*condition*])

▷ Return innermost restart *name*, or a list of all restarts, respectively, out of those either associated with *condition* or unassociated at all; or, without *condition*, out of all restarts. Return `NIL` if search is unsuccessful.

(${}_f$**restart-name** *restart*) ▷ Name of *restart*.

($\left\{\begin{array}{l}{}_f\textbf{abort}\\ {}_f\textbf{muffle-warning}\\ {}_f\textbf{continue}\\ {}_f\textbf{store-value}\ \textit{value}\\ {}_f\textbf{use-value}\ \textit{value}\end{array}\right\}$ [*condition*$_{\boxed{\texttt{NIL}}}$])

▷ Transfer control to innermost applicable restart with same name (i.e. **abort**, …, **continue** …) out of those either associated with *condition* or un-associated at all; or, without *condition*, out of all restarts. If no restart is found, signal **control-error** for $_f$**abort** and $_f$**muffle-warning**, or return NIL for the rest.

($_m$**with-condition-restarts** *condition restarts form*^{P*})
▷ Evaluate *forms* with *restarts* dynamically associated with *condition*. Return values of *forms*.

($_f$**arithmetic-error-operation** *condition*)
($_f$**arithmetic-error-operands** *condition*)
▷ List of function or of its operands respectively, used in the operation which caused *condition*.

($_f$**cell-error-name** *condition*)
▷ Name of cell which caused *condition*.

($_f$**unbound-slot-instance** *condition*)
▷ Instance with unbound slot which caused *condition*.

($_f$**print-not-readable-object** *condition*)
▷ The object not readably printable under *condition*.

($_f$**package-error-package** *condition*)
($_f$**file-error-pathname** *condition*)
($_f$**stream-error-stream** *condition*)
▷ Package, path, or stream, respectively, which caused the *condition* of indicated type.

($_f$**type-error-datum** *condition*)
($_f$**type-error-expected-type** *condition*)
▷ Object which caused *condition* of type **type-error**, or its expected type, respectively.

($_f$**simple-condition-format-control** *condition*)
($_f$**simple-condition-format-arguments** *condition*)
▷ Return $_f$**format** control or list of $_f$**format** arguments, respectively, of *condition*.

$_v$***break-on-signals***$_{\boxed{\texttt{NIL}}}$
▷ Condition type debugger is to be invoked on.

$_v$***debugger-hook***$_{\boxed{\texttt{NIL}}}$
▷ Function of condition and function itself. Called before debugger.

12 Types and Classes

For any class, there is always a corresponding type of the same name.

($_f$**typep** *foo type* [*environment*$_{\boxed{\texttt{NIL}}}$]) ▷ T if *foo* is of *type*.

($_f$**subtypep** *type-a type-b* [*environment*])
▷ Return T if *type-a* is a recognizable subtype of *type-b*, and NIL$_2$ if the relationship could not be determined.

($_s$**the** $\widehat{type}$ *form*) ▷ Declare values of *form* to be of *type*.

($_f$**coerce** *object type*) ▷ Coerce *object* into *type*.

($_m$**typecase** *foo* ($\widehat{type}$ *a-form*^{P*})* [($\left\{\begin{array}{l}\textbf{otherwise}\\ \texttt{T}\end{array}\right\}$ *b-form*$_{\boxed{\texttt{NIL}}}^{P*}$)])
▷ Return values of the first *a-form** whose *type* is *foo* of. Return values of *b-forms* if no *type* matches.

($\left\{\begin{array}{l}_m\textbf{etypecase}\\ _m\textbf{ctypecase}\end{array}\right\}$ *foo* ($\widehat{type}$ *form*^{P*})*)
▷ Return values of the first *form** whose *type* is *foo* of. Signal non-correctable/correctable **type-error** if no *type* matches.

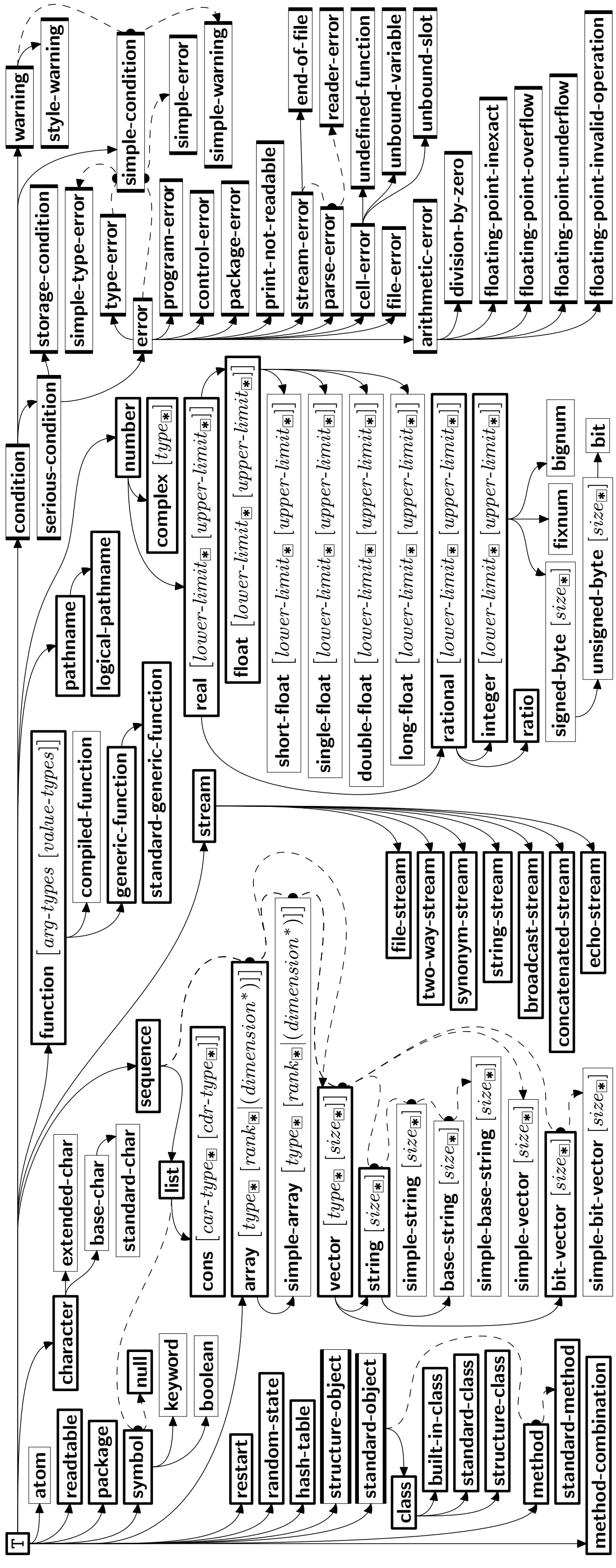

Figure 2: Precedence Order of System Classes (▭), Classes (▬), Types (▭), and Condition Types (▭).
Every type is also a supertype of NIL, the empty type.

($_f$**type-of** *foo*) ▷ Type of *foo*.

($_m$**check-type** *place type* [*string*$_{\boxed{\{\texttt{a}|\texttt{an}\}\ type}}$])
▷ Signal correctable **type-error** if *place* is not of *type*. Return `NIL`.

($_f$**stream-element-type** *stream*) ▷ Type of *stream* objects.

($_f$**array-element-type** *array*) ▷ Element type *array* can hold.

($_f$**upgraded-array-element-type** *type* [*environment*$_{\boxed{\texttt{NIL}}}$])
▷ Element type of most specialized array capable of holding elements of *type*.

($_m$**deftype** *foo* (*macro-λ**) $\left\{\begin{vmatrix}(\textbf{declare}\ \widehat{decl^*})^* \\ \widehat{doc}\end{vmatrix}\right\}$ *form*^{P_*})
▷ Define type *foo* which when referenced as (*foo* $\widehat{arg}^*$) (or as *foo* if *macro-λ* doesn't contain any required parameters) applies expanded *forms* to *args* returning the new type. For (*macro-λ**) see page 19 but with default value of `*` instead of `NIL`. *forms* are enclosed in an implicit $_s$**block** named *foo*.

(**eql** *foo*)
(**member** *foo**) ▷ Specifier for a type comprising *foo* or *foos*.

(**satisfies** *predicate*)
▷ Type specifier for all objects satisfying *predicate*.

(**mod** *n*) ▷ Type specifier for all non-negative integers $< n$.

(**not** *type*) ▷ Complement of type.

(**and** *type*$^*_{\boxed{\texttt{T}}}$) ▷ Type specifier for intersection of *types*.

(**or** *type*$^*_{\boxed{\texttt{NIL}}}$) ▷ Type specifier for union of *types*.

(**values** *type** [**&optional** *type** [**&rest** *other-args*]])
▷ Type specifier for multiple values.

`*` ▷ As a type argument (cf. Figure 2): no restriction.

13 Input/Output

13.1 Predicates

($_f$**streamp** *foo*)
($_f$**pathnamep** *foo*) ▷ `T` if *foo* is of indicated type.
($_f$**readtablep** *foo*)

($_f$**input-stream-p** *stream*)
($_f$**output-stream-p** *stream*)
($_f$**interactive-stream-p** *stream*)
($_f$**open-stream-p** *stream*)
▷ Return `T` if *stream* is for input, for output, interactive, or open, respectively.

($_f$**pathname-match-p** *path wildcard*)
▷ `T` if *path* matches *wildcard*.

($_f$**wild-pathname-p** *path* [{**:host**|**:device**|**:directory**|**:name**|**:type**|**:version**| `NIL`}])
▷ Return `T` if indicated component in *path* is wildcard. (`NIL` indicates any component.)

13.2 Reader

$\Big(\begin{Bmatrix}{}_f\textbf{y-or-n-p}\\ {}_f\textbf{yes-or-no-p}\end{Bmatrix}\ [\textit{control arg}^*]\Big)$

▷ Ask user a question and return $\underline{\texttt{T}}$ or $\underline{\texttt{NIL}}$ depending on their answer. See page 38, ${}_f$**format**, for *control* and *args*.

$({}_m\textbf{with-standard-io-syntax}\ \textit{form}^{P*})$

▷ Evaluate *forms* with standard behaviour of reader and printer. Return $\underline{\text{values of } \textit{forms}}$.

$\Big(\begin{Bmatrix}{}_f\textbf{read}\\ {}_f\textbf{read-preserving-whitespace}\end{Bmatrix}\ \big[\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-input*}}}\ [\textit{eof-err}_{\boxed{\texttt{T}}}$
$[\textit{eof-val}_{\boxed{\texttt{NIL}}}\ [\textit{recursive}_{\boxed{\texttt{NIL}}}]]]\big]\Big)$

▷ Read printed representation of $\underline{\text{object}}$.

$({}_f\textbf{read-from-string}\ \textit{string}\ \big[\textit{eof-error}_{\boxed{\texttt{T}}}\ [\textit{eof-val}_{\boxed{\texttt{NIL}}}$
$\left[\left\{\begin{array}{|l}\textbf{:start}\ \textit{start}_{\boxed{0}}\\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:preserve-whitespace}\ \textit{bool}_{\boxed{\texttt{NIL}}}\end{array}\right\}\right]]\big])$

▷ Return $\underline{\text{object}}$ read from string and zero-indexed $\underline{\text{position}}_2$ of next character.

$({}_f\textbf{read-delimited-list}\ \textit{char}\ \big[\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-input*}}}\ [\textit{recursive}_{\boxed{\texttt{NIL}}}]\big])$

▷ Continue reading until encountering *char*. Return $\underline{\text{list}}$ of objects read. Signal error if no *char* is found in stream.

$({}_f\textbf{read-char}\ \big[\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-input*}}}\ \big[\textit{eof-err}_{\boxed{\texttt{T}}}\ [\textit{eof-val}_{\boxed{\texttt{NIL}}}$
$[\textit{recursive}_{\boxed{\texttt{NIL}}}]]\big]\big])$

▷ Return $\underline{\text{next character}}$ from *stream*.

$({}_f\textbf{read-char-no-hang}\ \big[\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-input*}}}\ \big[\textit{eof-error}_{\boxed{\texttt{T}}}\ [\textit{eof-val}_{\boxed{\texttt{NIL}}}$
$[\textit{recursive}_{\boxed{\texttt{NIL}}}]]\big]\big])$

▷ $\underline{\text{Next character}}$ from *stream* or $\underline{\texttt{NIL}}$ if none is available.

$({}_f\textbf{peek-char}\ \big[\textit{mode}_{\boxed{\texttt{NIL}}}\ \big[\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-input*}}}\ \big[\textit{eof-error}_{\boxed{\texttt{T}}}\ [\textit{eof-val}_{\boxed{\texttt{NIL}}}$
$[\textit{recursive}_{\boxed{\texttt{NIL}}}]]\big]\big]\big])$

▷ Next, or if *mode* is `T`, next non-whitespace $\underline{\text{character}}$, or if *mode* is a character, $\underline{\text{next instance}}$ of it, from *stream* without removing it there.

$({}_f\textbf{unread-char}\ \textit{character}\ [\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-input*}}}])$

▷ Put last ${}_f$**read-char**ed *character* back into *stream*; return $\underline{\texttt{NIL}}$.

$({}_f\textbf{read-byte}\ \widetilde{\textit{stream}}\ \big[\textit{eof-err}_{\boxed{\texttt{T}}}\ [\textit{eof-val}_{\boxed{\texttt{NIL}}}]\big])$

▷ Read $\underline{\text{next byte}}$ from binary *stream*.

$({}_f\textbf{read-line}\ \big[\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-input*}}}\ \big[\textit{eof-err}_{\boxed{\texttt{T}}}\ [\textit{eof-val}_{\boxed{\texttt{NIL}}}$
$[\textit{recursive}_{\boxed{\texttt{NIL}}}]]\big]\big])$

▷ Return a $\underline{\text{line of text}}$ from *stream* and $\underline{\texttt{T}}_2$ if line has been ended by end of file.

$({}_f\textbf{read-sequence}\ \widetilde{\textit{sequence}}\ \widetilde{\textit{stream}}\ [\textbf{:start}\ \textit{start}_{\boxed{0}}][\textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}])$

▷ Replace elements of *sequence* between *start* and *end* with elements from binary or character *stream*. Return $\underline{\text{index}}$ of *sequence*'s first unmodified element.

$({}_f\textbf{readtable-case}\ \textit{readtable})_{\boxed{\textbf{:upcase}}}$

▷ $\underline{\text{Case sensitivity attribute}}$ (one of **:upcase**, **:downcase**, **:preserve**, **:invert**) of *readtable*. **setf**able.

$({}_f\textbf{copy-readtable}\ \big[\textit{from-readtable}_{\boxed{{}_v\textbf{*readtable*}}}\ [\widetilde{\textit{to-readtable}}_{\boxed{\texttt{NIL}}}]\big])$

▷ Return $\underline{\text{copy of } \textit{from-readtable}}$.

$({}_f\textbf{set-syntax-from-char}\ \textit{to-char}\ \textit{from-char}\ \big[\widetilde{\textit{to-readtable}}_{\boxed{{}_v\textbf{*readtable*}}}$
$[\textit{from-readtable}_{\boxed{\text{standard readtable}}}]\big])$

▷ Copy syntax of *from-char* to *to-readtable*. Return $\underline{\texttt{T}}$.

${}_v$***readtable*** ▷ Current readtable.

$_{v}\textbf{*read-base*}_{\boxed{10}}$ ▷ Radix for reading **integer**s and **ratio**s.

$_{v}\textbf{*read-default-float-format*}_{\boxed{\textsf{single-float}}}$
▷ Floating point format to use when not indicated in the number read.

$_{v}\textbf{*read-suppress*}_{\boxed{\texttt{NIL}}}$ ▷ If `T`, reader is syntactically more tolerant.

$({}_{f}\textbf{set-macro-character}\ \textit{char function}\ \big[\textit{non-term-p}_{\boxed{\texttt{NIL}}}\ [\widetilde{\textit{rt}}_{\boxed{_{v}\textbf{*readtable*}}}]\big])$
▷ Make *char* a macro character associated with *function* of stream and *char*. Return $\underline{\texttt{T}}$.

$({}_{f}\textbf{get-macro-character}\ \textit{char}\ [\textit{rt}_{\boxed{_{v}\textbf{*readtable*}}}])$
▷ $\underline{\text{Reader macro function}}$ associated with *char*, and $\underset{2}{\underline{\texttt{T}}}$ if *char* is a non-terminating macro character.

$({}_{f}\textbf{make-dispatch-macro-character}\ \textit{char}\ \big[\textit{non-term-p}_{\boxed{\texttt{NIL}}}\ [\textit{rt}_{\boxed{_{v}\textbf{*readtable*}}}]\big])$
▷ Make *char* a dispatching macro character. Return $\underline{\texttt{T}}$.

$({}_{f}\textbf{set-dispatch-macro-character}\ \textit{char sub-char function}\ [\widetilde{\textit{rt}}_{\boxed{_{v}\textbf{*readtable*}}}])$
▷ Make *function* of stream, *n*, *sub-char* a dispatch function of *char* followed by *n*, followed by *sub-char*. Return $\underline{\texttt{T}}$.

$({}_{f}\textbf{get-dispatch-macro-character}\ \textit{char sub-char}\ [\textit{rt}_{\boxed{_{v}\textbf{*readtable*}}}])$
▷ $\underline{\text{Dispatch function}}$ associated with *char* followed by *sub-char*.

13.3 Character Syntax

#| $\textit{multi-line-comment}^*$ **|#**
; $\textit{one-line-comment}^*$
▷ Comments. There are stylistic conventions:

;;;; *title* ▷ Short title for a block of code.

;;; *intro* ▷ Description before a block of code.

;; *state* ▷ State of program or of following code.

;*explanation*
; *continuation* ▷ Regarding line on which it appears.

$\textbf{(}\textit{foo}^*[\ \textbf{.}\ \textit{bar}_{\boxed{\texttt{NIL}}}]\textbf{)}$ ▷ List of *foo*s with the terminating cdr *bar*.

" ▷ Begin and end of a string.

'*foo* ▷ (${}_{s}$**quote** *foo*); *foo* unevaluated.

$\textbf{\`{}}([\textit{foo}]\ [\textbf{,}\textit{bar}]\ [\textbf{,@}\textit{baz}]\ [\textbf{,.}\widetilde{\textit{quux}}]\ [\textit{bing}])$
▷ Backquote. ${}_{s}$**quote** *foo* and *bing*; evaluate *bar* and splice the lists *baz* and *quux* into their elements. When nested, outermost commas inside the innermost backquote expression belong to this backquote.

#*c* ▷ (${}_{f}$**character** `"`*c*`"`), the character *c*.

#B*n*; **#O***n*; *n***.**; **#X***n*; **#***r***R***n*
▷ Integer of radix 2, 8, 10, 16, or *r*; $2 \le r \le 36$.

*n***/***d* ▷ The **ratio** $\frac{n}{d}$.

$\Big\{[m]\textbf{.}n\big[\{\textsf{S}|\textsf{F}|\textsf{D}|\textsf{L}|\textsf{E}\}x_{\boxed{\textsf{E0}}}\big]\Big|m\big[\textbf{.}[n]\big]\{\textsf{S}|\textsf{F}|\textsf{D}|\textsf{L}|\textsf{E}\}x\Big\}$
▷ $m.n \cdot 10^x$ as **short-float**, **single-float**, **double-float**, **long-float**, or the type from $\textbf{*read-default-float-format*}$.

#C(*a b***)** ▷ (${}_{f}$**complex** *a b*), the complex number $a + bi$.

#'*foo* ▷ (${}_{s}$**function** *foo*); the function named *foo*.

#*n***A***sequence* ▷ *n*-dimensional array.

$\textbf{\#}[n]\textbf{(}\textit{foo}^*\textbf{)}$
▷ Vector of some (or *n*) *foo*s filled with last *foo* if necessary.

$\#[n]{\ast}b^*$
▷ Bit vector of some (or n) bs filled with last b if necessary.

$\#\textbf{S(}type\ \{slot\ value\}^*\textbf{)}$ ▷ Structure of *type*.

$\#\textbf{P}string$ ▷ A pathname.

$\#\textbf{:}foo$ ▷ Uninterned symbol *foo*.

$\#\textbf{.}form$ ▷ Read-time value of *form*.

${}_v\textbf{*read-eval*}_{\boxed{\mathtt{T}}}$ ▷ If NIL, a **reader-error** is signalled at #..

$\#integer\textbf{=}\ foo$ ▷ Give *foo* the label *integer*.

$\#integer\#$ ▷ Object labelled *integer*.

$\#<$ ▷ Have the reader signal **reader-error**.

$\#{+}feature\ when\text{-}feature$
$\#{-}feature\ unless\text{-}feature$
▷ Means *when-feature* if *feature* is T; means *unless-feature* if *feature* is NIL. *feature* is a symbol from ${}_v\textbf{*features*}$, or ($\{\textbf{and}|\textbf{or}\}\ feature^*$), or (**not** *feature*).

${}_v\textbf{*features*}$
▷ List of symbols denoting implementation-dependent features.

$|c^*|;\ \backslash c$
▷ Treat arbitrary character(s) c as alphabetic preserving case.

13.4 Printer

$\left(\begin{Bmatrix}{}_f\textbf{prin1}\\ {}_f\textbf{print}\\ {}_f\textbf{pprint}\\ {}_f\textbf{princ}\end{Bmatrix} foo\ [\widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}]\right)$
▷ Print *foo* to *stream* ${}_f$**read**ably, ${}_f$**read**ably between a newline and a space, ${}_f$**read**ably after a newline, or human-readably without any extra characters, respectively. ${}_f$**prin1**, ${}_f$**print** and ${}_f$**princ** return $\underline{foo}$.

(${}_f$**prin1-to-string** *foo*)
(${}_f$**princ-to-string** *foo*)
▷ Print *foo* to $\underline{string}$ ${}_f$**read**ably or human-readably, respectively.

$({}_g\textbf{print-object}\ object\ \widetilde{stream})$
▷ Print $\underline{object}$ to *stream*. Called by the Lisp printer.

$({}_m\textbf{print-unreadable-object}\ (foo\ \widetilde{stream}\ \left\{\begin{vmatrix}\textbf{:type}\ bool_{\boxed{\mathtt{NIL}}}\\ \textbf{:identity}\ bool_{\boxed{\mathtt{NIL}}}\end{vmatrix}\right\})\ form^{\overset{\text{P}}{*}})$
▷ Enclosed in #< and >, print *foo* by means of *form*s to *stream*. Return $\underline{\mathtt{NIL}}$.

$({}_f\textbf{terpri}\ [\widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}])$
▷ Output a newline to *stream*. Return $\underline{\mathtt{NIL}}$.

$({}_f\textbf{fresh-line}\ [\widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}])$
▷ Output a newline to *stream* and return $\underline{\mathtt{T}}$ unless *stream* is already at the start of a line.

$({}_f\textbf{write-char}\ char\ [\widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}])$
▷ Output $\underline{char}$ to *stream*.

$\left(\begin{Bmatrix}{}_f\textbf{write-string}\\ {}_f\textbf{write-line}\end{Bmatrix} string\ [\widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}\ [\left\{\begin{vmatrix}\textbf{:start}\ start_{\boxed{0}}\\ \textbf{:end}\ end_{\boxed{\mathtt{NIL}}}\end{vmatrix}\right\}]]\right)$
▷ Write $\underline{string}$ to *stream* without/with a trailing newline.

$({}_f\textbf{write-byte}\ byte\ \widetilde{stream})$ ▷ Write $\underline{byte}$ to binary *stream*.

($_f$**write-sequence** *sequence* $\widetilde{stream}$ $\left\{\begin{array}{|l}\textbf{:start}\ start_{\boxed{0}}\\ \textbf{:end}\ end_{\boxed{\texttt{NIL}}}\end{array}\right\}$)

▷ Write elements of <u>*sequence*</u> to binary or character *stream*.

$$\left(\begin{Bmatrix}{}_f\textbf{write}\\ {}_f\textbf{write-to-string}\end{Bmatrix}\ foo\ \left\{\begin{array}{|l}\textbf{:array}\ bool\\ \textbf{:base}\ radix\\ \textbf{:case}\ \begin{cases}\textbf{:upcase}\\ \textbf{:downcase}\\ \textbf{:capitalize}\end{cases}\\ \textbf{:circle}\ bool\\ \textbf{:escape}\ bool\\ \textbf{:gensym}\ bool\\ \textbf{:length}\ \{int|\texttt{NIL}\}\\ \textbf{:level}\ \{int|\texttt{NIL}\}\\ \textbf{:lines}\ \{int|\texttt{NIL}\}\\ \textbf{:miser-width}\ \{int|\texttt{NIL}\}\\ \textbf{:pprint-dispatch}\ dispatch\text{-}table\\ \textbf{:pretty}\ bool\\ \textbf{:radix}\ bool\\ \textbf{:readably}\ bool\\ \textbf{:right-margin}\ \{int|\texttt{NIL}\}\\ \textbf{:stream}\ \widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}\end{array}\right\}\right)$$

▷ Print *foo* to *stream* and return <u>*foo*</u>, or print *foo* into <u>string</u>, respectively, after dynamically setting printer variables corresponding to keyword parameters (***print-***bar****** becoming **:***bar*). (**:stream** keyword with $_f$**write** only.)

($_f$**pprint-fill** $\widetilde{stream}$ *foo* $\left[parenthesis_{\boxed{\texttt{T}}}\ [noop]\right]$)
($_f$**pprint-tabular** $\widetilde{stream}$ *foo* $\left[parenthesis_{\boxed{\texttt{T}}}\ [noop\ [n_{\boxed{16}}]]\right]$)
($_f$**pprint-linear** $\widetilde{stream}$ *foo* $\left[parenthesis_{\boxed{\texttt{T}}}\ [noop]\right]$)

▷ Print *foo* to *stream*. If *foo* is a list, print as many elements per line as possible; do the same in a table with a column width of *n* ems; or print either all elements on one line or each on its own line, respectively. Return <u>`NIL`</u>. Usable with $_f$**format** directive ~//.

($_m$**pprint-logical-block** ($\widetilde{stream}$ *list* $\left\{\begin{array}{|l}\begin{Bmatrix}\textbf{:prefix}\ string\\ \textbf{:per-line-prefix}\ string\end{Bmatrix}\\ \textbf{:suffix}\ string_{\boxed{\texttt{""}}}\end{array}\right\}$)
(**declare** $\widehat{decl}$*)* $form^{\text{P}}$*)

▷ Evaluate *forms*, which should print *list*, with *stream* locally bound to a pretty printing stream which outputs to the original *stream*. If *list* is in fact not a list, it is printed by $_f$**write**. Return <u>`NIL`</u>.

($_m$**pprint-pop**)

▷ Take <u>next element</u> off *list*. If there is no remaining tail of *list*, or $_v$***print-length*** or $_v$***print-circle*** indicate printing should end, send element together with an appropriate indicator to *stream*.

($_f$**pprint-tab** $\begin{Bmatrix}\textbf{:line}\\ \textbf{:line-relative}\\ \textbf{:section}\\ \textbf{:section-relative}\end{Bmatrix}$ *c* *i* $[\widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}]$)

▷ Move cursor forward to column number $c + ki$, $k \geq 0$ being as small as possible.

($_f$**pprint-indent** $\begin{Bmatrix}\textbf{:block}\\ \textbf{:current}\end{Bmatrix}$ *n* $[\widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}]$)

▷ Specify indentation for innermost logical block relative to leftmost position/to current position. Return <u>`NIL`</u>.

($_m$**pprint-exit-if-list-exhausted**)

▷ If *list* is empty, terminate logical block. Return <u>`NIL`</u> otherwise.

($_f$**pprint-newline** $\begin{Bmatrix}\textbf{:linear}\\ \textbf{:fill}\\ \textbf{:miser}\\ \textbf{:mandatory}\end{Bmatrix}$ $[\widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}]$)

▷ Print a conditional newline if *stream* is a pretty printing stream. Return <u>`NIL`</u>.

$_v$***print-array*** ▷ If T, print arrays $_f$**read**ably.

$_v$***print-base***$\boxed{10}$ ▷ Radix for printing rationals, from 2 to 36.

$_v$***print-case***$\boxed{\text{:upcase}}$
▷ Print symbol names all uppercase (**:upcase**), all lowercase (**:downcase**), capitalized (**:capitalize**).

$_v$***print-circle***$\boxed{\text{NIL}}$
▷ If T, avoid indefinite recursion while printing circular structure.

$_v$***print-escape***$\boxed{\text{T}}$
▷ If NIL, do not print escape characters and package prefixes.

$_v$***print-gensym***$\boxed{\text{T}}$ ▷ If T, print **#:** before uninterned symbols.

$_v$***print-length***$\boxed{\text{NIL}}$
$_v$***print-level***$\boxed{\text{NIL}}$
$_v$***print-lines***$\boxed{\text{NIL}}$
▷ If integer, restrict printing of objects to that number of elements per level/to that depth/to that number of lines.

$_v$***print-miser-width***
▷ If integer and greater than the width available for printing a substructure, switch to the more compact miser style.

$_v$***print-pretty*** ▷ If T, print prettily.

$_v$***print-radix***$\boxed{\text{NIL}}$ ▷ If T, print rationals with a radix indicator.

$_v$***print-readably***$\boxed{\text{NIL}}$
▷ If T, print $_f$**read**ably or signal error **print-not-readable**.

$_v$***print-right-margin***$\boxed{\text{NIL}}$
▷ Right margin width in ems while pretty-printing.

($_f$**set-pprint-dispatch** *type function* [*priority*$\boxed{0}$ [*table*$\boxed{_v\text{*print-pprint-dispatch*}}$]])
▷ Install entry comprising *function* of arguments stream and object to print; and *priority* as *type* into *table*. If *function* is NIL, remove *type* from *table*. Return NIL.

($_f$**pprint-dispatch** *foo* [*table*$\boxed{_v\text{*print-pprint-dispatch*}}$])
▷ Return highest priority function associated with type of *foo* and $\underset{2}{\text{T}}$ if there was a matching type specifier in *table*.

($_f$**copy-pprint-dispatch** [*table*$\boxed{_v\text{*print-pprint-dispatch*}}$])
▷ Return copy of *table* or, if *table* is NIL, initial value of $_v$***print-pprint-dispatch***.

$_v$***print-pprint-dispatch*** ▷ Current pretty print dispatch table.

13.5 Format

($_m$**formatter** $\widehat{control}$)
▷ Return function of *stream* and *arg** applying $_f$**format** to *stream*, *control*, and *arg** returning NIL or any excess *args*.

($_f$**format** {T|NIL|*out-string*|*out-stream*} *control arg**)
▷ Output string *control* which may contain ~ directives possibly taking some *args*. Alternatively, *control* can be a function returned by $_m$**formatter** which is then applied to *out-stream* and *arg**. Output to *out-string*, *out-stream* or, if first argument is T, to $_v$***standard-output***. Return NIL. If first argument is NIL, return formatted output.

~ [*min-col*$\boxed{0}$] [,[*col-inc*$\boxed{1}$] [,[*min-pad*$\boxed{0}$] [,'*pad-char*$\boxed{␣}$]]]
[:] [**@**] {**A**|**S**}
▷ **Aesthetic/Standard.** Print argument of any type for consumption by humans/by the reader, respectively. With **:**, print NIL as () rather than nil; with **@**, add *pad-char*s on the left rather than on the right.

~ [*radix*$_{\boxed{10}}$] [,[*width*] [,['*pad-char*$_{\boxed{␣}}$] [,['*comma-char*$_{\boxed{,}}$] [,*comma-interval*$_{\boxed{3}}$]]]] [**:**] [**@**] **R**

▷ **Radix.** (With one or more prefix arguments.) Print argument as number; with **:**, group digits *comma-interval* each; with **@**, always prepend a sign.

{**~R**|**~:R**|**~@R**|**~@:R**}

▷ **Roman.** Take argument as number and print it as English cardinal number, as English ordinal number, as Roman numeral, or as old Roman numeral, respectively.

~ [*width*] [,['*pad-char*$_{\boxed{␣}}$] [,['*comma-char*$_{\boxed{,}}$] [,*comma-interval*$_{\boxed{3}}$]]] [**:**] [**@**] {**D**|**B**|**O**|**X**}

▷ **Decimal/Binary/Octal/Hexadecimal.** Print integer argument as number. With **:**, group digits *comma-interval* each; with **@**, always prepend a sign.

~ [*width*] [,[*dec-digits*] [,[*shift*$_{\boxed{0}}$] [,['*overflow-char*] [,'*pad-char*$_{\boxed{␣}}$]]]] [**@**] **F**

▷ **Fixed-Format Floating-Point.** With **@**, always prepend a sign.

~ [*width*] [,[*dec-digits*] [,[*exp-digits*] [,[*scale-factor*$_{\boxed{1}}$] [,['*overflow-char*] [,['*pad-char*$_{\boxed{␣}}$] [,'*exp-char*]]]]]] [**@**] {**E**|**G**}

▷ **Exponential/General Floating-Point.** Print argument as floating-point number with *dec-digits* after decimal point and *exp-digits* in the signed exponent. With **~G**, choose either **~E** or **~F**. With **@**, always prepend a sign.

~ [*dec-digits*$_{\boxed{2}}$] [,[*int-digits*$_{\boxed{1}}$] [,[*width*$_{\boxed{0}}$] [,'*pad-char*$_{\boxed{␣}}$]]] [**:**] [**@**] **$**

▷ **Monetary Floating-Point.** Print argument as fixed-format floating-point number. With **:**, put sign before any padding; with **@**, always prepend a sign.

{**~C**|**~:C**|**~@C**|**~@:C**}

▷ **Character.** Print, spell out, print in #\ syntax, or tell how to type, respectively, argument as (possibly non-printing) character.

{**~(** *text* **~)**|**~:(** *text* **~)**|**~@(** *text* **~)**|**~@:(** *text* **~)**}

▷ **Case-Conversion.** Convert *text* to lowercase, convert first letter of each word to uppercase, capitalize first word and convert the rest to lowercase, or convert to uppercase, respectively.

{**~P**|**~:P** |**~@P**|**~@:P**}

▷ **Plural.** If argument **eql** `1` print nothing, otherwise print `s`; do the same for the previous argument; if argument **eql** `1` print `y`, otherwise print `ies`; do the same for the previous argument, respectively.

~ [*n*$_{\boxed{1}}$] **%** ▷ **Newline.** Print *n* newlines.

~ [*n*$_{\boxed{1}}$] **&**

▷ **Fresh-Line.** Print $n - 1$ newlines if output stream is at the beginning of a line, or *n* newlines otherwise.

{**~_**|**~:_**|**~@_**|**~@:_**}

▷ **Conditional Newline.** Print a newline like **pprint-newline** with argument **:linear**, **:fill**, **:miser**, or **:mandatory**, respectively.

{**~:↩**|**~@↩**|**~↩**}

▷ **Ignored Newline.** Ignore newline, or whitespace following newline, or both, respectively.

~ [*n*$_{\boxed{1}}$] **|** ▷ **Page.** Print *n* page separators.

~ [*n*$_{\boxed{1}}$] **~** ▷ **Tilde.** Print *n* tildes.

~ [*min-col*$_{\boxed{0}}$] [,[*col-inc*$_{\boxed{1}}$] [,[*min-pad*$_{\boxed{0}}$] [,'*pad-char*$_{\boxed{␣}}$]]] [**:**] [**@**] **<** [*nl-text* **~**[*spare*$_{\boxed{0}}$ [,*width*]]**:;**] {*text* **~;**}* *text* **~>**

▷ **Justification.** Justify text produced by *text*s in a field of at least *min-col* columns. With **:**, right justify; with **@**, left justify. If this would leave less than *spare* characters on the current line, output *nl-text* first.

~ [**:**] [**@**] < {[*prefix*$_{\boxed{""}}$ **~;**]|[*per-line-prefix* **~@;**]} *body* [**~;** *suffix*$_{\boxed{""}}$] **~:** [**@**] >

▷ **Logical Block.** Act like **pprint-logical-block** using *body* as $_f$**format** control string on the elements of the list argument or, with **@**, on the remaining arguments, which are extracted by **pprint-pop**. With **:**, *prefix* and *suffix* default to (and). When closed by **~@:>**, spaces in *body* are replaced with conditional newlines.

{~ [$n_{\boxed{0}}$] **i**|~ [$n_{\boxed{0}}$] **:i**}

▷ **Indent.** Set indentation to n relative to leftmost/to current position.

~ [$c_{\boxed{1}}$] [,$i_{\boxed{1}}$] [**:**] [**@**] **T**

▷ **Tabulate.** Move cursor forward to column number $c+ki$, $k \geq 0$ being as small as possible. With **:**, calculate column numbers relative to the immediately enclosing section. With **@**, move to column number c_0+c+ki where c_0 is the current position.

{~ [$m_{\boxed{1}}$] $\ast$|~ [$m_{\boxed{1}}$] **:**$\ast$|~ [$n_{\boxed{0}}$] **@**$\ast$}

▷ **Go-To.** Jump m arguments forward, or backward, or to argument n.

~ [*limit*] [**:**] [**@**] **{** *text* **~}**

▷ **Iteration.** Use *text* repeatedly, up to *limit*, as control string for the elements of the list argument or (with **@**) for the remaining arguments. With **:** or **@:**, list elements or remaining arguments should be lists of which a new one is used at each iteration step.

~ [x [,y [,z]]] ^

▷ **Escape Upward.** Leave immediately **~< ~>**, **~< ~:>**, **~{ ~}**, **~?**, or the entire $_f$**format** operation. With one to three prefixes, act only if $x = 0$, $x = y$, or $x \leq y \leq z$, respectively.

~ [*i*] [**:**] [**@**] **[** [{*text* **~;**}* *text*] [**~:;** *default*] **~]**

▷ **Conditional Expression.** Use the zero-indexed argumenth (or *i*th if given) *text* as a $_f$**format** control subclause. With **:**, use the first *text* if the argument value is `NIL`, or the second *text* if it is `T`. With **@**, do nothing for an argument value of `NIL`. Use the only *text* and leave the argument to be read again if it is `T`.

{**~?**|**~@?**}

▷ **Recursive Processing.** Process two arguments as control string and argument list, or take one argument as control string and use then the rest of the original arguments.

~ [*prefix* {,*prefix*}*] [**:**] [**@**] **/**[*package* [:]:$\boxed{\texttt{cl-user:}}$]*function***/**

▷ **Call Function.** Call all-uppercase *package*::*function* with the arguments stream, format-argument, colon-p, at-sign-p and *prefix*es for printing format-argument.

~ [**:**] [**@**] **W**

▷ **Write.** Print argument of any type obeying every printer control variable. With **:**, pretty-print. With **@**, print without limits on length or depth.

{**V**|**#**}

▷ In place of the comma-separated prefix parameters: use next argument or number of remaining unprocessed arguments, respectively.

13.6 Streams

$$({}_f\textbf{open}\ \textit{path}\ \left\{\left|\begin{array}{l}
\textbf{:direction}\ \left\{\begin{array}{l}\textbf{:input}\\ \textbf{:output}\\ \textbf{:io}\\ \textbf{:probe}\end{array}\right\}_{\boxed{\textbf{:input}}}\\
\textbf{:element-type}\ \left\{\begin{array}{l}\textit{type}\\ \textbf{:default}\end{array}\right\}_{\boxed{\textbf{character}}}\\
\textbf{:if-exists}\ \left\{\begin{array}{l}\textbf{:new-version}\\ \textbf{:error}\\ \textbf{:rename}\\ \textbf{:rename-and-delete}\\ \textbf{:overwrite}\\ \textbf{:append}\\ \textbf{:supersede}\\ \texttt{NIL}\end{array}\right\}_{\boxed{\begin{array}{l}\textbf{:new-version}\text{ if }\textit{path}\\ \text{specifies }\textbf{:newest};\\ \texttt{NIL}\text{ otherwise}\end{array}}}\\
\textbf{:if-does-not-exist}\ \left\{\begin{array}{l}\textbf{:error}\\ \textbf{:create}\\ \texttt{NIL}\end{array}\right\}_{\boxed{\begin{array}{l}\texttt{NIL}\text{ for }\textbf{:direction :probe};\\ \{\textbf{:create}|\textbf{:error}\}\text{ otherwise}\end{array}}}\\
\textbf{:external-format}\ \textit{format}_{\boxed{\textbf{:default}}}
\end{array}\right.\right\})$$

▷ Open file-stream to *path*.

(*f* **make-concatenated-stream** *input-stream**)
(*f* **make-broadcast-stream** *output-stream**)
(*f* **make-two-way-stream** *input-stream-part output-stream-part*)
(*f* **make-echo-stream** *from-input-stream to-output-stream*)
(*f* **make-synonym-stream** *variable-bound-to-stream*)

▷ Return stream of indicated type.

(*f* **make-string-input-stream** *string* [*start*$_{\boxed{0}}$ [*end*$_{\boxed{\texttt{NIL}}}$]])

▷ Return a **string-stream** supplying the characters from *string*.

(*f* **make-string-output-stream** [**:element-type** *type*$_{\boxed{\textbf{character}}}$])

▷ Return a **string-stream** accepting characters (available via *f* **get-output-stream-string**).

(*f* **concatenated-stream-streams** *concatenated-stream*)
(*f* **broadcast-stream-streams** *broadcast-stream*)

▷ Return list of streams *concatenated-stream* still has to read from/*broadcast-stream* is broadcasting to.

(*f* **two-way-stream-input-stream** *two-way-stream*)
(*f* **two-way-stream-output-stream** *two-way-stream*)
(*f* **echo-stream-input-stream** *echo-stream*)
(*f* **echo-stream-output-stream** *echo-stream*)

▷ Return source stream or sink stream of *two-way-stream*/*echo-stream*, respectively.

(*f* **synonym-stream-symbol** *synonym-stream*)

▷ Return symbol of *synonym-stream*.

(*f* **get-output-stream-string** $\widetilde{\textit{string-stream}}$)

▷ Clear and return as a string characters on *string-stream*.

$$({}_f\textbf{file-position}\ \textit{stream}\ [\left\{\begin{array}{l}\textbf{:start}\\ \textbf{:end}\\ \textit{position}\end{array}\right\}])$$

▷ Return position within stream, or set it to *position* and return T on success.

(*f* **file-string-length** *stream foo*)

▷ Length *foo* would have in *stream*.

(*f* **listen** [*stream*$_{\boxed{{}_v\textbf{*standard-input*}}}$])

▷ T if there is a character in input *stream*.

(*f* **clear-input** [$\widetilde{\textit{stream}}$$_{\boxed{{}_v\textbf{*standard-input*}}}$])

▷ Clear input from *stream*, return NIL.

$$(\left\{\begin{array}{l}{}_f\textbf{clear-output}\\ {}_f\textbf{force-output}\\ {}_f\textbf{finish-output}\end{array}\right\}\ [\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-output*}}}])$$

▷ End output to *stream* and return NIL immediately, after initiating flushing of buffers, or after flushing of buffers, respectively.

(${}_f$**close** $\widetilde{\mathit{stream}}$ [**:abort** $\mathit{bool}_{\boxed{\texttt{NIL}}}$])

▷ Close *stream*. Return T if *stream* had been open. If **:abort** is T, delete associated file.

(${}_m$**with-open-file** (*stream path open-arg**) (**declare** $\widehat{\mathit{decl}^*}$)* $\mathit{form}^{\mathsf{P}_*}$)

▷ Use ${}_f$**open** with *open-arg*s to temporarily create *stream* to *path*; return values of *form*s.

(${}_m$**with-open-stream** (*foo* $\widetilde{\mathit{stream}}$) (**declare** $\widehat{\mathit{decl}^*}$)* $\mathit{form}^{\mathsf{P}_*}$)

▷ Evaluate *forms* with *foo* locally bound to *stream*. Return values of *form*s.

$$({}_m\textbf{with-input-from-string}\ (\mathit{foo}\ \mathit{string}\ \left\{\begin{array}{|l}\textbf{:index}\ \widetilde{\mathit{index}}\\ \textbf{:start}\ \mathit{start}_{\boxed{0}}\\ \textbf{:end}\ \mathit{end}_{\boxed{\texttt{NIL}}}\end{array}\right\})\ (\textbf{declare}\ \widehat{\mathit{decl}^*})^*\ \mathit{form}^{\mathsf{P}_*})$$

▷ Evaluate *forms* with *foo* locally bound to input **string-stream** from *string*. Return values of *forms*; store next reading position into *index*.

(${}_m$**with-output-to-string** (*foo* $\big[\widetilde{\mathit{string}_{\boxed{\texttt{NIL}}}}$ [**:element-type** $\mathit{type}_{\boxed{\textsf{character}}}$]$\big]$) (**declare** $\widehat{\mathit{decl}^*}$)* $\mathit{form}^{\mathsf{P}_*}$)

▷ Evaluate *forms* with *foo* locally bound to an output **string-stream**. Append output to *string* and return values of *forms* if *string* is given. Return string containing output otherwise.

(${}_f$**stream-external-format** *stream*)

▷ External file format designator.

${}_v$***terminal-io*** ▷ Bidirectional stream to user terminal.

${}_v$***standard-input***
${}_v$***standard-output***
${}_v$***error-output***

▷ Standard input stream, standard output stream, or standard error output stream, respectively.

${}_v$***debug-io***
${}_v$***query-io***

▷ Bidirectional streams for debugging and user interaction.

13.7 Pathnames and Files

$$({}_f\textbf{make-pathname}\ \left\{\begin{array}{|l}\textbf{:host}\ \{\mathit{host}|\texttt{NIL}|\textbf{:unspecific}\}\\ \textbf{:device}\ \{\mathit{device}|\texttt{NIL}|\textbf{:unspecific}\}\\ \textbf{:directory}\ \left\{\begin{array}{l}\{\mathit{directory}|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\}\\ (\left\{\begin{array}{l}\textbf{:absolute}\\ \textbf{:relative}\end{array}\right\}\left\{\begin{array}{l}\mathit{directory}\\ \textbf{:wild}\\ \textbf{:wild-inferiors}\\ \textbf{:up}\\ \textbf{:back}\end{array}\right\}^*)\end{array}\right.\\ \textbf{:name}\ \{\mathit{file\text{-}name}|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\}\\ \textbf{:type}\ \{\mathit{file\text{-}type}|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\}\\ \textbf{:version}\ \{\textbf{:newest}|\mathit{version}|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\}\\ \textbf{:defaults}\ \mathit{path}_{\boxed{\text{host from }{}_v\textbf{*default-pathname-defaults*}}}\\ \textbf{:case}\ \{\textbf{:local}|\textbf{:common}\}_{\boxed{\textbf{:local}}}\end{array}\right\})$$

▷ Construct a logical pathname if there is a logical pathname translation for *host*, otherwise construct a physical pathname. For **:case :local**, leave case of components unchanged. For **:case :common**, leave mixed-case components unchanged; convert all-uppercase components into local customary case; do the opposite with all-lowercase components.

$$(\left\{\begin{array}{l}{}_f\textbf{pathname-host}\\ {}_f\textbf{pathname-device}\\ {}_f\textbf{pathname-directory}\\ {}_f\textbf{pathname-name}\\ {}_f\textbf{pathname-type}\end{array}\right\}\ \mathit{path\text{-}or\text{-}stream}\ [\textbf{:case}\ \left\{\begin{array}{l}\textbf{:local}\\ \textbf{:common}\end{array}\right\}_{\boxed{\textbf{:local}}}])$$

(${}_f$**pathname-version** *path-or-stream*)

▷ Return pathname component.

($_f$**parse-namestring** *foo* [*host* [*default-pathname*$\boxed{_v\textbf{*default-pathname-defaults*}}$
$\left\{\begin{array}{|l} \textbf{:start } \textit{start}_{\boxed{0}} \\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}} \\ \textbf{:junk-allowed } \textit{bool}_{\boxed{\texttt{NIL}}} \end{array}\right\}$]])
▷ Return pathname converted from string, pathname, or stream *foo*; and $\underline{\text{position}}_2$ where parsing stopped.

($_f$**merge-pathnames** *path-or-stream*
[*default-path-or-stream*$\boxed{_v\textbf{*default-pathname-defaults*}}$
[*default-version*$\boxed{\textbf{:newest}}$]])
▷ Return pathname made by filling in components missing in *path-or-stream* from *default-path-or-stream*.

$_v$***default-pathname-defaults***
▷ Pathname to use if one is needed and none supplied.

($_f$**user-homedir-pathname** [*host*]) ▷ User's home directory.

($_f$**enough-namestring** *path-or-stream*
[*root-path*$\boxed{_v\textbf{*default-pathname-defaults*}}$])
▷ Return minimal path string that sufficiently describes the path of *path-or-stream* relative to *root-path.*

($_f$**namestring** *path-or-stream*)
($_f$**file-namestring** *path-or-stream*)
($_f$**directory-namestring** *path-or-stream*)
($_f$**host-namestring** *path-or-stream*)
▷ Return string representing full pathname; name, type, and version; directory name; or host name, respectively, of *path-or-stream*.

($_f$**translate-pathname** *path-or-stream wildcard-path-a wildcard-path-b*)
▷ Translate the path of *path-or-stream* from *wildcard-path-a* into *wildcard-path-b*. Return new path.

($_f$**pathname** *path-or-stream*) ▷ Pathname of *path-or-stream*.

($_f$**logical-pathname** *logical-path-or-stream*)
▷ Logical pathname of *logical-path-or-stream*. Logical pathnames are represented as all-uppercase
$\texttt{"}[\textit{host}\texttt{:}][\texttt{;}]\{\left\{\begin{array}{l}\{\textit{dir}|\texttt{*}\}^+\\ \texttt{**}\end{array}\right\}\texttt{;}\}^*\{\textit{name}|\texttt{*}\}^*[\texttt{.}\left\{\begin{array}{l}\{\textit{type}|\texttt{*}\}^+\\ \texttt{LISP}\end{array}\right\}[\texttt{.}\{\textit{version}|\texttt{*}$
$|\texttt{newest}|\texttt{NEWEST}\}]]\texttt{"}$.

($_f$**logical-pathname-translations** *logical-host*)
▷ List of (*from-wildcard to-wildcard*) translations for *logical-host*. **setf**able.

($_f$**load-logical-pathname-translations** *logical-host*)
▷ Load *logical-host*'s translations. Return `NIL` if already loaded; return `T` if successful.

($_f$**translate-logical-pathname** *path-or-stream*)
▷ Physical pathname corresponding to (possibly logical) pathname of *path-or-stream*.

($_f$**probe-file** *file*)
($_f$**truename** *file*)
▷ Canonical name of *file*. If *file* does not exist, return `NIL`/signal **file-error**, respectively.

($_f$**file-write-date** *file*) ▷ Time at which *file* was last written.

($_f$**file-author** *file*) ▷ Return name of *file* owner.

($_f$**file-length** *stream*) ▷ Return length of *stream*.

($_f$**rename-file** *foo bar*)
▷ Rename file *foo* to *bar*. Unspecified components of path *bar* default to those of *foo*. Return new pathname, $\underline{\text{old physical file name}}_2$, and $\underline{\text{new physical file name}}_3$.

($_f$**delete-file** *file*) ▷ Delete *file*. Return `T`.

($_f$**directory** *path*) ▷ List of pathnames matching *path*.

($_f$**ensure-directories-exist** *path* [**:verbose** *bool*])
▷ Create parts of *path* if necessary. Second return value is $\underset{2}{\underline{\texttt{T}}}$ if something has been created.

14 Packages and Symbols

The Loop Facility provides additional means of symbol handling; see **loop**, page 22.

14.1 Predicates

($_f$**symbolp** *foo*)
($_f$**packagep** *foo*) ▷ T if *foo* is of indicated type.
($_f$**keywordp** *foo*)

14.2 Packages

:*bar*|**keyword:***bar* ▷ Keyword, evaluates to *:bar*.

*package***:***symbol* ▷ Exported *symbol* of *package*.

*package***::***symbol* ▷ Possibly unexported *symbol* of *package*.

($_m$**defpackage** *foo* {|(**:nicknames** *nick**)*
|(**:documentation** *string*)
|(**:intern** *interned-symbol**)*
|(**:use** *used-package**)*
|(**:import-from** *pkg imported-symbol**)*
|(**:shadowing-import-from** *pkg shd-symbol**)*
|(**:shadow** *shd-symbol**)*
|(**:export** *exported-symbol**)*
|(**:size** *int*)})
▷ Create or modify package *foo* with *interned-symbols*, symbols from *used-packages*, *imported-symbols*, and *shd-symbols*. Add *shd-symbols* to *foo*'s shadowing list.

($_f$**make-package** *foo* {|**:nicknames** (*nick**)$_{\boxed{\texttt{NIL}}}$
|**:use** (*used-package**)})
▷ Create package *foo*.

($_f$**rename-package** *package new-name* [*new-nicknames*$_{\boxed{\texttt{NIL}}}$])
▷ Rename *package*. Return renamed package.

($_m$**in-package** $\widehat{foo}$) ▷ Make package *foo* current.

({$_f$**use-package**
$_f$**unuse-package**} *other-packages* [*package*$_{\boxed{_v\textbf{*package*}}}$])
▷ Make exported symbols of *other-packages* available in *package*, or remove them from *package*, respectively. Return T.

($_f$**package-use-list** *package*)
($_f$**package-used-by-list** *package*)
▷ List of other packages used by/using *package*.

($_f$**delete-package** $\widetilde{package}$)
▷ Delete *package*. Return T if successful.

$_v$***package***$_{\boxed{\textsf{common-lisp-user}}}$ ▷ The current package.

($_f$**list-all-packages**) ▷ List of registered packages.

($_f$**package-name** *package*) ▷ Name of *package*.

($_f$**package-nicknames** *package*) ▷ Nicknames of *package*.

(${}_f$**find-package** *name*) ▷ $\underline{\text{Package}}$ with *name* (case-sensitive).

(${}_f$**find-all-symbols** *foo*)
▷ $\underline{\text{List of symbols}}$ *foo* from all registered packages.

$\left(\left\{\begin{array}{l}{}_f\textbf{intern}\\ {}_f\textbf{find-symbol}\end{array}\right\} foo\ [package_{\boxed{{}_v\textbf{*package*}}}]\right)$
▷ Intern or find, respectively, symbol $\underline{foo}$ in *package*. Second return value is one of $\underset{2}{\underline{\textbf{:internal}}}$, $\underset{2}{\underline{\textbf{:external}}}$, or $\underset{2}{\underline{\textbf{:inherited}}}$ (or $\underset{2}{\underline{\texttt{NIL}}}$ if ${}_f$**intern** has created a fresh symbol).

(${}_f$**unintern** *symbol* $[package_{\boxed{{}_v\textbf{*package*}}}]$)
▷ Remove *symbol* from *package*, return $\underline{\texttt{T}}$ on success.

$\left(\left\{\begin{array}{l}{}_f\textbf{import}\\ {}_f\textbf{shadowing-import}\end{array}\right\} symbols\ [package_{\boxed{{}_v\textbf{*package*}}}]\right)$
▷ Make *symbols* internal to *package*. Return $\underline{\texttt{T}}$. In case of a name conflict signal correctable **package-error** or shadow the old symbol, respectively.

(${}_f$**shadow** *symbols* $[package_{\boxed{{}_v\textbf{*package*}}}]$)
▷ Make *symbols* of *package* shadow any otherwise accessible, equally named symbols from other packages. Return $\underline{\texttt{T}}$.

(${}_f$**package-shadowing-symbols** *package*)
▷ $\underline{\text{List of symbols}}$ of *package* that shadow any otherwise accessible, equally named symbols from other packages.

(${}_f$**export** *symbols* $[package_{\boxed{{}_v\textbf{*package*}}}]$)
▷ Make *symbols* external to *package*. Return $\underline{\texttt{T}}$.

(${}_f$**unexport** *symbols* $[package_{\boxed{{}_v\textbf{*package*}}}]$)
▷ Revert *symbols* to internal status. Return $\underline{\texttt{T}}$.

$\left(\left\{\begin{array}{l}{}_m\textbf{do-symbols}\\ {}_m\textbf{do-external-symbols}\\ {}_m\textbf{do-all-symbols}\ (var\ [result_{\boxed{\texttt{NIL}}}])\end{array}\right.\left.\begin{array}{l}\left.\begin{array}{l}\\ \\ \end{array}\right\}(\widehat{var}\ [package_{\boxed{{}_v\textbf{*package*}}}\ [result_{\boxed{\texttt{NIL}}}]])\\ \\ \end{array}\right\}\right.$
$(\textbf{declare}\ \widehat{decl}^*)^*\ \left\{\begin{array}{|l}\widehat{tag}\\ form\end{array}\right\}^*)$
▷ Evaluate ${}_s$**tagbody**-like body with *var* successively bound to every symbol from *package*, to every external symbol from *package*, or to every symbol from all registered packages, respectively. Return $\underline{\text{values of } result}$. Implicitly, the whole form is a ${}_s$**block** named `NIL`.

(${}_m$**with-package-iterator** (*foo packages* [**:internal**|**:external**|**:inherited**])
$(\textbf{declare}\ \widehat{decl}^*)^*\ form^{P_*})$
▷ Return $\underline{\text{values of } forms}$. In *forms*, successive invocations of (*foo*) return: `T` if a symbol is returned; a symbol from *packages*; accessibility (**:internal**, **:external**, or **:inherited**); and the package the symbol belongs to.

(${}_f$**require** *module* $[paths_{\boxed{\texttt{NIL}}}]$)
▷ If not in ${}_v$***modules***, try *paths* to load *module* from. Signal **error** if unsuccessful. Deprecated.

(${}_f$**provide** *module*)
▷ If not already there, add *module* to ${}_v$***modules***. Deprecated.

${}_v$***modules*** ▷ List of names of loaded modules.

14.3 Symbols

A **symbol** has the attributes *name*, home **package**, property list, and optionally value (of global constant or variable *name*) and function (**function**, macro, or special operator *name*).

(${}_f$**make-symbol** *name*)
▷ Make fresh, uninterned $\underline{\text{symbol } name}$.

($_f$**gensym** [*s*$_{\boxed{\text{G}}}$])

▷ Return fresh, uninterned symbol **#:***sn* with *n* from $_v$***gensym-counter***. Increment $_v$***gensym-counter***.

($_f$**gentemp** [*prefix*$_{\boxed{\texttt{T}}}$ [*package*$_{\boxed{_v\textbf{*package*}}}$]])

▷ Intern fresh symbol in package. Deprecated.

($_f$**copy-symbol** *symbol* [*props*$_{\boxed{\texttt{NIL}}}$])

▷ Return uninterned copy of *symbol*. If *props* is `T`, give copy the same value, function and property list.

($_f$**symbol-name** *symbol*)
($_f$**symbol-package** *symbol*)

▷ Name or package, respectively, of *symbol*.

($_f$**symbol-plist** *symbol*)
($_f$**symbol-value** *symbol*)
($_f$**symbol-function** *symbol*)

▷ Property list, value, or function, respectively, of *symbol*. **setf**able.

$$\left(\begin{Bmatrix}{}_g\textbf{documentation}\\(\textbf{setf }{}_g\textbf{documentation})\ \textit{new-doc}\end{Bmatrix}\ \textit{foo}\ \begin{Bmatrix}\textbf{'variable}\,|\,\textbf{'function}\\ \textbf{'compiler-macro}\\ \textbf{'method-combination}\\ \textbf{'structure}\,|\,\textbf{'type}\,|\,\textbf{'setf}\,|\,\texttt{T}\end{Bmatrix}\right)$$

▷ Get/set documentation string of *foo* of given type.

$_c$**t**

▷ Truth; the supertype of every type including **t**; the superclass of every class except **t**; $_v$***terminal-io***.

$_c$**nil**|$_c$**()**

▷ Falsity; the empty list; the empty type, subtype of every type; $_v$***standard-input***; $_v$***standard-output***; the global environment.

14.4 Standard Packages

common-lisp|**cl**

▷ Exports the defined names of Common Lisp except for those in the **keyword** package.

common-lisp-user|**cl-user**

▷ Current package after startup; uses package **common-lisp**.

keyword

▷ Contains symbols which are defined to be of type **keyword**.

15 Compiler

15.1 Predicates

($_f$**special-operator-p** *foo*) ▷ `T` if *foo* is a special operator.

($_f$**compiled-function-p** *foo*) ▷ `T` if *foo* is of type **compiled-function**.

15.2 Compilation

$$\left(_f\textbf{compile}\ \begin{Bmatrix}\texttt{NIL}\ \textit{definition}\\ \begin{Bmatrix}\textit{name}\\ (\textbf{setf}\ \textit{name})\end{Bmatrix}\ [\textit{definition}]\end{Bmatrix}\right)$$

▷ Return compiled function or replace *name*'s function definition with the compiled function. Return $\underset{2}{\texttt{T}}$ in case of **warning**s or **error**s, and $\underset{3}{\texttt{T}}$ in case of **warning**s or **error**s excluding **style-warning**s.

$$({}_{f}\textbf{compile-file}\ \textit{file}\ \left\{\begin{array}{|l}\textbf{:output-file}\ \textit{out-path}\\ \textbf{:verbose}\ \textit{bool}_{\boxed{{}_{v}\textbf{*compile-verbose*}}}\\ \textbf{:print}\ \textit{bool}_{\boxed{{}_{v}\textbf{*compile-print*}}}\\ \textbf{:external-format}\ \textit{file-format}_{\boxed{\textbf{:default}}}\end{array}\right\})$$

▷ Write compiled contents of *file* to *out-path*. Return $\underline{\text{true output path}}$ or $\underline{\texttt{NIL}}$, $\underset{2}{\underline{\texttt{T}}}$ in case of **warning**s or **error**s, $\underset{3}{\underline{\texttt{T}}}$ in case of **warning**s or **error**s excluding **style-warning**s.

(${}_f$**compile-file-pathname** *file* [**:output-file** *path*] [*other-keyargs*])

▷ $\underline{\text{Pathname}}$ ${}_f$**compile-file** writes to if invoked with the same arguments.

$$({}_{f}\textbf{load}\ \textit{path}\ \left\{\begin{array}{|l}\textbf{:verbose}\ \textit{bool}_{\boxed{{}_{v}\textbf{*load-verbose*}}}\\ \textbf{:print}\ \textit{bool}_{\boxed{{}_{v}\textbf{*load-print*}}}\\ \textbf{:if-does-not-exist}\ \textit{bool}_{\boxed{\texttt{T}}}\\ \textbf{:external-format}\ \textit{file-format}_{\boxed{\textbf{:default}}}\end{array}\right\})$$

▷ Load source file or compiled file into Lisp environment. Return $\underline{\texttt{T}}$ if successful.

$$\left.\begin{array}{l}{}_{v}\textbf{*compile-file}\\ {}_{v}\textbf{*load}\end{array}\right\}\text{-}\left\{\begin{array}{l}\textbf{pathname*}_{\boxed{\texttt{NIL}}}\\ \textbf{truename*}_{\boxed{\texttt{NIL}}}\end{array}\right.$$

▷ Input file used by ${}_f$**compile-file**/by ${}_f$**load**.

$$\left.\begin{array}{l}{}_{v}\textbf{*compile}\\ {}_{v}\textbf{*load}\end{array}\right\}\text{-}\left\{\begin{array}{l}\textbf{print*}\\ \textbf{verbose*}\end{array}\right.$$

▷ Defaults used by ${}_f$**compile-file**/by ${}_f$**load**.

$$({}_{s}\textbf{eval-when}\ (\left\{\begin{array}{|l}\{\textbf{:compile-toplevel}|\textbf{compile}\}\\ \{\textbf{:load-toplevel}|\textbf{load}\}\\ \{\textbf{:execute}|\textbf{eval}\}\end{array}\right\})\ \textit{form}^{P_*})$$

▷ Return $\underline{\text{values of } \textit{forms}}$ if ${}_s$**eval-when** is in the top-level of a file being compiled, in the top-level of a compiled file being loaded, or anywhere, respectively. Return $\underline{\texttt{NIL}}$ if *forms* are not evaluated. (**compile**, **load** and **eval** deprecated.)

(${}_s$**locally** (**declare** $\widehat{\textit{decl}}^*$)* $\textit{form}^{P_*}$)

▷ Evaluate *forms* in a lexical environment with declarations *decl* in effect. Return $\underline{\text{values of } \textit{forms}}$.

(${}_m$**with-compilation-unit** ([**:override** $\textit{bool}_{\boxed{\texttt{NIL}}}$]) $\textit{form}^{P_*}$)

▷ Return $\underline{\text{values of } \textit{forms}}$. Warnings deferred by the compiler until end of compilation are deferred until the end of evaluation of *forms*.

(${}_s$**load-time-value** *form* [$\widehat{\textit{read-only}}_{\boxed{\texttt{NIL}}}$])

▷ Evaluate *form* at compile time and treat $\underline{\text{its value}}$ as literal at run time.

(${}_s$**quote** $\widehat{\textit{foo}}$) ▷ Return $\underline{\text{unevaluated } \textit{foo}}$.

(${}_g$**make-load-form** *foo* [*environment*])

▷ Its methods are to return a $\underline{\text{creation form}}$ which on evaluation at ${}_f$**load** time returns an object equivalent to *foo*, and an optional $\underset{2}{\underline{\text{initialization form}}}$ which on evaluation performs some initialization of the object.

$$({}_{f}\textbf{make-load-form-saving-slots}\ \textit{foo}\ \left\{\begin{array}{|l}\textbf{:slot-names}\ \textit{slots}_{\boxed{\text{all local slots}}}\\ \textbf{:environment}\ \textit{environment}\end{array}\right\})$$

▷ Return a $\underline{\text{creation form}}$ and an $\underset{2}{\underline{\text{initialization form}}}$ which on evaluation construct an object equivalent to *foo* with *slots* initialized with the corresponding values from *foo*.

(${}_f$**macro-function** *symbol* [*environment*])

$$({}_{f}\textbf{compiler-macro-function}\ \left\{\begin{array}{l}\textit{name}\\ (\textbf{setf}\ \textit{name})\end{array}\right\}\ [\textit{environment}])$$

▷ Return specified $\underline{\text{macro function}}$, or $\underline{\text{compiler macro function}}$, respectively, if any. Return $\underline{\texttt{NIL}}$ otherwise. **setf**able.

(${}_f$**eval** *arg*)

▷ Return $\underline{\text{values of value of } \textit{arg}}$ evaluated in global environment.

15.3 REPL and Debugging

$_v$**+**|$_v$**++**|$_v$**+++**
$_v$***** | $_v$****** | $_v$*******
$_v$**/** | $_v$**//** | $_v$**///**

▷ Last, penultimate, or antepenultimate form evaluated in the REPL, or their respective primary value, or a list of their respective values.

$_v$**−** ▷ Form currently being evaluated by the REPL.

($_f$**apropos** *string* [*package*$_{\boxed{\texttt{NIL}}}$])
▷ Print interned symbols containing *string*.

($_f$**apropos-list** *string* [*package*$_{\boxed{\texttt{NIL}}}$])
▷ List of interned symbols containing *string*.

($_f$**dribble** [*path*])
▷ Save a record of interactive session to file at *path*. Without *path*, close that file.

($_f$**ed** [*file-or-function*$_{\boxed{\texttt{NIL}}}$]) ▷ Invoke editor if possible.

($\begin{Bmatrix} {}_f\textbf{macroexpand-1} \\ {}_f\textbf{macroexpand} \end{Bmatrix}$ *form* [*environment*$_{\boxed{\texttt{NIL}}}$])
▷ Return macro expansion, once or entirely, respectively, of *form* and $\underset{2}{\texttt{T}}$ if *form* was a macro form. Return *form* and $\underset{2}{\texttt{NIL}}$ otherwise.

$_v$***macroexpand-hook***
▷ Function of arguments expansion function, macro form, and environment called by $_f$**macroexpand-1** to generate macro expansions.

($_m$**trace** $\begin{Bmatrix} \textit{function} \\ (\textbf{setf}\ \textit{function}) \end{Bmatrix}^{*}$)
▷ Cause *function*s to be traced. With no arguments, return list of traced functions.

($_m$**untrace** $\begin{Bmatrix} \textit{function} \\ (\textbf{setf}\ \textit{function}) \end{Bmatrix}^{*}$)
▷ Stop *function*s, or each currently traced function, from being traced.

$_v$***trace-output***
▷ Output stream $_m$**trace** and $_m$**time** send their output to.

($_m$**step** *form*)
▷ Step through evaluation of *form*. Return values of *form*.

($_f$**break** [*control arg**])
▷ Jump directly into debugger; return NIL. See page 38, $_f$**format**, for *control* and *arg*s.

($_m$**time** *form*)
▷ Evaluate *form*s and print timing information to $_v$***trace-output***. Return values of *form*.

($_f$**inspect** *foo*) ▷ Interactively give information about *foo*.

($_f$**describe** *foo* [$\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-output*}}}$])
▷ Send information about *foo* to *stream*.

($_g$**describe-object** *foo* [$\widetilde{\textit{stream}}$])
▷ Send information about *foo* to *stream*. Called by $_f$**describe**.

($_f$**disassemble** *function*)
▷ Send disassembled representation of *function* to $_v$***standard-output***. Return NIL.

($_f$**room** [$\{\texttt{NIL}|\textbf{:default}|\texttt{T}\}_{\boxed{\textbf{:default}}}$])
▷ Print information about internal storage management to ***standard-output***.

15.4 Declarations

(${}_f$**proclaim** *decl*)
(${}_m$**declaim** $\widehat{decl*}$)

▷ Globally make declaration(s) *decl*. *decl* can be: **declaration**, **type**, **ftype**, **inline**, **notinline**, **optimize**, or **special**. See below.

(**declare** $\widehat{decl*}$)

▷ Inside certain forms, locally make declarations *decl**. *decl* can be: **dynamic-extent**, **type**, **ftype**, **ignorable**, **ignore**, **inline**, **notinline**, **optimize**, or **special**. See below.

(**declaration** foo*) ▷ Make *foo*s names of declarations.

(**dynamic-extent** *variable** (**function** *function*)*)

▷ Declare lifetime of *variable*s and/or *function*s to end when control leaves enclosing block.

([**type**] *type variable**)
(**ftype** *type function**)

▷ Declare *variable*s or *function*s to be of *type*.

$$\left(\begin{Bmatrix}\textbf{ignorable}\\ \textbf{ignore}\end{Bmatrix} \begin{Bmatrix} var \\ (\textbf{function}\ function)\end{Bmatrix}^{*}\right)$$

▷ Suppress warnings about used/unused bindings.

(**inline** *function**)
(**notinline** *function**)

▷ Tell compiler to integrate/not to integrate, respectively, called *function*s into the calling routine.

$$\left(\textbf{optimize} \begin{Bmatrix} \textbf{compilation-speed} \mid (\textbf{compilation-speed}\ n_{\boxed{3}}) \\ \textbf{debug} \mid (\textbf{debug}\ n_{\boxed{3}}) \\ \textbf{safety} \mid (\textbf{safety}\ n_{\boxed{3}}) \\ \textbf{space} \mid (\textbf{space}\ n_{\boxed{3}}) \\ \textbf{speed} \mid (\textbf{speed}\ n_{\boxed{3}}) \end{Bmatrix}\right)$$

▷ Tell compiler how to optimize. $n = 0$ means unimportant, $n = 1$ is neutral, $n = 3$ means important.

(**special** *var**) ▷ Declare *var*s to be dynamic.

16 External Environment

(${}_f$**get-internal-real-time**)
(${}_f$**get-internal-run-time**)

▷ $\underline{\text{Current time}}$, or $\underline{\text{computing time}}$, respectively, in clock ticks.

${}_c$**internal-time-units-per-second**

▷ Number of clock ticks per second.

(${}_f$**encode-universal-time** *sec min hour date month year* [$zone_{\boxed{\text{curr}}}$])
(${}_f$**get-universal-time**)

▷ $\underline{\text{Seconds from 1900-01-01, 00:00}}$, ignoring leap seconds.

(${}_f$**decode-universal-time** *universal-time* [$time\text{-}zone_{\boxed{\text{current}}}$])
(${}_f$**get-decoded-time**)

▷ Return $\underline{\text{second}}$, $\underset{2}{\underline{\text{minute}}}$, $\underset{3}{\underline{\text{hour}}}$, $\underset{4}{\underline{\text{date}}}$, $\underset{5}{\underline{\text{month}}}$, $\underset{6}{\underline{\text{year}}}$, $\underset{7}{\underline{\text{day}}}$, $\underset{8}{\underline{\text{daylight-p}}}$, and $\underset{9}{\underline{\text{zone}}}$.

(${}_f$**short-site-name**)
(${}_f$**long-site-name**)

▷ $\underline{\text{String}}$ representing physical location of computer.

$$\left(\begin{Bmatrix} {}_f\textbf{lisp-implementation} \\ {}_f\textbf{software} \\ {}_f\textbf{machine} \end{Bmatrix}\text{-}\begin{Bmatrix}\textbf{type}\\ \textbf{version}\end{Bmatrix}\right)$$

▷ $\underline{\text{Name}}$ or $\underline{\text{version}}$ of implementation, operating system, or hardware, respectively.

(${}_f$**machine-instance**) ▷ $\underline{\text{Computer name}}$.

Index

" 35
' 35
(35
() 46
) 35
* 3, 32, 33, 43, 48
** 43, 48
*** 48
BREAK-ON-SIGNALS 31
COMPILE-FILE-PATHNAME 47
COMPILE-FILE-TRUENAME 47
COMPILE-PRINT 47
COMPILE-VERBOSE 47
DEBUG-IO 42
DEBUGGER-HOOK 31
DEFAULT-PATHNAME-DEFAULTS 43
ERROR-OUTPUT 42
FEATURES 36
GENSYM-COUNTER 46
LOAD-PATHNAME 47
LOAD-PRINT 47
LOAD-TRUENAME 47
LOAD-VERBOSE 47
MACROEXPAND-HOOK 48
MODULES 45
PACKAGE 44
PRINT-ARRAY 38
PRINT-BASE 38
PRINT-CASE 38
PRINT-CIRCLE 38
PRINT-ESCAPE 38
PRINT-GENSYM 38
PRINT-LENGTH 38
PRINT-LEVEL 38
PRINT-LINES 38
PRINT-MISER-WIDTH 38
PRINT-PPRINT-DISPATCH 38
PRINT-PRETTY 38
PRINT-RADIX 38
PRINT-READABLY 38
PRINT-RIGHT-MARGIN 38
QUERY-IO 42
RANDOM-STATE 4
READ-BASE 35
READ-DEFAULT-FLOAT-FORMAT 35
READ-EVAL 36
READ-SUPPRESS 35
READTABLE 34
STANDARD-INPUT 42
STANDARD-OUTPUT 42
TERMINAL-IO 42
TRACE-OUTPUT 48
+ 3, 28, 48
++ 48
+++ 48
, 35
,. 35
,@ 35
− 3, 48
. 35
/ 3, 35, 48
// 48
/// 48
/= 3
: 44
:: 44
:ALLOW-OTHER-KEYS 21
; 35
< 3
<= 3
= 3, 22, 24
> 3
>= 3
\ 36
40
#\ 35
#' 35
#(35
#* 36
#+ 36
#− 36
#. 36
#: 36
#< 36
#= 36
#A 35
#B 35
#C(35
#O 35
#P 36
#R 35
#S(36
#X 35
36
#| |# 35
&ALLOW-OTHER-KEYS 21
&AUX 21
&BODY 20
&ENVIRONMENT 21
&KEY 20
&OPTIONAL 20
&REST 20
&WHOLE 20
∼(∼) 39
∼* 40
∼/ / 40
∼< ∼:> 40
∼< ∼> 39
∼? 40
∼A 38
∼B 39
∼C 39
∼D 39
∼E 39
∼F 39
∼G 39
∼I 40
∼O 39
∼P 39
∼R 39
∼S 38
∼T 40
∼W 40
∼X 39
∼[∼] 40
∼$ 39
∼% 39
∼& 39
∼^ 40
∼_ 39
∼| 39
∼{ ∼} 40
∼∼ 39
∼↩ 39
` 35
| | 36
1+ 3
1− 3

ABORT 30
ABOVE 24
ABS 4
ACONS 10
ACOS 3
ACOSH 4
ACROSS 24
ADD-METHOD 27
ADJOIN 9
ADJUST-ARRAY 11
ADJUSTABLE-ARRAY-P 11
ALLOCATE-INSTANCE 26
ALPHA-CHAR-P 7
ALPHANUMERICP 7
ALWAYS 25
AND 21, 22, 24, 28, 33, 36
APPEND 10, 24, 28
APPENDING 24
APPLY 18
APROPOS 48
APROPOS-LIST 48
AREF 11
ARITHMETIC-ERROR 32
ARITHMETIC-ERROR-OPERANDS 31
ARITHMETIC-ERROR-OPERATION 31
ARRAY 32
ARRAY-DIMENSION 11
ARRAY-DIMENSION-LIMIT 12
ARRAY-DIMENSIONS 11
ARRAY-DISPLACEMENT 12
ARRAY-ELEMENT-TYPE 33
ARRAY-HAS-FILL-POINTER-P 11
ARRAY-IN-BOUNDS-P 11
ARRAY-RANK 11
ARRAY-RANK-LIMIT 12
ARRAY-ROW-MAJOR-INDEX 11
ARRAY-TOTAL-SIZE 11
ARRAY-TOTAL-SIZE-LIMIT 12
ARRAYP 11
AS 22
ASH 6
ASIN 3
ASINH 4
ASSERT 29
ASSOC 10
ASSOC-IF 10
ASSOC-IF-NOT 10
ATAN 4
ATANH 4
ATOM 9, 32

BASE-CHAR 32
BASE-STRING 32
BEING 24
BELOW 24
BIGNUM 32
BIT 12, 32
BIT-AND 12
BIT-ANDC1 12
BIT-ANDC2 12
BIT-EQV 12
BIT-IOR 12
BIT-NAND 12
BIT-NOR 12
BIT-NOT 12
BIT-ORC1 12
BIT-ORC2 12
BIT-VECTOR 32
BIT-VECTOR-P 11
BIT-XOR 12
BLOCK 21
BOOLE 5
BOOLE-1 5
BOOLE-2 5
BOOLE-AND 5
BOOLE-ANDC1 5
BOOLE-ANDC2 5
BOOLE-C1 5
BOOLE-C2 5
BOOLE-CLR 5
BOOLE-EQV 5
BOOLE-IOR 5
BOOLE-NAND 5
BOOLE-NOR 5
BOOLE-ORC1 5
BOOLE-ORC2 5
BOOLE-SET 5
BOOLE-XOR 5
BOOLEAN 32
BOTH-CASE-P 7
BOUNDP 17
BREAK 48
BROADCAST-STREAM 32
BROADCAST-STREAM-STREAMS 41
BUILT-IN-CLASS 32
BUTLAST 9
BY 24
BYTE 6
BYTE-POSITION 6
BYTE-SIZE 6

CAAR 9
CADR 9
CALL-ARGUMENTS-LIMIT 19
CALL-METHOD 28
CALL-NEXT-METHOD 27
CAR 9
CASE 21
CATCH 22
CCASE 21
CDAR 9
CDDR 9
CDR 9
CEILING 4
CELL-ERROR 32
CELL-ERROR-NAME 31
CERROR 29
CHANGE-CLASS 26
CHAR 8
CHAR-CODE 7
CHAR-CODE-LIMIT 7
CHAR-DOWNCASE 7
CHAR-EQUAL 7
CHAR-GREATERP 7
CHAR-INT 7
CHAR-LESSP 7
CHAR-NAME 7
CHAR-NOT-EQUAL 7
CHAR-NOT-GREATERP 7
CHAR-NOT-LESSP 7
CHAR-UPCASE 7
CHAR/= 7
CHAR< 7
CHAR<= 7
CHAR= 7
CHAR> 7
CHAR>= 7
CHARACTER 7, 32, 35
CHARACTERP 7
CHECK-TYPE 33
CIS 4
CL 46
CL-USER 46
CLASS 32
CLASS-NAME 26
CLASS-OF 26
CLEAR-INPUT 41
CLEAR-OUTPUT 41
CLOSE 42
CLQR 1
CLRHASH 15
CODE-CHAR 7
COERCE 31
COLLECT 24
COLLECTING 24
COMMON-LISP 46
COMMON-LISP-USER 46
COMPILATION-SPEED 49
COMPILE 46
COMPILE-FILE 47
COMPILE-FILE-PATHNAME 47
COMPILED-FUNCTION 32
COMPILED-FUNCTION-P 46
COMPILER-MACRO 46
COMPILER-MACRO-FUNCTION 47
COMPLEMENT 19
COMPLEX 4, 32, 35
COMPLEXP 3
COMPUTE-APPLICABLE-METHODS 27
COMPUTE-RESTARTS 30
CONCATENATE 13
CONCATENATED-STREAM 32
CONCATENATED-STREAM-STREAMS 41
COND 21
CONDITION 32
CONJUGATE 4
CONS 9, 32
CONSP 8
CONSTANTLY 19
CONSTANTP 17
CONTINUE 30
CONTROL-ERROR 32
COPY-ALIST 10
COPY-LIST 10
COPY-PPRINT-DISPATCH 38
COPY-READTABLE 34
COPY-SEQ 15
COPY-STRUCTURE 16
COPY-SYMBOL 46
COPY-TREE 11
COS 3
COSH 4
COUNT 13, 24
COUNT-IF 13
COUNT-IF-NOT 13
COUNTING 24
CTYPECASE 31

DEBUG 49
DECF 3
DECLAIM 49
DECLARATION 49
DECLARE 49
DECODE-FLOAT 6
DECODE-UNIVERSAL-TIME 49
DEFCLASS 25
DEFCONSTANT 17
DEFGENERIC 26
DEFINE-COMPILER-MACRO 19
DEFINE-CONDITION 29
DEFINE-METHOD-COMBINATION 28
DEFINE-MODIFY-MACRO 20
DEFINE-SETF-EXPANDER 20
DEFINE-SYMBOL-MACRO 20
DEFMACRO 19
DEFMETHOD 27
DEFPACKAGE 44
DEFPARAMETER 17
DEFSETF 20
DEFSTRUCT 16
DEFTYPE 33
DEFUN 18
DEFVAR 17
DELETE 14
DELETE-DUPLICATES 14
DELETE-FILE 43
DELETE-IF 14
DELETE-IF-NOT 14
DELETE-PACKAGE 44
DENOMINATOR 4
DEPOSIT-FIELD 6
DESCRIBE 48
DESCRIBE-OBJECT 48
DESTRUCTURING-BIND 18
DIGIT-CHAR 7
DIGIT-CHAR-P 7
DIRECTORY 44
DIRECTORY-NAMESTRING 43

DISASSEMBLE 48
DIVISION-BY-ZERO 32
DO 22, 24
DO-ALL-SYMBOLS 45
DO-EXTERNAL-SYMBOLS 45
DO-SYMBOLS 45
DO∗ 22
DOCUMENTATION 46
DOING 24
DOLIST 22
DOTIMES 22
DOUBLE-FLOAT 32, 35
DOUBLE-FLOAT-EPSILON 6
DOUBLE-FLOAT-NEGATIVE-EPSILON 6
DOWNFROM 24
DOWNTO 24
DPB 6
DRIBBLE 48
DYNAMIC-EXTENT 49

EACH 24
ECASE 21
ECHO-STREAM 32
ECHO-STREAM-INPUT-STREAM 41
ECHO-STREAM-OUTPUT-STREAM 41
ED 48
EIGHTH 9
ELSE 24
ELT 13
ENCODE-UNIVERSAL-TIME 49
END 24
END-OF-FILE 32
ENDP 8
ENOUGH-NAMESTRING 43
ENSURE-DIRECTORIES-EXIST 44
ENSURE-GENERIC-FUNCTION 27
EQ 16
EQL 16, 33
EQUAL 16
EQUALP 16
ERROR 29, 32
ETYPECASE 31
EVAL 47
EVAL-WHEN 47
EVENP 3
EVERY 12
EXP 3
EXPORT 45
EXPT 3
EXTENDED-CHAR 32
EXTERNAL-SYMBOL 24
EXTERNAL-SYMBOLS 24

FBOUNDP 17
FCEILING 4
FDEFINITION 19
FFLOOR 4
FIFTH 9
FILE-AUTHOR 43
FILE-ERROR 32
FILE-ERROR-PATHNAME 31
FILE-LENGTH 43
FILE-NAMESTRING 43
FILE-POSITION 41
FILE-STREAM 32
FILE-STRING-LENGTH 41
FILE-WRITE-DATE 43
FILL 13
FILL-POINTER 12
FINALLY 25
FIND 14
FIND-ALL-SYMBOLS 45
FIND-CLASS 25
FIND-IF 14
FIND-IF-NOT 14
FIND-METHOD 27
FIND-PACKAGE 45
FIND-RESTART 30
FIND-SYMBOL 45
FINISH-OUTPUT 41
FIRST 9
FIXNUM 32
FLET 18
FLOAT 4, 32
FLOAT-DIGITS 6
FLOAT-PRECISION 6
FLOAT-RADIX 6
FLOAT-SIGN 4
FLOATING-POINT-INEXACT 32
FLOATING-POINT-INVALID-OPERATION 32
FLOATING-POINT-OVERFLOW 32
FLOATING-POINT-UNDERFLOW 32
FLOATP 3

FLOOR 4
FMAKUNBOUND 19
FOR 22
FORCE-OUTPUT 41
FORMAT 38
FORMATTER 38
FOURTH 9
FRESH-LINE 36
FROM 24
FROUND 4
FTRUNCATE 4
FTYPE 49
FUNCALL 18
FUNCTION 18, 32, 35, 46
FUNCTION-KEYWORDS 28
FUNCTION-LAMBDA-EXPRESSION 19
FUNCTIONP 17

GCD 3
GENERIC-FUNCTION 32
GENSYM 46
GENTEMP 46
GET 17
GET-DECODED-TIME 49
GET-DISPATCH-MACRO-CHARACTER 35
GET-INTERNAL-REAL-TIME 49
GET-INTERNAL-RUN-TIME 49
GET-MACRO-CHARACTER 35
GET-OUTPUT-STREAM-STRING 41
GET-PROPERTIES 17
GET-SETF-EXPANSION 20
GET-UNIVERSAL-TIME 49
GETF 17
GETHASH 15
GO 22
GRAPHIC-CHAR-P 7

HANDLER-BIND 30
HANDLER-CASE 30
HASH-KEY 24
HASH-KEYS 24
HASH-TABLE 32
HASH-TABLE-COUNT 15
HASH-TABLE-P 15
HASH-TABLE-REHASH-SIZE 15
HASH-TABLE-REHASH-THRESHOLD 15
HASH-TABLE-SIZE 15
HASH-TABLE-TEST 15
HASH-VALUE 24
HASH-VALUES 24
HOST-NAMESTRING 43

IDENTITY 19
IF 21, 24
IGNORABLE 49
IGNORE 49
IGNORE-ERRORS 29
IMAGPART 4
IMPORT 45
IN 24
IN-PACKAGE 44
INCF 3
INITIALIZE-INSTANCE 26
INITIALLY 25
INLINE 49
INPUT-STREAM-P 33
INSPECT 48
INTEGER 32
INTEGER-DECODE-FLOAT 6
INTEGER-LENGTH 6
INTEGERP 3
INTERACTIVE-STREAM-P 33
INTERN 45
INTERNAL-TIME-UNITS-PER-SECOND 49
INTERSECTION 11
INTO 24
INVALID-METHOD-ERROR 27
INVOKE-DEBUGGER 29
INVOKE-RESTART 30
INVOKE-RESTART-INTERACTIVELY 30
ISQRT 3
IT 24

KEYWORD 32, 44, 46
KEYWORDP 44

LABELS 18

LAMBDA 18
LAMBDA-LIST-KEYWORDS 20
LAMBDA-PARAMETERS-LIMIT 19
LAST 9
LCM 3
LDB 6
LDB-TEST 6
LDIFF 9
LEAST-NEGATIVE-DOUBLE-FLOAT 6
LEAST-NEGATIVE-LONG-FLOAT 6
LEAST-NEGATIVE-NORMALIZED-DOUBLE-FLOAT 6
LEAST-NEGATIVE-NORMALIZED-LONG-FLOAT 6
LEAST-NEGATIVE-NORMALIZED-SHORT-FLOAT 6
LEAST-NEGATIVE-NORMALIZED-SINGLE-FLOAT 6
LEAST-NEGATIVE-SHORT-FLOAT 6
LEAST-NEGATIVE-SINGLE-FLOAT 6
LEAST-POSITIVE-DOUBLE-FLOAT 6
LEAST-POSITIVE-LONG-FLOAT 6
LEAST-POSITIVE-NORMALIZED-DOUBLE-FLOAT 6
LEAST-POSITIVE-NORMALIZED-LONG-FLOAT 6
LEAST-POSITIVE-NORMALIZED-SHORT-FLOAT 6
LEAST-POSITIVE-NORMALIZED-SINGLE-FLOAT 6
LEAST-POSITIVE-SHORT-FLOAT 6
LEAST-POSITIVE-SINGLE-FLOAT 6
LENGTH 13
LET 18
LET∗ 18
LISP-IMPLEMENTATION-TYPE 49
LISP-IMPLEMENTATION-VERSION 49
LIST 9, 28, 32
LIST-ALL-PACKAGES 44
LIST-LENGTH 9
LIST∗ 9
LISTEN 41
LISTP 8
LOAD 47
LOAD-LOGICAL-PATHNAME-TRANSLATIONS 43
LOAD-TIME-VALUE 47
LOCALLY 47
LOG 3
LOGAND 5
LOGANDC1 5
LOGANDC2 5
LOGBITP 5
LOGCOUNT 5
LOGEQV 5
LOGICAL-PATHNAME 32, 43
LOGICAL-PATHNAME-TRANSLATIONS 43
LOGIOR 5
LOGNAND 5
LOGNOR 5
LOGNOT 5
LOGORC1 5
LOGORC2 5
LOGTEST 5
LOGXOR 5
LONG-FLOAT 32, 35
LONG-FLOAT-EPSILON 6
LONG-FLOAT-NEGATIVE-EPSILON 6
LONG-SITE-NAME 49
LOOP 22
LOOP-FINISH 25
LOWER-CASE-P 7

MACHINE-INSTANCE 49
MACHINE-TYPE 49
MACHINE-VERSION 49
MACRO-FUNCTION 47
MACROEXPAND 48
MACROEXPAND-1 48
MACROLET 20
MAKE-ARRAY 11
MAKE-BROADCAST-STREAM 41

MAKE-CONCATENATED-STREAM 41
MAKE-CONDITION 29
MAKE-DISPATCH-MACRO-CHARACTER 35
MAKE-ECHO-STREAM 41
MAKE-HASH-TABLE 15
MAKE-INSTANCE 25
MAKE-INSTANCES-OBSOLETE 26
MAKE-LIST 9
MAKE-LOAD-FORM 47
MAKE-LOAD-FORM-SAVING-SLOTS 47
MAKE-METHOD 28
MAKE-PACKAGE 44
MAKE-PATHNAME 42
MAKE-RANDOM-STATE 4
MAKE-SEQUENCE 13
MAKE-STRING 8
MAKE-STRING-INPUT-STREAM 41
MAKE-STRING-OUTPUT-STREAM 41
MAKE-SYMBOL 45
MAKE-SYNONYM-STREAM 41
MAKE-TWO-WAY-STREAM 41
MAKUNBOUND 17
MAP 15
MAP-INTO 15
MAPC 10
MAPCAN 10
MAPCAR 10
MAPCON 10
MAPHASH 15
MAPL 10
MAPLIST 10
MASK-FIELD 6
MAX 4, 28
MAXIMIZE 24
MAXIMIZING 24
MEMBER 9, 33
MEMBER-IF 9
MEMBER-IF-NOT 9
MERGE 13
MERGE-PATHNAMES 43
METHOD 32
METHOD-COMBINATION 32, 46
METHOD-COMBINATION-ERROR 27
METHOD-QUALIFIERS 28
MIN 4, 28
MINIMIZE 24
MINIMIZING 24
MINUSP 3
MISMATCH 13
MOD 4, 33
MOST-NEGATIVE-DOUBLE-FLOAT 6
MOST-NEGATIVE-FIXNUM 6
MOST-NEGATIVE-LONG-FLOAT 6
MOST-NEGATIVE-SHORT-FLOAT 6
MOST-NEGATIVE-SINGLE-FLOAT 6
MOST-POSITIVE-DOUBLE-FLOAT 6
MOST-POSITIVE-FIXNUM 6
MOST-POSITIVE-LONG-FLOAT 6
MOST-POSITIVE-SHORT-FLOAT 6
MOST-POSITIVE-SINGLE-FLOAT 6
MUFFLE-WARNING 30
MULTIPLE-VALUE-BIND 18
MULTIPLE-VALUE-CALL 18
MULTIPLE-VALUE-LIST 18
MULTIPLE-VALUE-PROG1 21
MULTIPLE-VALUE-SETQ 17
MULTIPLE-VALUES-LIMIT 19

NAME-CHAR 7
NAMED 22
NAMESTRING 43
NBUTLAST 9
NCONC 10, 24, 28
NCONCING 24
NEVER 25
NEWLINE 7
NEXT-METHOD-P 26
NIL 2, 46
NINTERSECTION 11

NINTH 9
NO-APPLICABLE-METHOD 27
NO-NEXT-METHOD 27
NOT 17, 33, 36
NOTANY 13
NOTEVERY 12
NOTINLINE 49
NRECONC 10
NREVERSE 13
NSET-DIFFERENCE 11
NSET-EXCLUSIVE-OR 11
NSTRING-CAPITALIZE 8
NSTRING-DOWNCASE 8
NSTRING-UPCASE 8
NSUBLIS 11
NSUBST 10
NSUBST-IF 10
NSUBST-IF-NOT 10
NSUBSTITUTE 14
NSUBSTITUTE-IF 14
NSUBSTITUTE-IF-NOT 14
NTH 9
NTH-VALUE 19
NTHCDR 9
NULL 8, 32
NUMBER 32
NUMBERP 3
NUMERATOR 4
NUNION 11

ODDP 3
OF 24
OF-TYPE 22
ON 24
OPEN 41
OPEN-STREAM-P 33
OPTIMIZE 49
OR 21, 28, 33, 36
OTHERWISE 21, 31
OUTPUT-STREAM-P 33

PACKAGE 32
PACKAGE-ERROR 32
PACKAGE-ERROR-PACKAGE 31
PACKAGE-NAME 44
PACKAGE-NICKNAMES 44
PACKAGE-SHADOWING-SYMBOLS 45
PACKAGE-USE-LIST 44
PACKAGE-USED-BY-LIST 44
PACKAGEP 44
PAIRLIS 10
PARSE-ERROR 32
PARSE-INTEGER 8
PARSE-NAMESTRING 43
PATHNAME 32, 43
PATHNAME-DEVICE 42
PATHNAME-DIRECTORY 42
PATHNAME-HOST 42
PATHNAME-MATCH-P 33
PATHNAME-NAME 42
PATHNAME-TYPE 42
PATHNAME-VERSION 42
PATHNAMEP 33
PEEK-CHAR 34
PHASE 4
PI 3
PLUSP 3
POP 9
POSITION 14
POSITION-IF 14
POSITION-IF-NOT 14
PPRINT 36
PPRINT-DISPATCH 38
PPRINT-EXIT-IF-LIST-EXHAUSTED 37
PPRINT-FILL 37
PPRINT-INDENT 37
PPRINT-LINEAR 37
PPRINT-LOGICAL-BLOCK 37
PPRINT-NEWLINE 37
PPRINT-POP 37
PPRINT-TAB 37
PPRINT-TABULAR 37
PRESENT-SYMBOL 24
PRESENT-SYMBOLS 24
PRIN1 36
PRIN1-TO-STRING 36
PRINC 36
PRINC-TO-STRING 36
PRINT 36
PRINT-NOT-READABLE 32
PRINT-NOT-READABLE-OBJECT 31
PRINT-OBJECT 36
PRINT-UNREADABLE-OBJECT 36
PROBE-FILE 43
PROCLAIM 49
PROG 21
PROG1 21
PROG2 21
PROG* 21
PROGN 21, 28
PROGRAM-ERROR 32
PROGV 17
PROVIDE 45
PSETF 17
PSETQ 17
PUSH 10
PUSHNEW 10

QUOTE 35, 47

RANDOM 4
RANDOM-STATE 32
RANDOM-STATE-P 3
RASSOC 10
RASSOC-IF 10
RASSOC-IF-NOT 10
RATIO 32, 35
RATIONAL 4, 32
RATIONALIZE 4
RATIONALP 3
READ 34
READ-BYTE 34
READ-CHAR 34
READ-CHAR-NO-HANG 34
READ-DELIMITED-LIST 34
READ-FROM-STRING 34
READ-LINE 34
READ-PRESERVING-WHITESPACE 34
READ-SEQUENCE 34
READER-ERROR 32
READTABLE 32
READTABLE-CASE 34
READTABLEP 33
REAL 32
REALP 3
REALPART 4
REDUCE 15
REINITIALIZE-INSTANCE 25
REM 4
REMF 17
REMHASH 15
REMOVE 14
REMOVE-DUPLICATES 14
REMOVE-IF 14
REMOVE-IF-NOT 14
REMOVE-METHOD 27
REMPROP 17
RENAME-FILE 43
RENAME-PACKAGE 44
REPEAT 25
REPLACE 15
REQUIRE 45
REST 9
RESTART 32
RESTART-BIND 30
RESTART-CASE 30
RESTART-NAME 30
RETURN 21, 24
RETURN-FROM 21
REVAPPEND 10
REVERSE 13
ROOM 48
ROTATEF 17
ROUND 4
ROW-MAJOR-AREF 11
RPLACA 9
RPLACD 9

SAFETY 49
SATISFIES 33
SBIT 12
SCALE-FLOAT 6
SCHAR 8
SEARCH 14
SECOND 9
SEQUENCE 32
SERIOUS-CONDITION 32
SET 17
SET-DIFFERENCE 11
SET-DISPATCH-MACRO-CHARACTER 35
SET-EXCLUSIVE-OR 11
SET-MACRO-CHARACTER 35
SET-PPRINT-DISPATCH 38
SET-SYNTAX-FROM-CHAR 34
SETF 17, 46
SETQ 17
SEVENTH 9
SHADOW 45
SHADOWING-IMPORT 45
SHARED-INITIALIZE 26
SHIFTF 17
SHORT-FLOAT 32, 35
SHORT-FLOAT-EPSILON 6
SHORT-FLOAT-NEGATIVE-EPSILON 6
SHORT-SITE-NAME 49
SIGNAL 29
SIGNED-BYTE 32
SIGNUM 4
SIMPLE-ARRAY 32
SIMPLE-BASE-STRING 32
SIMPLE-BIT-VECTOR 32
SIMPLE-BIT-VECTOR-P 11
SIMPLE-CONDITION 32
SIMPLE-CONDITION-FORMAT-ARGUMENTS 31
SIMPLE-CONDITION-FORMAT-CONTROL 31
SIMPLE-ERROR 32
SIMPLE-STRING 32
SIMPLE-STRING-P 8
SIMPLE-TYPE-ERROR 32
SIMPLE-VECTOR 32
SIMPLE-VECTOR-P 11
SIMPLE-WARNING 32
SIN 3
SINGLE-FLOAT 32, 35
SINGLE-FLOAT-EPSILON 6
SINGLE-FLOAT-NEGATIVE-EPSILON 6
SINH 4
SIXTH 9
SLEEP 22
SLOT-BOUNDP 25
SLOT-EXISTS-P 25
SLOT-MAKUNBOUND 25
SLOT-MISSING 26
SLOT-UNBOUND 26
SLOT-VALUE 25
SOFTWARE-TYPE 49
SOFTWARE-VERSION 49
SOME 13
SORT 13
SPACE 7, 49
SPECIAL 49
SPECIAL-OPERATOR-P 46
SPEED 49
SQRT 3
STABLE-SORT 13
STANDARD 28
STANDARD-CHAR 7, 32
STANDARD-CHAR-P 7
STANDARD-CLASS 32
STANDARD-GENERIC-FUNCTION 32
STANDARD-METHOD 32
STANDARD-OBJECT 32
STEP 48
STORAGE-CONDITION 32
STORE-VALUE 30
STREAM 32
STREAM-ELEMENT-TYPE 33
STREAM-ERROR 32
STREAM-ERROR-STREAM 31
STREAM-EXTERNAL-FORMAT 42
STREAMP 33
STRING 8, 32
STRING-CAPITALIZE 8
STRING-DOWNCASE 8
STRING-EQUAL 8
STRING-GREATERP 8
STRING-LEFT-TRIM 8
STRING-LESSP 8
STRING-NOT-EQUAL 8
STRING-NOT-GREATERP 8
STRING-NOT-LESSP 8
STRING-RIGHT-TRIM 8
STRING-STREAM 32
STRING-TRIM 8
STRING-UPCASE 8
STRING/= 8
STRING< 8
STRING<= 8
STRING= 8
STRING> 8
STRING>= 8
STRINGP 8
STRUCTURE 46
STRUCTURE-CLASS 32
STRUCTURE-OBJECT 32
STYLE-WARNING 32
SUBLIS 11
SUBSEQ 13
SUBSETP 9
SUBST 10
SUBST-IF 10
SUBST-IF-NOT 10
SUBSTITUTE 14
SUBSTITUTE-IF 14
SUBSTITUTE-IF-NOT 14
SUBTYPEP 31
SUM 24
SUMMING 24
SVREF 12
SXHASH 16
SYMBOL 24, 32, 45
SYMBOL-FUNCTION 46
SYMBOL-MACROLET 20
SYMBOL-NAME 46
SYMBOL-PACKAGE 46
SYMBOL-PLIST 46
SYMBOL-VALUE 46
SYMBOLP 44
SYMBOLS 24
SYNONYM-STREAM 32
SYNONYM-STREAM-SYMBOL 41

T 2, 32, 46
TAGBODY 22
TAILP 9
TAN 3
TANH 4
TENTH 9
TERPRI 36
THE 24, 31
THEN 24
THEREIS 25
THIRD 9
THROW 22
TIME 48
TO 24
TRACE 48
TRANSLATE-LOGICAL-PATHNAME 43
TRANSLATE-PATHNAME 43
TREE-EQUAL 10
TRUENAME 43
TRUNCATE 4
TWO-WAY-STREAM 32
TWO-WAY-STREAM-INPUT-STREAM 41
TWO-WAY-STREAM-OUTPUT-STREAM 41
TYPE 46, 49
TYPE-ERROR 32
TYPE-ERROR-DATUM 31
TYPE-ERROR-EXPECTED-TYPE 31
TYPE-OF 33
TYPECASE 31
TYPEP 31

UNBOUND-SLOT 32
UNBOUND-SLOT-INSTANCE 31
UNBOUND-VARIABLE 32
UNDEFINED-FUNCTION 32
UNEXPORT 45
UNINTERN 45
UNION 11
UNLESS 21, 24
UNREAD-CHAR 34
UNSIGNED-BYTE 32
UNTIL 25
UNTRACE 48
UNUSE-PACKAGE 44
UNWIND-PROTECT 21
UPDATE-INSTANCE-FOR-DIFFERENT-CLASS 26
UPDATE-INSTANCE-FOR-REDEFINED-CLASS 26
UPFROM 24
UPGRADED-ARRAY-ELEMENT-TYPE 33
UPGRADED-COMPLEX-PART-TYPE 6
UPPER-CASE-P 7
UPTO 24
USE-PACKAGE 44
USE-VALUE 30
USER-HOMEDIR-PATHNAME 43
USING 24

V 40
VALUES 18, 33
VALUES-LIST 18
VARIABLE 46
VECTOR 12, 32
VECTOR-POP 12
VECTOR-PUSH 12
VECTOR-PUSH-EXTEND 12
VECTORP 11

WARN 29
WARNING 32

WHEN 21, 24
WHILE 25
WILD-PATHNAME-P 33
WITH 22
WITH-ACCESSORS 26
WITH-COMPILATION-UNIT 47
WITH-CONDITION-RESTARTS 31
WITH-HASH-TABLE-ITERATOR 15
WITH-INPUT-FROM-STRING 42
WITH-OPEN-FILE 42
WITH-OPEN-STREAM 42
WITH-OUTPUT-TO-STRING 42
WITH-PACKAGE-ITERATOR 45
WITH-SIMPLE-RESTART 30
WITH-SLOTS 26
WITH-STANDARD-IO-SYNTAX 34
WRITE 37
WRITE-BYTE 36
WRITE-CHAR 36
WRITE-LINE 36
WRITE-SEQUENCE 37
WRITE-STRING 36
WRITE-TO-STRING 37

Y-OR-N-P 34
YES-OR-NO-P 34

ZEROP 3

Common Lisp Quick Reference Revision 148 [2018-10-10]

LaTeX source: http://clqr.boundp.org