Quick Reference

cl

Common lisp

Bert Burgemeister

Contents

Typographic Conventions

name; $\overset{\text{Fu}}{\textbf{name}}$; $\overset{\text{M}}{\textbf{name}}$; $\overset{\text{sO}}{\textbf{name}}$; $\overset{\text{gF}}{\textbf{name}}$; $\overset{\text{var}}{\textbf{*name*}}$; $\overset{\text{co}}{\textbf{name}}$

▷ Symbol defined in Common Lisp; esp. function, macro, special operator, generic function, variable, constant.

them ▷ Placeholder for actual code.

`me` ▷ Literal text.

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

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

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

foos ▷ English plural denotes a list argument.

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

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

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

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

$foo^{\text{P}*}$ ▷ foo^* is evaluated as in $\overset{\text{sO}}{\textbf{progn}}$; see p. 19.

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

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

1 Numbers

1.1 Predicates

($\overset{\text{Fu}}{=}$ *number*$^+$)
($\overset{\text{Fu}}{/=}$ *number*$^+$)

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

($\overset{\text{Fu}}{>}$ *number*$^+$)
($\overset{\text{Fu}}{>=}$ *number*$^+$)
($\overset{\text{Fu}}{<}$ *number*$^+$)
($\overset{\text{Fu}}{<=}$ *number*$^+$)

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

($\overset{\text{Fu}}{\textsf{minusp}}$ a)
($\overset{\text{Fu}}{\textsf{zerop}}$ a)
($\overset{\text{Fu}}{\textsf{plusp}}$ a)

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

($\overset{\text{Fu}}{\textsf{evenp}}$ *integer*)
($\overset{\text{Fu}}{\textsf{oddp}}$ *integer*)

▷ $\underline{\texttt{T}}$ if *integer* is even or odd, respectively.

($\overset{\text{Fu}}{\textsf{numberp}}$ *foo*)
($\overset{\text{Fu}}{\textsf{realp}}$ *foo*)
($\overset{\text{Fu}}{\textsf{rationalp}}$ *foo*)
($\overset{\text{Fu}}{\textsf{floatp}}$ *foo*)
($\overset{\text{Fu}}{\textsf{integerp}}$ *foo*)
($\overset{\text{Fu}}{\textsf{complexp}}$ *foo*)
($\overset{\text{Fu}}{\textsf{random-state-p}}$ *foo*)

▷ $\underline{\texttt{T}}$ if *foo* is of indicated type.

1.2 Numeric Functions

($\overset{\text{Fu}}{+}$ $a_{\boxed{0}}{}^*$)
($\overset{\text{Fu}}{*}$ $a_{\boxed{1}}{}^*$)

▷ Return $\underline{\sum a}$ or $\underline{\prod a}$, respectively.

($\overset{\text{Fu}}{-}$ a b^*)
($\overset{\text{Fu}}{/}$ a b^*)

▷ Return $\underline{a - \sum b}$ or $\underline{a/\prod b}$, respectively. Without any bs, return $\underline{-a}$ or $\underline{1/a}$, respectively.

($\overset{\text{Fu}}{\textsf{1+}}$ a)
($\overset{\text{Fu}}{\textsf{1-}}$ a)

▷ Return $\underline{a+1}$ or $\underline{a-1}$, respectively.

($\left\{\begin{matrix}\overset{\text{M}}{\textsf{incf}}\\ \overset{\text{M}}{\textsf{decf}}\end{matrix}\right\}$ $\widetilde{place}$ [$delta_{\boxed{1}}$])

▷ Increment or decrement the value of *place* by *delta*. Return $\underline{\text{new value}}$.

($\overset{\text{Fu}}{\textsf{exp}}$ p)
($\overset{\text{Fu}}{\textsf{expt}}$ b p)

▷ Return $\underline{e^p}$ or $\underline{b^p}$, respectively.

($\overset{\text{Fu}}{\textsf{log}}$ a [b]) ▷ Return $\underline{\log_b a}$ or, without b, $\underline{\ln a}$.

($\overset{\text{Fu}}{\textsf{sqrt}}$ n)
($\overset{\text{Fu}}{\textsf{isqrt}}$ n)

▷ $\underline{\sqrt{n}}$ in complex or natural numbers, respectively.

($\overset{\text{Fu}}{\textsf{lcm}}$ $integer^*{}_{\boxed{1}}$)
($\overset{\text{Fu}}{\textsf{gcd}}$ *integer**)

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

$\overset{\text{co}}{\textsf{pi}}$ ▷ **long-float** approximation of π, Ludolph's number.

($\overset{\text{Fu}}{\textsf{sin}}$ a)
($\overset{\text{Fu}}{\textsf{cos}}$ a)
($\overset{\text{Fu}}{\textsf{tan}}$ a)

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

($\overset{\text{Fu}}{\textsf{asin}}$ a)
($\overset{\text{Fu}}{\textsf{acos}}$ a)

▷ $\underline{\arcsin a}$ or $\underline{\arccos a}$, respectively, in radians.

($\overset{\text{Fu}}{\textsf{atan}}$ a [$b_{\boxed{1}}$]) ▷ $\underline{\arctan \frac{a}{b}}$ in radians.

($\overset{\text{Fu}}{\textsf{sinh}}$ a)
($\overset{\text{Fu}}{\textsf{cosh}}$ a)
($\overset{\text{Fu}}{\textsf{tanh}}$ a)

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

($\overset{\text{Fu}}{\textbf{asinh}}$ *a*)
($\overset{\text{Fu}}{\textbf{acosh}}$ *a*)
($\overset{\text{Fu}}{\textbf{atanh}}$ *a*)
▷ asinh *a*, acosh *a*, or atanh *a*, respectively.

($\overset{\text{Fu}}{\textbf{cis}}$ *a*) ▷ Return $e^{i\,a} = \cos a + i \sin a$.

($\overset{\text{Fu}}{\textbf{conjugate}}$ *a*) ▷ Return complex conjugate of *a*.

($\overset{\text{Fu}}{\textbf{max}}$ *num*$^{+}$)
($\overset{\text{Fu}}{\textbf{min}}$ *num*$^{+}$)
▷ Greatest or least, respectively, of *num*s.

({$\overset{\text{Fu}}{\textbf{round}}$|$\overset{\text{Fu}}{\textbf{fround}}$}
{$\overset{\text{Fu}}{\textbf{floor}}$|$\overset{\text{Fu}}{\textbf{ffloor}}$}
{$\overset{\text{Fu}}{\textbf{ceiling}}$|$\overset{\text{Fu}}{\textbf{fceiling}}$}
{$\overset{\text{Fu}}{\textbf{truncate}}$|$\overset{\text{Fu}}{\textbf{ftruncate}}$} *n* [*d*$_{\boxed{1}}$])

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

({$\overset{\text{Fu}}{\textbf{mod}}$ | $\overset{\text{Fu}}{\textbf{rem}}$} *n d*)

▷ Same as $\overset{\text{Fu}}{\textbf{floor}}$ or $\overset{\text{Fu}}{\textbf{truncate}}$, respectively, but return remainder only.

($\overset{\text{Fu}}{\textbf{random}}$ *limit* [*state*$_{\boxed{\overset{\text{var}}{\textbf{*random-state*}}}}$])

▷ Return non-negative random number less than *limit*, and of the same type.

($\overset{\text{Fu}}{\textbf{make-random-state}}$ [{*state*|`NIL`|`T`}$_{\boxed{\texttt{NIL}}}$])

▷ Copy of **random-state** object *state* or of the current random state; or a randomly initialized fresh random state.

$\overset{\text{var}}{\textbf{*random-state*}}$ ▷ Current random state.

($\overset{\text{Fu}}{\textbf{float-sign}}$ *num-a* [*num-b*$_{\boxed{1}}$]) ▷ *num-b* with *num-a*'s sign.

($\overset{\text{Fu}}{\textbf{signum}}$ *n*)

▷ Number of magnitude 1 representing sign or phase of *n*.

($\overset{\text{Fu}}{\textbf{numerator}}$ *rational*)
($\overset{\text{Fu}}{\textbf{denominator}}$ *rational*)

▷ Numerator or denominator, respectively, of *rational*'s canonical form.

($\overset{\text{Fu}}{\textbf{realpart}}$ *number*)
($\overset{\text{Fu}}{\textbf{imagpart}}$ *number*)

▷ Real part or imaginary part, respectively, of *number*.

($\overset{\text{Fu}}{\textbf{complex}}$ *real* [*imag*$_{\boxed{0}}$]) ▷ Make a complex number.

($\overset{\text{Fu}}{\textbf{phase}}$ *number*) ▷ Angle of *number*'s polar representation.

($\overset{\text{Fu}}{\textbf{abs}}$ *n*) ▷ Return $|n|$.

($\overset{\text{Fu}}{\textbf{rational}}$ *real*)
($\overset{\text{Fu}}{\textbf{rationalize}}$ *real*)

▷ Convert *real* to rational. Assume complete/limited accuracy for *real*.

($\overset{\text{Fu}}{\textbf{float}}$ *real* [*prototype*$_{\boxed{\texttt{0.0F0}}}$])

▷ Convert *real* into float with type of *prototype*.

1.3 Logic Functions

Negative integers are used in two's complement representation.

($\overset{\text{Fu}}{\textbf{boole}}$ *operation int-a int-b*)

▷ Return value of bitwise logical *operation*. *operation*s are

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

boole-eqv$^{\text{co}}$	▷ $\underline{\textit{int-a} \equiv \textit{int-b}}$.
boole-and$^{\text{co}}$	▷ $\underline{\textit{int-a} \wedge \textit{int-b}}$.
boole-andc1$^{\text{co}}$	▷ $\underline{\neg \textit{int-a} \wedge \textit{int-b}}$.
boole-andc2$^{\text{co}}$	▷ $\underline{\textit{int-a} \wedge \neg \textit{int-b}}$.
boole-nand$^{\text{co}}$	▷ $\underline{\neg(\textit{int-a} \wedge \textit{int-b})}$.
boole-ior$^{\text{co}}$	▷ $\underline{\textit{int-a} \vee \textit{int-b}}$.
boole-orc1$^{\text{co}}$	▷ $\underline{\neg \textit{int-a} \vee \textit{int-b}}$.
boole-orc2$^{\text{co}}$	▷ $\underline{\textit{int-a} \vee \neg \textit{int-b}}$.
boole-xor$^{\text{co}}$	▷ $\underline{\neg(\textit{int-a} \equiv \textit{int-b})}$.
boole-nor$^{\text{co}}$	▷ $\underline{\neg(\textit{int-a} \vee \textit{int-b})}$.

(**lognot**$^{\text{Fu}}$ *integer*) ▷ $\underline{\neg \textit{integer}}$.

(**logeqv**$^{\text{Fu}}$ *integer**)
(**logand**$^{\text{Fu}}$ *integer**)

▷ Return $\underline{\text{value of exclusive-nored or anded } \textit{integer}}$s, respectively. Without any *integer*, return $\underline{-1}$.

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

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

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

(**logxor**$^{\text{Fu}}$ *integer**)
(**logior**$^{\text{Fu}}$ *integer**)

▷ Return $\underline{\text{value of exclusive-ored or ored } \textit{integer}}$s, respectively. Without any *integer*, return $\underline{0}$.

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

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

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

(**logbitp**$^{\text{Fu}}$ *i integer*)

▷ $\underline{\texttt{T}}$ if zero-indexed *i*th bit of *integer* is set.

(**logtest**$^{\text{Fu}}$ *int-a int-b*)

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

(**logcount**$^{\text{Fu}}$ *int*)

▷ $\underline{\text{Number of 1 bits}}$ in *int* ≥ 0, $\underline{\text{number of 0 bits}}$ in *int* < 0.

1.4 Integer Functions

(**integer-length**$^{\text{Fu}}$ *integer*)

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

(**ldb-test**$^{\text{Fu}}$ *byte-spec integer*)

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

(**ash**$^{\text{Fu}}$ *integer count*)

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

(**ldb**$^{\text{Fu}}$ *byte-spec integer*)

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

($\begin{Bmatrix}\textbf{deposit-field}^{\text{Fu}}\\ \textbf{dpb}^{\text{Fu}}\end{Bmatrix}$ *int-a byte-spec int-b*)

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

(**mask-field**$^{\text{Fu}}$ *byte-spec integer*)

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

(**byte**$^{\text{Fu}}$ *size position*)

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

(**byte-size**$^{\text{Fu}}$ *byte-spec*)
(**byte-position**$^{\text{Fu}}$ *byte-spec*)

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

1.5 Implementation-Dependent

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

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

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

▷ Available numbers closest to −0 or +0, respectively.

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

▷ Available numbers closest to −∞ or +∞, respectively.

($^{\text{Fu}}$**decode-float** *n*)
($^{\text{Fu}}$**integer-decode-float** *n*)

▷ Return $\underline{\text{significand}}$, $\underset{2}{\underline{\text{exponent}}}$, and $\underset{3}{\underline{\text{sign}}}$ of **float** *n*.

($^{\text{Fu}}$**scale-float** *n* [*i*]) ▷ With *n*'s radix *b*, return nb^i.

($^{\text{Fu}}$**float-radix** *n*)
($^{\text{Fu}}$**float-digits** *n*)
($^{\text{Fu}}$**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*.

($^{\text{Fu}}$**upgraded-complex-part-type** *foo* [*environment*$_{\boxed{\text{NIL}}}$])

▷ $\underline{\text{Type}}$ 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 ``!?$"'`.:,;*+-/|\~_^<=>#%@&()[]{}``.

($^{\text{Fu}}$**characterp** *foo*)
($^{\text{Fu}}$**standard-char-p** *char*) ▷ $\underline{\texttt{T}}$ if argument is of indicated type.

($^{\text{Fu}}$**graphic-char-p** *character*)
($^{\text{Fu}}$**alpha-char-p** *character*)
($^{\text{Fu}}$**alphanumericp** *character*)

▷ $\underline{\texttt{T}}$ if *character* is visible, alphabetic, or alphanumeric, respectively.

($^{\text{Fu}}$**upper-case-p** *character*)
($^{\text{Fu}}$**lower-case-p** *character*)
($^{\text{Fu}}$**both-case-p** *character*)

▷ Return $\underline{\texttt{T}}$ if *character* is uppercase, lowercase, or able to be in another case, respectively.

($^{\text{Fu}}$**digit-char-p** *character* [*radix*$_{\boxed{10}}$])

▷ Return $\underline{\text{its weight}}$ if *character* is a digit, or $\underline{\texttt{NIL}}$ otherwise.

($^{\text{Fu}}$**char=** *character*$^+$)
($^{\text{Fu}}$**char/=** *character*$^+$)

▷ Return $\underline{\texttt{T}}$ if all *character*s, or none, respectively, are equal.

($^{\text{Fu}}$**char-equal** *character*$^+$)
($^{\text{Fu}}$**char-not-equal** *character*$^+$)

▷ Return $\underline{\texttt{T}}$ if all *character*s, or none, respectively, are equal ignoring case.

($^{\text{Fu}}$**char>** *character*$^+$)
($^{\text{Fu}}$**char>=** *character*$^+$)
($^{\text{Fu}}$**char<** *character*$^+$)
($^{\text{Fu}}$**char<=** *character*$^+$)

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

($^{\text{Fu}}$**char-greaterp** *character*$^{+}$)
($^{\text{Fu}}$**char-not-lessp** *character*$^{+}$)
($^{\text{Fu}}$**char-lessp** *character*$^{+}$)
($^{\text{Fu}}$**char-not-greaterp** *character*$^{+}$)

▷ Return T if *character*s are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively, ignoring case.

($^{\text{Fu}}$**char-upcase** *character*)
($^{\text{Fu}}$**char-downcase** *character*)

▷ Return corresponding uppercase/lowercase character, respectively.

($^{\text{Fu}}$**digit-char** *i* [*radix*$_{\boxed{10}}$]) ▷ Character representing digit *i*.

($^{\text{Fu}}$**char-name** *character*) ▷ *character*'s name if any, or NIL.

($^{\text{Fu}}$**name-char** *foo*) ▷ Character named *foo* if any, or NIL.

($^{\text{Fu}}$**char-int** *character*)
($^{\text{Fu}}$**char-code** *character*) ▷ Code of *character*.

($^{\text{Fu}}$**code-char** *code*) ▷ Character with *code*.

$^{\text{co}}$**char-code-limit** ▷ Upper bound of ($^{\text{Fu}}$**char-code** *char*); ≥ 96.

($^{\text{Fu}}$**character** *c*) ▷ Return #\c.

3 Strings

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

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

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

▷ Return T if subsequences of *foo* and *bar* are equal. Obey/ignore, respectively, case.

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

▷ 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.

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

▷ Return string of length *size*.

($^{\text{Fu}}$**string** *x*)

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

▷ Convert *x* (**symbol**, **string**, or **character**) into a string, a string with capitalized words, an all-uppercase string, or an all-lowercase string, respectively.

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

▷ Convert *string* into a string with capitalized words, an all-uppercase string, or an all-lowercase string, respectively.

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

▷ Return *string* with all characters in sequence *char-bag* removed from both ends, from the beginning, or from the end, respectively.

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

(**parse-integer**$^{\text{Fu}}$ *string* {‖ **:start** *start*$\boxed{0}$ | **:end** *end*$\boxed{\text{NIL}}$ | **:radix** *int*$\boxed{10}$ | **:junk-allowed** *bool*$\boxed{\text{NIL}}$ })
▷ Return integer parsed from *string* and index$_2$ of parse end.

4 Conses

4.1 Predicates

(**consp**$^{\text{Fu}}$ *foo*)
(**listp**$^{\text{Fu}}$ *foo*)
▷ Return `T` if *foo* is of indicated type.

(**endp**$^{\text{Fu}}$ *list*)
(**null**$^{\text{Fu}}$ *foo*)
▷ Return `T` if *list*/*foo* is `NIL`.

(**atom**$^{\text{Fu}}$ *foo*) ▷ Return `T` if *foo* is not a **cons**.

(**tailp**$^{\text{Fu}}$ *foo list*) ▷ Return `T` if *foo* is a tail of *list*.

(**member**$^{\text{Fu}}$ *foo list* {‖ { **:test** *function*$\boxed{\text{\#'eql}}$ | **:test-not** *function* } | **:key** *function* })
▷ Return tail of *list* starting with its first element matching *foo*. Return `NIL` if there is no such element.

({**member-if**$^{\text{Fu}}$ | **member-if-not**$^{\text{Fu}}$} *test list* [**:key** *function*])
▷ Return tail of *list* starting with its first element satisfying *test*. Return `NIL` if there is no such element.

(**subsetp**$^{\text{Fu}}$ *list-a list-b* {‖ { **:test** *function*$\boxed{\text{\#'eql}}$ | **:test-not** *function* } | **:key** *function* })
▷ Return `T` if *list-a* is a subset of *list-b*.

4.2 Lists

(**cons**$^{\text{Fu}}$ *foo bar*) ▷ Return new cons (*foo* **.** *bar*).

(**list**$^{\text{Fu}}$ *foo**) ▷ Return list of *foo*s.

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

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

(**list-length**$^{\text{Fu}}$ *list*) ▷ Length of *list*; `NIL` for circular *list*.

(**car**$^{\text{Fu}}$ *list*) ▷ Car of *list* or `NIL` if *list* is `NIL`. **setf**able.

(**cdr**$^{\text{Fu}}$ *list*)
(**rest**$^{\text{Fu}}$ *list*)
▷ Cdr of *list* or `NIL` if *list* is `NIL`. **setf**able.

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

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

(**nth**$^{\text{Fu}}$ *n list*) ▷ Zero-indexed *n*th element of *list*. **setf**able.

(**c*X*r**$^{\text{Fu}}$ *list*)
▷ With *X* being one to four **a**s and **d**s representing **car**$^{\text{Fu}}$s and **cdr**$^{\text{Fu}}$s, e.g. (**cadr**$^{\text{Fu}}$ *bar*) is equivalent to (**car**$^{\text{Fu}}$ (**cdr**$^{\text{Fu}}$ *bar*)). **setf**able.

(**last**$^{\text{Fu}}$ *list* [*num*$\boxed{1}$]) ▷ Return list of last *num* conses of *list*.

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{butlast}}\ \textit{list}\\ \overset{\text{Fu}}{\textbf{nbutlast}}\ \widetilde{\textit{list}}\end{Bmatrix}[\textit{num}_{\boxed{1}}]\right)$ ▷ $\underline{\textit{list}}$ excluding last *num* conses.

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{rplaca}}\\ \overset{\text{Fu}}{\textbf{rplacd}}\end{Bmatrix}\ \widetilde{\textit{cons}}\ \textit{object}\right)$

▷ Replace car, or cdr, respectively, of $\underline{\textit{cons}}$ with *object*.

$(\overset{\text{Fu}}{\textbf{ldiff}}\ \textit{list foo})$

▷ If *foo* is a tail of *list*, return $\underline{\text{preceding part of } \textit{list}}$. Otherwise return $\underline{\textit{list}}$.

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

▷ Return $\underline{\textit{list}}$ if *foo* is already member of *list*. If not, return $\underline{(\overset{\text{Fu}}{\textbf{cons}}\ \textit{foo list})}$.

$(\overset{\text{M}}{\textbf{pop}}\ \widetilde{\textit{place}})$ ▷ Set *place* to $(\overset{\text{Fu}}{\textbf{cdr}}\ \textit{place})$, return $\underline{(\overset{\text{Fu}}{\textbf{car}}\ \textit{place})}$.

$(\overset{\text{M}}{\textbf{push}}\ \textit{foo}\ \widetilde{\textit{place}})$ ▷ Set *place* to $\underline{(\overset{\text{Fu}}{\textbf{cons}}\ \textit{foo place})}$.

$\left(\overset{\text{M}}{\textbf{pushnew}}\ \textit{foo}\ \widetilde{\textit{place}}\ \left\{\begin{array}{l}\left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\#\text{'eql}}}\\ \textbf{:test-not}\ \textit{function}\end{array}\right\}\\ \textbf{:key}\ \textit{function}\end{array}\right\}\right)$

▷ Set *place* to $\underline{(\overset{\text{Fu}}{\textbf{adjoin}}\ \textit{foo place})}$.

$(\overset{\text{Fu}}{\textbf{append}}\ [\textit{proper-list}^{*}\ \textit{foo}_{\boxed{\text{NIL}}}])$

$(\overset{\text{Fu}}{\textbf{nconc}}\ [\widetilde{\textit{non-circular-list}}^{*}\ \textit{foo}_{\boxed{\text{NIL}}}])$

▷ Return $\underline{\text{concatenated list}}$ or, with only one argument, $\underline{\textit{foo}}$. *foo* can be of any type.

$(\overset{\text{Fu}}{\textbf{revappend}}\ \textit{list foo})$

$(\overset{\text{Fu}}{\textbf{nreconc}}\ \widetilde{\textit{list}}\ \textit{foo})$

▷ Return $\underline{\text{concatenated list}}$ after reversing order in *list*.

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{mapcar}}\\ \overset{\text{Fu}}{\textbf{maplist}}\end{Bmatrix}\ \textit{function list}^{+}\right)$

▷ Return $\underline{\text{list of return values}}$ of *function* successively invoked with corresponding arguments, either cars or cdrs, respectively, from each *list*.

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{mapcan}}\\ \overset{\text{Fu}}{\textbf{mapcon}}\end{Bmatrix}\ \textit{function list}^{+}\right)$

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

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{mapc}}\\ \overset{\text{Fu}}{\textbf{mapl}}\end{Bmatrix}\ \textit{function list}^{+}\right)$

▷ Return $\underline{\text{first } \textit{list}}$ after successively applying *function* to corresponding arguments, either cars or cdrs, respectively, from each *list*. *function* should have some side effects.

$(\overset{\text{Fu}}{\textbf{copy-list}}\ \textit{list})$ ▷ Return $\underline{\text{copy}}$ of *list* with shared elements.

4.3 Association Lists

$(\overset{\text{Fu}}{\textbf{pairlis}}\ \textit{keys values}\ [\textit{alist}_{\boxed{\text{NIL}}}])$

▷ Prepend to $\underline{\textit{alist}}$ an association list made from lists *keys* and *values*.

$(\overset{\text{Fu}}{\textbf{acons}}\ \textit{key value alist})$

▷ Return $\underline{\textit{alist}}$ with a (*key* **.** *value*) pair added.

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{assoc}}\\ \overset{\text{Fu}}{\textbf{rassoc}}\end{Bmatrix}\ \textit{foo alist}\ \left\{\begin{array}{l}\left\{\begin{array}{l}\textbf{:test}\ \textit{test}_{\boxed{\#\text{'eql}}}\\ \textbf{:test-not}\ \textit{test}\end{array}\right\}\\ \textbf{:key}\ \textit{function}\end{array}\right\}\right)$

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{assoc-if}}[\textbf{-not}]\\ \overset{\text{Fu}}{\textbf{rassoc-if}}[\textbf{-not}]\end{Bmatrix}\ \textit{test alist}\ [\textbf{:key}\ \textit{function}]\right)$

▷ First $\underline{\text{cons}}$ whose car, or cdr, respectively, satisfies *test*.

$(\overset{\text{Fu}}{\textbf{copy-alist}}\ \textit{alist})$ ▷ Return $\underline{\text{copy}}$ of *alist*.

4.4 Trees

$$(\overset{\mathsf{Fu}}{\textbf{tree-equal}}\ \textit{foo bar} \left\{\begin{array}{l}\textbf{:test}\ \textit{test}_{\boxed{\texttt{\#'eql}}}\\ \textbf{:test-not}\ \textit{test}\end{array}\right\})$$

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

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

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

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

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

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

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

$(\overset{\mathsf{Fu}}{\textbf{copy-tree}}\ \textit{tree})$ ▷ $\underline{\text{Copy of } \textit{tree}}$ with same shape and leaves.

4.5 Sets

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

▷ 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

$(\overset{\mathsf{Fu}}{\textbf{arrayp}}\ \textit{foo})$
$(\overset{\mathsf{Fu}}{\textbf{vectorp}}\ \textit{foo})$
$(\overset{\mathsf{Fu}}{\textbf{simple-vector-p}}\ \textit{foo})$ ▷ $\underline{\texttt{T}}$ if *foo* is of indicated type.
$(\overset{\mathsf{Fu}}{\textbf{bit-vector-p}}\ \textit{foo})$
$(\overset{\mathsf{Fu}}{\textbf{simple-bit-vector-p}}\ \textit{foo})$

$(\overset{\mathsf{Fu}}{\textbf{adjustable-array-p}}\ \textit{array})$
$(\overset{\mathsf{Fu}}{\textbf{array-has-fill-pointer-p}}\ \textit{array})$

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

$(\overset{\mathsf{Fu}}{\textbf{array-in-bounds-p}}\ \textit{array}\ [\textit{subscripts}])$

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

5.2 Array Functions

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

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

$(\overset{\mathsf{Fu}}{\textbf{aref}}\ \textit{array}\ \left[\textit{subscripts}\right])$

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

$(\overset{\mathsf{Fu}}{\textbf{row-major-aref}}\ \textit{array}\ i)$

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

(${}^{\text{Fu}}$**array-row-major-index** *array* [*subscripts*])

▷ $\underline{\text{Index}}$ in row-major order of the element denoted by *subscripts*.

(${}^{\text{Fu}}$**array-dimensions** *array*)

▷ $\underline{\text{List}}$ containing the lengths of *array*'s dimensions.

(${}^{\text{Fu}}$**array-dimension** *array* *i*)

▷ $\underline{\text{Length of } i\text{th dimension}}$ of *array*.

(${}^{\text{Fu}}$**array-total-size** *array*) ▷ $\underline{\text{Number of elements}}$ in *array*.

(${}^{\text{Fu}}$**array-rank** *array*) ▷ $\underline{\text{Number of dimensions}}$ of *array*.

(${}^{\text{Fu}}$**array-displacement** *array*) ▷ $\underline{\text{Target array}}$ and $\underset{2}{\underline{\text{offset}}}$.

(${}^{\text{Fu}}$**bit** *bit-array* [*subscripts*])
(${}^{\text{Fu}}$**sbit** *simple-bit-array* [*subscripts*])

▷ Return $\underline{\text{element}}$ of *bit-array* or of *simple-bit-array*. **setf**able.

(${}^{\text{Fu}}$**bit-not** $\widetilde{\textit{bit-array}}$ [$\widetilde{\textit{result-bit-array}}_{\boxed{\texttt{NIL}}}$])

▷ Return $\underline{\text{result}}$ of bitwise negation of *bit-array*. If *result-bit-array* is `T`, put result in *bit-array*; if it is `NIL`, make a new array for result.

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

▷ Return $\underline{\text{result}}$ of bitwise logical operations (cf. operations of ${}^{\text{Fu}}$**boole**, p. 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.

${}^{\text{co}}$**array-rank-limit** ▷ Upper bound of array rank; ≥ 8.

${}^{\text{co}}$**array-dimension-limit**

▷ Upper bound of an array dimension; ≥ 1024.

${}^{\text{co}}$**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.

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

(${}^{\text{Fu}}$**svref** *vector* *i*) ▷ Return $\underline{\text{element } i}$ of simple *vector*. **setf**able.

(${}^{\text{Fu}}$**vector-push** *foo* $\widetilde{\textit{vector}}$)

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

(${}^{\text{Fu}}$**vector-push-extend** *foo* $\widetilde{\textit{vector}}$ [*num*])

▷ Replace element of *vector* pointed to by $\underline{\text{fill pointer}}$ with *foo*, then increment fill pointer. Extend *vector*'s size by ≥ *num* if necessary.

(${}^{\text{Fu}}$**vector-pop** $\widetilde{\textit{vector}}$)

▷ Return $\underline{\text{element of } \textit{vector}}$ its fillpointer points to after decrementation.

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

6 Sequences

6.1 Sequence Predicates

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{every}}\\ \overset{\text{Fu}}{\textbf{notevery}}\end{Bmatrix}\ \textit{test sequence}^{+}\right)$

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

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{some}}\\ \overset{\text{Fu}}{\textbf{notany}}\end{Bmatrix}\ \textit{test sequence}^{+}\right)$

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

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

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

6.2 Sequence Functions

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

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

$(\overset{\text{Fu}}{\textbf{concatenate}}\ \textit{type sequence}^{*})$

▷ Return concatenated sequence of *type*.

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

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

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

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

$(\overset{\text{Fu}}{\textbf{length}}\ \textit{sequence})$

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

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

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

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

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

$(\overset{\text{Fu}}{\textbf{elt}}\ \textit{sequence index})$

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

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

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

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{sort}}\\ \overset{\text{Fu}}{\textbf{stable-sort}}\end{Bmatrix}\ \widetilde{\textit{sequence}}\ \textit{test}\ [\textbf{:key}\ \textit{function}]\right)$

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

$(\overset{\text{Fu}}{\textbf{reverse}}\ \textit{sequence})$
$(\overset{\text{Fu}}{\textbf{nreverse}}\ \widetilde{\textit{sequence}})$

▷ Return *sequence* in reverse order.

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

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

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

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

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

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

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

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

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

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

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

▷ Make copy of *sequence* without duplicates.

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

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

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

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

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

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

($\overset{\textsf{Fu}}{\textbf{map}}$ *type function sequence*$^+$)

▷ Apply *function* successively to corresponding elements of the *sequence*s. Return values as a $\underline{\text{sequence}}$ of *type*. If *type* is `NIL`, return $\underline{\texttt{NIL}}$.

($\overset{\textsf{Fu}}{\textbf{map-into}}$ $\widetilde{\textit{result-sequence}}$ *function sequence**)

▷ Store into $\underline{\textit{result-sequence}}$ successively values of *function* applied to corresponding elements of the *sequence*s.

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

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

($\overset{\textsf{Fu}}{\textbf{copy-seq}}$ *sequence*)

▷ $\underline{\text{Copy of } \textit{sequence}}$ with shared elements.

7 Hash Tables

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

($\overset{\textsf{Fu}}{\textbf{hash-table-p}}$ *foo*) ▷ Return $\underline{\texttt{T}}$ if *foo* is of type **hash-table**.

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

▷ Make a $\underline{\text{hash table}}$.

($\overset{\textsf{Fu}}{\textbf{gethash}}$ *key hash-table* [*default*$_{\boxed{\texttt{NIL}}}$])

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

($\overset{\textsf{Fu}}{\textbf{hash-table-count}}$ *hash-table*)

▷ $\underline{\text{Number of entries}}$ in *hash-table*.

($\overset{\textsf{Fu}}{\textbf{remhash}}$ *key* $\widetilde{\textit{hash-table}}$)

▷ Remove from *hash-table* entry with *key* and return $\underline{\texttt{T}}$ if it existed. Return $\underline{\texttt{NIL}}$ otherwise.

($\overset{\textsf{Fu}}{\textbf{clrhash}}$ $\widetilde{\textit{hash-table}}$) ▷ Empty $\underline{\textit{hash-table}}$.

($\overset{\textsf{Fu}}{\textbf{maphash}}$ *function hash-table*)

▷ Iterate over *hash-table* calling *function* on key and value. Return $\underline{\texttt{NIL}}$.

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

▷ Return $\underline{\text{values of } \textit{forms}}$. In *forms*, invocations of (*foo*) return: `T` if an entry is returned; its key; its value.

($\overset{\textsf{Fu}}{\textbf{hash-table-test}}$ *hash-table*)

▷ $\underline{\text{Test function}}$ used in *hash-table*.

($\overset{\textsf{Fu}}{\textbf{hash-table-size}}$ *hash-table*)
($\overset{\textsf{Fu}}{\textbf{hash-table-rehash-size}}$ *hash-table*)
($\overset{\textsf{Fu}}{\textbf{hash-table-rehash-threshold}}$ *hash-table*)

▷ Current $\underline{\text{size}}$, $\underline{\text{rehash-size}}$, or $\underline{\text{rehash-threshold}}$, respectively, as used in $\overset{\textsf{Fu}}{\textbf{make-hash-table}}$.

($\overset{\textsf{Fu}}{\textbf{sxhash}}$ *foo*)

▷ $\underline{\text{Hash code}}$ unique for any argument $\overset{\textsf{Fu}}{\textbf{equal}}$ *foo*.

8 Structures

$$(\overset{\text{M}}{\textbf{defstruct}}$$

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

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

▷ Define structure <u>*foo*</u> 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 p. 16) 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 $\overset{\text{gF}}{\textbf{print-object}}$ 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.

($\overset{\text{Fu}}{\textbf{copy-structure}}$ *structure*)

▷ Return <u>copy of *structure*</u> with shared slot values.

9 Control Structure

9.1 Predicates

($\overset{\text{Fu}}{\textbf{eq}}$ *foo bar*) ▷ <u>`T`</u> if *foo* and *bar* are identical.

($\overset{\text{Fu}}{\textbf{eql}}$ *foo bar*)

▷ <u>`T`</u> if *foo* and *bar* are identical, or the same **character**, or **number**s of the same type and value.

($\overset{\text{Fu}}{\textbf{equal}}$ *foo bar*)

▷ <u>`T`</u> if *foo* and *bar* are $\overset{\text{Fu}}{\textbf{eql}}$, or are equivalent **pathname**s, or are **cons**es with $\overset{\text{Fu}}{\textbf{equal}}$ cars and cdrs, or are **string**s or **bit-vector**s with $\overset{\text{Fu}}{\textbf{eql}}$ elements below their fill pointers.

($\overset{\text{Fu}}{\textbf{equalp}}$ *foo bar*)

▷ <u>`T`</u> 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 $\overset{\text{Fu}}{\textbf{equalp}}$ elements; or are structures of the same type with $\overset{\text{Fu}}{\textbf{equalp}}$ elements; or are **hash-table**s of the same size with the same **:test** function, the same keys in terms of **:test** function, and $\overset{\text{Fu}}{\textbf{equalp}}$ elements.

($\overset{\text{Fu}}{\textbf{not}}$ *foo*) ▷ <u>`T`</u> if *foo* is `NIL`; <u>`NIL`</u> otherwise.

($\overset{\text{Fu}}{\textbf{boundp}}$ *symbol*) ▷ <u>`T`</u> if *symbol* is a special variable.

($\overset{\text{Fu}}{\textbf{constantp}}$ *foo* [*environment*$_{\boxed{\texttt{NIL}}}$])

▷ <u>`T`</u> if *foo* is a constant form.

($\overset{\text{Fu}}{\textbf{functionp}}$ *foo*) ▷ <u>`T`</u> if *foo* is of type **function**.

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

9.2 Variables

$(\left\{\begin{array}{l}\overset{\text{M}}{\textbf{defconstant}}\\\overset{\text{M}}{\textbf{defparameter}}\end{array}\right\}\ \widehat{foo}\ form\ [\widehat{doc}])$

▷ Assign value of *form* to global constant/dynamic variable *foo*.

$(\overset{\text{M}}{\textbf{defvar}}\ \widehat{foo}\ [form\ [\widehat{doc}]])$

▷ Unless bound already, assign value of *form* to dynamic variable *foo*.

$(\left\{\begin{array}{l}\overset{\text{M}}{\textbf{setf}}\\\overset{\text{M}}{\textbf{psetf}}\end{array}\right\}\ \{place\ form\}^*)$

▷ Set *place*s to primary values of *form*s. Return values of last *form*/NIL; work sequentially/in parallel, respectively.

$(\left\{\begin{array}{l}\overset{\text{sO}}{\textbf{setq}}\\\overset{\text{M}}{\textbf{psetq}}\end{array}\right\}\ \{symbol\ form\}^*)$

▷ Set *symbol*s to primary values of *form*s. Return value of last *form*/NIL; work sequentially/in parallel, respectively.

$(\overset{\text{Fu}}{\textbf{set}}\ \widetilde{symbol}\ foo)$

▷ Set *symbol*'s value cell to *foo*. Deprecated.

$(\overset{\text{M}}{\textbf{multiple-value-setq}}\ vars\ form)$

▷ Set elements of *vars* to the values of *form*. Return *form*'s primary value.

$(\overset{\text{M}}{\textbf{shiftf}}\ \widetilde{place}^+\ foo)$

▷ Store value of *foo* in rightmost *place* shifting values of *place*s left, returning first *place*.

$(\overset{\text{M}}{\textbf{rotatef}}\ \widetilde{place}^*)$

▷ Rotate values of *place*s left, old first becoming new last *place*'s value. Return NIL.

$(\overset{\text{Fu}}{\textbf{makunbound}}\ \widetilde{foo})$ ▷ Delete special variable *foo* if any.

$(\overset{\text{Fu}}{\textbf{get}}\ symbol\ key\ [default_{\boxed{\texttt{NIL}}}])$
$(\overset{\text{Fu}}{\textbf{getf}}\ place\ key\ [default_{\boxed{\texttt{NIL}}}])$

▷ First entry *key* from property list stored in *symbol*/in *place*, respectively, or *default* if there is no *key*. **setf**able.

$(\overset{\text{Fu}}{\textbf{get-properties}}\ property\text{-}list\ keys)$

▷ Return key and value$_2$ of first entry from *property-list* matching a key from *keys*, and tail of *property-list*$_3$ starting with that key. Return NIL, NIL$_2$, and NIL$_3$ if there was no matching key in *property-list*.

$(\overset{\text{Fu}}{\textbf{remprop}}\ \widetilde{symbol}\ key)$
$(\overset{\text{M}}{\textbf{remf}}\ \widetilde{place}\ key)$

▷ Remove first entry *key* from property list stored in *symbol*/in *place*, respectively. Return T if *key* was there, or NIL otherwise.

9.3 Functions

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

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

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

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

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

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

▷ 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 $\overset{\text{sO}}{\textbf{block}}$ around its corresponding *local-form**. Only for $\overset{\text{sO}}{\textbf{labels}}$, functions *foo* are visible inside *local-forms*. Return $\underline{\text{values of }\textit{forms}}$.

$\left(\overset{\text{sO}}{\textbf{function}}\ \left\{\begin{array}{l}\textit{foo}\\ (\overset{\text{M}}{\textbf{lambda}}\ \textit{form}^*)\end{array}\right\}\right)$

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

$\left(\overset{\text{Fu}}{\textbf{apply}}\ \left\{\begin{array}{l}\textit{function}\\ (\textbf{setf}\ \textit{function})\end{array}\right\}\ \textit{arg}^*\ \textit{args}\right)$

▷ $\underline{\text{Values of }\textit{function}}$ called with *args* and the list elements of *args*. **setf**able if *function* is one of $\overset{\text{Fu}}{\textbf{aref}}$, $\overset{\text{Fu}}{\textbf{bit}}$, and $\overset{\text{Fu}}{\textbf{sbit}}$.

$(\overset{\text{Fu}}{\textbf{funcall}}\ \textit{function}\ \text{arg}^*)$ ▷ $\underline{\text{Values of }\textit{function}}$ called with *args*.

$(\overset{\text{sO}}{\textbf{multiple-value-call}}\ \textit{function form}^*)$

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

$(\overset{\text{Fu}}{\textbf{values-list}}\ \textit{list})$ ▷ Return $\underline{\text{elements of }\textit{list}}$.

$(\overset{\text{Fu}}{\textbf{values}}\ \textit{foo}^*)$

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

$(\overset{\text{Fu}}{\textbf{multiple-value-list}}\ \textit{form})$ ▷ $\underline{\text{List of the values of }\textit{form}}$.

$(\overset{\text{M}}{\textbf{nth-value}}\ n\ \textit{form})$

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

$(\overset{\text{Fu}}{\textbf{complement}}\ \textit{function})$

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

$(\overset{\text{Fu}}{\textbf{constantly}}\ \textit{foo})$

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

$(\overset{\text{Fu}}{\textbf{identity}}\ \textit{foo})$ ▷ Return $\underline{\textit{foo}}$.

$(\overset{\text{Fu}}{\textbf{function-lambda-expression}}\ \textit{function})$

▷ 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*.

$\left(\overset{\text{Fu}}{\textbf{fdefinition}}\ \left\{\begin{array}{l}\textit{foo}\\ (\textbf{setf}\ \textit{foo})\end{array}\right\}\right)$

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

$(\overset{\text{Fu}}{\textbf{fmakunbound}}\ \textit{foo})$

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

$\overset{\text{co}}{\textbf{call-arguments-limit}}$
$\overset{\text{co}}{\textbf{lambda-parameters-limit}}$

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

$\overset{\text{co}}{\textbf{multiple-values-limit}}$

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

9.4 Macros

Below, macro lambda list ($macro\text{-}\lambda^*$) has the form of either

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

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

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

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

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

or

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

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

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

$(\begin{Bmatrix} \overset{\text{M}}{\textbf{defmacro}} \\ \overset{\text{Fu}}{\textbf{define-compiler-macro}} \end{Bmatrix}\ \begin{Bmatrix} foo \\ (\textbf{setf}\ foo) \end{Bmatrix}\ (macro\text{-}\lambda^*)$ (**declare** $\widehat{decl}^*)^*\ [\widehat{doc}]\ form^{\text{P}*})$

▷ Define macro $\underline{foo}$ which on evaluation as (*foo tree*) applies expanded *forms* to arguments from *tree*, which corresponds to *tree*-shaped *macro*-λs. *forms* are enclosed in an implicit $\overset{\text{sO}}{\textbf{block}}$ named *foo*.

($\overset{\text{M}}{\textbf{define-symbol-macro}}$ *foo form*)

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

($\overset{\text{sO}}{\textbf{macrolet}}$ $((foo\ (macro\text{-}\lambda^*)$ (**declare** $\widehat{local\text{-}decl}^*)^*\ [\widehat{doc}]$ $macro\text{-}form^{\text{P}*})^*)$ (**declare** $\widehat{decl}^*)^*\ form^{\text{P}*})$

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

($\overset{\text{sO}}{\textbf{symbol-macrolet}}$ $((foo\ expansion\text{-}form)^*)$ (**declare** $\widehat{decl}^*)^*\ form^{\text{P}*})$

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

($\overset{\text{M}}{\textbf{defsetf}}$ $\widehat{function}$

$\left\{\begin{matrix} \widehat{updater}\ [\widehat{doc}] \\ (setf\text{-}\lambda^*)\ (s\text{-}var^*)\ (\textbf{declare}\ \widehat{decl}^*)^*\ [\widehat{doc}]\ form^{\text{P}*} \end{matrix}\right\})$

where defsetf lambda list ($setf\text{-}\lambda^*$) has the form (var^*

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

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

$$[\textbf{\&allow-other-keys}]]\ \left[\textbf{\&environment}\ var\right])$$

▷ 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*), *forms* 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**. *forms* are enclosed in an implicit $\overset{\text{sO}}{\textbf{block}}$ named *function*.

($\overset{\text{M}}{\textbf{define-setf-expander}}$ *function* ($macro\text{-}\lambda^*$) (**declare** $\widehat{decl}^*)^*\ [\widehat{doc}]$ $form^{\text{P}*})$

▷ 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 $\overset{\text{Fu}}{\textbf{get-setf-expansion}}$ where the elements of macro lambda list *macro*-λ* are bound to corresponding *args*. *forms* are enclosed in an implicit $\overset{\text{sO}}{\textbf{block}}$ named *function*.

$(\overset{\text{Fu}}{\textbf{get-setf-expansion}}\ \textit{place}\ [\textit{environment}_{\boxed{\texttt{NIL}}}])$

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

$(\overset{\text{M}}{\textbf{define-modify-macro}}\ \textit{foo}\ ([\textbf{\&optional}\ \left\{\begin{array}{l}\textit{var}\\ (\textit{var}\ [\textit{init}_{\boxed{\texttt{NIL}}}\ [\textit{supplied-p}]])\end{array}\right\}^{*}]\ [\textbf{\&rest}\ \textit{var}])\ \textit{function}\ [\widehat{\textit{doc}}])$

▷ Define macro $\underline{\textit{foo}}$ 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.

$\overset{\text{co}}{\textbf{lambda-list-keywords}}$

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

&whole *var*

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

&optional *var**

▷ Bind *vars* to corresponding arguments if any.

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

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

&key *var**

▷ Bind *vars* 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 *vars* as in $\overset{\text{sO}}{\textbf{let*}}$.

9.5 Control Flow

$(\overset{\text{sO}}{\textbf{if}}\ \textit{test}\ \textit{then}\ [\textit{else}_{\boxed{\texttt{NIL}}}])$

▷ Return values of $\underline{\textit{then}}$ if *test* returns `T`; return values of $\underline{\textit{else}}$ otherwise.

$(\overset{\text{M}}{\textbf{cond}}\ (\textit{test}\ \textit{then}^{\overset{\text{P}}{*}}{}_{\boxed{\overline{\textit{test}}}})^{*})$

▷ Return the $\underline{\text{values}}$ of the first *then** whose *test* returns `T`; return $\underline{\texttt{NIL}}$ if all *tests* return `NIL`.

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

▷ Evaluate *foos* and return $\underline{\text{their values}}$ if *test* returns `T` or `NIL`, respectively. Return $\underline{\texttt{NIL}}$ otherwise.

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

▷ Return the $\underline{\text{values}}$ of the first *foo** one of whose *keys* is **eql** *test*. Return $\underline{\text{values of } \textit{bars}}$ if there is no matching *key*.

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

▷ Return the $\underline{\text{values}}$ of the first *foo** one of whose *keys* is **eql** *test*. Signal non-correctable/correctable **type-error** and return $\underline{\texttt{NIL}}$ if there is no matching *key*.

$(\overset{\text{M}}{\textbf{and}}\ \textit{form}^{*}{}_{\boxed{\texttt{T}}})$

▷ Evaluate *forms* from left to right. Immediately return $\underline{\texttt{NIL}}$ if one *form*'s value is `NIL`. Return $\underline{\text{values of last } \textit{form}}$ otherwise.

$(\overset{\text{M}}{\textbf{or}}\ \textit{form}^{*}{}_{\boxed{\texttt{NIL}}})$

▷ Evaluate *forms* from left to right. Immediately return $\underline{\text{primary value}}$ of first non-`NIL`-evaluating form, or $\underline{\text{all values}}$ if last *form* is reached. Return $\underline{\texttt{NIL}}$ if no *form* returns `T`.

$(\overset{\text{sO}}{\textbf{progn}}\ \textit{form}^{*}{}_{\boxed{\texttt{NIL}}})$

▷ Evaluate *forms* sequentially. Return $\underline{\text{values of last } \textit{form}}$.

($^{\text{sO}}$**multiple-value-prog1** *form-r form**)
($^{\text{M}}$**prog1** *form-r form**)
($^{\text{M}}$**prog2** *form-a form-r form**)

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

($\left\{\begin{array}{l}\overset{\text{sO}}{\textbf{let}}\\ \overset{\text{sO}}{\textbf{let*}}\end{array}\right\}$ ($\left\{\begin{array}{|l}name\\ (name\ [value_{\boxed{\texttt{NIL}}}])\end{array}\right\}^*$) (**declare** $\widehat{decl^*}$)* $form^{\text{P}*}$)

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

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

▷ Evaluate $^{\text{sO}}$**tagbody**-like body with *names* lexically bound (in parallel or sequentially, respectively) to *values*. Return NIL or explicitly $^{\text{M}}$**return**ed values. Implicitly, the whole form is a $^{\text{sO}}$**block** named NIL.

($^{\text{sO}}$**progv** *symbols values* $form^{\text{P}*}$)

▷ Evaluate *forms* with locally established dynamic bindings of *symbols* to *values* or NIL. Return values of *forms*.

($^{\text{sO}}$**unwind-protect** *protected cleanup**)

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

($^{\text{M}}$**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 their values. *destruct-λ* resembles *macro-λ* (section 9.4), but without any **&environment** clause.

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

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

($^{\text{sO}}$**block** *name* $form^{\text{P}*}$)

▷ Evaluate *forms* in a lexical environment, and return their values unless interrupted by $^{\text{sO}}$**return-from**.

($^{\text{sO}}$**return-from** *foo* [$result_{\boxed{\texttt{NIL}}}$])
($^{\text{M}}$**return** [$result_{\boxed{\texttt{NIL}}}$])

▷ Have nearest enclosing $^{\text{sO}}$**block** named *foo*/named NIL, respectively, return with values of *result*.

($^{\text{sO}}$**tagbody** {$\widehat{tag}$|*form*}*)

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

($^{\text{sO}}$**go** $\widehat{tag}$)

▷ Within the innermost possible enclosing $^{\text{sO}}$**tagbody**, jump to a tag **eql** *tag*.

($^{\text{sO}}$**catch** *tag* $form^{\text{P}*}$)

▷ Evaluate *forms* and return their values unless interrupted by $^{\text{sO}}$**throw**.

($^{\text{sO}}$**throw** *tag form*)

▷ Have the nearest dynamically enclosing $^{\text{sO}}$**catch** with a tag $^{\text{Fu}}$**eq** *tag* return with the values of *form*.

($^{\text{Fu}}$**sleep** *n*) ▷ Wait *n* seconds, return NIL.

9.6 Iteration

($\left\{\begin{array}{l}\overset{\text{M}}{\textbf{do}}\\ \overset{\text{M}}{\textbf{do*}}\end{array}\right\}$ ($\left\{\begin{array}{l}var\\ (var\ \big[start\ [step]\big])\end{array}\right\}^*$) (*stop* $result^{\text{P}*}$) (**declare** $\widehat{decl^*}$)* $\left\{\begin{array}{l}\widehat{tag}\\ form\end{array}\right\}^*$)

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

($\overset{\text{M}}{\textbf{dotimes}}$ (*var i* [$result_{\boxed{\texttt{NIL}}}$]) (**declare** $\widehat{decl^*}$)* {$\widehat{tag}$|*form*}*)

▷ Evaluate $\overset{\text{sO}}{\textbf{tagbody}}$-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 $\overset{\text{sO}}{\textbf{block}}$ named `NIL`.

($\overset{\text{M}}{\textbf{dolist}}$ (*var list* [$result_{\boxed{\texttt{NIL}}}$]) (**declare** $\widehat{decl^*}$)* {$\widehat{tag}$|*form*}*)

▷ Evaluate $\overset{\text{sO}}{\textbf{tagbody}}$-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 $\overset{\text{sO}}{\textbf{block}}$ named `NIL`.

9.7 Loop Facility

($\overset{\text{M}}{\textbf{loop}}$ *form**)

▷ **Simple Loop.** If *forms* do not contain any atomic Loop Facility keywords, evaluate them forever in an implicit $\overset{\text{sO}}{\textbf{block}}$ named `NIL`.

($\overset{\text{M}}{\textbf{loop}}$ *clause**)

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

named $n_{\boxed{\texttt{NIL}}}$ ▷ Give $\overset{\text{M}}{\textbf{loop}}$'s implicit $\overset{\text{sO}}{\textbf{block}}$ a name.

{**with** $\left\{\begin{matrix} var\text{-}s \\ (var\text{-}s^*) \end{matrix}\right\}$ [*d-type*] = *foo*}$^+$

{**and** $\left\{\begin{matrix} var\text{-}p \\ (var\text{-}p^*) \end{matrix}\right\}$ [*d-type*] = *bar*}*

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

$\left\{\textbf{fixnum}\middle|\textbf{float}\middle|\texttt{T}\middle|\texttt{NIL}\middle|\left\{\textbf{of-type} \left\{\begin{matrix} type \\ (type^*) \end{matrix}\right\}\right\}\right\}$

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

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

▷ 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.

{**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**} $package_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$]

▷ Bind *var* successively to the accessible symbols, or the present symbols, or the external symbols respectively, of *package*.

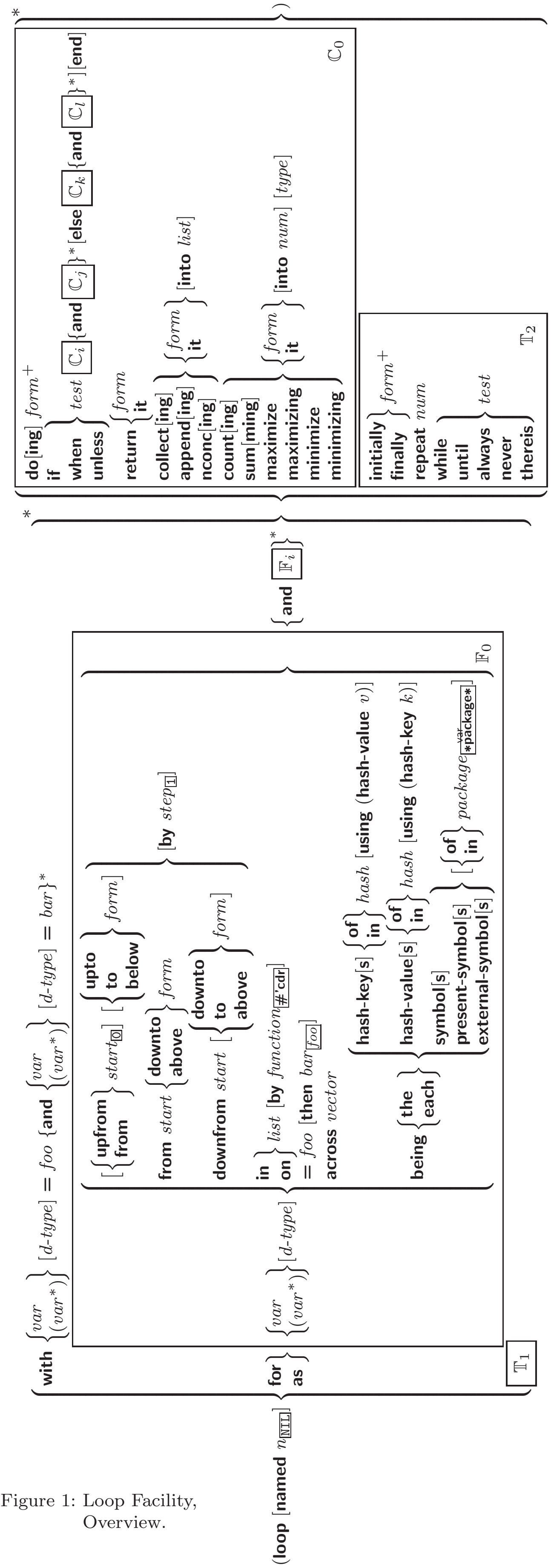
(loop [named n NIL]
with {var | (var*)} [d-type] = foo {and {var | (var*)} [d-type] = bar}*
for | as {var | (var*)} [d-type]
[{upfrom | from} start 0] [{upto | to | below} form] [by step 1]
from start {downto | above} form
downfrom start [{downto | to | above} form]
in | on list [by function #'cdr]
= 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 var *package*]
F0
{and F i}*
T1
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]
C0
initially | finally form+
repeat num
while | until | always | never | thereis test
T2
*
)

Figure 1: Loop Facility, Overview.

{**do**|**doing**} *form*$^+$

▷ Evaluate *form*s in every iteration.

{**if**|**when**|**unless**} *test i-clause* {**and** *j-clause*}* [**else** *k-clause* {**and** *l-clause*}*] [**end**]

▷ If *test* returns `T`, `T`, or `NIL`, respectively, evaluate *i-clause* and *j-clause*s; otherwise, evaluate *k-clause* and *l-clause*s.

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

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 $\overset{\text{Fu}}{\textbf{append}}$ or $\overset{\text{Fu}}{\textbf{nconc}}$, respectively. If no *list* is given, collect into an anonymous list which is returned after termination.

{**count**|**counting**} {*form*|**it**} [**into** *n*] [*type*]

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

{**sum**|**summing**} {*form*|**it**} [**into** *sum*] [*type*]

▷ Calculate the sum of the primary values of *form* or of **it**. If no *sum* is given, sum into an anonymous variable which is returned after termination.

{**maximize**|**maximizing**|**minimize**|**minimizing**} {*form*|**it**} [**into** *max-min*] [*type*]

▷ Determine the maximum or minimum, respectively, of the primary values of *form* or of **it**. If no *max-min* is given, use an anonymous variable which is returned after termination.

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

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

repeat *num*

▷ Terminate $\overset{\text{M}}{\textbf{loop}}$ after *num* iterations; *num* is evaluated once.

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

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

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

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

thereis *test*

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

($\overset{\text{M}}{\textbf{loop-finish}}$)

▷ Terminate $\overset{\text{M}}{\textbf{loop}}$ immediately executing any **finally** clauses and returning any accumulated results.

10 CLOS

10.1 Classes

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

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

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

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

$$\left\{\begin{vmatrix}(\textbf{:default-initargs}\ \{\textit{name value}\}^*) \\ (\textbf{:documentation}\ \textit{class-doc}) \\ (\textbf{:metaclass}\ \textit{name}_{\boxed{\textbf{standard-class}}})\end{vmatrix}\right\})$$

▷ Define, as a subclass of *superclass*es, class *foo*. In a new instance *i*, a *slot*'s value defaults to *form* unless set via **:***initarg-name*; it is readable via (*reader i*) or (*accessor i*), and writeable via (*writer i value*) or (**setf** (*accessor i*) *value*). With **:allocation :class**, *slot* is shared by all instances of class *foo*.

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

($^{\text{gF}}$**make-instance** *class* {**:***initarg value*}* *other-keyarg**)
▷ Make new instance of *class*.

($^{\text{gF}}$**reinitialize-instance** *instance* {**:***initarg value*}* *other-keyarg**)
▷ Change local slots of *instance* according to *initarg*s.

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

($^{\text{Fu}}$**slot-makunbound** *instance slot*)
▷ Make *slot* in *instance* unbound.

$$\left(\begin{Bmatrix}\overset{\text{M}}{\textbf{with-slots}}\ (\{\widehat{slot}\,|(\widehat{var}\ \widehat{slot})\}^*) \\ \overset{\text{M}}{\textbf{with-accessors}}\ ((\widehat{var}\ \widehat{accessor})^*)\end{Bmatrix}\ \textit{instance}\ (\textbf{declare}\ \widehat{decl}^*)^*\right.$$
form$^{\overset{\text{P}}{*}}$)
▷ Return values of *forms* after evaluating them in a lexical environment with slots of *instance* visible as **setf**able *slot*s or *var*s/with *accessor*s of *instance* visible as **setf**able *var*s.

($^{\text{gF}}$**class-name** *class*)
((**setf** $^{\text{gF}}$**class-name**) *new-name class*) ▷ Get/set name of *class*.

($^{\text{Fu}}$**class-of** *foo*) ▷ Class *foo* is a direct instance of.

($^{\text{gF}}$**change-class** $\widetilde{instance}$ *new-class* {**:***initarg value*}* *other-keyarg**)
▷ Change class of *instance* to *new-class*.

($^{\text{gF}}$**make-instances-obsolete** *class*)
▷ Update instances of *class*.

$$\left(\begin{Bmatrix}\overset{\text{gF}}{\textbf{initialize-instance}}\ (\textit{instance}) \\ \overset{\text{gF}}{\textbf{update-instance-for-different-class}}\ \textit{previous current}\end{Bmatrix}\right.$$
{**:***initarg value*}* *other-keyarg**)
▷ Its primary method sets slots on behalf of $^{\text{gF}}$**make-instance**/of $^{\text{gF}}$**change-class** by means of $^{\text{gF}}$**shared-initialize**.

($^{\text{gF}}$**update-instance-for-redefined-class** *instances added-slots discarded-slots property-list* {**:***initarg value*}* *other-keyarg**)
▷ Its primary method sets slots on behalf of $^{\text{gF}}$**make-instances-obsolete** by means of $^{\text{gF}}$**shared-initialize**.

($^{\text{gF}}$**allocate-instance** *class* {**:***initarg value*}* *other-keyarg**)
▷ Return uninitialized instance of *class*. Called by $^{\text{gF}}$**make-instance**.

$$(\overset{\text{gF}}{\textbf{shared-initialize}}\ \textit{instance} \begin{Bmatrix}\textit{slots}\\ \texttt{T}\end{Bmatrix} \{\textbf{:}\textit{initarg value}\}^*\ \textit{other-keyarg}^*)$$
▷ Fill *instance*'s *slots* using *initarg*s and **:initform** forms.

$$(\overset{\text{gF}}{\textbf{slot-missing}}\ \textit{class object slot} \begin{Bmatrix}\textbf{setf}\\ \textbf{slot-boundp}\\ \textbf{slot-makunbound}\\ \textbf{slot-value}\end{Bmatrix} [\textit{value}])$$
▷ Called in case of attempted access to missing *slot*. Its primary method signals **error**.

$(\overset{\text{gF}}{\textbf{slot-unbound}}\ \textit{class instance slot})$

▷ Called by $\overset{\text{Fu}}{\textbf{slot-value}}$ in case of unbound *slot*. Its primary method signals **unbound-slot**.

10.2 Generic Functions

$(\overset{\text{Fu}}{\textbf{next-method-p}})$

▷ $\underline{\texttt{T}}$ if enclosing method has a next method.

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

$$\big[\textbf{\&rest}\ \textit{var}\big]\ \Big[\textbf{\&key} \begin{Bmatrix}\textit{var}\\(\textit{var}\,\big|\,(\textbf{:}\textit{key var}))\end{Bmatrix}^*$$

$$[\textbf{\&allow-other-keys}]\Big])\ \left\{\left\| \begin{matrix}(\textbf{:argument-precedence-order}\ \textit{required-var}^+)\\ (\textbf{declare}\ (\textbf{optimize}\ \textit{arg}^*)^+)\\ (\textbf{:documentation}\ \widehat{\textit{string}})\\ (\textbf{:generic-function-class}\ \textit{class}_{\boxed{\textbf{standard-generic-function}}})\\ (\textbf{:method-class}\ \textit{class}_{\boxed{\textbf{standard-method}}})\\ (\textbf{:method-combination}\ \textit{c-type}_{\boxed{\textbf{standard}}}\ \textit{c-arg}^*)\\ (\textbf{:method}\ \textit{defmethod-args})^*\end{matrix}\right.\right\})$$

▷ Define $\underline{\text{generic function}}$ *foo*. *defmethod-args* resemble those of $\overset{\text{M}}{\textbf{defmethod}}$. For *c-type* see section 10.3.

$$(\overset{\text{Fu}}{\textbf{ensure-generic-function}} \begin{Bmatrix} \textit{foo} \\ (\textbf{setf}\ \textit{foo}) \end{Bmatrix}$$

$$\left\{\left\| \begin{matrix}\textbf{:argument-precedence-order}\ \textit{required-var}^+\\ \textbf{:declare}\ (\textbf{optimize}\ \textit{arg}^*)^+\\ \textbf{:documentation}\ \textit{string}\\ \textbf{:generic-function-class}\ \textit{class}\\ \textbf{:method-class}\ \textit{class}\\ \textbf{:method-combination}\ \textit{c-type}\ \textit{c-arg}^*\\ \textbf{:lambda-list}\ \textit{lambda-list}\\ \textbf{:environment}\ \textit{environment}\end{matrix}\right.\right\})$$

▷ Define or modify $\underline{\text{generic function}}$ *foo*. **:generic-function-class** and **:lambda-list** have to 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.

$$(\overset{\text{M}}{\textbf{defmethod}} \begin{Bmatrix}\textit{foo}\\(\textbf{setf}\ \textit{foo})\end{Bmatrix} \Big[\begin{Bmatrix}\textbf{:before}\\\textbf{:after}\\\textbf{:around}\\\textit{qualifier}^*\end{Bmatrix}_{\boxed{\text{primary method}}}\Big]$$

$$(\begin{Bmatrix}\textit{var}\\(\textit{spec-var} \begin{Bmatrix}\textit{class}\\(\textbf{eql}\ \textit{bar})\end{Bmatrix})\end{Bmatrix}^* \Big[\textbf{\&optional}$$

$$\begin{Bmatrix}\textit{var}\\(\textit{var}\ [\textit{init}\ [\textit{supplied-p}]])\end{Bmatrix}^*\Big]\ [\textbf{\&rest}\ \textit{var}]\ \Big[\textbf{\&key}$$

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

$$\Big[\textbf{\&aux} \begin{Bmatrix}\textit{var}\\(\textit{var}\ [\textit{init}])\end{Bmatrix}^*\Big])\ \left\{\left\| \begin{matrix}(\textbf{declare}\ \widehat{\textit{decl}}^*)^*\\ \widehat{\textit{doc}}\end{matrix}\right.\right\} \textit{form}^{\overset{\text{P}}{*}})$$

▷ Define $\underline{\text{new method}}$ 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 $\underline{\text{new method}}$ act like parameters of a function with body *form**. *forms* are enclosed in an implicit $\overset{\text{sO}}{\textbf{block}}$ *foo*. Applicable *qualifier*s depend on the **method-combination** type; see section 10.3.

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

▷ Add (if necessary) or remove (if any) *method* to/from $\underline{\textit{generic-function}}$.

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

▷ Return suitable $\underline{\text{method}}$, or signal **error**.

$(\overset{\text{gF}}{\textbf{compute-applicable-methods}}\ \textit{generic-function args})$

▷ $\underline{\text{List of methods}}$ suitable for *args*, most specific first.

($\overset{\text{Fu}}{\textbf{call-next-method}}$ *arg** $\boxed{\text{current args}}$)

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

($\overset{\text{gF}}{\textbf{no-applicable-method}}$ *generic-function arg**)

▷ Called on invocation of *generic-function* on *args* if there is no applicable method. Default method signals **error**.

($\left\{\begin{array}{l}\overset{\text{Fu}}{\textbf{invalid-method-error}}\ \textit{method}\\ \overset{\text{Fu}}{\textbf{method-combination-error}}\end{array}\right\}$ *control arg**)

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

($\overset{\text{gF}}{\textbf{no-next-method}}$ *generic-function method arg**)

▷ Called on invocation of **call-next-method** when there is no next method. Default method signals **error**.

($\overset{\text{gF}}{\textbf{function-keywords}}$ *method*)

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

($\overset{\text{gF}}{\textbf{method-qualifiers}}$ *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, $\overset{\text{Fu}}{\textbf{call-next-method}}$ 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 $\overset{\text{Fu}}{\textbf{call-next-method}}$ if any, or of the generic function; and which can call less specific primary methods via $\overset{\text{Fu}}{\textbf{call-next-method}}$. 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-types* defined by the short form of $\overset{\text{M}}{\textbf{define-method-combination}}$.

($\overset{\text{M}}{\textbf{define-method-combination}}$ *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, $\overset{\text{Fu}}{\textbf{call-next-method}}$ 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-methods* are ordered $\left[\left\{\begin{array}{l}\textbf{:most-specific-first}\\ \textbf{:most-specific-last}\end{array}\right\}_{\boxed{\textbf{:most-specific-first}}}\right]$ (specified as *c-arg* in $\overset{\text{M}}{\textbf{defgeneric}}$). Using *c-type* as the *qualifier* in $\overset{\text{M}}{\textbf{defmethod}}$ makes the method primary.

($\overset{\text{M}}{\textbf{define-method-combination}}$ *c-type* (*ord-λ**) ((*group*

$\left\{\begin{array}{l}\textbf{*}\\ (\textit{qualifier}^*\ [\textbf{*}])\\ \textit{predicate}\end{array}\right\}$

$\left\{\begin{array}{|l}\textbf{:description}\ \textit{control}\\ \textbf{:order}\ \left\{\begin{array}{l}\textbf{:most-specific-first}\\ \textbf{:most-specific-last}\end{array}\right\}_{\boxed{\textbf{:most-specific-first}}}\\ \textbf{:required}\ \textit{bool}\end{array}\right\}$)*)

$\left\{\begin{array}{|l}(\textbf{:arguments}\ \textit{method-combination-}\lambda^*)\\ (\textbf{:generic-function}\ \textit{symbol})\\ (\textbf{declare}\ \widehat{\textit{decl}^*})^*\\ \widehat{\textit{doc}}\end{array}\right\}$ $\textit{body}^{\text{P}}{}^*$)

▷ **Long Form.** Define new **method-combination** *c-type*. A call to a generic function using *c-type* will be equivalent to a call to the forms returned by *body** with *ord-λ** bound to *c-arg** (cf. $\overset{\text{M}}{\textbf{defgeneric}}$), with *symbol* bound to the generic function, with *method-combination-λ** bound to the arguments of the generic function, and with *groups* bound to lists of methods. An applicable method becomes a member of the leftmost *group* whose *predicate* or *qualifiers* match. Methods can be called via $\overset{\text{M}}{\textbf{call-method}}$. Lambda lists (*ord-λ**) and (*method-combination-λ**) according to *ord-λ* on p. 16, the latter enhanced by an optional **&whole** argument.

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

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

11 Conditions and Errors

For standardized condition types cf. Figure 2 on page 30.

$$(\overset{\text{M}}{\textbf{define-condition}}\ foo\ (parent\text{-}type^{*}\boxed{\textbf{condition}})$$

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

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

▷ Define, as a subtype of *parent-types*, 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 writeable via (*writer i value*) 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.

$$(\overset{\text{Fu}}{\textbf{make-condition}}\ type\ \{\textbf{:}initarg\text{-}name\ value\}^{*})$$

▷ Return new condition of *type*.

$$(\begin{Bmatrix} \overset{\text{Fu}}{\textbf{signal}} \\ \overset{\text{Fu}}{\textbf{warn}} \\ \overset{\text{Fu}}{\textbf{error}} \end{Bmatrix} \begin{Bmatrix} condition \\ type\ \{\textbf{:}initarg\text{-}name\ value\}^{*} \\ control\ arg^{*} \end{Bmatrix})$$

▷ Unless handled, signal as **condition**, **warning** or **error**, respectively, *condition* or a new condition of *type* or, with $\overset{\text{Fu}}{\textbf{format}}$ *control* and *args* (see p. 36), **simple-condition**, **simple-warning**, or **simple-error**, respectively. From $\overset{\text{Fu}}{\textbf{signal}}$ and $\overset{\text{Fu}}{\textbf{warn}}$, return `NIL`.

$$(\overset{\text{Fu}}{\textbf{cerror}}\ continue\text{-}control \begin{Bmatrix} condition\ continue\text{-}arg^{*} \\ type\ \{\textbf{:}initarg\text{-}name\ value\}^{*} \\ control\ arg^{*} \end{Bmatrix})$$

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

$$(\overset{\text{M}}{\textbf{ignore-errors}}\ \overset{\text{P}}{form}^{*})$$

▷ Return values of *forms* or, in case of **errors**, `NIL` and the $\underline{\text{condition}}_{2}$.

$$(\overset{\text{Fu}}{\textbf{invoke-debugger}}\ condition)$$

▷ Invoke debugger with *condition*.

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

▷ If *test*, which may depend on *place*s, returns NIL, signal as correctable **error** *condition* or a new condition of *type* or, with $\overset{\text{Fu}}{\textbf{format}}$ *control* and *arg*s (see p. 36), **error**. When using the debugger's continue option, *place*s can be altered before re-evaluation of *test*. Return $\underline{\texttt{NIL}}$.

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

▷ If, on evaluation of *foo*, a condition of *type* is signalled, evaluate matching *condition-form*s with *var* bound to the condition, and return $\underline{\text{their values}}$. Without a condition, bind *ord-λ*s to values of *foo* and return $\underline{\text{values of } \textit{forms}}$ or, without a **:no-error** clause, return $\underline{\text{values of } \textit{foo}}$. See p. 16 for (*ord-λ**).

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

▷ Return $\underline{\text{values of } \textit{forms}}$ after evaluating them with *condition-type*s dynamically bound to their respective *handler-function*s of argument condition.

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

▷ Return $\underline{\text{values of } \textit{forms}}$ unless *restart* is called during their evaluation. In this case, describe restart using $\overset{\text{Fu}}{\textbf{format}}$ *control* and *arg*s (see p. 36) and return $\underline{\texttt{NIL}}$ and $\underline{\texttt{T}}_{2}$.

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

▷ Evaluate *form* with dynamically established restarts *foo*. Return $\underline{\text{values of } \textit{form}}$ or, if by ($\overset{\text{Fu}}{\textbf{invoke-restart}}$ *foo arg**) one restart *foo* is called, use *string* or *report-function* (of a stream) to print a description of restart *foo* and return the $\underline{\text{values of its } \textit{restart-forms}}$. *arg-function* supplies appropriate *arg*s if *foo* is called by $\overset{\text{Fu}}{\textbf{invoke-restart-interactively}}$. If (*test-function condition*) returns T, *foo* is made visible under *condition*. *arg** matches (*ord-λ**); see p. 16 for the latter.

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

▷ Return $\underline{\text{values of } \textit{forms}}$ evaluated with *restart*s dynamically bound to *restart-function*s.

$$(\overset{\text{Fu}}{\textbf{invoke-restart}}\ \textit{restart arg}^*)$$
$$(\overset{\text{Fu}}{\textbf{invoke-restart-interactively}}\ \textit{restart})$$

▷ Call function associated with *restart* with arguments given or prompted for, respectively. If *restart* function returns, return $\underline{\text{its values}}$.

$$(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{compute-restarts}}\\ \overset{\text{Fu}}{\textbf{find-restart}}\ \textit{name}\end{Bmatrix}\ [\textit{condition}])$$

▷ Return list of $\underline{\text{all restarts}}$, or innermost $\underline{\textit{restart}\ \textit{name}}$, respectively, out of those either associated with *condition* or un-associated at all; or, without *condition*, out of all restarts. Return $\underline{\texttt{NIL}}$ if search is unsuccessful.

$(\overset{\text{Fu}}{\textbf{restart-name}}\ \textit{restart})$ ▷ $\underline{\text{Name of } \textit{restart}}$.

$$(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{abort}}\\ \overset{\text{Fu}}{\textbf{muffle-warning}}\\ \overset{\text{Fu}}{\textbf{continue}}\\ \overset{\text{Fu}}{\textbf{store-value}}\ \textit{value}\\ \overset{\text{Fu}}{\textbf{use-value}}\ \textit{value}\end{Bmatrix}\ [\textit{condition}_{\boxed{\texttt{NIL}}}])$$

▷ 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 $\overset{\text{Fu}}{\textbf{abort}}$ and $\overset{\text{Fu}}{\textbf{muffle-warning}}$, or return $\underline{\texttt{NIL}}$ for the rest.

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

(${}^{\text{Fu}}$**arithmetic-error-operation** *condition*)
(${}^{\text{Fu}}$**arithmetic-error-operands** *condition*)
▷ List of function or of its operands respectively, used in the operation which caused *condition*.

(${}^{\text{Fu}}$**cell-error-name** *condition*)
▷ Name of cell which caused *condition*.

(${}^{\text{Fu}}$**unbound-slot-instance** *condition*)
▷ Instance with unbound slot which caused *condition*.

(${}^{\text{Fu}}$**print-not-readable-object** *condition*)
▷ The object not readably printable under *condition*.

(${}^{\text{Fu}}$**package-error-package** *condition*)
(${}^{\text{Fu}}$**file-error-pathname** *condition*)
(${}^{\text{Fu}}$**stream-error-stream** *condition*)
▷ Package, path, or stream, respectively, which caused the *condition* of indicated type.

(${}^{\text{Fu}}$**type-error-datum** *condition*)
(${}^{\text{Fu}}$**type-error-expected-type** *condition*)
▷ Object which caused *condition* of type **type-error**, or its expected type, respectively.

(${}^{\text{Fu}}$**simple-condition-format-control** *condition*)
(${}^{\text{Fu}}$**simple-condition-format-arguments** *condition*)
▷ Return ${}^{\text{Fu}}$**format** control or list of ${}^{\text{Fu}}$**format** arguments, respectively, of *condition*.

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

${}^{\text{var}}$**∗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.

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

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

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

(${}^{\text{Fu}}$**coerce** *object type*) ▷ Coerce *object* into *type*.

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

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

(${}^{\text{Fu}}$**type-of** *foo*) ▷ Type of *foo*.

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

(${}^{\text{Fu}}$**stream-element-type** *stream*) ▷ Return type of *stream* objects.

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

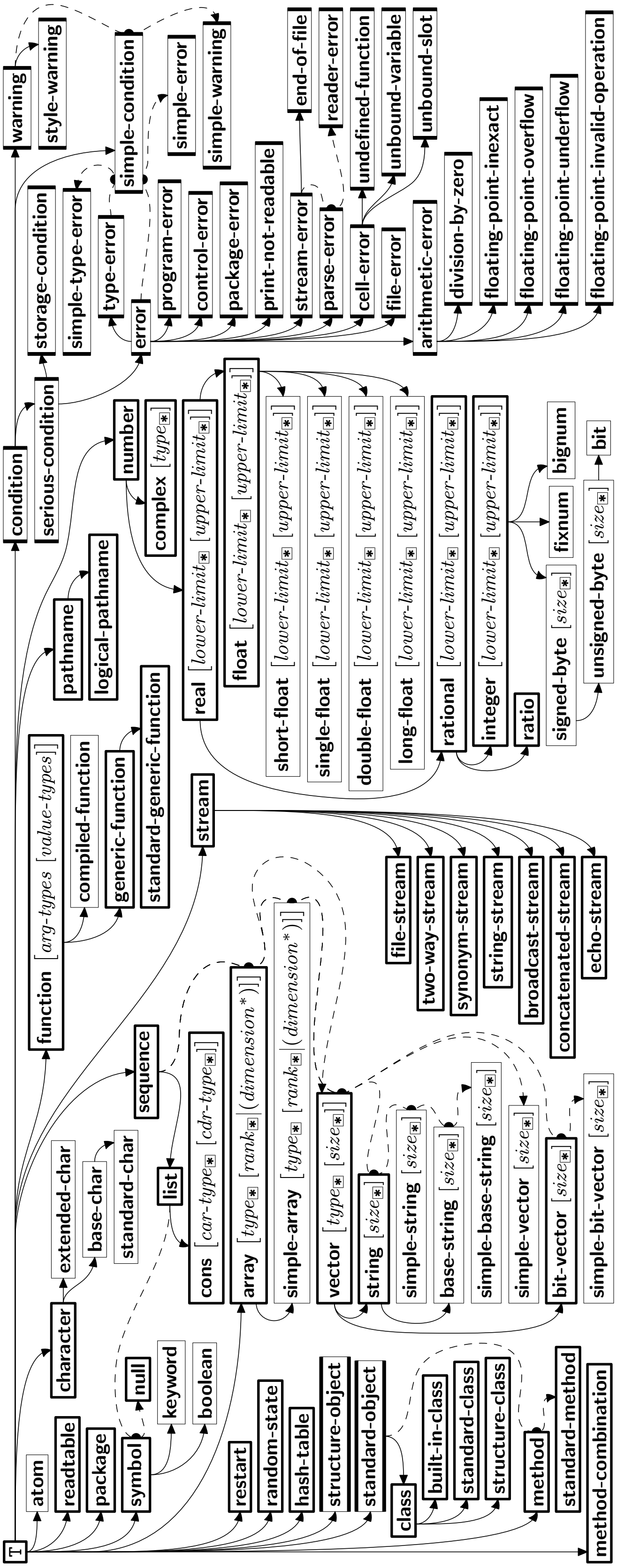

Figure 2: Precedence Order of System Classes (▭), Classes (▬), Types (▭), and Condition Types (▭).

($\overset{\text{Fu}}{\textbf{upgraded-array-element-type}}$ *type* [*environment*$_{\boxed{\texttt{NIL}}}$])

▷ <u>Element type</u> of most specialized array capable of holding elements of *type*.

($\overset{\text{M}}{\textbf{deftype}}$ *foo* (*macro-λ**) (**declare** $\widehat{decl}$*)* [$\widehat{doc}$] *form*$^{\text{P}}$*)

▷ Define type <u>*foo*</u> which when referenced as (*foo* $\widehat{arg}$*) applies expanded *forms* to *args* returning the new type. For (*macro-λ**) see p. 18 but with default value of ∗ instead of NIL. *forms* are enclosed in an implicit $\overset{\text{sO}}{\textbf{block}}$ named *foo*.

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

(**satisfies** *predicate*)

▷ Type specifier for all objects satisfying *predicate*.

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

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

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

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

(**values** *type** [**&optional** *type** [**&rest** *other-args*]])

▷ Type specifier for multiple values.

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

13 Input/Output

13.1 Predicates

($\overset{\text{Fu}}{\textbf{streamp}}$ *foo*)
($\overset{\text{Fu}}{\textbf{pathnamep}}$ *foo*) ▷ <u>T</u> if *foo* is of indicated type.
($\overset{\text{Fu}}{\textbf{readtablep}}$ *foo*)

($\overset{\text{Fu}}{\textbf{input-stream-p}}$ *stream*)
($\overset{\text{Fu}}{\textbf{output-stream-p}}$ *stream*)
($\overset{\text{Fu}}{\textbf{interactive-stream-p}}$ *stream*)
($\overset{\text{Fu}}{\textbf{open-stream-p}}$ *stream*)

▷ Return <u>T</u> if *stream* is for input, for output, interactive, or open, respectively.

($\overset{\text{Fu}}{\textbf{pathname-match-p}}$ *path wildcard*)

▷ <u>T</u> if *path* matches *wildcard*.

($\overset{\text{Fu}}{\textbf{wild-pathname-p}}$ *path* [{**:host**|**:device**|**:directory**|**:name**|**:type**|**:version**|NIL}])

▷ Return <u>T</u> if indicated component in *path* is wildcard. (NIL indicates any component.)

13.2 Reader

($\left\{\begin{array}{l}\overset{\text{Fu}}{\textbf{y-or-n-p}}\\ \overset{\text{Fu}}{\textbf{yes-or-no-p}}\end{array}\right\}$ [*control arg**])

▷ Ask user a question and return <u>T</u> or <u>NIL</u> depending on their answer. See p. 36, $\overset{\text{Fu}}{\textbf{format}}$, for *control* and *args*.

($\overset{\text{M}}{\textbf{with-standard-io-syntax}}$ *form*$^{\text{P}}$*)

▷ Evaluate *forms* with standard behaviour of reader and printer. Return <u>values of *forms*</u>.

($\left\{\begin{array}{l}\overset{\text{Fu}}{\textbf{read}}\\ \overset{\text{Fu}}{\textbf{read-preserving-whitespace}}\end{array}\right\}$ [$\widetilde{stream}$$_{\boxed{\overset{\text{var}}{\textbf{*standard-input*}}}}$ [*eof-err*$_{\boxed{\texttt{T}}}$ [*eof-val*$_{\boxed{\texttt{NIL}}}$ [*recursive*$_{\boxed{\texttt{NIL}}}$]]]])

▷ Read printed representation of <u>object</u>.

($\overset{\text{Fu}}{\textbf{read-from-string}}$ *string* [*eof-error*$_{\boxed{\texttt{T}}}$ [*eof-val*$_{\boxed{\texttt{NIL}}}$ [$\left\{\begin{array}{l}\textbf{:start}\ start_{\boxed{0}}\\ \textbf{:end}\ end_{\boxed{\texttt{NIL}}}\\ \textbf{:preserve-whitespace}\ bool_{\boxed{\texttt{NIL}}}\end{array}\right\}$]]])

▷ Return <u>object</u> read from string and zero-indexed $\underline{\text{position}}_{2}$ of next character.

($^{\text{Fu}}$**read-delimited-list** *char* [$\widetilde{\textit{stream}}$$_{\boxed{\overset{\text{var}}{\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.

($^{\text{Fu}}$**read-char** [$\widetilde{\textit{stream}}$$_{\boxed{\overset{\text{var}}{\text{*standard-input*}}}}$ [*eof-err*$_{\boxed{\texttt{T}}}$ [*eof-val*$_{\boxed{\texttt{NIL}}}$ [*recursive*$_{\boxed{\texttt{NIL}}}$]]]])

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

($^{\text{Fu}}$**read-char-no-hang** [$\widetilde{\textit{stream}}$$_{\boxed{\overset{\text{var}}{\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.

($^{\text{Fu}}$**peek-char** [*mode*$_{\boxed{\texttt{NIL}}}$ [$\widetilde{\textit{stream}}$$_{\boxed{\overset{\text{var}}{\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.

($^{\text{Fu}}$**unread-char** *character* [$\widetilde{\textit{stream}}$$_{\boxed{\overset{\text{var}}{\text{*standard-input*}}}}$])

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

($^{\text{Fu}}$**read-byte** $\widetilde{\textit{stream}}$ [*eof-err*$_{\boxed{\texttt{T}}}$ [*eof-val*$_{\boxed{\texttt{NIL}}}$]])

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

($^{\text{Fu}}$**read-line** [$\widetilde{\textit{stream}}$$_{\boxed{\overset{\text{var}}{\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.

($^{\text{Fu}}$**read-sequence** $\widetilde{\textit{sequence}}$ $\widetilde{\textit{stream}}$ [**:start** *start*$_{\boxed{0}}$][**:end** *end*$_{\boxed{\texttt{NIL}}}$])

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

($^{\text{Fu}}$**readtable-case** *readtable*)$_{\boxed{\text{:upcase}}}$

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

($^{\text{Fu}}$**copy-readtable** [*from-readtable*$_{\boxed{\overset{\text{var}}{\text{*readtable*}}}}$ [$\widetilde{\textit{to-readtable}}$$_{\boxed{\texttt{NIL}}}$]])

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

($^{\text{Fu}}$**set-syntax-from-char** *to-char* *from-char* [$\widetilde{\textit{to-readtable}}$$_{\boxed{\overset{\text{var}}{\text{*readtable*}}}}$ [*from-readtable*$_{\boxed{\text{standard readtable}}}$]])

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

$^{\text{var}}$***readtable*** ▷ Current readtable.

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

$^{\text{var}}$***read-default-float-format***$_{\boxed{\text{single-float}}}$

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

$^{\text{var}}$***read-suppress***$_{\boxed{\texttt{NIL}}}$

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

($^{\text{Fu}}$**set-macro-character** *char* *function* [*non-term-p*$_{\boxed{\texttt{NIL}}}$ [$\widetilde{\textit{rt}}$$_{\boxed{\overset{\text{var}}{\text{*readtable*}}}}$]])

▷ Make *char* a macro character associated with *function* of stream and *char*. Return $\underline{\texttt{T}}$.

($^{\text{Fu}}$**get-macro-character** *char* [*rt*$_{\boxed{\overset{\text{var}}{\text{*readtable*}}}}$])

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

($^{\text{Fu}}$**make-dispatch-macro-character** *char* [*non-term-p*$_{\boxed{\texttt{NIL}}}$ [*rt*$_{\boxed{\overset{\text{var}}{\text{*readtable*}}}}$]])

▷ Make *char* a dispatching macro character. Return $\underline{\texttt{T}}$.

($^{\text{Fu}}$**set-dispatch-macro-character** *char* *sub-char* *function* [$\widetilde{\textit{rt}}$$_{\boxed{\overset{\text{var}}{\text{*readtable*}}}}$])

▷ Make *function* of stream, *n*, *sub-char* a dispatch function of *char* followed by *n*, followed by *sub-char*. Return $\underline{\texttt{T}}$.

($^{\text{Fu}}$**get-dispatch-macro-character** *char* *sub-char* [*rt*$_{\boxed{\overset{\text{var}}{\text{*readtable*}}}}$])

▷ $\underline{\text{Dispatch function}}$ 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* ▷ ($\overset{\text{sO}}{\textbf{quote}}$ *foo*); *foo* unevaluated.

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

▷ Backquote. $\overset{\text{sO}}{\textbf{quote}}$ *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* ▷ ($\overset{\text{Fu}}{\textbf{character}}$ "*c*"), the character *c*.

#B*n*; **#O***n*; *n***.**; **#X***n*; **#***r***R***n*

▷ Integer of radix 2, 8, 10, 16, or *r*; $2 \leq r \leq 36$.

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

$\Big\{[m].n\big[\{\textsf{S}|\textsf{F}|\textsf{D}|\textsf{L}|\textsf{E}\}x_{\boxed{\textsf{E0}}}\big]\,\big|\,m\,[.[n]]\,\{\textsf{S}|\textsf{F}|\textsf{D}|\textsf{L}|\textsf{E}\}x\Big\}$

▷ $m.n \cdot 10^x$ as **short-float**, **single-float**, **double-float**, **long-float**, or the type from ***read-default-float-format***.

#C(*a b***)** ▷ ($\overset{\text{Fu}}{\textbf{complex}}$ *a b*), the complex number $a + b$i.

#'*foo* ▷ ($\overset{\text{sO}}{\textbf{function}}$ *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*.

$\overset{\text{var}}{\textbf{*read-eval*}}$$\boxed{\texttt{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 $\overset{\text{var}}{\textbf{*features*}}$, or ({**and**|**or**} *feature**), or (**not** *feature*).

$\overset{\text{var}}{\textbf{*features*}}$

▷ List of symbols denoting implementation-dependent features.

|*c****|**; **\***c*

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

13.4 Printer

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{prin1}}\\ \overset{\text{Fu}}{\textbf{print}}\\ \overset{\text{Fu}}{\textbf{pprint}}\\ \overset{\text{Fu}}{\textbf{princ}}\end{Bmatrix} \textit{foo}\ [\widetilde{\textit{stream}}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}]\right)$

▷ Print *foo* to *stream* $\overset{\text{Fu}}{\textbf{read}}$ably, $\overset{\text{Fu}}{\textbf{read}}$ably between a newline and a space, $\overset{\text{Fu}}{\textbf{read}}$ably after a newline, or human-readably without any extra characters, respectively. $\overset{\text{Fu}}{\textbf{prin1}}$, $\overset{\text{Fu}}{\textbf{print}}$ and $\overset{\text{Fu}}{\textbf{princ}}$ return $\underline{\textit{foo}}$.

$(\overset{\text{Fu}}{\textbf{prin1-to-string}}\ \textit{foo})$
$(\overset{\text{Fu}}{\textbf{princ-to-string}}\ \textit{foo})$

▷ Print *foo* to $\underline{\textit{string}}$ $\overset{\text{Fu}}{\textbf{read}}$ably or human-readably, respectively.

$(\overset{\text{gF}}{\textbf{print-object}}\ \textit{object}\ \widetilde{\textit{stream}})$

▷ Print $\underline{\textit{object}}$ to *stream*. Called by the Lisp printer.

$(\overset{\text{M}}{\textbf{print-unreadable-object}}\ (\textit{foo}\ \widetilde{\textit{stream}}\ \left\{\begin{vmatrix}\textbf{:type}\ \textit{bool}_{\boxed{\text{NIL}}}\\ \textbf{:identity}\ \textit{bool}_{\boxed{\text{NIL}}}\end{vmatrix}\right\})\ \textit{form}^{\text{P}_*})$

▷ Enclosed in **#<** and **>**, print *foo* by means of *forms* to *stream*. Return $\underline{\texttt{NIL}}$.

$(\overset{\text{Fu}}{\textbf{terpri}}\ [\widetilde{\textit{stream}}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}])$

▷ Output a newline to *stream*. Return $\underline{\texttt{NIL}}$.

$(\overset{\text{Fu}}{\textbf{fresh-line}})\ [\widetilde{\textit{stream}}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}]$

▷ Output a newline to *stream* and return $\underline{\texttt{T}}$ unless *stream* is already at the start of a line.

$(\overset{\text{Fu}}{\textbf{write-char}}\ \textit{char}\ [\widetilde{\textit{stream}}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}])$

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

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{write-string}}\\ \overset{\text{Fu}}{\textbf{write-line}}\end{Bmatrix}\ \textit{string}\ \left[\widetilde{\textit{stream}}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}\ \left[\left\{\begin{vmatrix}\textbf{:start}\ \textit{start}_{\boxed{0}}\\ \textbf{:end}\ \textit{end}_{\boxed{\text{NIL}}}\end{vmatrix}\right\}\right]\right]\right)$

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

$(\overset{\text{Fu}}{\textbf{write-byte}}\ \textit{byte}\ \widetilde{\textit{stream}})$ ▷ Write $\underline{\textit{byte}}$ to binary *stream*.

$(\overset{\text{Fu}}{\textbf{write-sequence}}\ \textit{sequence}\ \widetilde{\textit{stream}}\ \left\{\begin{vmatrix}\textbf{:start}\ \textit{start}_{\boxed{0}}\\ \textbf{:end}\ \textit{end}_{\boxed{\text{NIL}}}\end{vmatrix}\right\})$

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

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

▷ Print *foo* to *stream* and return $\underline{\textit{foo}}$, or print *foo* into $\underline{\textit{string}}$, respectively, after dynamically setting printer variables corresponding to keyword parameters (***print-***bar****** becoming **:***bar*). (**:stream** keyword with $\overset{\text{Fu}}{\textbf{write}}$ only.)

$(\overset{\text{Fu}}{\textbf{pprint-fill}}\ \widetilde{\textit{stream}\ \textit{foo}}\ [\textit{parenthesis}_{\boxed{\text{T}}}\ [\textit{noop}]])$
$(\overset{\text{Fu}}{\textbf{pprint-tabular}}\ \widetilde{\textit{stream}}\ \textit{foo}\ [\textit{parenthesis}_{\boxed{\text{T}}}\ [\textit{noop}\ [\textit{n}_{\boxed{16}}]]])$
$(\overset{\text{Fu}}{\textbf{pprint-linear}}\ \widetilde{\textit{stream}}\ \textit{foo}\ [\textit{parenthesis}_{\boxed{\text{T}}}\ [\textit{noop}]])$

▷ 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 $\overset{\text{Fu}}{\textbf{format}}$ directive ~//.

(${}^{\text{M}}$**pprint-logical-block** ($\widetilde{stream}$ *list* $\left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:prefix}\ \textit{string}\\ \textbf{:per-line-prefix}\ \textit{string}\end{array}\right\}\\ \textbf{:suffix}\ \textit{string}_{\boxed{""}}\end{array}\right.\right\}$)

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

▷ Evaluate *forms*, which should print *list*, with *stream* locally bound to a pretty printing stream which outputs to the original *stream*. If *list* is in fact not a list, it is printed by ${}^{\text{Fu}}$**write**. Return $\underline{\texttt{NIL}}$.

(${}^{\text{M}}$**pprint-pop**)

▷ Take $\underline{\text{next element}}$ off *list*. If there is no remaining tail of *list*, or ${}^{\text{var}}$***print-length*** or ${}^{\text{var}}$***print-circle*** indicate printing should end, send element together with an appropriate indicator to *stream*.

(${}^{\text{Fu}}$**pprint-tab** $\left\{\begin{array}{l}\textbf{:line}\\ \textbf{:line-relative}\\ \textbf{:section}\\ \textbf{:section-relative}\end{array}\right\}$ *c i* [$\widetilde{stream}_{\boxed{\text{*standard-output*}}}$])

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

(${}^{\text{Fu}}$**pprint-indent** $\left\{\begin{array}{l}\textbf{:block}\\ \textbf{:current}\end{array}\right\}$ *n* [$\widetilde{stream}_{\boxed{\text{*standard-output*}}}$])

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

(${}^{\text{M}}$**pprint-exit-if-list-exhausted**)

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

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

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

${}^{\text{var}}$***print-array*** ▷ If T, print arrays ${}^{\text{Fu}}$**read**ably.

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

${}^{\text{var}}$***print-case***$_{\boxed{\text{:upcase}}}$

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

${}^{\text{var}}$***print-circle***$_{\boxed{\text{NIL}}}$

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

${}^{\text{var}}$***print-escape***$_{\boxed{\text{T}}}$

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

${}^{\text{var}}$***print-gensym***$_{\boxed{\text{T}}}$

▷ If T, print **#:** before uninterned symbols.

${}^{\text{var}}$***print-length***$_{\boxed{\text{NIL}}}$
${}^{\text{var}}$***print-level***$_{\boxed{\text{NIL}}}$
${}^{\text{var}}$***print-lines***$_{\boxed{\text{NIL}}}$

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

${}^{\text{var}}$***print-miser-width***

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

${}^{\text{var}}$***print-pretty*** ▷ If T, print pretty.

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

${}^{\text{var}}$***print-readably***$_{\boxed{\text{NIL}}}$

▷ If T, print ${}^{\text{Fu}}$**read**ably or signal error **print-not-readable**.

${}^{\text{var}}$***print-right-margin***$_{\boxed{\text{NIL}}}$

▷ Right margin width in ems while pretty-printing.

($\overset{\text{Fu}}{\textbf{set-pprint-dispatch}}$ *type function* [*priority*$_{\boxed{0}}$
[*table*$_{\boxed{\overset{\text{var}}{\textbf{*print-pprint-dispatch*}}}}$]])

▷ 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 $\underline{\texttt{NIL}}$.

($\overset{\text{Fu}}{\textbf{pprint-dispatch}}$ *foo* [*table*$_{\boxed{\overset{\text{var}}{\textbf{*print-pprint-dispatch*}}}}$])

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

($\overset{\text{Fu}}{\textbf{copy-pprint-dispatch}}$ [*table*$_{\boxed{\overset{\text{var}}{\textbf{*print-pprint-dispatch*}}}}$])

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

$\overset{\text{var}}{\textbf{*print-pprint-dispatch*}}$ ▷ Current pretty print dispatch table.

13.5 Format

($\overset{\text{M}}{\textbf{formatter}}$ $\widehat{\textit{control}}$)

▷ Return $\underline{\text{function}}$ of stream and a **&rest** argument applying $\overset{\text{Fu}}{\textbf{format}}$ to stream, *control*, and the **&rest** argument returning `NIL` or any excess arguments.

($\overset{\text{Fu}}{\textbf{format}}$ {`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 $\overset{\text{M}}{\textbf{formatter}}$ which is then applied to *out-stream* and *arg**. Output to *out-string*, *out-stream* or, if first argument is `T`, to $\overset{\text{var}}{\textbf{*standard-output*}}$. Return $\underline{\texttt{NIL}}$. If first argument is `NIL`, return $\underline{\text{formatted output}}$.

~ [*min-col*$_{\boxed{0}}$] [,[*col-inc*$_{\boxed{1}}$] [,[*min-pad*$_{\boxed{0}}$] [,'*pad-char*$_{\boxed{␣}}$]]]
[**:**] [**@**] {**A**|**S**}

▷ **Aesthetic/Standard.** Print argument of any type for consumption by humans/by the reader, respectively. With **:**, print `NIL` as **()** rather than **nil**; with **@**, add *pad-char*s on the left rather than on the right.

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

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

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

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

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

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

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

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

~ [*width*] [,[*int-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 *int-digits* before decimal point and *exp-digits* in the signed exponent. With **~G**, choose either **~E** or **~F**. With **@**, always prepend a sign.

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

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

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

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

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

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

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

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

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

~ [$n_{\boxed{1}}$] **&**

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

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

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

{**~:**↩ | **~@**↩ | **~**↩}

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

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

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

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

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

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

▷ **Logical Block.** Act like **pprint-logical-block** using *body* as $\overset{\text{Fu}}{\textbf{format}}$ 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 $\overset{\text{Fu}}{\textbf{format}}$ 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 ith if given) *text* as a $\overset{\text{Fu}}{\textbf{format}}$ 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. With **@**, take one argument as control string and use then the rest of the original arguments.

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

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

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

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

{**V**|**#**}

▷ In place of the comma-separated prefix parameters: use next argument or number of remaining unprocessed arguments, respectively.

13.6 Streams

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

▷ Open **file-stream** to *path*.

($^{\text{Fu}}$**make-concatenated-stream** *input-stream**)
($^{\text{Fu}}$**make-broadcast-stream** *output-stream**)
($^{\text{Fu}}$**make-two-way-stream** *input-stream-part output-stream-part*)
($^{\text{Fu}}$**make-echo-stream** *from-input-stream to-output-stream*)
($^{\text{Fu}}$**make-synonym-stream** *variable-bound-to-stream*)

▷ Return stream of indicated type.

($^{\text{Fu}}$**make-string-input-stream** *string* $\left[start_{\boxed{0}}\ \left[end_{\boxed{\texttt{NIL}}}\right]\right]$)

▷ Return a **string-stream** supplying the characters from *string*.

($^{\text{Fu}}$**make-string-output-stream** [**:element-type** $type_{\boxed{\textbf{character}}}$])

▷ Return a **string-stream** accepting characters (available via $^{\text{Fu}}$**get-output-stream-string**).

($^{\text{Fu}}$**concatenated-stream-streams** *concatenated-stream*)
($^{\text{Fu}}$**broadcast-stream-streams** *broadcast-stream*)

▷ Return list of streams *concatenated-stream* still has to read from/*broadcast-stream* is broadcasting to.

($^{\text{Fu}}$**two-way-stream-input-stream** *two-way-stream*)
($^{\text{Fu}}$**two-way-stream-output-stream** *two-way-stream*)
($^{\text{Fu}}$**echo-stream-input-stream** *echo-stream*)
($^{\text{Fu}}$**echo-stream-output-stream** *echo-stream*)

▷ Return source stream or sink stream of *two-way-stream*/*echo-stream*, respectively.

($^{\text{Fu}}$**synonym-stream-symbol** *synonym-stream*)

▷ Return symbol of *synonym-stream*.

($^{\text{Fu}}$**get-output-stream-string** $\widetilde{string\text{-}stream}$)

▷ Clear and return as a string characters on *string-stream*.

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

▷ Return position within stream, or set it to *position* and return `T` on success.

($\overset{\text{Fu}}{\textbf{file-string-length}}$ *stream foo*)

▷ Length *foo* would have in *stream*.

($\overset{\text{Fu}}{\textbf{listen}}$ [*stream*$_{\boxed{\overset{\text{var}}{\textbf{*standard-input*}}}}$])

▷ `T` if there is a character in input *stream*.

($\overset{\text{Fu}}{\textbf{clear-input}}$ [$\widetilde{\textit{stream}}_{\boxed{\overset{\text{var}}{\textbf{*standard-input*}}}}$])

▷ Clear input from *stream*, return `NIL`.

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

▷ End output to *stream* and return `NIL` immediately, after initiating flushing of buffers, or after flushing of buffers, respectively.

($\overset{\text{Fu}}{\textbf{close}}$ $\widetilde{\textit{stream}}$ [**:abort** *bool*$_{\boxed{\texttt{NIL}}}$])

▷ Close *stream*. Return `T` if *stream* had been open. If **:abort** is `T`, delete associated file.

($\overset{\text{M}}{\textbf{with-open-file}}$ (*stream path open-arg**) (**declare** $\widehat{\textit{decl}}$*)* $\overset{\text{P}}{\textit{form}}$*)

▷ Use $\overset{\text{Fu}}{\textbf{open}}$ with *open-arg*s to temporarily create *stream* to *path*; return values of *forms*.

($\overset{\text{M}}{\textbf{with-open-stream}}$ (*foo* $\widetilde{\textit{stream}}$) (**declare** $\widehat{\textit{decl}}$*)* $\overset{\text{P}}{\textit{form}}$*)

▷ Evaluate *forms* with *foo* locally bound to *stream*. Return values of *forms*.

($\overset{\text{M}}{\textbf{with-input-from-string}}$ (*foo string* $\left\{\begin{array}{|l}\textbf{:index}\ \widetilde{\textit{index}}\\ \textbf{:start}\ \textit{start}_{\boxed{0}}\\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}\end{array}\right\}$) (**declare** $\widehat{\textit{decl}}$*)* $\overset{\text{P}}{\textit{form}}$*)

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

($\overset{\text{M}}{\textbf{with-output-to-string}}$ (*foo* [$\widetilde{\textit{string}}_{\boxed{\texttt{NIL}}}$] [**:element-type** *type*$_{\boxed{\textbf{character}}}$]) (**declare** $\widehat{\textit{decl}}$*)* $\overset{\text{P}}{\textit{form}}$*)

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

($\overset{\text{Fu}}{\textbf{stream-external-format}}$ *stream*)

▷ External file format designator.

$\overset{\text{var}}{\textbf{*terminal-io*}}$ ▷ Bidirectional stream to user terminal.

$\overset{\text{var}}{\textbf{*standard-input*}}$
$\overset{\text{var}}{\textbf{*standard-output*}}$
$\overset{\text{var}}{\textbf{*error-output*}}$

▷ Standard input stream, standard output stream, or standard error output stream, respectively.

$\overset{\text{var}}{\textbf{*debug-io*}}$
$\overset{\text{var}}{\textbf{*query-io*}}$

▷ Bidirectional streams for debugging and user interaction.

13.7 Pathnames and Files

($^{\text{Fu}}$**make-pathname**

$$\left\{\begin{array}{|l}
\textbf{:host } \{host|\texttt{NIL}|\textbf{:unspecific}\} \\
\textbf{:device } \{device|\texttt{NIL}|\textbf{:unspecific}\} \\
\textbf{:directory } \left\{\begin{array}{l}\{directory|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\} \\ (\left\{\begin{array}{l}\textbf{:absolute}\\ \textbf{:relative}\end{array}\right\} \left\{\begin{array}{l}directory\\ \textbf{:wild}\\ \textbf{:wild-inferiors}\\ \textbf{:up}\\ \textbf{:back}\end{array}\right\}^{*})\end{array}\right. \\
\textbf{:name } \{file\text{-}name|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\} \\
\textbf{:type } \{file\text{-}type|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\} \\
\textbf{:version } \{\textbf{:newest}|version|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\} \\
\textbf{:defaults } path_{\boxed{\text{host from } \overset{\text{var}}{\textbf{*default-pathname-defaults*}}}} \\
\textbf{:case } \{\textbf{:local}|\textbf{:common}\}_{\boxed{\textbf{:local}}}
\end{array}\right\})$$

▷ Construct pathname. For **:case :local**, leave case of components unchanged. For **:case :common**, leave mixed-case components unchanged; convert all-uppercase components into local customary case; do the opposite with all-lowercase components.

$$(\left\{\begin{array}{l}\overset{\text{Fu}}{\textbf{pathname-host}}\\ \overset{\text{Fu}}{\textbf{pathname-device}}\\ \overset{\text{Fu}}{\textbf{pathname-directory}}\\ \overset{\text{Fu}}{\textbf{pathname-name}}\\ \overset{\text{Fu}}{\textbf{pathname-type}}\end{array}\right\} path\ [\textbf{:case} \left\{\begin{array}{l}\textbf{:local}\\ \textbf{:common}\end{array}\right\}_{\boxed{\textbf{:local}}}])$$

($^{\text{Fu}}$**pathname-version** *path*)

▷ Return pathname component.

($^{\text{Fu}}$**parse-namestring** *foo* [*host*

$$[default\text{-}pathname_{\boxed{\overset{\text{var}}{\textbf{*default-pathname-defaults*}}}} \left\{\begin{array}{|l}\textbf{:start } start_{\boxed{0}}\\ \textbf{:end } end_{\boxed{\texttt{NIL}}}\\ \textbf{:junk-allowed } bool_{\boxed{\texttt{NIL}}}\end{array}\right\}]])$$

▷ Return pathname converted from string, pathname, or stream *foo*; and $\underset{2}{\text{position}}$ where parsing stopped.

($^{\text{Fu}}$**merge-pathnames** *pathname*

$[default\text{-}pathname_{\boxed{\overset{\text{var}}{\textbf{*default-pathname-defaults*}}}}$

$[default\text{-}version_{\boxed{\textbf{:newest}}}]])$

▷ Return *pathname* after filling in missing components from *default-pathname*.

$^{\text{var}}$***default-pathname-defaults***

▷ Pathname to use if one is needed and none supplied.

($^{\text{Fu}}$**user-homedir-pathname** [*host*]) ▷ User's home directory.

($^{\text{Fu}}$**enough-namestring** *path* [$root\text{-}path_{\boxed{\overset{\text{var}}{\textbf{*default-pathname-defaults*}}}}$])

▷ Return minimal path string to sufficiently describe *path* relative to *root-path.*

($^{\text{Fu}}$**namestring** *path*)
($^{\text{Fu}}$**file-namestring** *path*)
($^{\text{Fu}}$**directory-namestring** *path*)
($^{\text{Fu}}$**host-namestring** *path*)

▷ Return string representing full pathname; name, type, and version; directory name; or host name, respectively, of *path*.

($^{\text{Fu}}$**translate-pathname** *path wildcard-path-a wildcard-path-b*)

▷ Translate *path* from *wildcard-path-a* into *wildcard-path-b*. Return new path.

($^{\text{Fu}}$**pathname** *path*) ▷ Pathname of *path*.

($^{\text{Fu}}$**logical-pathname** *logical-path*)

▷ Logical pathname of *logical-path*. Logical pathnames are represented as all-uppercase $\texttt{\#P"}[host\texttt{:}][\texttt{:}]\{\left\{\begin{array}{l}\{dir|\texttt{*}\}^{+}\\ \texttt{**}\end{array}\right\}\texttt{;}\}^{*}$ $\{name|\texttt{*}\}^{*}[\texttt{.}\left\{\begin{array}{l}\{type|\texttt{*}\}^{+}\\ \texttt{LISP}\end{array}\right\}[\texttt{.}\{version|\texttt{*}|\texttt{newest}|\texttt{NEWEST}\}]]\texttt{"}$.

($^{\text{Fu}}$**logical-pathname-translations** *logical-host*)

▷ List of (*from-wildcard to-wildcard*) translations for *logical-host*. **setf**able.

($^{\text{Fu}}$**load-logical-pathname-translations** *logical-host*)
▷ Load *logical-host*'s translations. Return $\underline{\texttt{NIL}}$ if already loaded; return $\underline{\texttt{T}}$ if successful.

($^{\text{Fu}}$**translate-logical-pathname** *pathname*)
▷ $\underline{\text{Physical pathname}}$ corresponding to (possibly logical) *pathname*.

($^{\text{Fu}}$**probe-file** *file*)
($^{\text{Fu}}$**truename** *file*)
▷ $\underline{\text{Canonical name}}$ of *file*. If *file* does not exist, return $\underline{\texttt{NIL}}$/signal **file-error**, respectively.

($^{\text{Fu}}$**file-write-date** *file*) ▷ $\underline{\text{Time}}$ at which *file* was last written.

($^{\text{Fu}}$**file-author** *file*) ▷ Return $\underline{\text{name of } \textit{file} \text{ owner}}$.

($^{\text{Fu}}$**file-length** *stream*) ▷ Return $\underline{\text{length of } \textit{stream}}$.

($^{\text{Fu}}$**rename-file** *foo bar*)
▷ Rename file *foo* to *bar*. Unspecified components of path *bar* default to those of *foo*. Return $\underline{\text{new pathname}}$, $\underline{\text{old physical file name}}_2$, and $\underline{\text{new physical file name}}_3$.

($^{\text{Fu}}$**delete-file** *file*) ▷ Delete *file*. Return $\underline{\texttt{T}}$.

($^{\text{Fu}}$**directory** *path*) ▷ $\underline{\text{List of pathnames}}$ matching *path*.

($^{\text{Fu}}$**ensure-directories-exist** *path* [**:verbose** *bool*])
▷ Create parts of $\underline{\textit{path}}$ if necessary. Second return value is $\underline{\texttt{T}}_2$ if something has been created.

14 Packages and Symbols

14.1 Predicates

($^{\text{Fu}}$**symbolp** *foo*)
($^{\text{Fu}}$**packagep** *foo*) ▷ $\underline{\texttt{T}}$ if *foo* is of indicated type.
($^{\text{Fu}}$**keywordp** *foo*)

14.2 Packages

:*bar*|**keyword:***bar* ▷ Keyword, evaluates to $\underline{:\!\textit{bar}}$.

*package***:***symbol* ▷ Exported *symbol* of *package*.

*package***::***symbol* ▷ Possibly unexported *symbol* of *package*.

($^{\text{M}}$**defpackage** *foo* {
|(**:nicknames** *nick**)*
|(**:documentation** *string*)
|(**:intern** *interned-symbol**)*
|(**:use** *used-package**)*
|(**:import-from** *pkg imported-symbol**)*
|(**:shadowing-import-from** *pkg shd-symbol**)*
|(**:shadow** *shd-symbol**)*
|(**:export** *exported-symbol**)*
|(**:size** *int*)
})

▷ Create or modify $\underline{\text{package } \textit{foo}}$ with *interned-symbols*, symbols from *used-packages*, *imported-symbols*, and *shd-symbols*. Add *shd-symbols* to *foo*'s shadowing list.

($^{\text{Fu}}$**make-package** *foo* {|**:nicknames** (*nick**)$_{\boxed{\texttt{NIL}}}$ |**:use** (*used-package**)})
▷ Create $\underline{\text{package } \textit{foo}}$.

($^{\text{Fu}}$**rename-package** *package new-name* [*new-nicknames*$_{\boxed{\texttt{NIL}}}$])
▷ Rename *package*. Return $\underline{\text{renamed package}}$.

($^{\text{M}}$**in-package** $\widehat{\textit{foo}}$) ▷ Make $\underline{\text{package } \textit{foo}}$ current.

({$^{\text{Fu}}$**use-package** | $^{\text{Fu}}$**unuse-package**} *other-packages* [*package*$_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$])
▷ Make exported symbols of *other-packages* available in *package*, or remove them from *package*, respectively. Return $\underline{\texttt{T}}$.

($\overset{\text{Fu}}{\textbf{package-use-list}}$ *package*)
($\overset{\text{Fu}}{\textbf{package-used-by-list}}$ *package*)

▷ $\underline{\text{List of other packages}}$ used by/using *package*.

($\overset{\text{Fu}}{\textbf{delete-package}}$ $\widetilde{package}$)

▷ Delete *package*. Return $\underline{\texttt{T}}$ if successful.

$\overset{\text{var}}{\textbf{*package*}}_{\boxed{\text{common-lisp-user}}}$ ▷ The current package.

($\overset{\text{Fu}}{\textbf{list-all-packages}}$) ▷ $\underline{\text{List of registered packages}}$.

($\overset{\text{Fu}}{\textbf{package-name}}$ *package*) ▷ $\underline{\text{Name of } package}$.

($\overset{\text{Fu}}{\textbf{package-nicknames}}$ *package*) ▷ $\underline{\text{List of nicknames}}$ of *package*.

($\overset{\text{Fu}}{\textbf{find-package}}$ *name*) ▷ $\underline{\text{Package}}$ with *name* (case-sensitive).

($\overset{\text{Fu}}{\textbf{find-all-symbols}}$ *foo*)

▷ $\underline{\text{List of symbols}}$ *foo* from all registered packages.

($\begin{Bmatrix}\overset{\text{Fu}}{\textbf{intern}}\\ \overset{\text{Fu}}{\textbf{find-symbol}}\end{Bmatrix}$ *foo* [$package_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$])

▷ Intern or find, respectively, symbol $\underline{foo}$ in *package*. Second return value is one of $\underset{2}{\underline{\textbf{:internal}}}$, $\underset{2}{\underline{\textbf{:external}}}$, or $\underset{2}{\underline{\textbf{:inherited}}}$ (or $\underset{2}{\underline{\texttt{NIL}}}$ if $\overset{\text{Fu}}{\textbf{intern}}$ created a fresh symbol).

($\overset{\text{Fu}}{\textbf{unintern}}$ *symbol* [$package_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$])

▷ Remove *symbol* from *package*, return $\underline{\texttt{T}}$ on success.

($\begin{Bmatrix}\overset{\text{Fu}}{\textbf{import}}\\ \overset{\text{Fu}}{\textbf{shadowing-import}}\end{Bmatrix}$ *symbols* [$package_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$])

▷ 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.

($\overset{\text{Fu}}{\textbf{shadow}}$ *symbols* [$package_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$])

▷ Make *symbols* of *package* shadow any otherwise accessible, equally named symbols from other packages. Return $\underline{\texttt{T}}$.

($\overset{\text{Fu}}{\textbf{package-shadowing-symbols}}$ *package*)

▷ $\underline{\text{List of symbols}}$ of *package* that shadow any otherwise accessible, equally named symbols from other packages.

($\overset{\text{Fu}}{\textbf{export}}$ *symbols* [$package_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$])

▷ Make *symbols* external to *package*. Return $\underline{\texttt{T}}$.

($\overset{\text{Fu}}{\textbf{unexport}}$ *symbols* [$package_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$])

▷ Revert *symbols* to internal status. Return $\underline{\texttt{T}}$.

($\begin{Bmatrix}\begin{Bmatrix}\overset{\text{M}}{\textbf{do-symbols}}\\ \overset{\text{M}}{\textbf{do-external-symbols}}\end{Bmatrix} (\widehat{var}\ [package_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}\ [result_{\boxed{\texttt{NIL}}}]])\\ \overset{\text{M}}{\textbf{do-all-symbols}}\ (var\ [result_{\boxed{\texttt{NIL}}}])\end{Bmatrix}$

(**declare** $\widehat{decl}$*)* $\begin{Bmatrix}\vert\widehat{tag}\\ \vert form\end{Bmatrix}$\*)

▷ Evaluate $\overset{\text{sO}}{\textbf{tagbody}}$-like body with *var* successively bound to every symbol from *package*, to every external symbol from *package*, or to every symbol from all registered packages, respectively. Return $\underline{\text{values of } result}$. Implicitly, the whole form is a $\overset{\text{sO}}{\textbf{block}}$ named `NIL`.

($\overset{\text{M}}{\textbf{with-package-iterator}}$ (*foo packages* [**:internal**|**:external**|**:inherited**])

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

▷ Return $\underline{\text{values of } forms}$. In *forms*, successive invocations of (*foo*) return: `T` if a symbol is returned; a symbol from *packages*; accessibility (**:internal**, **:external**, or **:inherited**); and the package the symbol belongs to.

($\overset{\text{Fu}}{\textbf{require}}$ *module* [$paths_{\boxed{\texttt{NIL}}}$])

▷ If not in $\overset{\text{var}}{\textbf{*modules*}}$, try *paths* to load *module* from. Signal **error** if unsuccessful. Deprecated.

($\overset{\text{Fu}}{\textbf{provide}}$ *module*)

▷ If not already there, add *module* to $\overset{\text{var}}{\textbf{*modules*}}$. Deprecated.

$\overset{\text{var}}{\textbf{*modules*}}$ ▷ 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*).

($^{\text{Fu}}$**make-symbol** *name*)

▷ Make fresh, uninterned symbol *name*.

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

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

($^{\text{Fu}}$**gentemp** [$\textit{prefix}_{\boxed{\text{T}}}$ [$\textit{package}_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$]])

▷ Intern fresh symbol in package. Deprecated.

($^{\text{Fu}}$**copy-symbol** *symbol* [$\textit{props}_{\boxed{\texttt{NIL}}}$])

▷ Return uninterned copy of *symbol*. If *props* is `T`, give copy the same value, function and property list.

($^{\text{Fu}}$**symbol-name** *symbol*)
($^{\text{Fu}}$**symbol-package** *symbol*)
($^{\text{Fu}}$**symbol-plist** *symbol*)
($^{\text{Fu}}$**symbol-value** *symbol*)
($^{\text{Fu}}$**symbol-function** *symbol*)

▷ Name, package, property list, value, or function, respectively, of *symbol*. **setf**able.

$$(\left\{\begin{array}{l}\overset{\text{gF}}{\textbf{documentation}}\\ (\textbf{setf}\ \overset{\text{gF}}{\textbf{documentation}})\ \textit{new-doc}\end{array}\right\}\ \textit{foo}\ \left\{\begin{array}{l}\textbf{'variable}\,|\,\textbf{'function}\\ \textbf{'compiler-macro}\\ \textbf{'method-combination}\\ \textbf{'structure}\,|\,\textbf{'type}\,|\,\textbf{'setf}\,|\,\texttt{T}\end{array}\right\})$$

▷ Get/set documentation string of *foo* of given type.

$^{\text{co}}$**t**

▷ Truth; the supertype of every type including **t**; the superclass of every class except **t**; $^{\text{var}}$***terminal-io***.

$^{\text{co}}$**nil**|$^{\text{co}}$**()**

▷ Falsity; the empty list; the empty type, subtype of every type; $^{\text{var}}$***standard-input***; $^{\text{var}}$***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

($^{\text{Fu}}$**special-operator-p** *foo*) ▷ `T` if *foo* is a special operator.

($^{\text{Fu}}$**compiled-function-p** *foo*)

▷ `T` if *foo* is of type **compiled-function**.

15.2 Compilation

$$(\overset{\text{Fu}}{\textbf{compile}}\ \left\{\begin{array}{l}\texttt{NIL}\ \textit{definition}\\ \left\{\begin{array}{l}\textit{name}\\ (\textbf{setf}\ \textit{name})\end{array}\right\}\ [\textit{definition}]\end{array}\right\})$$

▷ Return compiled function or replace *name*'s function definition with the compiled function. Return $\underset{2}{\texttt{T}}$ in case of warnings or errors, and $\underset{3}{\texttt{T}}$ in case of warnings or errors excluding style warnings.

$(\overset{\text{Fu}}{\textbf{compile-file}}\ \textit{file}\ \left\{\begin{array}{|l}\textbf{:output-file}\ \textit{out-path}\\ \textbf{:verbose}\ \textit{bool}_{\boxed{\overset{\text{var}}{\textbf{*compile-verbose*}}}}\\ \textbf{:print}\ \textit{bool}_{\boxed{\overset{\text{var}}{\textbf{*compile-print*}}}}\\ \textbf{:external-format}\ \textit{file-format}_{\boxed{\textbf{:default}}}\end{array}\right\})$

▷ Write compiled contents of *file* to *out-path*. Return $\underline{\text{true output path}}$ or $\underline{\texttt{NIL}}$, $\underset{2}{\underline{\texttt{T}}}$ in case of warnings or errors, $\underset{3}{\underline{\texttt{T}}}$ in case of warnings or errors excluding style warnings.

$(\overset{\text{Fu}}{\textbf{compile-file-pathname}}\ \textit{file}\ [\textbf{:output-file}\ \textit{path}]\ [\textit{other-keyargs}])$

▷ $\underline{\text{Pathname}}$ $\overset{\text{Fu}}{\textbf{compile-file}}$ writes to if invoked with the same arguments.

$(\overset{\text{Fu}}{\textbf{load}}\ \textit{path}\ \left\{\begin{array}{|l}\textbf{:verbose}\ \textit{bool}_{\boxed{\overset{\text{var}}{\textbf{*load-verbose*}}}}\\ \textbf{:print}\ \textit{bool}_{\boxed{\overset{\text{var}}{\textbf{*load-print*}}}}\\ \textbf{:if-does-not-exist}\ \textit{bool}_{\boxed{\texttt{T}}}\\ \textbf{:external-format}\ \textit{file-format}_{\boxed{\textbf{:default}}}\end{array}\right\})$

▷ Load source file or compiled file into Lisp environment. Return $\underline{\texttt{T}}$ if successful.

$\left.\begin{array}{l}\overset{\text{var}}{\textbf{*compile-file}}\\ \overset{\text{var}}{\textbf{*load}}\end{array}\right\}\text{-}\left\{\begin{array}{l}\textbf{pathname*}_{\boxed{\texttt{NIL}}}\\ \textbf{truename*}_{\boxed{\texttt{NIL}}}\end{array}\right.$

▷ Input file used by $\overset{\text{Fu}}{\textbf{compile-file}}$/by $\overset{\text{Fu}}{\textbf{load}}$.

$\left.\begin{array}{l}\overset{\text{var}}{\textbf{*compile}}\\ \overset{\text{var}}{\textbf{*load}}\end{array}\right\}\text{-}\left\{\begin{array}{l}\textbf{print*}\\ \textbf{verbose*}\end{array}\right.$

▷ Defaults used by $\overset{\text{Fu}}{\textbf{compile-file}}$/by $\overset{\text{Fu}}{\textbf{load}}$.

$(\overset{\text{sO}}{\textbf{eval-when}}\ (\left\{\begin{array}{|l}\{\textbf{:compile-toplevel}|\textbf{compile}\}\\ \{\textbf{:load-toplevel}|\textbf{load}\}\\ \{\textbf{:execute}|\textbf{eval}\}\end{array}\right\})\ \textit{form}^{\text{P}}{*})$

▷ Return $\underline{\text{values of } \textit{forms}}$ if $\overset{\text{sO}}{\textbf{eval-when}}$ 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.)

$(\overset{\text{sO}}{\textbf{locally}}\ (\textbf{declare}\ \widehat{\textit{decl}*})*\ \textit{form}^{\text{P}}{*})$

▷ Evaluate *forms* in a lexical environment with declarations *decl* in effect. Return $\underline{\text{values of } \textit{forms}}$.

$(\overset{\text{M}}{\textbf{with-compilation-unit}}\ ([\textbf{:override}\ \textit{bool}_{\boxed{\texttt{NIL}}}])\ \textit{form}^{\text{P}}{*})$

▷ 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*.

$(\overset{\text{sO}}{\textbf{load-time-value}}\ \textit{form}\ [\widehat{\textit{read-only}_{\boxed{\texttt{NIL}}}}])$

▷ Evaluate *form* at compile time and treat $\underline{\text{its value}}$ as literal at run time.

$(\overset{\text{sO}}{\textbf{quote}}\ \widehat{\textit{foo}})$ ▷ Return $\underline{\text{unevaluated } \textit{foo}}$.

$(\overset{\text{gF}}{\textbf{make-load-form}}\ \textit{foo}\ [\textit{environment}])$

▷ Its methods are to return a $\underline{\text{creation form}}$ which on evaluation at $\overset{\text{Fu}}{\textbf{load}}$ 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.

$(\overset{\text{Fu}}{\textbf{make-load-form-saving-slots}}\ \textit{foo}\ \left\{\begin{array}{|l}\textbf{:slot-names}\ \textit{slots}_{\boxed{\text{all local slots}}}\\ \textbf{:environment}\ \textit{environment}\end{array}\right\})$

▷ Return a $\underline{\text{creation form}}$ and an $\underset{2}{\underline{\text{initialization form}}}$ which on evaluation construct an object equivalent to *foo* with *slots* initialized with the corresponding values from *foo*.

$(\overset{\text{Fu}}{\textbf{macro-function}}\ \textit{symbol}\ [\textit{environment}])$

$(\overset{\text{Fu}}{\textbf{compiler-macro-function}}\ \left\{\begin{array}{l}\textit{name}\\ (\textbf{setf}\ \textit{name})\end{array}\right\}\ [\textit{environment}])$

▷ Return specified $\underline{\text{macro function}}$, or $\underline{\text{compiler macro function}}$, respectively, if any. Return $\underline{\texttt{NIL}}$ otherwise. **setf**able.

$(\overset{\text{Fu}}{\textbf{eval}}\ \textit{arg})$

▷ Return $\underline{\text{values of value of } \textit{arg}}$ evaluated in global environment.

15.3 REPL and Debugging

$\overset{\text{var}}{+}|\overset{\text{var}}{++}|\overset{\text{var}}{+++}$
$\overset{\text{var}}{*}\,|\,\overset{\text{var}}{**}\,|\,\overset{\text{var}}{***}$
$\overset{\text{var}}{/}\,|\,\overset{\text{var}}{//}\,|\,\overset{\text{var}}{///}$

▷ 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.

$\overset{\text{var}}{-}$ ▷ $\underline{\text{Form}}$ currently being evaluated by the REPL.

($\overset{\text{Fu}}{\textbf{apropos}}$ *string* [*package*$_{\boxed{\texttt{NIL}}}$])

▷ Print interned symbols containing *string*.

($\overset{\text{Fu}}{\textbf{apropos-list}}$ *string* [*package*$_{\boxed{\texttt{NIL}}}$])

▷ $\underline{\text{List of interned symbols}}$ containing *string*.

($\overset{\text{Fu}}{\textbf{dribble}}$ [*path*])

▷ Save a record of interactive session to file at *path*. Without *path*, close that file.

($\overset{\text{Fu}}{\textbf{ed}}$ [*file-or-function*$_{\boxed{\texttt{NIL}}}$]) ▷ Invoke editor if possible.

($\left\{\begin{array}{l}\overset{\text{Fu}}{\textbf{macroexpand-1}}\\ \overset{\text{Fu}}{\textbf{macroexpand}}\end{array}\right\}$ *form* [*environment*$_{\boxed{\texttt{NIL}}}$])

▷ Return $\underline{\text{macro expansion}}$, once or entirely, respectively, of *form* and $\underline{\texttt{T}}_{2}$ if *form* was a macro form. Return $\underline{\textit{form}}$ and $\underline{\texttt{NIL}}_{2}$ otherwise.

$\overset{\text{var}}{\textbf{*macroexpand-hook*}}$

▷ Function of arguments expansion function, macro form, and environment called by $\overset{\text{Fu}}{\textbf{macroexpand-1}}$ to generate macro expansions.

($\overset{\text{M}}{\textbf{trace}}$ $\left\{\begin{array}{l}\textit{function}\\ (\textbf{setf}\ \textit{function})\end{array}\right\}^{*}$)

▷ Cause *function*s to be traced. With no arguments, return $\underline{\text{list of traced functions}}$.

($\overset{\text{M}}{\textbf{untrace}}$ $\left\{\begin{array}{l}\textit{function}\\ (\textbf{setf}\ \textit{function})\end{array}\right\}^{*}$)

▷ Stop *function*s, or each currently traced function, from being traced.

$\overset{\text{var}}{\textbf{*trace-output*}}$

▷ Stream $\overset{\text{M}}{\textbf{trace}}$ and $\overset{\text{M}}{\textbf{time}}$ print their output on.

($\overset{\text{M}}{\textbf{step}}$ *form*)

▷ Step through evaluation of *form*. Return $\underline{\text{values of }\textit{form}}$.

($\overset{\text{Fu}}{\textbf{break}}$ [*control arg**])

▷ Jump directly into debugger; return $\underline{\texttt{NIL}}$. See p. 36, $\overset{\text{Fu}}{\textbf{format}}$, for *control* and *arg*s.

($\overset{\text{M}}{\textbf{time}}$ *form*)

▷ Evaluate *form*s and print timing information to $\overset{\text{var}}{\textbf{*trace-output*}}$. Return $\underline{\text{values of }\textit{form}}$.

($\overset{\text{Fu}}{\textbf{inspect}}$ *foo*) ▷ Interactively give information about *foo*.

($\overset{\text{Fu}}{\textbf{describe}}$ *foo* [$\widetilde{\textit{stream}}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}$])

▷ Send information about *foo* to *stream*.

($\overset{\text{gF}}{\textbf{describe-object}}$ *foo* [$\widetilde{\textit{stream}}$])

▷ Send information about *foo* to *stream*. Not to be called by user.

($\overset{\text{Fu}}{\textbf{disassemble}}$ *function*)

▷ Send disassembled representation of *function* to $\overset{\text{var}}{\textbf{*standard-output*}}$. Return $\underline{\texttt{NIL}}$.

15.4 Declarations

($^{\text{Fu}}$**proclaim** *decl*)
($^{\text{M}}$**declaim** $\widehat{decl*}$)

▷ Globally make declaration(s) *decl*. *decl* can be: **declaration**, **type**, **ftype**, **inline**, **notinline**, **optimize**, or **special**. See below.

(**declare** $\widehat{decl*}$)

▷ Inside certain forms, locally make declarations *decl**. *decl* can be: **dynamic-extent**, **type**, **ftype**, **ignorable**, **ignore**, **inline**, **notinline**, **optimize**, or **special**. See below.

(**declaration** foo*)

▷ Make *foo*s names of declarations.

(**dynamic-extent** *variable** ($^{\text{sO}}$**function** *function*)*)

▷ Declare lifetime of *variable*s and/or *function*s to end when control leaves enclosing block.

([**type**] *type variable**)
(**ftype** *type function**)

▷ Declare *variable*s or *function*s to be of *type*.

$$(\begin{Bmatrix}\textbf{ignorable}\\\textbf{ignore}\end{Bmatrix}\begin{Bmatrix}var\\(\overset{\text{sO}}{\textbf{function}}\ function)\end{Bmatrix}^{*})$$

▷ Suppress warnings about used/unused bindings.

(**inline** *function**)
(**notinline** *function**)

▷ Tell compiler to integrate/not to integrate, respectively, called *function*s into the calling routine.

$$(\textbf{optimize}\begin{Bmatrix}\textbf{compilation-speed}\,|\,(\textbf{compilation-speed}\ n_{\boxed{3}})\\\textbf{debug}\,|\,(\textbf{debug}\ n_{\boxed{3}})\\\textbf{safety}\,|\,(\textbf{safety}\ n_{\boxed{3}})\\\textbf{space}\,|\,(\textbf{space}\ n_{\boxed{3}})\\\textbf{speed}\,|\,(\textbf{speed}\ n_{\boxed{3}})\end{Bmatrix})$$

▷ Tell compiler how to optimize. $n = 0$ means unimportant, $n = 1$ is neutral, $n = 3$ means important.

(**special** *var**) ▷ Declare *var*s to be dynamic.

16 External Environment

($^{\text{Fu}}$**get-internal-real-time**)
($^{\text{Fu}}$**get-internal-run-time**)

▷ $\underline{\text{Current time}}$, or $\underline{\text{computing time}}$, respectively, in clock ticks.

$^{\text{co}}$**internal-time-units-per-second**

▷ Number of clock ticks per second.

($^{\text{Fu}}$**encode-universal-time** *sec min hour date month year* [$zone_{\boxed{\text{curr}}}$])
($^{\text{Fu}}$**get-universal-time**)

▷ $\underline{\text{Seconds from 1900-01-01, 00:00}}$, ignoring leap seconds.

($^{\text{Fu}}$**decode-universal-time** *universal-time* [$time\text{-}zone_{\boxed{\text{current}}}$])
($^{\text{Fu}}$**get-decoded-time**)

▷ Return $\underline{\text{second}}$, $\underset{2}{\underline{\text{minute}}}$, $\underset{3}{\underline{\text{hour}}}$, $\underset{4}{\underline{\text{date}}}$, $\underset{5}{\underline{\text{month}}}$, $\underset{6}{\underline{\text{year}}}$, $\underset{7}{\underline{\text{day}}}$, $\underset{8}{\underline{\text{daylight-p}}}$, and $\underset{9}{\underline{\text{zone}}}$.

($^{\text{Fu}}$**room** [{`NIL`|**:default**|`T`}])

▷ Print information about internal storage management.

($^{\text{Fu}}$**short-site-name**)
($^{\text{Fu}}$**long-site-name**)

▷ $\underline{\text{String}}$ representing physical location of computer.

$$(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{lisp-implementation}}\\\overset{\text{Fu}}{\textbf{software}}\\\overset{\text{Fu}}{\textbf{machine}}\end{Bmatrix}\text{-}\begin{Bmatrix}\textbf{type}\\\textbf{version}\end{Bmatrix})$$

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

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

Index

Common Lisp Quick Reference Revision 124 [2011-04-09]

LaTeX source: http://clqr.berlios.de