Quick Reference

Common

lisp

Bert Burgemeister

Contents

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.

$\{foo \big| bar \big| baz\}$; $\begin{cases} foo \\ bar \\ baz \end{cases}$ ▷ 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.

$foo^{\overset{\mathsf{P}}{*}}$ ▷ foo^* is evaluated as in ${}_s$**progn**; see page 20.

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

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

1 Numbers

1.1 Predicates

($_f$**=** *number*$^+$)
($_f$**/=** *number*$^+$)

▷ $\underline{\texttt{T}}$ if all *numbers*, or none, respectively, are equal in value.

($_f$**>** *number*$^+$)
($_f$**>=** *number*$^+$)
($_f$**<** *number*$^+$)
($_f$**<=** *number*$^+$)

▷ Return $\underline{\texttt{T}}$ if *numbers* are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively.

($_f$**minusp** *a*)
($_f$**zerop** *a*)
($_f$**plusp** *a*)

▷ $\underline{\texttt{T}}$ if $a < 0$, $a = 0$, or $a > 0$, respectively.

($_f$**evenp** *int*)
($_f$**oddp** *int*)

▷ $\underline{\texttt{T}}$ 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*)

▷ $\underline{\texttt{T}}$ 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 *b*s, 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 *integers*. (**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*)
($_f$**tanh** *a*)

▷ $\underline{\sinh a}$, $\underline{\cosh a}$, or $\underline{\tanh a}$, respectively.

(${}_f$**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*.

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

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

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

▷ 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** [$\{state|\texttt{NIL}|\texttt{T}\}_{\boxed{\texttt{NIL}}}$])
▷ $\underline{\text{Copy}}$ 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 $\underline{\text{value}}$ of bitwise logical *operation*. *operations* are

${}_c$**boole-1** ▷ $\underline{int\text{-}a}$.
${}_c$**boole-2** ▷ $\underline{int\text{-}b}$.
${}_c$**boole-c1** ▷ $\underline{\neg int\text{-}a}$.
${}_c$**boole-c2** ▷ $\underline{\neg int\text{-}b}$.
${}_c$**boole-set** ▷ $\underline{\text{All bits set}}$.
${}_c$**boole-clr** ▷ $\underline{\text{All bits zero}}$.

Index

$_c$**boole-eqv** ▷ $\underline{int\text{-}a \equiv int\text{-}b}$.

$_c$**boole-and** ▷ $\underline{int\text{-}a \wedge int\text{-}b}$.

$_c$**boole-andc1** ▷ $\underline{\neg int\text{-}a \wedge int\text{-}b}$.

$_c$**boole-andc2** ▷ $\underline{int\text{-}a \wedge \neg int\text{-}b}$.

$_c$**boole-nand** ▷ $\underline{\neg(int\text{-}a \wedge int\text{-}b)}$.

$_c$**boole-ior** ▷ $\underline{int\text{-}a \vee int\text{-}b}$.

$_c$**boole-orc1** ▷ $\underline{\neg int\text{-}a \vee int\text{-}b}$.

$_c$**boole-orc2** ▷ $\underline{int\text{-}a \vee \neg int\text{-}b}$.

$_c$**boole-xor** ▷ $\underline{\neg(int\text{-}a \equiv int\text{-}b)}$.

$_c$**boole-nor** ▷ $\underline{\neg(int\text{-}a \vee int\text{-}b)}$.

($_f$**lognot** *integer*) ▷ $\underline{\neg integer}$.

($_f$**logeqv** *integer**)
($_f$**logand** *integer**)
▷ Return $\underline{\text{value of exclusive-nored or anded } integers}$, respectively. Without any *integer*, return $\underline{-1}$.

($_f$**logandc1** *int-a int-b*) ▷ $\underline{\neg int\text{-}a \wedge int\text{-}b}$.

($_f$**logandc2** *int-a int-b*) ▷ $\underline{int\text{-}a \wedge \neg int\text{-}b}$.

($_f$**lognand** *int-a int-b*) ▷ $\underline{\neg(int\text{-}a \wedge int\text{-}b)}$.

($_f$**logxor** *integer**)
($_f$**logior** *integer**)
▷ Return $\underline{\text{value of exclusive-ored or ored } integers}$, respectively. Without any *integer*, return $\underline{0}$.

($_f$**logorc1** *int-a int-b*) ▷ $\underline{\neg int\text{-}a \vee int\text{-}b}$.

($_f$**logorc2** *int-a int-b*) ▷ $\underline{int\text{-}a \vee \neg int\text{-}b}$.

($_f$**lognor** *int-a int-b*) ▷ $\underline{\neg(int\text{-}a \vee int\text{-}b)}$.

($_f$**logbitp** *i int*) ▷ $\underline{\mathtt{T}}$ if zero-indexed *i*th bit of *int* is set.

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

($_f$**logcount** *int*)
▷ $\underline{\text{Number of 1 bits}}$ in $int \geq 0$, $\underline{\text{number of 0 bits}}$ 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 $\underline{\mathtt{T}}$ if any bit specified by *byte-spec* in *integer* is set.

($_f$**ash** *integer count*)
▷ Return copy of $\underline{integer}$ 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.

$\left(\left\{\begin{array}{l}{}_f\textbf{deposit-field}\\ {}_f\textbf{dpb}\end{array}\right\}\right.$ *int-a byte-spec int-b*)
▷ Return $\underline{int\text{-}b}$ 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 $\underline{integer}$ 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*)
▷ $\underline{\texttt{T}}$ if argument is of indicated type.

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

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

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

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

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

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

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 $\underline{\text{second}}$, $\underline{\text{minute}}_2$, $\underline{\text{hour}}_3$, $\underline{\text{date}}_4$, $\underline{\text{month}}_5$, $\underline{\text{year}}_6$, $\underline{\text{day}}_7$, $\underline{\text{daylight-p}}_8$, and $\underline{\text{zone}}_9$.

($_f$**short-site-name**)
($_f$**long-site-name**)
▷ $\underline{\text{String}}$ representing physical location of computer.

$$\left(\left\{\begin{array}{l}{}_f\textbf{lisp-implementation}\\{}_f\textbf{software}\\{}_f\textbf{machine}\end{array}\right\}\text{-}\left\{\begin{array}{l}\textbf{type}\\\textbf{version}\end{array}\right\}\right)$$

▷ $\underline{\text{Name}}$ or $\underline{\text{version}}$ of implementation, operating system, or hardware, respectively.

($_f$**machine-instance**) ▷ $\underline{\text{Computer name}}$.

(${}_m$**trace** $\left\{\begin{array}{l}\textit{function}\\(\textbf{setf}\ \textit{function})\end{array}\right\}^{*}$)
▷ Cause *functions* to be traced. With no arguments, return list of traced functions.

(${}_m$**untrace** $\left\{\begin{array}{l}\textit{function}\\(\textbf{setf}\ \textit{function})\end{array}\right\}^{*}$)
▷ Stop *functions*, 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 36, ${}_f$**format**, for *control* and *args*.

(${}_m$**time** *form*)
▷ Evaluate *forms* 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** $\widehat{\textit{decl}}^*$)
▷ Globally make declaration(s) *decl*. *decl* can be: **declaration**, **type**, **ftype**, **inline**, **notinline**, **optimize**, or **special**. See below.

(**declare** $\widehat{\textit{decl}}^*$)
▷ 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 *variables* and/or *functions* to end when control leaves enclosing block.

([**type**] *type variable**)
(**ftype** *type function**)
▷ Declare *variables* or *functions* to be of *type*.

($\left\{\begin{array}{l}\textbf{ignorable}\\\textbf{ignore}\end{array}\right\}$ $\left\{\begin{array}{l}\textit{var}\\(\textbf{function}\ \textit{function})\end{array}\right\}^{*}$)
▷ Suppress warnings about used/unused bindings.

(**inline** *function**)
(**notinline** *function**)
▷ Tell compiler to integrate/not to integrate, respectively, called *functions* into the calling routine.

(**optimize** $\left\{\begin{array}{|l}\textbf{compilation-speed}|(\textbf{compilation-speed}\ n_{\boxed{3}})\\\textbf{debug}|(\textbf{debug}\ n_{\boxed{3}})\\\textbf{safety}|(\textbf{safety}\ n_{\boxed{3}})\\\textbf{space}|(\textbf{space}\ n_{\boxed{3}})\\\textbf{speed}|(\textbf{speed}\ n_{\boxed{3}})\end{array}\right\}$)
▷ 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.

(${}_f$**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*); ≥ 96.

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

3 Strings

Strings can as well be manipulated by array and sequence functions; see pages 10 and 12.

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

($\left\{\begin{array}{l}{}_f\textbf{string=}\\{}_f\textbf{string-equal}\end{array}\right\}$ *foo bar* $\left\{\begin{array}{|l}\textbf{:start1}\ \textit{start-foo}_{\boxed{0}}\\\textbf{:start2}\ \textit{start-bar}_{\boxed{0}}\\\textbf{:end1}\ \textit{end-foo}_{\boxed{\texttt{NIL}}}\\\textbf{:end2}\ \textit{end-bar}_{\boxed{\texttt{NIL}}}\end{array}\right\}$)
▷ Return T if subsequences of *foo* and *bar* are equal. Obey/ignore, respectively, case.

($\left\{\begin{array}{l}{}_f\textbf{string}\{\textbf{/=}\ |\textbf{-not-equal}\}\\{}_f\textbf{string}\{\textbf{>}\ |\textbf{-greaterp}\}\\{}_f\textbf{string}\{\textbf{>=}\ |\textbf{-not-lessp}\}\\{}_f\textbf{string}\{\textbf{<}\ |\textbf{-lessp}\}\\{}_f\textbf{string}\{\textbf{<=}\ |\textbf{-not-greaterp}\}\end{array}\right\}$ *foo bar* $\left\{\begin{array}{|l}\textbf{:start1}\ \textit{start-foo}_{\boxed{0}}\\\textbf{:start2}\ \textit{start-bar}_{\boxed{0}}\\\textbf{:end1}\ \textit{end-foo}_{\boxed{\texttt{NIL}}}\\\textbf{:end2}\ \textit{end-bar}_{\boxed{\texttt{NIL}}}\end{array}\right\}$)
▷ If *foo* is lexicographically not equal, greater, not less, less, or not greater, respectively, then return position of first mismatching character in *foo*. Otherwise return NIL. Obey/ignore, respectively, case.

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

(${}_f$**string** x)
($\left\{\begin{array}{l}{}_f\textbf{string-capitalize}\\{}_f\textbf{string-upcase}\\{}_f\textbf{string-downcase}\end{array}\right\}$ x $\left\{\begin{array}{|l}\textbf{:start}\ \textit{start}_{\boxed{0}}\\\textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}\end{array}\right\}$)
▷ Convert x (**symbol**, **string**, or **character**) into a string, a string with capitalized words, an all-uppercase string, or an all-lowercase string, respectively.

($\left\{\begin{array}{l}{}_f\textbf{nstring-capitalize}\\{}_f\textbf{nstring-upcase}\\{}_f\textbf{nstring-downcase}\end{array}\right\}$ $\widetilde{\textit{string}}$ $\left\{\begin{array}{|l}\textbf{:start}\ \textit{start}_{\boxed{0}}\\\textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}\end{array}\right\}$)
▷ Convert *string* into a string with capitalized words, an all-uppercase string, or an all-lowercase string, respectively.

($\left\{\begin{array}{l}{}_f\textbf{string-trim}\\{}_f\textbf{string-left-trim}\\{}_f\textbf{string-right-trim}\end{array}\right\}$ *char-bag string*)
▷ Return *string* with all characters in sequence *char-bag* removed from both ends, from the beginning, or from the end, respectively.

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

($_f$**parse-integer** *string* $\left\{\begin{array}{|l}\textbf{:start } \textit{start}_{\boxed{0}}\\ \textbf{:end } \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:radix } \textit{int}_{\boxed{10}}\\ \textbf{:junk-allowed } \textit{bool}_{\boxed{\texttt{NIL}}}\end{array}\right\}$)
▷ Return $\underline{\text{integer}}$ parsed from *string* and $\underset{2}{\underline{\text{index}}}$ of parse end.

4 Conses

4.1 Predicates

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

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

($_f$**atom** *foo*) ▷ Return $\underline{\texttt{T}}$ if *foo* is not a **cons**.

($_f$**tailp** *foo list*) ▷ Return $\underline{\texttt{T}}$ if *foo* is a tail of *list*.

($_f$**member** *foo list* $\left\{\begin{array}{|l}\left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\#\text{'eql}}}\\ \textbf{:test-not } \textit{function}\end{array}\right\}\\ \textbf{:key } \textit{function}\end{array}\right\}$)
▷ Return $\underline{\text{tail of } \textit{list}}$ starting with its first element matching *foo*. Return $\underline{\texttt{NIL}}$ 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 $\underline{\text{tail of } \textit{list}}$ starting with its first element satisfying *test*. Return $\underline{\texttt{NIL}}$ if there is no such element.

($_f$**subsetp** *list-a list-b* $\left\{\begin{array}{|l}\left\{\begin{array}{l}\textbf{:test } \textit{function}_{\boxed{\#\text{'eql}}}\\ \textbf{:test-not } \textit{function}\end{array}\right\}\\ \textbf{:key } \textit{function}\end{array}\right\}$)
▷ Return $\underline{\texttt{T}}$ if *list-a* is a subset of *list-b*.

4.2 Lists

($_f$**cons** *foo bar*) ▷ Return new cons $\underline{(\textit{foo} \ . \ \textit{bar})}$.

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

($_f$**list*** *foo*$^+$)
▷ Return $\underline{\text{list of } \textit{foo}\text{s}}$ with last *foo* becoming cdr of last cons. Return $\underline{\textit{foo}}$ if only one *foo* given.

($_f$**make-list** *num* [**:initial-element** *foo*$_{\boxed{\texttt{NIL}}}$])
▷ New $\underline{\text{list}}$ with *num* elements set to *foo*.

($_f$**list-length** *list*) ▷ $\underline{\text{Length}}$ of *list*; $\underline{\texttt{NIL}}$ for circular *list*.

($_f$**car** *list*) ▷ $\underline{\text{Car of } \textit{list}}$ or $\underline{\texttt{NIL}}$ if *list* is NIL. **setf**able.

($_f$**cdr** *list*)
($_f$**rest** *list*) ▷ $\underline{\text{Cdr of } \textit{list}}$ or $\underline{\texttt{NIL}}$ if *list* is NIL. **setf**able.

($_f$**nthcdr** *n list*) ▷ Return $\underline{\text{tail of } \textit{list}}$ after calling $_f$**cdr** *n* times.

({$_f$**first**|$_f$**second**|$_f$**third**|$_f$**fourth**|$_f$**fifth**|$_f$**sixth**|. . . |$_f$**ninth**|$_f$**tenth**} *list*)
▷ Return $\underline{\text{nth element of } \textit{list}}$ if any, or $\underline{\texttt{NIL}}$ otherwise. **setf**able.

($_f$**nth** *n list*) ▷ Zero-indexed $\underline{n\text{th element}}$ 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 $\underline{\text{last } \textit{num} \text{ conses}}$ of *list*.

($_s$**eval-when** ($\left\{\begin{array}{|l}\{\textbf{:compile-toplevel}|\textbf{compile}\}\\ \{\textbf{:load-toplevel}|\textbf{load}\}\\ \{\textbf{:execute}|\textbf{eval}\}\end{array}\right\}$) *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** $\widehat{\textit{decl}^*}$)* *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** *bool*$_{\boxed{\texttt{NIL}}}$]) *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.

($_f$**make-load-form-saving-slots** *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*])
($_f$**compiler-macro-function** $\left\{\begin{array}{l}\textit{name}\\ (\textbf{setf } \textit{name})\end{array}\right\}$ [*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 $\underline{\text{form}}$ evaluated in the REPL, or their respective $\underline{\text{primary value}}$, or a $\underline{\text{list}}$ of their respective values.

$_v$− ▷ $\underline{\text{Form}}$ currently being evaluated by the REPL.

($_f$**apropos** *string* [*package*$_{\boxed{\texttt{NIL}}}$])
▷ Print interned symbols containing *string*.

($_f$**apropos-list** *string* [*package*$_{\boxed{\texttt{NIL}}}$])
▷ $\underline{\text{List of interned symbols}}$ 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.

($\left\{\begin{array}{l}{}_f\textbf{macroexpand-1}\\ {}_f\textbf{macroexpand}\end{array}\right\}$ *form* [*environment*$_{\boxed{\texttt{NIL}}}$])
▷ Return $\underline{\text{macro expansion}}$, once or entirely, respectively, of *form* and $\underset{2}{\underline{\texttt{T}}}$ if *form* was a macro form. Return $\underline{\textit{form}}$ and $\underset{2}{\underline{\texttt{NIL}}}$ otherwise.

$_v$***macroexpand-hook***
▷ Function of arguments expansion function, macro form, and environment called by $_f$**macroexpand-1** to generate macro expansions.

({$_g$**documentation** | (**setf** $_g$**documentation**) *new-doc*} *foo* {**'variable**|**'function** | **'compiler-macro** | **'method-combination** | **'structure**|**'type**|**'setf**|T})

▷ 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

($_f$**compile** {NIL *definition* | {*name* | (**setf** *name*)} [*definition*]})

▷ Return compiled function or replace *name*'s function definition with the compiled function. Return T$_2$ in case of **warning**s or **error**s, and T$_3$ in case of **warning**s or **error**s excluding **style-warning**s.

($_f$**compile-file** *file* {‖ **:output-file** *out-path* | **:verbose** *bool*$_{\boxed{v\text{*compile-verbose*}}}$ | **:print** *bool*$_{\boxed{v\text{*compile-print*}}}$ | **:external-format** *file-format*$_{\boxed{\text{:default}}}$ })

▷ Write compiled contents of *file* to *out-path*. Return true output path or NIL, T$_2$ in case of **warning**s or **error**s, T$_3$ in case of **warning**s or **error**s excluding **style-warning**s.

($_f$**compile-file-pathname** *file* [**:output-file** *path*] [*other-keyargs*])

▷ Pathname $_f$**compile-file** writes to if invoked with the same arguments.

($_f$**load** *path* {‖ **:verbose** *bool*$_{\boxed{v\text{*load-verbose*}}}$ | **:print** *bool*$_{\boxed{v\text{*load-print*}}}$ | **:if-does-not-exist** *bool*$_{\boxed{\text{T}}}$ | **:external-format** *file-format*$_{\boxed{\text{:default}}}$ })

▷ Load source file or compiled file into Lisp environment. Return T if successful.

{$_v$***compile-file** | $_v$***load**}-{**pathname***$_{\boxed{\text{NIL}}}$ | **truename***$_{\boxed{\text{NIL}}}$}

▷ Input file used by $_f$**compile-file**/by $_f$**load**.

{$_v$***compile** | $_v$***load**}-{**print*** | **verbose***}

▷ Defaults used by $_f$**compile-file**/by $_f$**load**.

({$_f$**butlast** *list* | $_f$**nbutlast** $\widetilde{list}$} [*num*$_{\boxed{1}}$]) ▷ list excluding last *num* conses.

({$_f$**rplaca** | $_f$**rplacd**} $\widetilde{cons}$ *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* {‖ {**:test** *function*$_{\boxed{\text{\#'eql}}}$ | **:test-not** *function*} | **:key** *function* })

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

($_m$**pop** $\widetilde{place}$)

▷ Set *place* to ($_f$**cdr** *place*), return ($_f$**car** *place*).

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

($_m$**pushnew** *foo* $\widetilde{place}$ {‖ {**:test** *function*$_{\boxed{\text{\#'eql}}}$ | **:test-not** *function*} | **:key** *function* })

▷ 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*.

({$_f$**mapcar** | $_f$**maplist**} *function list*$^+$)

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

({$_f$**mapcan** | $_f$**mapcon**} *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.

({$_f$**mapc** | $_f$**mapl**} *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.

({$_f$**assoc** | $_f$**rassoc**} *foo alist* {‖ {**:test** *test*$_{\boxed{\text{\#'eql}}}$ | **:test-not** *test*} | **:key** *function* })

({$_f$**assoc-if**[**-not**] | $_f$**rassoc-if**[**-not**]} *test alist* [**:key** *function*])

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

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

4.4 Trees

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

▷ Return $\underline{\texttt{T}}$ if trees *foo* and *bar* have same shape and leaves satisfying *test*.

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

▷ Make $\underline{\text{copy of } \textit{tree}}$ with each subtree or leaf matching *old* replaced by *new*.

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

▷ Make $\underline{\text{copy of } \textit{tree}}$ with each subtree or leaf satisfying *test* replaced by *new*.

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

▷ Make $\underline{\text{copy of } \textit{tree}}$ with each subtree or leaf matching a key in *association-list* replaced by that key's value.

($_f$**copy-tree** *tree*) ▷ $\underline{\text{Copy of } \textit{tree}}$ with same shape and leaves.

4.5 Sets

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

▷ Return $\underline{a \cap b}$, $\underline{a \setminus b}$, $\underline{a \cup b}$, or $\underline{a \bigtriangleup b}$, respectively, of lists *a* and *b*.

5 Arrays

5.1 Predicates

($_f$**arrayp** *foo*)
($_f$**vectorp** *foo*)
($_f$**simple-vector-p** *foo*) ▷ $\underline{\texttt{T}}$ 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*)

▷ $\underline{\texttt{T}}$ if *array* is adjustable/has a fill pointer, respectively.

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

▷ Return $\underline{\texttt{T}}$ if *subscripts* are in *array*'s bounds.

5.2 Array Functions

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

▷ Return fresh, or readjust, respectively, $\underline{\text{vector}}$ or $\underline{\text{array}}$.

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

▷ Return $\underline{\text{array element}}$ pointed to by *subscripts*. **setf**able.

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

▷ Return $\underline{i\text{th element}}$ of *array* in row-major order. **setf**able.

($\left\{\begin{array}{l}{}_f\textbf{import}\\ {}_f\textbf{shadowing-import}\end{array}\right\}$ *symbols* [*package*$_{\boxed{_v\textbf{*package*}}}$])

▷ 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 $\underline{\texttt{T}}$.

($\left\{\begin{array}{l}\left.\begin{array}{l}{}_m\textbf{do-symbols}\\ {}_m\textbf{do-external-symbols}\end{array}\right\}\ (\widehat{\textit{var}}\ [\textit{package}_{\boxed{_v\textbf{*package*}}}\ [\textit{result}_{\boxed{\texttt{NIL}}}]])\\ {}_m\textbf{do-all-symbols}\ (\textit{var}\ [\textit{result}_{\boxed{\texttt{NIL}}}])\end{array}\right\}$
(**declare** $\widehat{\textit{decl}}$*)* $\left\{\left|\begin{array}{l}\widehat{\textit{tag}}\\ \textit{form}\end{array}\right.\right\}$*)

▷ 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 $\underline{\text{values of } \textit{result}}$. Implicitly, the whole form is a $_s$**block** named NIL.

($_m$**with-package-iterator** (*foo packages* [**:internal**|**:external**|**:inherited**])
(**declare** $\widehat{\textit{decl}}$*)* *form*^{P_*})

▷ Return $\underline{\text{values of } \textit{forms}}$. 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 $\underline{\text{symbol } \textit{name}}$.

($_f$**gensym** [$s_{\boxed{\text{G}}}$])

▷ Return fresh, uninterned symbol $\underline{\#\text{:}sn}$ with *n* from $_v$***gensym-counter***. Increment $_v$***gensym-counter***.

($_f$**gentemp** [*prefix*$_{\boxed{\texttt{T}}}$ [*package*$_{\boxed{_v\textbf{*package*}}}$]])

▷ Intern fresh $\underline{\text{symbol}}$ in $\underline{\text{package}}$. Deprecated.

($_f$**copy-symbol** *symbol* [*props*$_{\boxed{\texttt{NIL}}}$])

▷ Return uninterned $\underline{\text{copy of } \textit{symbol}}$. If *props* is T, give copy the same value, function and property list.

($_f$**symbol-name** *symbol*)
($_f$**symbol-package** *symbol*)

▷ $\underline{\text{Name}}$ or $\underline{\text{package}}$, respectively, of *symbol*.

($_f$**symbol-plist** *symbol*)
($_f$**symbol-value** *symbol*)
($_f$**symbol-function** *symbol*)

▷ $\underline{\text{Property list}}$, $\underline{\text{value}}$, or $\underline{\text{function}}$, respectively, of *symbol*. **setf**able.

14 Packages and Symbols

The Loop Facility provides additional means of symbol handling; see **loop**, page 21.

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*.

$$({}_m\textbf{defpackage}\ \textit{foo}\ \left\{\left|\begin{array}{l}(\textbf{:nicknames}\ \textit{nick}^*)^*\\(\textbf{:documentation}\ \textit{string})\\(\textbf{:intern}\ \textit{interned-symbol}^*)^*\\(\textbf{:use}\ \textit{used-package}^*)^*\\(\textbf{:import-from}\ \textit{pkg}\ \textit{imported-symbol}^*)^*\\(\textbf{:shadowing-import-from}\ \textit{pkg}\ \textit{shd-symbol}^*)^*\\(\textbf{:shadow}\ \textit{shd-symbol}^*)^*\\(\textbf{:export}\ \textit{exported-symbol}^*)^*\\(\textbf{:size}\ \textit{int})\end{array}\right.\right\})$$

▷ Create or modify package *foo* with *interned-symbol*s, symbols from *used-package*s, *imported-symbol*s, and *shd-symbol*s. Add *shd-symbols* to *foo*'s shadowing list.

$$({}_f\textbf{make-package}\ \textit{foo}\ \left\{\left|\begin{array}{l}\textbf{:nicknames}\ (\textit{nick}^*)_{\boxed{\texttt{NIL}}}\\\textbf{:use}\ (\textit{used-package}^*)\end{array}\right.\right\})$$

▷ 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.

($\left\{\begin{array}{l}{}_f\textbf{use-package}\\{}_f\textbf{unuse-package}\end{array}\right\}$ *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*) ▷ Package with *name* (case-sensitive).

($_f$**find-all-symbols** *foo*)
▷ List of symbols *foo* from all registered packages.

($\left\{\begin{array}{l}{}_f\textbf{intern}\\{}_f\textbf{find-symbol}\end{array}\right\}$ *foo* [*package*$_{\boxed{{}_v\textbf{*package*}}}$])
▷ Intern or find, respectively, symbol *foo* in *package*. Second return value is one of $\underset{2}{\textbf{:internal}}$, $\underset{2}{\textbf{:external}}$, or $\underset{2}{\textbf{:inherited}}$ (or $\underset{2}{\texttt{NIL}}$ if $_f$**intern** has created a fresh symbol).

($_f$**unintern** *symbol* [*package*$_{\boxed{{}_v\textbf{*package*}}}$])
▷ Remove *symbol* from *package*, return T on success.

($_f$**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*) ▷ Target array and $\underset{2}{\text{offset}}$.

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

($_f$**bit-not** $\widetilde{bit\text{-}array}$ [$\widetilde{result\text{-}bit\text{-}array}_{\boxed{\texttt{NIL}}}$])
▷ Return 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.

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

▷ Return result of bitwise logical operations (cf. operations of $_f$**boole**, page 4) 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 simple vector of *foo*s.

($_f$**svref** *vector i*) ▷ Element *i* of simple *vector*. **setf**able.

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

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

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

($_f$**fill-pointer** *vector*) ▷ Fill pointer of *vector*. **setf**able.

6 Sequences

6.1 Sequence Predicates

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

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

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

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

(${}_f$**mismatch** *sequence-a sequence-b* $\left\{\begin{array}{l}\textbf{:from-end}\ bool_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test}\ function_{\boxed{\texttt{\#'eql}}}\\ \textbf{:test-not}\ function\end{array}\right.\\ \textbf{:start1}\ start\text{-}a_{\boxed{0}}\\ \textbf{:start2}\ start\text{-}b_{\boxed{0}}\\ \textbf{:end1}\ end\text{-}a_{\boxed{\texttt{NIL}}}\\ \textbf{:end2}\ end\text{-}b_{\boxed{\texttt{NIL}}}\\ \textbf{:key}\ function\end{array}\right\}$)

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

6.2 Sequence Functions

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

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

(${}_f$**concatenate** *type sequence**)

▷ Return concatenated sequence of *type*.

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

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

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

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

(${}_f$**length** *sequence*)

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

(${}_f$**count** *foo sequence* $\left\{\begin{array}{l}\textbf{:from-end}\ bool_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test}\ function_{\boxed{\texttt{\#'eql}}}\\ \textbf{:test-not}\ function\end{array}\right.\\ \textbf{:start}\ start_{\boxed{0}}\\ \textbf{:end}\ end_{\boxed{\texttt{NIL}}}\\ \textbf{:key}\ function\end{array}\right\}$)

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

($\left\{\begin{array}{l}{}_f\textbf{count-if}\\ {}_f\textbf{count-if-not}\end{array}\right\}$ *test sequence* $\left\{\begin{array}{l}\textbf{:from-end}\ bool_{\boxed{\texttt{NIL}}}\\ \textbf{:start}\ start_{\boxed{0}}\\ \textbf{:end}\ end_{\boxed{\texttt{NIL}}}\\ \textbf{:key}\ function\end{array}\right\}$)

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

(${}_f$**elt** *sequence index*)

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

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

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

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

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

(${}_f$**reverse** *sequence*)
(${}_f$**nreverse** $\widetilde{sequence}$) ▷ Return *sequence* in reverse order.

(${}_f$**parse-namestring** *foo* [*host*
[*default-pathname*$_{\boxed{{}_v\textbf{*default-pathname-defaults*}}}$
$\left\{\begin{array}{l}\textbf{:start}\ start_{\boxed{0}}\\ \textbf{:end}\ end_{\boxed{\texttt{NIL}}}\\ \textbf{:junk-allowed}\ bool_{\boxed{\texttt{NIL}}}\end{array}\right\}$]]])

▷ Return pathname converted from string, pathname, or stream *foo*; and 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 "[*host*:][;]{$\left\{\begin{array}{l}\{dir|\texttt{*}\}^+\\ \texttt{**}\end{array}\right\}$;}* {*name*|*}* [.$\left\{\begin{array}{l}\{type|\texttt{*}\}^+\\ \texttt{LISP}\end{array}\right\}$[.{*version* |*|newest|NEWEST}]]".

(${}_f$**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, old physical file name$_2$, and 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 T$_2$ if something has been created.

($_f$**close** $\widetilde{stream}$ [**:abort** $bool_{\boxed{\texttt{NIL}}}$])

▷ Close *stream*. Return $\underline{\texttt{T}}$ if *stream* had been open. If **:abort** is T, delete associated file.

($_m$**with-open-file** (*stream path open-arg**) (**declare** $\widehat{decl}$*)* $form^{P_*}$)

▷ Use $_f$**open** with *open-arg*s to temporarily create *stream* to *path*; return <u>values of *forms*</u>.

($_m$**with-open-stream** (*foo* $\widetilde{stream}$) (**declare** $\widehat{decl}$*)* $form^{P_*}$)

▷ Evaluate *forms* with *foo* locally bound to *stream*. Return <u>values of *forms*</u>.

($_m$**with-input-from-string** (*foo string* {|**:index** $\widetilde{index}$ | **:start** $start_{\boxed{0}}$ | **:end** $end_{\boxed{\texttt{NIL}}}$|}) (**declare** $\widehat{decl}$*)* $form^{P_*}$)

▷ Evaluate *forms* with *foo* locally bound to input **string-stream** from *string*. Return <u>values of *forms*</u>; store next reading position into *index*.

($_m$**with-output-to-string** (*foo* $\widetilde{[string_{\boxed{\texttt{NIL}}}}$ [**:element-type** $type_{\boxed{\textsf{character}}}$]]) (**declare** $\widehat{decl}$*)* $form^{P_*}$)

▷ Evaluate *forms* with *foo* locally bound to an output **string-stream**. Append output to *string* and return <u>values of *forms*</u> if *string* is given. Return <u>string containing output</u> otherwise.

($_f$**stream-external-format** *stream*)

▷ <u>External file format designator</u>.

$_v$***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

($_f$**make-pathname**
{|
:host {*host*|NIL|**:unspecific**}
:device {*device*|NIL|**:unspecific**}
:directory { {*directory*|**:wild**|NIL|**:unspecific**} | ({**:absolute** | **:relative**} {*directory* | **:wild** | **:wild-inferiors** | **:up** | **:back**}*) }
:name {*file-name*|**:wild**|NIL|**:unspecific**}
:type {*file-type*|**:wild**|NIL|**:unspecific**}
:version {**:newest**|*version*|**:wild**|NIL|**:unspecific**}
:defaults $path_{\boxed{\text{host from }{}_v\textbf{*default-pathname-defaults*}}}$
:case {**:local**|**:common**}$_{\boxed{\textbf{:local}}}$
|})

▷ Construct a <u>logical pathname</u> if there is a logical pathname translation for *host*, otherwise construct a <u>physical pathname</u>. 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.

({$_f$**pathname-host** | $_f$**pathname-device** | $_f$**pathname-directory** | $_f$**pathname-name** | $_f$**pathname-type**} *path-or-stream* [**:case** {**:local** | **:common**}$_{\boxed{\textbf{:local}}}$])

($_f$**pathname-version** *path-or-stream*)

▷ Return <u>pathname component</u>.

({$_f$**find** | $_f$**position**} *foo sequence* {|**:from-end** $bool_{\boxed{\texttt{NIL}}}$ | {**:test** $function_{\boxed{\#\textbf{'eql}}}$ | **:test-not** *test*} | **:start** $start_{\boxed{0}}$ | **:end** $end_{\boxed{\texttt{NIL}}}$ | **:key** *function*|})

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

({$_f$**find-if** | $_f$**find-if-not** | $_f$**position-if** | $_f$**position-if-not**} *test sequence* {|**:from-end** $bool_{\boxed{\texttt{NIL}}}$ | **:start** $start_{\boxed{0}}$ | **:end** $end_{\boxed{\texttt{NIL}}}$ | **:key** *function*|})

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

($_f$**search** *sequence-a sequence-b* {|**:from-end** $bool_{\boxed{\texttt{NIL}}}$ | {**:test** $function_{\boxed{\#\textbf{'eql}}}$ | **:test-not** *function*} | **:start1** $start\text{-}a_{\boxed{0}}$ | **:start2** $start\text{-}b_{\boxed{0}}$ | **:end1** $end\text{-}a_{\boxed{\texttt{NIL}}}$ | **:end2** $end\text{-}b_{\boxed{\texttt{NIL}}}$ | **:key** *function*|})

▷ Search *sequence-b* for a subsequence matching *sequence-a*. Return <u>position</u> in *sequence-b*, or <u>NIL</u>.

({$_f$**remove** *foo sequence* | $_f$**delete** *foo* $\widetilde{sequence}$} {|**:from-end** $bool_{\boxed{\texttt{NIL}}}$ | {**:test** $function_{\boxed{\#\textbf{'eql}}}$ | **:test-not** *function*} | **:start** $start_{\boxed{0}}$ | **:end** $end_{\boxed{\texttt{NIL}}}$ | **:key** *function* | **:count** $count_{\boxed{\texttt{NIL}}}$|})

▷ Make <u>copy of *sequence*</u> without elements matching *foo*.

({ {$_f$**remove-if** | $_f$**remove-if-not**} *test sequence* | {$_f$**delete-if** | $_f$**delete-if-not**} *test* $\widetilde{sequence}$ } {|**:from-end** $bool_{\boxed{\texttt{NIL}}}$ | **:start** $start_{\boxed{0}}$ | **:end** $end_{\boxed{\texttt{NIL}}}$ | **:key** *function* | **:count** $count_{\boxed{\texttt{NIL}}}$|})

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

({$_f$**remove-duplicates** *sequence* | $_f$**delete-duplicates** $\widetilde{sequence}$} {|**:from-end** $bool_{\boxed{\texttt{NIL}}}$ | {**:test** $function_{\boxed{\#\textbf{'eql}}}$ | **:test-not** *function*} | **:start** $start_{\boxed{0}}$ | **:end** $end_{\boxed{\texttt{NIL}}}$ | **:key** *function*|})

▷ Make <u>copy of *sequence*</u> without duplicates.

({$_f$**substitute** *new old sequence* | $_f$**nsubstitute** *new old* $\widetilde{sequence}$} {|**:from-end** $bool_{\boxed{\texttt{NIL}}}$ | {**:test** $function_{\boxed{\#\textbf{'eql}}}$ | **:test-not** *function*} | **:start** $start_{\boxed{0}}$ | **:end** $end_{\boxed{\texttt{NIL}}}$ | **:key** *function* | **:count** $count_{\boxed{\texttt{NIL}}}$|})

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

({ {$_f$**substitute-if** | $_f$**substitute-if-not**} *new test sequence* | {$_f$**nsubstitute-if** | $_f$**nsubstitute-if-not**} *new test* $\widetilde{sequence}$ } {|**:from-end** $bool_{\boxed{\texttt{NIL}}}$ | **:start** $start_{\boxed{0}}$ | **:end** $end_{\boxed{\texttt{NIL}}}$ | **:key** *function* | **:count** $count_{\boxed{\texttt{NIL}}}$|})

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

($_f$**replace** $\widetilde{sequence\text{-}a}$ *sequence-b* {|**:start1** $start\text{-}a_{\boxed{0}}$ | **:start2** $start\text{-}b_{\boxed{0}}$ | **:end1** $end\text{-}a_{\boxed{\texttt{NIL}}}$ | **:end2** $end\text{-}b_{\boxed{\texttt{NIL}}}$|})

▷ Replace elements of <u>*sequence-a*</u> with elements of *sequence-b*.

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

▷ Apply *function* successively to corresponding elements of the *sequence*s. Return values as a <u>sequence</u> of *type*. If *type* is NIL, return <u>NIL</u>.

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

▷ Store into <u>*result-sequence*</u> successively values of *function* applied to corresponding elements of the *sequence*s.

($_f$**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 <u>last value</u> of function.

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

▷ <u>Copy of *sequence*</u> with shared elements.

7 Hash Tables

The Loop Facility provides additional hash table-related functionality; see **loop**, page 21.

Key-value storage similar to hash tables can as well be achieved using association lists and property lists; see pages 9 and 16.

($_f$**hash-table-p** *foo*) ▷ Return <u>T</u> if *foo* is of type **hash-table**.

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

▷ Make a <u>hash table</u>.

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

▷ Return <u>object</u> with *key* if any or <u>*default*</u> otherwise; and $\underline{\texttt{T}}_2$ if found, $\underline{\texttt{NIL}}_2$ otherwise. **setf**able.

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

▷ <u>Number of entries</u> in *hash-table*.

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

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

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

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

▷ Iterate over *hash-table* calling *function* on key and value. Return <u>NIL</u>.

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

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

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

▷ <u>Test function</u> 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 <u>size</u>, <u>rehash-size</u>, or <u>rehash-threshold</u>, respectively, as used in $_f$**make-hash-table**.

($_f$**sxhash** *foo*)

▷ <u>Hash code</u> unique for any argument $_f$**equal** *foo*.

13.6 Streams

($_f$**open** *path* $\left\{\begin{array}{l}\textbf{:direction } \left\{\begin{array}{l}\textbf{:input}\\ \textbf{:output}\\ \textbf{:io}\\ \textbf{:probe}\end{array}\right\}_{\boxed{\textbf{:input}}}\\ \textbf{:element-type } \left\{\begin{array}{l}\textit{type}\\ \textbf{:default}\end{array}\right\}_{\boxed{\textbf{character}}}\\ \textbf{:if-exists } \left\{\begin{array}{l}\textbf{:new-version}\\ \textbf{:error}\\ \textbf{:rename}\\ \textbf{:rename-and-delete}\\ \textbf{:overwrite}\\ \textbf{:append}\\ \textbf{:supersede}\\ \texttt{NIL}\end{array}\right\}_{\boxed{\textbf{:new-version} \text{ if } \textit{path} \text{ specifies } \textbf{:newest}\text{; } \texttt{NIL} \text{ otherwise}}}\\ \textbf{:if-does-not-exist } \left\{\begin{array}{l}\textbf{:error}\\ \textbf{:create}\\ \texttt{NIL}\end{array}\right\}_{\boxed{\texttt{NIL} \text{ for } \textbf{:direction :probe}\text{; } \{\textbf{:create}|\textbf{:error}\} \text{ otherwise}}}\\ \textbf{:external-format } \textit{format}_{\boxed{\textbf{:default}}}\end{array}\right\}$)

▷ Open <u>**file-stream** to *path*</u>.

($_f$**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 <u>stream</u> of indicated type.

($_f$**make-string-input-stream** *string* [*start*$_{\boxed{0}}$ [*end*$_{\boxed{\texttt{NIL}}}$]])

▷ Return a <u>**string-stream**</u> supplying the characters from *string*.

($_f$**make-string-output-stream** [**:element-type** *type*$_{\boxed{\textbf{character}}}$])

▷ Return a <u>**string-stream**</u> accepting characters (available via $_f$**get-output-stream-string**).

($_f$**concatenated-stream-streams** *concatenated-stream*)
($_f$**broadcast-stream-streams** *broadcast-stream*)

▷ Return <u>list of streams</u> *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 <u>source stream</u> or <u>sink stream</u> of *two-way-stream*/ *echo-stream*, respectively.

($_f$**synonym-stream-symbol** *synonym-stream*)

▷ Return <u>symbol</u> of *synonym-stream*.

($_f$**get-output-stream-string** $\widetilde{\textit{string-stream}}$)

▷ Clear and return as a <u>string</u> characters on *string-stream*.

($_f$**file-position** *stream* [$\left\{\begin{array}{l}\textbf{:start}\\ \textbf{:end}\\ \textit{position}\end{array}\right\}$])

▷ Return <u>position within stream</u>, or set it to <u>*position*</u> and return <u>T</u> on success.

($_f$**file-string-length** *stream foo*)

▷ <u>Length</u> *foo* would have in *stream*.

($_f$**listen** [*stream*$_{\boxed{{}_v\textbf{*standard-input*}}}$])

▷ <u>T</u> if there is a character in input *stream*.

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

▷ Clear input from *stream*, return <u>NIL</u>.

($\left\{\begin{array}{l}{}_f\textbf{clear-output}\\ {}_f\textbf{force-output}\\ {}_f\textbf{finish-output}\end{array}\right\}$ [$\widetilde{\textit{stream}}_{\boxed{{}_v\textbf{*standard-output*}}}$])

▷ End output to *stream* and return <u>NIL</u> immediately, after initiating flushing of buffers, or after flushing of buffers, respectively.

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

▷ **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.

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

▷ **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, atsign-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.

8 Structures

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

▷ 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 17) 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*) ▷ T if *foo* is `NIL`; `NIL` otherwise.

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

(${}_f$**constantp** *foo* [*environment*$_{\boxed{\texttt{NIL}}}$])

▷ T if *foo* is a constant form.

($_f$**functionp** *foo*) ▷ T if *foo* is of type **function**.

($_f$**fboundp** $\left\{\begin{array}{l}foo\\ (\textbf{setf}\ foo)\end{array}\right\}$) ▷ 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 *foo*.

($_m$**defvar** $\widehat{foo}$ [*form* [$\widehat{doc}$]])
▷ Unless bound already, assign value of *form* to dynamic variable *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 values of last *form*/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 value of last *form*/NIL; work sequentially/in parallel, respectively.

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

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

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

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

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

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

($_f$**get-properties** *property-list keys*)
▷ Return key and $\underset{2}{\text{value}}$ of first entry from *property-list* matching a key from *keys*, and $\underset{3}{\text{tail of } property\text{-}list}$ starting with that key. Return NIL, $\underset{2}{\texttt{NIL}}$, and $\underset{3}{\texttt{NIL}}$ 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 T if *key* was there, or NIL otherwise.

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

($\left\{\begin{array}{l}{}_s\textbf{let}\\ {}_s\textbf{let*}\end{array}\right\}$ ($\left\{\begin{array}{l}name\\ (name\ [value_{\boxed{\texttt{NIL}}}])\end{array}\right\}^{*}$) (**declare** $\widehat{decl}^{*}$)* $form^{P_*}$)
▷ Evaluate *form*s with *name*s lexically bound (in parallel or sequentially, respectively) to *value*s. Return values of *forms*.

($_m$**multiple-value-bind** ($\widehat{var}^{*}$) *values-form* (**declare** $\widehat{decl}^{*}$)* $body\text{-}form^{P_*}$)
▷ Evaluate *body-form*s with *var*s lexically bound to the return values of *values-form*. Return values of *body-forms*.

~ [*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 nonprinting) 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.

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

▷ Print a conditional newline if *stream* is a pretty printing stream. Return $\underline{\texttt{NIL}}$.

$_v\textbf{*print-array*}$ ▷ If T, print arrays $_f$**read**ably.

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

$_v\textbf{*print-case*}_{\boxed{\textbf{:upcase}}}$

▷ Print symbol names all uppercase (**:upcase**), all lowercase (**:downcase**), capitalized (**:capitalize**).

$_v\textbf{*print-circle*}_{\boxed{\texttt{NIL}}}$

▷ If T, avoid indefinite recursion while printing circular structure.

$_v\textbf{*print-escape*}_{\boxed{\texttt{T}}}$

▷ If NIL, do not print escape characters and package prefixes.

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

$_v\textbf{*print-length*}_{\boxed{\texttt{NIL}}}$
$_v\textbf{*print-level*}_{\boxed{\texttt{NIL}}}$
$_v\textbf{*print-lines*}_{\boxed{\texttt{NIL}}}$

▷ If integer, restrict printing of objects to that number of elements per level/to that depth/to that number of lines.

$_v\textbf{*print-miser-width*}$

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

$_v\textbf{*print-pretty*}$ ▷ If T, print prettily.

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

$_v\textbf{*print-readably*}_{\boxed{\texttt{NIL}}}$

▷ If T, print $_f$**read**ably or signal error **print-not-readable**.

$_v\textbf{*print-right-margin*}_{\boxed{\texttt{NIL}}}$

▷ Right margin width in ems while pretty-printing.

($_f$**set-pprint-dispatch** *type function* $[priority_{\boxed{0}}$ $[table_{\boxed{{}_v\textbf{*print-pprint-dispatch*}}}]]$)

▷ 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 $\underline{\texttt{NIL}}$.

($_f$**pprint-dispatch** *foo* $[table_{\boxed{{}_v\textbf{*print-pprint-dispatch*}}}]$)

▷ Return highest priority $\underline{function}$ associated with type of *foo* and $\underset{2}{\underline{\texttt{T}}}$ if there was a matching type specifier in *table*.

($_f$**copy-pprint-dispatch** $[table_{\boxed{{}_v\textbf{*print-pprint-dispatch*}}}]$)

▷ Return $\underline{\text{copy of } table}$ or, if *table* is NIL, initial value of $_v\textbf{*print-pprint-dispatch*}$.

$_v\textbf{*print-pprint-dispatch*}$ ▷ Current pretty print dispatch table.

13.5 Format

($_m$**formatter** $\widehat{control}$)

▷ Return $\underline{\text{function}}$ 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 $_v\textbf{*standard-output*}$. Return $\underline{\texttt{NIL}}$. If first argument is NIL, return $\underline{\text{formatted output}}$.

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

▷ Evaluate *forms* with variables from tree *destruct-λ* bound to corresponding elements of tree *bar*, and return $\underline{\text{their values}}$. *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

$$(var^*\ [\textbf{\&optional}\ \left\{\begin{array}{l}var\\ (var\ [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]])\end{array}\right\}^*]\ [\textbf{\&rest}\ var]$$
$$[\textbf{\&key}\ \left\{\begin{array}{l}var\\ (\left\{\begin{array}{l}var\\ (\textit{:key}\ var)\end{array}\right\}\ [init_{\boxed{\texttt{NIL}}}\ [supplied\text{-}p]])\end{array}\right\}^*\ [\textbf{\&allow-other-keys}]]$$
$$[\textbf{\&aux}\ \left\{\begin{array}{l}var\\ (var\ [init_{\boxed{\texttt{NIL}}}])\end{array}\right\}^*]).$$

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

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

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

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

▷ 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 $\underline{\text{values of } forms}$.

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

▷ Return lexically innermost $\underline{\text{function}}$ named *foo* or a lexical closure of the $\underline{{}_m\textbf{lambda}\text{ expression}}$.

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

▷ $\underline{\text{Values of } function}$ 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*) ▷ $\underline{\text{Values of } function}$ called with *args*.

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

▷ Call *function* with all the values of each *form* as its arguments. Return $\underline{\text{values returned by } function}$.

($_f$**values-list** *list*) ▷ Return $\underline{\text{elements of } list}$.

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

▷ Return as multiple values the $\underline{\text{primary values}}$ of the *foo*s. **setf**able.

($_f$**multiple-value-list** *form*) ▷ $\underline{\text{List of the values of } form}$.

($_m$**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*, $\underline{\texttt{NIL}}_2$ if *function* was defined in an environment without bindings, and $\underline{\text{name}}_3$ of *function*.

($_f$**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

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

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

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

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

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

or

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

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

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.

$$(\begin{Bmatrix} {}_m\textbf{defmacro} \\ {}_m\textbf{define-compiler-macro} \end{Bmatrix} \begin{Bmatrix} foo \\ (\textbf{setf}\ foo) \end{Bmatrix}\ (macro\text{-}\lambda^*)$$
$$\begin{Bmatrix} \begin{vmatrix} (\textbf{declare}\ \widehat{decl}^*)^* \\ \widehat{doc} \end{vmatrix} \end{Bmatrix}\ form^{P*})$$

▷ Define macro $\underline{foo}$ which on evaluation as (*foo tree*) applies expanded *forms* to arguments from *tree*, which corresponds to *tree*-shaped *macro-λ*s. *forms* 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*.

$$({}_s\textbf{macrolet}\ ((foo\ (macro\text{-}\lambda^*)\ \begin{Bmatrix} \begin{vmatrix} (\textbf{declare}\ \widehat{local\text{-}decl}^*)^* \\ \widehat{doc} \end{vmatrix} \end{Bmatrix}$$
$$macro\text{-}form^{P*})^*)\ (\textbf{declare}\ \widehat{decl}^*)^*\ form^{P*})$$

▷ 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*.

$$({}_m\textbf{defsetf}\ \widehat{function} \begin{Bmatrix} \widehat{updater}\ [\widehat{doc}] \\ (setf\text{-}\lambda^*)\ (s\text{-}var^*)\ \begin{Bmatrix} \begin{vmatrix} (\textbf{declare}\ \widehat{decl}^*)^* \\ \widehat{doc} \end{vmatrix} \end{Bmatrix}\ form^{P*} \end{Bmatrix})$$

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

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

▷ Output $\underline{char}$ to *stream*.

$$(\begin{Bmatrix} {}_f\textbf{write-string} \\ {}_f\textbf{write-line} \end{Bmatrix}\ string\ [\widetilde{stream}_{\boxed{{}_v\textbf{*standard-output*}}}\ [\begin{Bmatrix} \begin{vmatrix} \textbf{:start}\ start_{\boxed{0}} \\ \textbf{:end}\ end_{\boxed{\texttt{NIL}}} \end{vmatrix} \end{Bmatrix}]])$$

▷ Write $\underline{string}$ to *stream* without/with a trailing newline.

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

$$({}_f\textbf{write-sequence}\ sequence\ \widetilde{stream}\ \begin{Bmatrix} \begin{vmatrix} \textbf{:start}\ start_{\boxed{0}} \\ \textbf{:end}\ end_{\boxed{\texttt{NIL}}} \end{vmatrix} \end{Bmatrix})$$

▷ Write elements of $\underline{sequence}$ to binary or character *stream*.

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

▷ Print *foo* to *stream* and return $\underline{foo}$, or print *foo* into $\underline{string}$, 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* [*parenthesis*$_{\boxed{\texttt{T}}}$ [*noop*]])
($_f$**pprint-tabular** $\widetilde{stream}$ *foo* [*parenthesis*$_{\boxed{\texttt{T}}}$ [*noop* [$n_{\boxed{16}}$]]])
($_f$**pprint-linear** $\widetilde{stream}$ *foo* [*parenthesis*$_{\boxed{\texttt{T}}}$ [*noop*]])

▷ 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 $\underline{\texttt{NIL}}$. Usable with $_f$**format** directive ~//.

$$({}_m\textbf{pprint-logical-block}\ (\widetilde{stream}\ list\ \begin{Bmatrix} \begin{vmatrix} \begin{Bmatrix} \textbf{:prefix}\ string \\ \textbf{:per-line-prefix}\ string \end{Bmatrix} \\ \textbf{:suffix}\ string_{\boxed{\texttt{""}}} \end{vmatrix} \end{Bmatrix})$$

(**declare** $\widehat{decl}$*)* $form^{P*}$)

▷ 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 $\underline{\texttt{NIL}}$.

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

▷ Take $\underline{\text{next element}}$ 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*.

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

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

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

▷ Specify indentation for innermost logical block relative to leftmost position/to current position. Return $\underline{\texttt{NIL}}$.

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

▷ If *list* is empty, terminate logical block. Return $\underline{\texttt{NIL}}$ otherwise.

n/*d* ▷ The **ratio** $\frac{n}{d}$.

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

#C(*a b***)** ▷ ($_f$**complex** *a b*), the complex number $a + b$i.

#'*foo* ▷ ($_s$**function** *foo*); the function named *foo*.

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

#[*n*]**(***foo****)**
▷ Vector of some (or *n*) *foo*s filled with last *foo* if necessary.

#[*n*]******b**
▷ Bit vector of some (or *n*) *b*s filled with last *b* if necessary.

#S(*type* {*slot value*}***)** ▷ Structure of *type*.

#P*string* ▷ A pathname.

#:*foo* ▷ Uninterned symbol *foo*.

#.*form* ▷ Read-time value of *form*.

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

#*integer***=** *foo* ▷ Give *foo* the label *integer*.

#*integer***#** ▷ Object labelled *integer*.

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

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

$_v$***features***
▷ List of symbols denoting implementation-dependent features.

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

13.4 Printer

($\left\{\begin{array}{l}{}_f\textbf{prin1}\\{}_f\textbf{print}\\{}_f\textbf{pprint}\\{}_f\textbf{princ}\end{array}\right\}$ *foo* [$\widetilde{stream}_{\boxed{_v\textbf{*standard-output*}}}$])
▷ 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.

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

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

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

($_f$**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.

(*var** [**&optional** $\left\{\begin{array}{l}var\\(var\ [init_{\boxed{\mathtt{NIL}}}\ [supplied\text{-}p]])\end{array}\right\}^*$] [**&rest** *var*]
[**&key** $\left\{\begin{array}{l}var\\(\left\{\begin{array}{l}var\\(\textit{:key var})\end{array}\right\}\ [init_{\boxed{\mathtt{NIL}}}\ [supplied\text{-}p]])\end{array}\right\}^*$
[**&allow-other-keys**]] [**&environment** *var*])
▷ 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*$_{\boxed{\mathtt{NIL}}}$])
▷ 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* ([**&optional** $\left\{\begin{array}{l}var\\(var\ [init_{\boxed{\mathtt{NIL}}}\ [supplied\text{-}p]])\end{array}\right\}^*$] [**&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

($_s$**if** *test then* [*else*$_{\boxed{\mathtt{NIL}}}$])
▷ Return values of $\underline{then}$ if *test* returns T; return values of $\underline{else}$ otherwise.

($_m$**cond** (*test then*$^{P*}_{\boxed{test}}$)*)
▷ Return the $\underline{\text{values}}$ of the first *then** whose *test* returns T; return $\underline{\mathtt{NIL}}$ if all *test*s return NIL.

($\left\{\begin{array}{l}{}_m\textbf{when}\\{}_m\textbf{unless}\end{array}\right\}$ *test foo*^{P*})
▷ Evaluate *foo*s and return $\underline{\text{their values}}$ if *test* returns T or NIL, respectively. Return $\underline{\mathtt{NIL}}$ otherwise.

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

▷ 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*.

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

▷ 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*.

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

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

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

▷ Evaluate *forms* 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`.

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

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

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

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

$(\left\{\begin{matrix}_m\textsf{prog}\\ _m\textsf{prog*}\end{matrix}\right\}\ (\left\{\begin{vmatrix}name\\ (name\ [value_{\boxed{\texttt{NIL}}}])\end{vmatrix}\right\}^*)\ (\textsf{declare}\ \widehat{decl^*})^*\ \left\{\begin{matrix}\widehat{tag}\\ form\end{matrix}\right\}^*)$

▷ Evaluate $_s$**tagbody**-like body with *names* 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`.

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

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

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

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

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

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

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

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

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

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

$(_s\textsf{catch}\ tag\ form^{\overset{P}{*}})$

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

$(_s\textsf{throw}\ tag\ form)$

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

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

$(_f\textsf{read-sequence}\ \widetilde{sequence}\ \widetilde{stream}\ [\textsf{:start}\ start_{\boxed{0}}][\textsf{:end}\ end_{\boxed{\texttt{NIL}}}])$

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

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

▷ Case sensitivity attribute (one of **:upcase**, **:downcase**, **:preserve**, **:invert**) of *readtable*. **setf**able.

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

▷ Return copy of *from-readtable*.

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

▷ Copy syntax of *from-char* to *to-readtable*. Return `T`.

$_v$***readtable*** ▷ Current readtable.

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

$_v$***read-default-float-format***$_{\boxed{\textsf{single-float}}}$

▷ Floating point format to use when not indicated in the number read.

$_v$***read-suppress***$_{\boxed{\texttt{NIL}}}$

▷ If `T`, reader is syntactically more tolerant.

$(_f\textsf{set-macro-character}\ char\ function\ \big[non\text{-}term\text{-}p_{\boxed{\texttt{NIL}}}\ [\widetilde{rt}_{\boxed{_v\textsf{*readtable*}}}]\big])$

▷ Make *char* a macro character associated with *function* of stream and *char*. Return `T`.

$(_f\textsf{get-macro-character}\ char\ [rt_{\boxed{_v\textsf{*readtable*}}}])$

▷ Reader macro function associated with *char*, and $\underset{2}{\texttt{T}}$ if *char* is a non-terminating macro character.

$(_f\textsf{make-dispatch-macro-character}\ char\ \big[non\text{-}term\text{-}p_{\boxed{\texttt{NIL}}}\ [rt_{\boxed{_v\textsf{*readtable*}}}]\big])$

▷ Make *char* a dispatching macro character. Return `T`.

$(_f\textsf{set-dispatch-macro-character}\ char\ sub\text{-}char\ function\ [\widetilde{rt}_{\boxed{_v\textsf{*readtable*}}}])$

▷ Make *function* of stream, *n*, *sub-char* a dispatch function of *char* followed by *n*, followed by *sub-char*. Return `T`.

$(_f\textsf{get-dispatch-macro-character}\ char\ sub\text{-}char\ [rt_{\boxed{_v\textsf{*readtable*}}}])$

▷ Dispatch function associated with *char* followed by *sub-char*.

13.3 Character Syntax

#| *multi-line-comment** |#
; *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.

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

" ▷ Begin and end of a string.

'*foo* ▷ ($_s$**quote** *foo*); *foo* unevaluated.

$`([foo]\ [,bar]\ [,@baz]\ [,.\widetilde{quux}]\ [bing])$

▷ 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*; $2 \le r \le 36$.

13 Input/Output

13.1 Predicates

($_f$**streamp** *foo*)
($_f$**pathnamep** *foo*)   ▷ $\underline{\texttt{T}}$ 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 $\underline{\texttt{T}}$ if *stream* is for input, for output, interactive, or open, respectively.

($_f$**pathname-match-p** *path wildcard*)
▷ $\underline{\texttt{T}}$ if *path* matches *wildcard*.

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

13.2 Reader

({$_f$**y-or-n-p** | $_f$**yes-or-no-p**} [*control arg**])
▷ Ask user a question and return $\underline{\texttt{T}}$ or $\underline{\texttt{NIL}}$ depending on their answer. See page 36, $_f$**format**, for *control* and *args*.

($_m$**with-standard-io-syntax** *form*$^{P}_{*}$)
▷ Evaluate *forms* with standard behaviour of reader and printer. Return $\underline{\text{values of } forms}$.

({$_f$**read** | $_f$**read-preserving-whitespace**} [$\widetilde{stream}_{\boxed{v\text{*standard-input*}}}$ [*eof-err*$_{\boxed{\texttt{T}}}$ [*eof-val*$_{\boxed{\texttt{NIL}}}$ [*recursive*$_{\boxed{\texttt{NIL}}}$]]]])
▷ Read printed representation of $\underline{\text{object}}$.

($_f$**read-from-string** *string* [*eof-error*$_{\boxed{\texttt{T}}}$ [*eof-val*$_{\boxed{\texttt{NIL}}}$ [{|**:start** *start*$_{\boxed{0}}$ | **:end** *end*$_{\boxed{\texttt{NIL}}}$ | **:preserve-whitespace** *bool*$_{\boxed{\texttt{NIL}}}$}]]])
▷ Return $\underline{\text{object}}$ read from string and zero-indexed $\underset{2}{\underline{\text{position}}}$ of next character.

($_f$**read-delimited-list** *char* [$\widetilde{stream}_{\boxed{v\text{*standard-input*}}}$ [*recursive*$_{\boxed{\texttt{NIL}}}$]])
▷ Continue reading until encountering *char*. Return $\underline{\text{list}}$ of objects read. Signal error if no *char* is found in stream.

($_f$**read-char** [$\widetilde{stream}_{\boxed{v\text{*standard-input*}}}$ [*eof-err*$_{\boxed{\texttt{T}}}$ [*eof-val*$_{\boxed{\texttt{NIL}}}$ [*recursive*$_{\boxed{\texttt{NIL}}}$]]]])
▷ Return $\underline{\text{next character}}$ from *stream*.

($_f$**read-char-no-hang** [$\widetilde{stream}_{\boxed{v\text{*standard-input*}}}$ [*eof-error*$_{\boxed{\texttt{T}}}$ [*eof-val*$_{\boxed{\texttt{NIL}}}$ [*recursive*$_{\boxed{\texttt{NIL}}}$]]]])
▷ $\underline{\text{Next character}}$ from *stream* or $\underline{\texttt{NIL}}$ if none is available.

($_f$**peek-char** [*mode*$_{\boxed{\texttt{NIL}}}$ [$\widetilde{stream}_{\boxed{v\text{*standard-input*}}}$ [*eof-error*$_{\boxed{\texttt{T}}}$ [*eof-val*$_{\boxed{\texttt{NIL}}}$ [*recursive*$_{\boxed{\texttt{NIL}}}$]]]]])
▷ 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.

($_f$**unread-char** *character* [$\widetilde{stream}_{\boxed{v\text{*standard-input*}}}$])
▷ Put last $_f$**read-char**ed *character* back into *stream*; return $\underline{\texttt{NIL}}$.

($_f$**read-byte** $\widetilde{stream}$ [*eof-err*$_{\boxed{\texttt{T}}}$ [*eof-val*$_{\boxed{\texttt{NIL}}}$]])
▷ Read $\underline{\text{next byte}}$ from binary *stream*.

($_f$**read-line** [$\widetilde{stream}_{\boxed{v\text{*standard-input*}}}$ [*eof-err*$_{\boxed{\texttt{T}}}$ [*eof-val*$_{\boxed{\texttt{NIL}}}$ [*recursive*$_{\boxed{\texttt{NIL}}}$]]]])
▷ Return a $\underline{\text{line of text}}$ from *stream* and $\underset{2}{\underline{\texttt{T}}}$ if line has been ended by end of file.

9.6 Iteration

({$_m$**do** | $_m$**do***} ({*var* | (*var* [*start* [*step*]])}*) (*stop result*$^{P}_{*}$) (**declare** $\widehat{decl}$*)* {$\widehat{tag}$ | *form*}*)
▷ 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 $\underline{\text{values of } result*}$. Implicitly, the whole form is a $_s$**block** named `NIL`.

($_m$**dotimes** (*var i* [*result*$_{\boxed{\texttt{NIL}}}$]) (**declare** $\widehat{decl}$*)* {$\widehat{tag}$|*form*}*)
▷ Evaluate $_s$**tagbody**-like body with *var* successively bound to integers from 0 to $i-1$. Upon evaluation of $\underline{result}$, *var* is *i*. Implicitly, the whole form is a $_s$**block** named `NIL`.

($_m$**dolist** (*var list* [*result*$_{\boxed{\texttt{NIL}}}$]) (**declare** $\widehat{decl}$*)* {$\widehat{tag}$|*form*}*)
▷ Evaluate $_s$**tagbody**-like body with *var* successively bound to the elements of *list*. Upon evaluation of $\underline{result}$, *var* is `NIL`. Implicitly, the whole form is a $_s$**block** named `NIL`.

9.7 Loop Facility

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

($_m$**loop** *clause**)
▷ **Loop Facility.** For Loop Facility keywords see below and Figure 1.

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

{**with** {*var-s* | (*var-s**)} [*d-type*] [= *foo*]}$^+$
{**and** {*var-p* | (*var-p**)} [*d-type*] [= *bar*]}*
where destructuring type specifier *d-type* has the form
{**fixnum**|**float**|`T`|`NIL`|{**of-type** {*type* | (*type**)}}}
▷ Initialize (possibly trees of) local variables *var-s* sequentially and *var-p* in parallel.

{{**for**|**as**} {*var-s* | (*var-s**)} [*d-type*]}$^+$ {**and** {*var-p* | (*var-p**)} [*d-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**.

{**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.

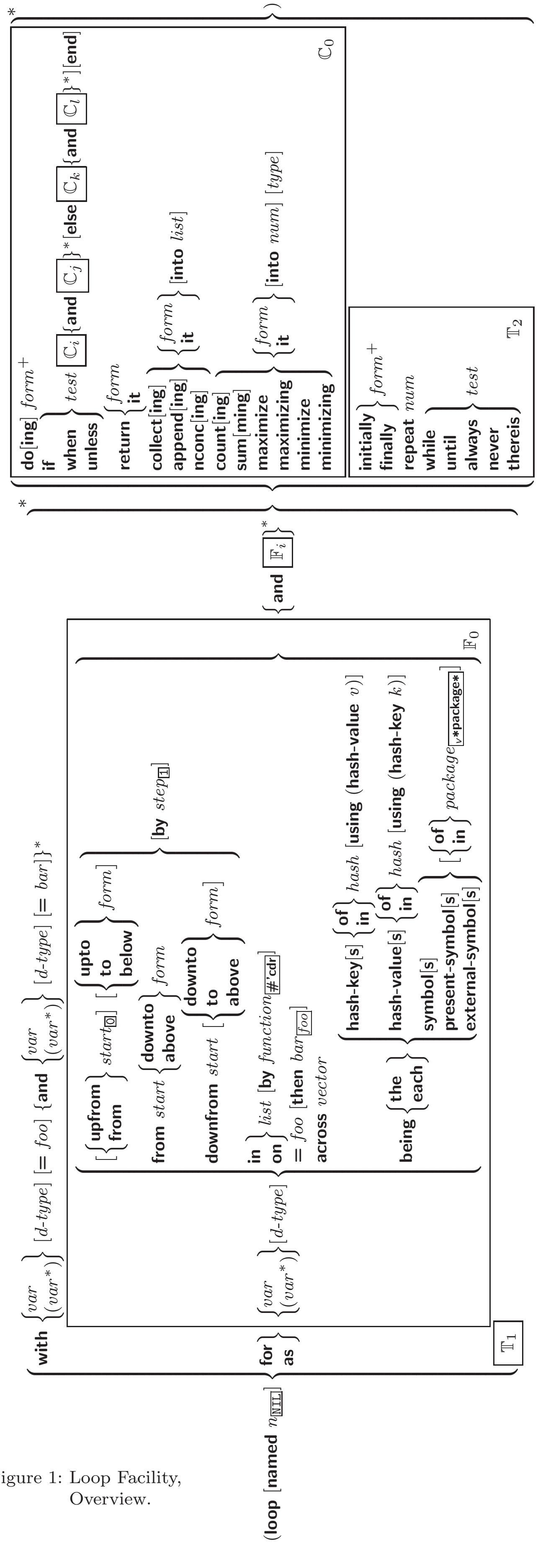
(loop [named n_NIL]
with {var | (var*)} [d-type] [= foo] {and {var | (var*)} [d-type] [= bar]}*
for | as {var | (var*)} [d-type]
[{upfrom | from} start_0] [{upto | to | below} form]
from start {downto | above} form
downfrom start [{downto | to | above} form]
[by step_1]
{in | on} list [by function_#'cdr]
= 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*]
{and F_i}*
F_0
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.

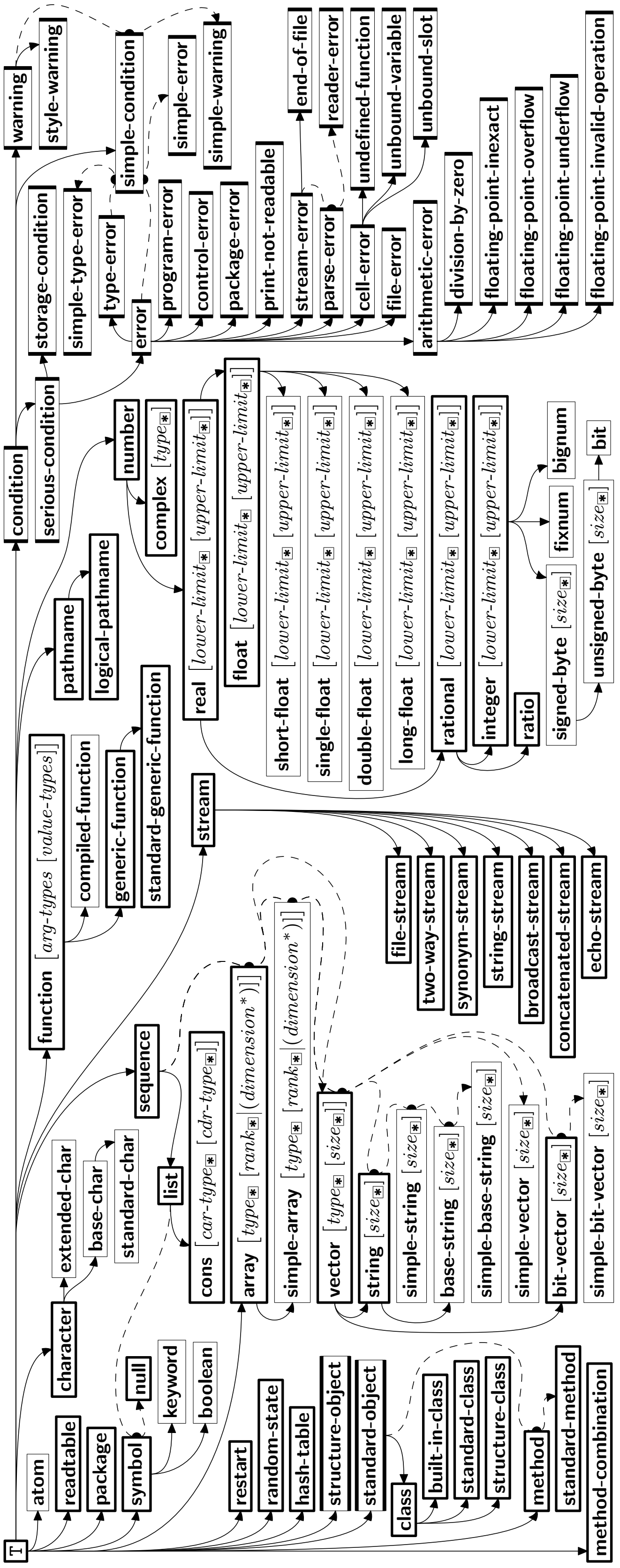

Figure 2: Precedence Order of System Classes (▭), Classes (▬), Types (▭), and Condition Types (▭).
Every type is also a supertype of NIL, the empty type.

$_v$***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*$_{\boxed{\texttt{NIL}}}$]) ▷ $\underline{\texttt{T}}$ if *foo* is of *type*.

($_f$**subtypep** *type-a type-b* [*environment*])
▷ Return $\underline{\texttt{T}}$ if *type-a* is a recognizable subtype of *type-b*, and $\underbrace{\texttt{NIL}}_{2}$ if the relationship could not be determined.

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

($_f$**coerce** *object type*) ▷ Coerce $\underline{object}$ into *type*.

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

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

($_f$**type-of** *foo*) ▷ $\underline{\text{Type of } foo}$.

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

($_f$**stream-element-type** *stream*) ▷ $\underline{\text{Type}}$ of *stream* objects.

($_f$**array-element-type** *array*) ▷ Element $\underline{\text{type}}$ *array* can hold.

($_f$**upgraded-array-element-type** *type* [*environment*$_{\boxed{\texttt{NIL}}}$])
▷ $\underline{\text{Element type}}$ 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 $\underline{foo}$ 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 18 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.

{**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 *forms* 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-clauses*; otherwise, evaluate *k-clause* and *l-clauses*.

it ▷ Inside *i-clause* or *k-clause*: $\underline{\text{value of } test}$.

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 *forms* 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.

(${}_m$**defclass** *foo* (*superclass**$_{\boxed{\text{standard-object}}}$)
$\left(\begin{Bmatrix} \textit{slot} \\ (\textit{slot} \left\{ \begin{vmatrix} \{\textbf{:reader}\ \textit{reader}\}^* \\ \{\textbf{:writer} \begin{Bmatrix} \textit{writer} \\ (\textbf{setf}\ \textit{writer}) \end{Bmatrix}\}^* \\ \{\textbf{:accessor}\ \textit{accessor}\}^* \\ \textbf{:allocation} \begin{Bmatrix} \textbf{:instance} \\ \textbf{:class} \end{Bmatrix}_{\boxed{\text{:instance}}} \\ \{\textbf{:initarg}\ [\textbf{:}]\textit{initarg-name}\}^* \\ \textbf{:initform}\ \textit{form} \\ \textbf{:type}\ \textit{type} \\ \textbf{:documentation}\ \textit{slot-doc} \end{vmatrix} \right\}) \end{Bmatrix}^* \right)$
$\left\{ \begin{vmatrix} (\textbf{:default-initargs}\ \{\textit{name value}\}^*) \\ (\textbf{:documentation}\ \textit{class-doc}) \\ (\textbf{:metaclass}\ \textit{name}_{\boxed{\text{standard-class}}}) \end{vmatrix} \right\})$

▷ Define or modify 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*). *slots* with **:allocation :class** are shared by all instances of class *foo*.

(${}_f$**find-class** *symbol* [*errorp*$_{\boxed{\texttt{T}}}$ [*environment*]])
▷ Return class named *symbol*. **setf**able.

(${}_g$**make-instance** *class* {[**:**]*initarg value*}* *other-keyarg**)
▷ Make new instance of *class*.

(${}_g$**reinitialize-instance** *instance* {[**:**]*initarg value*}* *other-keyarg**)
▷ Change local slots of *instance* according to *initarg*s by means of ${}_g$**shared-initialize**.

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

(${}_f$**slot-makunbound** *instance slot*)
▷ Make *slot* in *instance* unbound.

$\left(\begin{Bmatrix} {}_m\textbf{with-slots}\ (\{\widehat{\textit{slot}}|(\widehat{\textit{var}}\ \widehat{\textit{slot}})\}^*) \\ {}_m\textbf{with-accessors}\ ((\widehat{\textit{var}}\ \widehat{\textit{accessor}})^*) \end{Bmatrix}\right.$ *instance* (**declare** $\widehat{\textit{decl}}$*)*
form^{P_*})
▷ Return values of *forms* after evaluating them in a lexical environment with slots of *instance* visible as **setf**able *slots* or *vars*/with *accessor*s 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** $\widetilde{\textit{instance}}$ *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 *value*s of the corresponding *initarg*s 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**.

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

(${}_m$**restart-bind** (($\begin{Bmatrix} \widehat{\textit{restart}} \\ \texttt{NIL} \end{Bmatrix}$ *restart-function*
$\left\{\begin{vmatrix} \textbf{:interactive-function}\ \textit{arg-function} \\ \textbf{:report-function}\ \textit{report-function} \\ \textbf{:test-function}\ \textit{test-function} \end{vmatrix}\right\}$)*) *form*^{P_*})
▷ Return values of *forms* evaluated with dynamically established *restarts* whose *restart-functions* 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, *restarts* are described by *restart-function* (of a stream). A *restart* can be called by (**invoke-restart** *restart arg**), where *args* must be suitable for the corresponding *restart-function*, or by (**invoke-restart-interactively** *restart*) where a list of the respective *args* 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.

$\left(\begin{Bmatrix} {}_f\textbf{find-restart} \\ {}_f\textbf{compute-restarts}\ \textit{name} \end{Bmatrix}\right.$ [*condition*])
▷ Return innermost restart *name*, or a list of all restarts, respectively, out of those either associated with *condition* or un-associated at all; or, without *condition*, out of all restarts. Return NIL if search is unsuccessful.

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

$\left(\begin{Bmatrix} {}_f\textbf{abort} \\ {}_f\textbf{muffle-warning} \\ {}_f\textbf{continue} \\ {}_f\textbf{store-value}\ \textit{value} \\ {}_f\textbf{use-value}\ \textit{value} \end{Bmatrix}\right.$ [*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.

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

▷ Return new instance of *condition-type*.

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

▷ 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 36), **simple-condition**, **simple-warning**, or **simple-error**, respectively. From ${}_f$**signal** and ${}_f$**warn**, return NIL.

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

▷ Unless handled, signal as correctable **error** *condition* or a new instance of *condition-type* or, with ${}_f$**format** *control* and *args* (see page 36), **simple-error**. In the debugger, use ${}_f$**format** arguments *continue-control* and *continue-args* to tag the continue option. Return NIL.

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

▷ Return values of *forms* or, in case of **errors**, NIL and the condition_2.

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

▷ Invoke debugger with *condition*.

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

▷ If *test*, which may depend on *places*, returns NIL, signal as correctable **error** *condition* or a new instance of *condition-type* or, with ${}_f$**format** *control* and *args* (see page 36), **error**. When using the debugger's continue option, *places* can be altered before re-evaluation of *test*. Return NIL.

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

▷ If, on evaluation of *foo*, a condition of *type* is signalled, evaluate matching *condition-forms* 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 *forms* or, without a **:no-error** clause, return values of *foo*. See page 17 for (*ord-λ**).

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

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

$$({}_m\textbf{with-simple-restart}\ (\begin{Bmatrix}\textit{restart}\\ \texttt{NIL}\end{Bmatrix}\ \textit{control arg}^*)\ \textit{form}^{\text{P}*})$$

▷ Return values of *forms* unless *restart* is called during their evaluation. In this case, describe *restart* using ${}_f$**format** *control* and *args* (see page 36) and return NIL and $\texttt{T}_2$.

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

▷ Return values of *form* or, if during evaluation of *form* one of the dynamically established *restarts* is called, the values of its *restart-forms*. A *restart* is visible under *condition* if (**funcall #'***test-function condition*) returns T. If presented in the debugger, *restarts* are described by *string* or by **#'***report-function* (of a stream). A *restart* can be called by (**invoke-restart** *restart arg**), where *args* match *ord-λ**, or by (**invoke-restart-interactively** *restart*) where a list of the respective *args* is supplied by **#'***arg-function*. See page 17 for *ord-λ**.

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

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

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

▷ 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.

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

$({}_g\textbf{slot-unbound}\ \textit{class instance slot})$

▷ 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

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

$$({}_m\textbf{defgeneric}\ \begin{Bmatrix}\textit{foo}\\ (\textbf{setf}\ \textit{foo})\end{Bmatrix}\ (\textit{required-var}^*\ \Big[\textbf{\&optional}\ \begin{Bmatrix}\textit{var}\\ (\textit{var})\end{Bmatrix}^*\Big]\ \big[\textbf{\&rest}\ \textit{var}\big]\ \Big[\textbf{\&key}\ \begin{Bmatrix}\textit{var}\\ (\textit{var}\big|(\textbf{:}\textit{key var}))\end{Bmatrix}^*\ [\textbf{\&allow-other-keys}]\Big])$$
$$\begin{Bmatrix}\left|\begin{array}{l}(\textbf{:argument-precedence-order}\ \textit{required-var}^+)\\ (\textbf{declare}\ (\textbf{optimize}\ \textit{method-selection-optimization})^+)\\ (\textbf{:documentation}\ \widehat{\textit{string}})\\ (\textbf{:generic-function-class}\ \textit{gf-class}_{\boxed{\textsf{standard-generic-function}}})\\ (\textbf{:method-class}\ \textit{method-class}_{\boxed{\textsf{standard-method}}})\\ (\textbf{:method-combination}\ \textit{c-type}_{\boxed{\textsf{standard}}}\ \textit{c-arg}^*)\\ (\textbf{:method}\ \textit{defmethod-args})^*\end{array}\right.\end{Bmatrix})$$

▷ 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.

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

▷ 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.

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

▷ 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.

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

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

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

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

(${}_g$**compute-applicable-methods** *generic-function args*)

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

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

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

(${}_g$**no-applicable-method** *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.

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

▷ Signal **error** on applicable method with invalid qualifiers, or on method combination. For *control* and *args* see **format**, page 36.

(${}_g$**no-next-method** *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 $\underset{2}{\texttt{T}}$ 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**.

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

▷ **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 $\left[\begin{Bmatrix}\textbf{:most-specific-first} \\ \textbf{:most-specific-last}\end{Bmatrix}_{\boxed{\textbf{:most-specific-first}}}\right]$ (specified as *c-arg* in ${}_m$**defgeneric**). Using *c-type* as the *qualifier* in ${}_m$**defmethod** makes the method primary.

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

▷ **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 17, the latter enhanced by an optional **&whole** argument.

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

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

11 Conditions and Errors

For standardized condition types cf. Figure 2 on page 31.

$$({}_m\textbf{define-condition}\ \textit{foo}\ (\textit{parent-type}^*_{\boxed{\textbf{condition}}})$$
$$\left(\left\{\begin{array}{l} \textit{slot} \\ (\textit{slot} \left\{\begin{array}{|l} \{\textbf{:reader}\ \textit{reader}\}^* \\ \{\textbf{:writer} \begin{Bmatrix} \textit{writer} \\ (\textbf{setf}\ \textit{writer}) \end{Bmatrix}\}^* \\ \{\textbf{:accessor}\ \textit{accessor}\}^* \\ \textbf{:allocation} \begin{Bmatrix} \textbf{:instance} \\ \textbf{:class} \end{Bmatrix}_{\boxed{\textbf{:instance}}} \\ \{\textbf{:initarg}\ [\textbf{:}]\textit{initarg-name}\}^* \\ \textbf{:initform}\ \textit{form} \\ \textbf{:type}\ \textit{type} \\ \textbf{:documentation}\ \textit{slot-doc} \end{array}\right\}) \end{array}\right\}^*\right)$$
$$\left\{\begin{array}{|l} (\textbf{:default-initargs}\ \{\textit{name value}\}^*) \\ (\textbf{:documentation}\ \textit{condition-doc}) \\ (\textbf{:report} \begin{Bmatrix} \textit{string} \\ \textit{report-function} \end{Bmatrix}) \end{array}\right\})$$

▷ 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.