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

($\overset{\text{Fu}}{\textbf{cis}}$ *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.

($\left\{\begin{array}{l}\{\overset{\text{Fu}}{\textbf{round}} \mid \overset{\text{Fu}}{\textbf{fround}}\} \\ \{\overset{\text{Fu}}{\textbf{floor}} \mid \overset{\text{Fu}}{\textbf{ffloor}}\} \\ \{\overset{\text{Fu}}{\textbf{ceiling}} \mid \overset{\text{Fu}}{\textbf{fceiling}}\} \\ \{\overset{\text{Fu}}{\textbf{truncate}} \mid \overset{\text{Fu}}{\textbf{ftruncate}}\}\end{array}\right\}$ *n* [$d_{\boxed{1}}$])

▷ Return as **integer** or **float**, respectively, *n/d* rounded, or rounded towards −∞, +∞, or 0, respectively; and $\underset{2}{\text{remainder}}$.

($\left\{\begin{array}{l}\overset{\text{Fu}}{\textbf{mod}} \\ \overset{\text{Fu}}{\textbf{rem}}\end{array}\right\}$ *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.

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

▷ 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* [$num\text{-}b_{\boxed{1}}$]) ▷ *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*.
$\overset{\text{co}}{\textbf{boole-c1}}$	▷ ¬*int-a*.
$\overset{\text{co}}{\textbf{boole-c2}}$	▷ ¬*int-b*.
$\overset{\text{co}}{\textbf{boole-set}}$	▷ All bits set.
$\overset{\text{co}}{\textbf{boole-clr}}$	▷ All bits zero.

Quick Reference

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.

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

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

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

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

$foo\overset{\text{P}}{*}$ ▷ 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

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

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

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

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

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

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

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

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

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

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

1.2 Numeric Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(**char** *string i*)
(**schar** *string i*)
▷ Return zero-indexed *i*th character of string ignoring/obeying, respectively, fill pointer. **setf**able.

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

4 Conses

4.1 Predicates

(**consp** *foo*)
(**listp** *foo*) ▷ Return T if *foo* is of indicated type.

(**endp** *list*)
(**null** *foo*) ▷ Return T if *list*/*foo* is NIL.

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

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

(**member** *foo list* {|{**:test** *function*$\boxed{\texttt{\#'eql}}$ | **: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** | **member-if-not**} *test list* [**:key** *function*])
▷ Return tail of *list* starting with its first element satisfying *test*. Return NIL if there is no such element.

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

4.2 Lists

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

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

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

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

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

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

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

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

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

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

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

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

boole-eqv ▷ *int-a* ≡ *int-b*.
boole-and ▷ *int-a* ∧ *int-b*.
boole-andc1 ▷ ¬*int-a* ∧ *int-b*.
boole-andc2 ▷ *int-a* ∧ ¬*int-b*.
boole-nand ▷ ¬(*int-a* ∧ *int-b*).
boole-ior ▷ *int-a* ∨ *int-b*.
boole-orc1 ▷ ¬*int-a* ∨ *int-b*.
boole-orc2 ▷ *int-a* ∨ ¬*int-b*.
boole-xor ▷ ¬(*int-a* ≡ *int-b*).
boole-nor ▷ ¬(*int-a* ∨ *int-b*).

(**lognot** *integer*) ▷ ¬*integer*.

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

(**logandc1** *int-a int-b*) ▷ ¬*int-a* ∧ *int-b*.

(**logandc2** *int-a int-b*) ▷ *int-a* ∧ ¬*int-b*.

(**lognand** *int-a int-b*) ▷ ¬(*int-a* ∧ *int-b*).

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

(**logorc1** *int-a int-b*) ▷ ¬*int-a* ∨ *int-b*.

(**logorc2** *int-a int-b*) ▷ *int-a* ∨ ¬*int-b*.

(**lognor** *int-a int-b*) ▷ ¬(*int-a* ∨ *int-b*).

(**logbitp** *i integer*)
▷ T if zero-indexed *i*th bit of *integer* is set.

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

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

1.4 Integer Functions

(**integer-length** *integer*)
▷ Number of bits necessary to represent *integer*.

(**ldb-test** *byte-spec integer*)
▷ Return T if any bit specified by *byte-spec* in *integer* is set.

(**ash** *integer count*)
▷ Return copy of *integer* arithmetically shifted left by *count* adding zeros at the right, or, for *count* < 0, shifted right discarding bits.

(**ldb** *byte-spec integer*)
▷ Extract byte denoted by *byte-spec* from *integer*. **setf**able.

({**deposit-field** | **dpb**} *int-a byte-spec int-b*)
▷ Return *int-b* with bits denoted by *byte-spec* replaced by corresponding bits of *int-a*, or by the low (**byte-size** *byte-spec*) bits of *int-a*, respectively.

(**mask-field** *byte-spec integer*)
▷ Return copy of *integer* with all bits unset but those denoted by *byte-spec*. **setf**able.

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

(**byte-size** *byte-spec*)
(**byte-position** *byte-spec*)
▷ Size or 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 significand, exponent$_2$, and sign$_3$ 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*)

▷ Radix, number of digits in that radix, or precision in that radix, respectively, of float *n*.

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

▷ 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*)
▷ T if argument is of indicated type.

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

▷ 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 T if *character* is uppercase, lowercase, or able to be in another case, respectively.

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

▷ Return its weight if *character* is a digit, or NIL otherwise.

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

▷ Return T if all *character*s, or none, respectively, are equal.

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

▷ Return 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 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.

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

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

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

▷ 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.

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

▷ Return string of length *size*.

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

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

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

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

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

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

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

6 Sequences

6.1 Sequence Predicates

({**every**$^{\text{Fu}}$ | **notevery**$^{\text{Fu}}$} *test sequence*$^+$)
▷ Return `NIL` or `T`, respectively, as soon as *test* on any set of corresponding elements of *sequence*s returns `NIL`.

({**some**$^{\text{Fu}}$ | **notany**$^{\text{Fu}}$} *test sequence*$^+$)
▷ Return value of *test* or `NIL`, respectively, as soon as *test* on any set of corresponding elements of *sequence*s returns non-`NIL`.

(**mismatch**$^{\text{Fu}}$ *sequence-a sequence-b* {| **:from-end** *bool*$_{\boxed{\text{NIL}}}$ | {**:test** *function*$_{\boxed{\text{\#'eql}}}$ | **:test-not** *function*} | **:start1** *start-a*$_{\boxed{0}}$ | **:start2** *start-b*$_{\boxed{0}}$ | **:end1** *end-a*$_{\boxed{\text{NIL}}}$ | **:end2** *end-b*$_{\boxed{\text{NIL}}}$ | **:key** *function*})
▷ Return position in *sequence-a* where *sequence-a* and *sequence-b* begin to mismatch. Return `NIL` if they match entirely.

6.2 Sequence Functions

(**make-sequence**$^{\text{Fu}}$ *sequence-type size* [**:initial-element** *foo*])
▷ Make sequence of *sequence-type* with *size* elements.

(**concatenate**$^{\text{Fu}}$ *type sequence**)
▷ Return concatenated sequence of *type*.

(**merge**$^{\text{Fu}}$ *type sequence-a sequence-b test* [**:key** *function*$_{\boxed{\text{NIL}}}$])
▷ Return interleaved sequence of *type*. Merged sequence will be sorted if both *sequence-a* and *sequence-b* are sorted.

(**fill**$^{\text{Fu}}$ *sequence foo* {| **:start** *start*$_{\boxed{0}}$ | **:end** *end*$_{\boxed{\text{NIL}}}$})
▷ Return *sequence* after setting elements between *start* and *end* to *foo*.

(**length**$^{\text{Fu}}$ *sequence*)
▷ Return length of *sequence* (being value of fill pointer if applicable).

(**count**$^{\text{Fu}}$ *foo sequence* {| **:from-end** *bool*$_{\boxed{\text{NIL}}}$ | {**:test** *function*$_{\boxed{\text{\#'eql}}}$ | **:test-not** *function*} | **:start** *start*$_{\boxed{0}}$ | **:end** *end*$_{\boxed{\text{NIL}}}$ | **:key** *function*})
▷ Return number of elements in *sequence* which match *foo*.

({**count-if**$^{\text{Fu}}$ | **count-if-not**$^{\text{Fu}}$} *test sequence* {| **:from-end** *bool*$_{\boxed{\text{NIL}}}$ | **:start** *start*$_{\boxed{0}}$ | **:end** *end*$_{\boxed{\text{NIL}}}$ | **:key** *function*})
▷ Return number of elements in *sequence* which satisfy *test*.

(**elt**$^{\text{Fu}}$ *sequence index*)
▷ Return element of *sequence* pointed to by zero-indexed *index*. **setf**able.

(**subseq**$^{\text{Fu}}$ *sequence start* [*end*$_{\boxed{\text{NIL}}}$])
▷ Return subsequence of *sequence* between *start* and *end*. **setf**able.

({**sort**$^{\text{Fu}}$ | **stable-sort**$^{\text{Fu}}$} *sequence test* [**:key** *function*])
▷ Return *sequence* sorted. Order of elements considered equal is not guaranteed/retained, respectively.

(**reverse**$^{\text{Fu}}$ *sequence*)
(**nreverse**$^{\text{Fu}}$ *sequence*)
▷ Return *sequence* in reverse order.

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

({**rplaca**$^{\text{Fu}}$ | **rplacd**$^{\text{Fu}}$} *cons object*)
▷ Replace car, or cdr, respectively, of *cons* with *object*.

(**ldiff**$^{\text{Fu}}$ *list foo*)
▷ If *foo* is a tail of *list*, return preceding part of *list*. Otherwise return *list*.

(**adjoin**$^{\text{Fu}}$ *foo list* {| {**:test** *function*$_{\boxed{\text{\#'eql}}}$ | **:test-not** *function*} | **:key** *function*})
▷ Return *list* if *foo* is already member of *list*. If not, return (**cons**$^{\text{Fu}}$ *foo list*).

(**pop**$^{\text{M}}$ *place*)
▷ Set *place* to (**cdr**$^{\text{Fu}}$ *place*), return (**car**$^{\text{Fu}}$ *place*).

(**push**$^{\text{M}}$ *foo place*)
▷ Set *place* to (**cons**$^{\text{Fu}}$ *foo place*).

(**pushnew**$^{\text{M}}$ *foo place* {| {**:test** *function*$_{\boxed{\text{\#'eql}}}$ | **:test-not** *function*} | **:key** *function*})
▷ Set *place* to (**adjoin**$^{\text{Fu}}$ *foo place*).

(**append**$^{\text{Fu}}$ [*proper-list** *foo*$_{\boxed{\text{NIL}}}$])
(**nconc**$^{\text{Fu}}$ [*non-circular-list** *foo*$_{\boxed{\text{NIL}}}$])
▷ Return concatenated list or, with only one argument, *foo*. *foo* can be of any type.

(**revappend**$^{\text{Fu}}$ *list foo*)
(**nreconc**$^{\text{Fu}}$ *list foo*)
▷ Return concatenated list after reversing order in *list*.

({**mapcar**$^{\text{Fu}}$ | **maplist**$^{\text{Fu}}$} *function list*$^+$)
▷ Return list of return values of *function* successively invoked with corresponding arguments, either cars or cdrs, respectively, from each *list*.

({**mapcan**$^{\text{Fu}}$ | **mapcon**$^{\text{Fu}}$} *function list*$^+$)
▷ Return list of concatenated return values of *function* successively invoked with corresponding arguments, either cars or cdrs, respectively, from each *list*. *function* should return a list.

({**mapc**$^{\text{Fu}}$ | **mapl**$^{\text{Fu}}$} *function list*$^+$)
▷ Return first *list* after successively applying *function* to corresponding arguments, either cars or cdrs, respectively, from each *list*. *function* should have some side effects.

(**copy-list**$^{\text{Fu}}$ *list*)
▷ Return copy of *list* with shared elements.

4.3 Association Lists

(**pairlis**$^{\text{Fu}}$ *keys values* [*alist*$_{\boxed{\text{NIL}}}$])
▷ Prepend to *alist* an association list made from lists *keys* and *values*.

(**acons**$^{\text{Fu}}$ *key value alist*)
▷ Return *alist* with a (*key* **.** *value*) pair added.

({**assoc**$^{\text{Fu}}$ | **rassoc**$^{\text{Fu}}$} *foo alist* {| {**:test** *test*$_{\boxed{\text{\#'eql}}}$ | **:test-not** *test*} | **:key** *function*})
({**assoc-if**$^{\text{Fu}}$[**-not**] | **rassoc-if**$^{\text{Fu}}$[**-not**]} *test alist* [**:key** *function*])
▷ First cons whose car, or cdr, respectively, satisfies *test*.

(**copy-alist**$^{\text{Fu}}$ *alist*)
▷ Return copy of *alist*.

4.4 Trees

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

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

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

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

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

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

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

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

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

4.5 Sets

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

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

5 Arrays

5.1 Predicates

(**arrayp**$^{\text{Fu}}$ *foo*)
(**vectorp**$^{\text{Fu}}$ *foo*)
(**simple-vector-p**$^{\text{Fu}}$ *foo*) ▷ T if *foo* is of indicated type.
(**bit-vector-p**$^{\text{Fu}}$ *foo*)
(**simple-bit-vector-p**$^{\text{Fu}}$ *foo*)

(**adjustable-array-p**$^{\text{Fu}}$ *array*)
(**array-has-fill-pointer-p**$^{\text{Fu}}$ *array*)

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

(**array-in-bounds-p**$^{\text{Fu}}$ *array* [*subscripts*])

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

5.2 Array Functions

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

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

(**aref**$^{\text{Fu}}$ *array* [*subscripts*])

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

(**row-major-aref**$^{\text{Fu}}$ *array i*)

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

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

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

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

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

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

▷ Length of *i*th dimension of *array*.

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

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

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

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

▷ Return element of *bit-array* or of *simple-bit-array*. **setf**able.

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

▷ Return result of bitwise negation of *bit-array*. If *result-bit-array* is T, put result in *bit-array*; if it is NIL, make a new array for result.

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

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

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

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

▷ Upper bound of an array dimension; ≥ 1024.

array-total-size-limit$^{\text{co}}$ ▷ Upper bound of array size; ≥ 1024.

5.3 Vector Functions

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

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

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

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

▷ Return NIL if *vector*'s fill pointer equals size of *vector*. Otherwise replace element of *vector* pointed to by fill pointer with *foo*; then increment fill pointer.

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

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

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

▷ Return element of *vector* its fillpointer points to after decrementation.

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

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

9.2 Variables

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

▷ 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*.

$(\begin{Bmatrix}\overset{\text{M}}{\textbf{setf}}\\ \overset{\text{M}}{\textbf{psetf}}\end{Bmatrix}\ \{place\ form\}^*)$

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

$(\begin{Bmatrix}\overset{\text{sO}}{\textbf{setq}}\\ \overset{\text{M}}{\textbf{psetq}}\end{Bmatrix}\ \{symbol\ form\}^*)$

▷ 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.

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

▷ 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 $\underset{2}{\text{value}}$ of first entry from *property-list* matching a key from *keys*, and $\underset{3}{\text{tail of } property\text{-}list}$ starting with that key. Return NIL, $\underset{2}{\text{NIL}}$, and $\underset{3}{\text{NIL}}$ 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

▷ Make copy of *sequence* without duplicates.

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

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

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

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

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

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

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

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

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

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

($\overset{\text{Fu}}{\textbf{reduce}}$ *function sequence* $\left\{\begin{array}{|l}\textbf{:initial-value}\ foo_{\boxed{\texttt{NIL}}}\\ \textbf{:from-end}\ bool_{\boxed{\texttt{NIL}}}\\ \textbf{:start}\ start_{\boxed{0}}\\ \textbf{:end}\ end_{\boxed{\texttt{NIL}}}\\ \textbf{:key}\ function\end{array}\right\}$)

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

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

▷ Copy of *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.

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

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

▷ Make a hash table.

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

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

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

▷ Number of entries in *hash-table*.

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

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

($\overset{\text{Fu}}{\textbf{clrhash}}$ $\widetilde{\mathit{hash\text{-}table}}$) ▷ Empty *hash-table*.

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

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

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

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

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

▷ Test function used in *hash-table*.

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

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

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

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

8 Structures

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

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

▷ Define structure *foo* together with functions `MAKE-`*foo*, `COPY-`*foo* and *foo*`-P`; and **setf**able accessors *foo*`-`*slot*. Instances are of class *foo* or, if **defstruct** option **:type** is given, of the specified type. They can be created by (`MAKE-`*foo* {:*slot value*}*) or, if *ord-λ* (see 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 copy of *structure* with shared slot values.

9 Control Structure

9.1 Predicates

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

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

▷ `T` 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*)

▷ `T` 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*)

▷ `T` if *foo* and *bar* are identical; or are the same **character** ignoring case; or are **number**s of the same value ignoring type; or are equivalent **pathname**s; or are **cons**es or **array**s of the same shape with $\overset{\text{Fu}}{\textbf{equalp}}$ 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*) ▷ `T` if *foo* is `NIL`; `NIL` otherwise.

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

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

▷ `T` if *foo* is a constant form.

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

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

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

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

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

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

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

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

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

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

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

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

▷ 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.

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

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

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

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

$(\overset{\text{sO}}{\textbf{return-from}}\ \textit{foo}\ [\textit{result}_{\boxed{\texttt{NIL}}}])$
$(\overset{\text{M}}{\textbf{return}}\ [\textit{result}_{\boxed{\texttt{NIL}}}])$

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

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

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

$(\overset{\text{sO}}{\textbf{go}}\ \widehat{\textit{tag}})$

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

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

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

$(\overset{\text{sO}}{\textbf{throw}}\ \textit{tag}\ \textit{form})$

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

$(\overset{\text{Fu}}{\textbf{sleep}}\ n)$ ▷ Wait n seconds, return `NIL`.

9.6 Iteration

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

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

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

▷ Define a function named *foo* or (**setf** *foo*), or an anonymous 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*.

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

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

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

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

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

▷ Values of *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 $\overset{\text{Fu}}{\textbf{sbit}}$.

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

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

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

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

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

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

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

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

▷ Zero-indexed nth return value of *form*.

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

▷ Return new function with same arguments and same side effects as *function*, but with complementary truth value.

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

▷ Function of any number of arguments returning *foo*.

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

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

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

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

▷ Definition of global function *foo*. **setf**able.

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

▷ Remove global function or macro definition *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-λ**) has the form of either

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

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

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

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

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

or

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

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

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.

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

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

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

▷ 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.

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

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

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

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

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

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

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

▷ Specify how to **setf** a place accessed by $\underline{function}$. **Short form:** (**setf** (*function arg**) *value-form*) is replaced by (*updater arg* value-form*); the latter must return *value-form*. **Long form:** on invocation of (**setf** (*function arg**) *value-form*), *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*.

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

▷ 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*.

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

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

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

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

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

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

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

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

{**&rest**|**&body**} *var*
▷ Bind *var* to a list of remaining arguments.

&key *var**
▷ Bind *var*s to corresponding keyword arguments.

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

&environment *var*
▷ Bind *var* to the lexical compilation environment.

&aux *var** ▷ Bind *var*s as in $\overset{\text{sO}}{\textbf{let*}}$.

9.5 Control Flow

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

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

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

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

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

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

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

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

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

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

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

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

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

▷ 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`.

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

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

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

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

▷ 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 value i*) or (**setf** (*accessor i*) *value*). With **:allocation :class**, *slot* is shared by all instances of class *foo*.

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

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

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

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

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

$$\left(\left\{\begin{array}{l}\overset{\text{M}}{\textsf{with-slots}}\ (\{\widehat{slot}|(\widehat{var}\ \widehat{slot})\}^*)\\ \overset{\text{M}}{\textsf{with-accessors}}\ ((\widehat{var}\ \widehat{accessor})^*)\end{array}\right\}\ \mathit{instance}\ (\textsf{declare}\ \widehat{decl^*})^*\ \mathit{form}^{P*})$$

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

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

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

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

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

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

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

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

$$(\overset{\text{gF}}{\textsf{shared-initialize}}\ \mathit{instance}\ \left\{\begin{array}{l}\mathit{slots}\\ \texttt{T}\end{array}\right\}\ \{\textsf{:}\mathit{initarg}\ \mathit{value}\}^*\ \mathit{other\text{-}keyarg}^*)$$

▷ Fill *instance*'s *slots* using *initarg*s and **:initform** forms.

$$(\overset{\text{gF}}{\textsf{slot-missing}}\ \mathit{class}\ \mathit{object}\ \mathit{slot}\ \left\{\begin{array}{l}\textsf{setf}\\ \textsf{slot-boundp}\\ \textsf{slot-makunbound}\\ \textsf{slot-value}\end{array}\right\}\ [\mathit{value}])$$

▷ Called in case of attempted access to missing *slot*. Its primary method signals **error**.

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

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

9.7 Loop Facility

(**loop**$^{\text{M}}$ *form**)
▷ **Simple Loop.** If *form*s do not contain any atomic Loop Facility keywords, evaluate them forever in an implicit **block**$^{\text{sO}}$ named `NIL`.

(**loop**$^{\text{M}}$ *clause**)
▷ **Loop Facility.** For Loop Facility keywords see below and Figure 1.

named *n*$_{\boxed{\texttt{NIL}}}$ ▷ Give **loop**$^{\text{M}}$'s implicit **block**$^{\text{sO}}$ a name.

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

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

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

$$\left\{\textsf{fixnum}\middle|\textsf{float}\middle|\texttt{T}\middle|\texttt{NIL}\middle|\left\{\textsf{of-type}\left\{\begin{array}{l}\mathit{type}\\(\mathit{type}^*)\end{array}\right\}\right\}\right\}$$

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

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

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

{**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*$_{\boxed{\textsf{\#'cdr}}}$}
▷ 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*$_{\boxed{\overset{\text{var}}{\textsf{*package*}}}}$]
▷ Bind *var* successively to the accessible symbols, or the present symbols, or the external symbols respectively, of *package*.

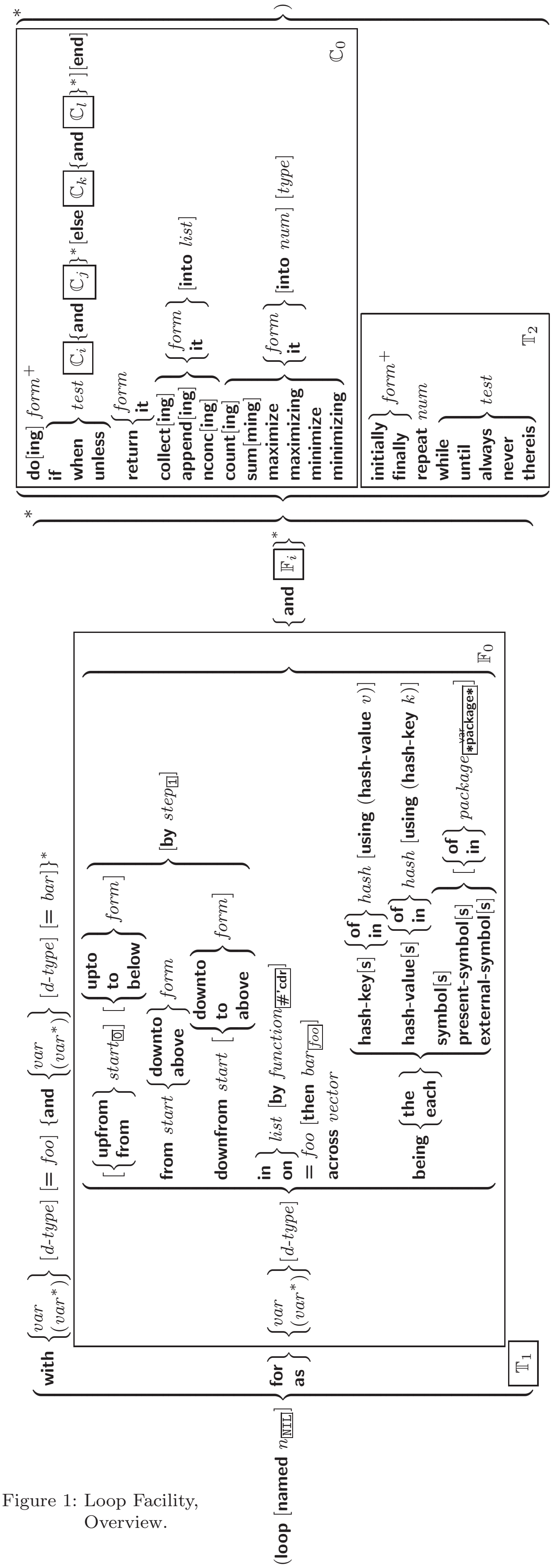
(loop [named n NIL]
with {var | (var*)} [d-type] [= foo] {and {var | (var*)} [d-type] [= bar]}*
for | as {var | (var*)} [d-type]
{and F_i}*
[{upfrom | from} start 0] [{upto | to | below} form]
from start {downto | above} form
downfrom start [{downto | to | above} form]
[by step 1]
in | on list [by function #'cdr]
= foo [then bar foo]
across vector
being {the | each}
hash-key[s] {of | in} hash [using (hash-value v)]
hash-value[s] {of | in} hash [using (hash-key k)]
symbol[s] | present-symbol[s] | external-symbol[s] [{of | in} package *package*]
F_0
T_1
do[ing] form+
if | when | unless test C_i {and C_j}* [else C_k {and C_l}*] [end]
return {form | it}
collect[ing] | append[ing] | nconc[ing] {form | it} [into list]
count[ing] | sum[ming] | maximize | maximizing | minimize | minimizing {form | it} [into num] [type]
C_0
initially | finally form+
repeat num
while | until | always | never | thereis test
T_2

Figure 1: Loop Facility, Overview.

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

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

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

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

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

{**append**|**appending**|**nconc**|**nconcing**} {*form*|**it**} [**into** *list*]
▷ Concatenate values of *form* or **it**, which should be lists, into *list* by the means of $\overset{\text{Fu}}{\textbf{append}}$ 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 *forms* 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*) ▷ T if *foo* has a slot *bar*.

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

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

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

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

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

▷ If, on evaluation of *foo*, a condition of *type* is signalled, evaluate matching *condition-forms* with *var* bound to the condition, and return <u>their values</u>. Without a condition, bind *ord-λ*s to values of *foo* and return <u>values of *forms*</u> or, without a **:no-error** clause, return <u>values of *foo*</u>. See p. 16 for (*ord-λ**).

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

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

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

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

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

▷ Evaluate *form* with dynamically established restarts *foo*. Return <u>values of *form*</u> 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 <u>values of its *restart-forms*</u>. *arg-function* supplies appropriate *args* 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.

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

▷ Return <u>values of *forms*</u> evaluated with *restarts* dynamically bound to *restart-functions*.

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

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

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

▷ Return list of <u>all restarts</u>, or innermost <u>restart *name*</u>, 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.

($\overset{\text{Fu}}{\textbf{restart-name}}$ *restart*) ▷ <u>Name of *restart*</u>.

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

▷ 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.

($\overset{\text{gF}}{\textbf{slot-unbound}}$ *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.

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

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

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

▷ Define or modify <u>generic function *foo*</u>. **: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.

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

▷ Define <u>new method</u> for generic function *foo*. *spec-vars* specialize to either being of *class* or being **eql** *bar*, respectively. On invocation, *vars* and *spec-vars* of the <u>new method</u> act like parameters of a function with body *form**. *forms* are enclosed in an implicit $\overset{\text{sO}}{\textbf{block}}$ *foo*. Applicable *qualifiers* 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 <u>*generic-function*</u>.

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

▷ Return suitable <u>method</u>, or signal **error**.

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

▷ <u>List of methods</u> suitable for *args*, most specific first.

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

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

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

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

({ $^{\text{Fu}}$**invalid-method-error** *method* | $^{\text{Fu}}$**method-combination-error** } *control arg**)

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

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

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

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

▷ Return list of keyword parameters of *method* and $\underline{\texttt{T}}_2$ if other keys are allowed.

($^{\text{gF}}$**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, $^{\text{Fu}}$**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 $^{\text{Fu}}$**call-next-method** if any, or of the generic function; and which can call less specific primary methods via $^{\text{Fu}}$**call-next-method**. After its return, call all **:after** methods, least specific first.

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

▷ Simple built-in **method-combination** types; have the same usage as the *c-type*s defined by the short form of $^{\text{M}}$**define-method-combination**.

($^{\text{M}}$**define-method-combination** *c-type*
{ | **:documentation** $\widehat{string}$
| **:identity-with-one-argument** *bool*$_{\boxed{\text{NIL}}}$
| **:operator** *operator*$_{\boxed{c\text{-type}}}$ })

▷ **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, $^{\text{Fu}}$**call-next-method** can call less specific **:around** methods if there are any. If not, or if there are no **:around** methods at all, return from the calling **call-next-method** or from the generic function, respectively, the values of (*operator* (*primary-method gen-arg**)*), *gen-arg** being the arguments of the generic function. The *primary-method*s are ordered [{ **:most-specific-first** | **:most-specific-last** }$_{\boxed{\text{:most-specific-first}}}$] (specified as *c-arg* in $^{\text{M}}$**defgeneric**). Using *c-type* as the *qualifier* in $^{\text{M}}$**defmethod** makes the method primary.

($^{\text{M}}$**define-method-combination** *c-type* (*ord-λ**) ((*group*
{ ***** | (*qualifier** [*****]) | *predicate* }
{ | **:description** *control*
| **:order** { **:most-specific-first** | **:most-specific-last** }$_{\boxed{\text{:most-specific-first}}}$
| **:required** *bool* })*)
{ | (**:arguments** *method-combination-λ**)
| (**:generic-function** *symbol*)
| (**declare** $\widehat{decl^*}$)*
| $\widehat{doc}$ } *body*^{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. $^{\text{M}}$**defgeneric**), with *symbol* bound to the generic function, with *method-combination-λ** bound to the arguments of the generic function, and with *group*s bound to lists of methods. An applicable method becomes a member of the leftmost *group* whose *predicate* or *qualifier*s match. Methods can be called via $^{\text{M}}$**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.

($^{\text{M}}$**call-method** { $\widehat{method}$ | ($^{\text{M}}$**make-method** $\widehat{form}$) } [({ $\widehat{next\text{-}method}$ | ($^{\text{M}}$**make-method** $\widehat{form}$) }*)])

▷ 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.

($^{\text{M}}$**define-condition** *foo* (*parent-type**$_{\boxed{\text{condition}}}$)
({ *slot*
| (*slot* { | {**:reader** *reader*}*
| {**:writer** { *writer* | (**setf** *writer*) }}*
| {**:accessor** *accessor*}*
| **:allocation** { **:instance** | **:class** }$_{\boxed{\text{:instance}}}$
| {**:initarg** **:***initarg-name*}*
| **:initform** *form*
| **:type** *type*
| **:documentation** *slot-doc* }) }*)
{ | (**:default-initargs** {*name value*}*)
| (**:documentation** *condition-doc*)
| (**:report** { *string* | *report-function* }) })

▷ 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 value i*) or (**setf** (*accessor i*) *value*). With **:allocation :class**, *slot* is shared by all conditions of type *foo*. A condition is reported by *string* or by *report-function* of arguments condition and stream.

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

▷ Return new condition of *type*.

({ $^{\text{Fu}}$**signal** | $^{\text{Fu}}$**warn** | $^{\text{Fu}}$**error** } { *condition* | *type* {**:***initarg-name value*}* | *control arg** })

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

($^{\text{Fu}}$**cerror** *continue-control* { *condition continue-arg** | *type* {**:***initarg-name value*}* | *control arg** })

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

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

▷ Return values of *forms* or, in case of **error**s, $\underline{\texttt{NIL}}$ and the $\underline{\text{condition}}_2$.

($^{\text{Fu}}$**invoke-debugger** *condition*)

▷ Invoke debugger with *condition*.

($^{\text{Fu}}$**read-delimited-list** *char* [$\widetilde{stream}_{\boxed{\text{*standard-input*}}}$ [*recursive*$_{\boxed{\text{NIL}}}$]])

▷ Continue reading until encountering *char*. Return list of objects read. Signal error if no *char* is found in stream.

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

▷ Return next character from *stream*.

($^{\text{Fu}}$**read-char-no-hang** [$\widetilde{stream}_{\boxed{\text{*standard-input*}}}$ [*eof-error*$_{\boxed{\text{T}}}$ [*eof-val*$_{\boxed{\text{NIL}}}$ [*recursive*$_{\boxed{\text{NIL}}}$]]]])

▷ Next character from *stream* or `NIL` if none is available.

($^{\text{Fu}}$**peek-char** [*mode*$_{\boxed{\text{NIL}}}$ [$\widetilde{stream}_{\boxed{\text{*standard-input*}}}$ [*eof-error*$_{\boxed{\text{T}}}$ [*eof-val*$_{\boxed{\text{NIL}}}$ [*recursive*$_{\boxed{\text{NIL}}}$]]]]])

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

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

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

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

▷ Read next byte from binary *stream*.

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

▷ Return a line of text from *stream* and $\underset{2}{\texttt{T}}$ if line has been ended by end of file.

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

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

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

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

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

▷ Return copy of *from-readtable*.

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

▷ Copy syntax of *from-char* to *to-readtable*. Return `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***$_{\boxed{\text{NIL}}}$

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

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

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

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

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

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

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

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

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

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

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

($^{\text{M}}$**with-condition-restarts** *condition restarts* *form*$^{\text{P}}{}^{*}$)

▷ 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***$_{\boxed{\text{NIL}}}$

▷ Condition type debugger is to be invoked on.

$^{\text{var}}$***debugger-hook***$_{\boxed{\text{NIL}}}$

▷ 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*$_{\boxed{\text{NIL}}}$]) ▷ `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*$^{\text{P}}{}^{*}$)* [($\left\{\begin{matrix}\textbf{otherwise}\\ \texttt{T}\end{matrix}\right\}$ *b-form*$_{\boxed{\text{NIL}}}^{\text{P}}{}^{*}$)])

▷ Return values of the *a-forms* whose *type* is *foo* of. Return values of *b-forms* if no *type* matches.

($\left\{\begin{matrix}^{\text{M}}\textbf{ctypecase}\\ ^{\text{M}}\textbf{etypecase}\end{matrix}\right\}$ *foo* ($\widehat{type}$ *form*$^{\text{P}}{}^{*}$)*)

▷ 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.

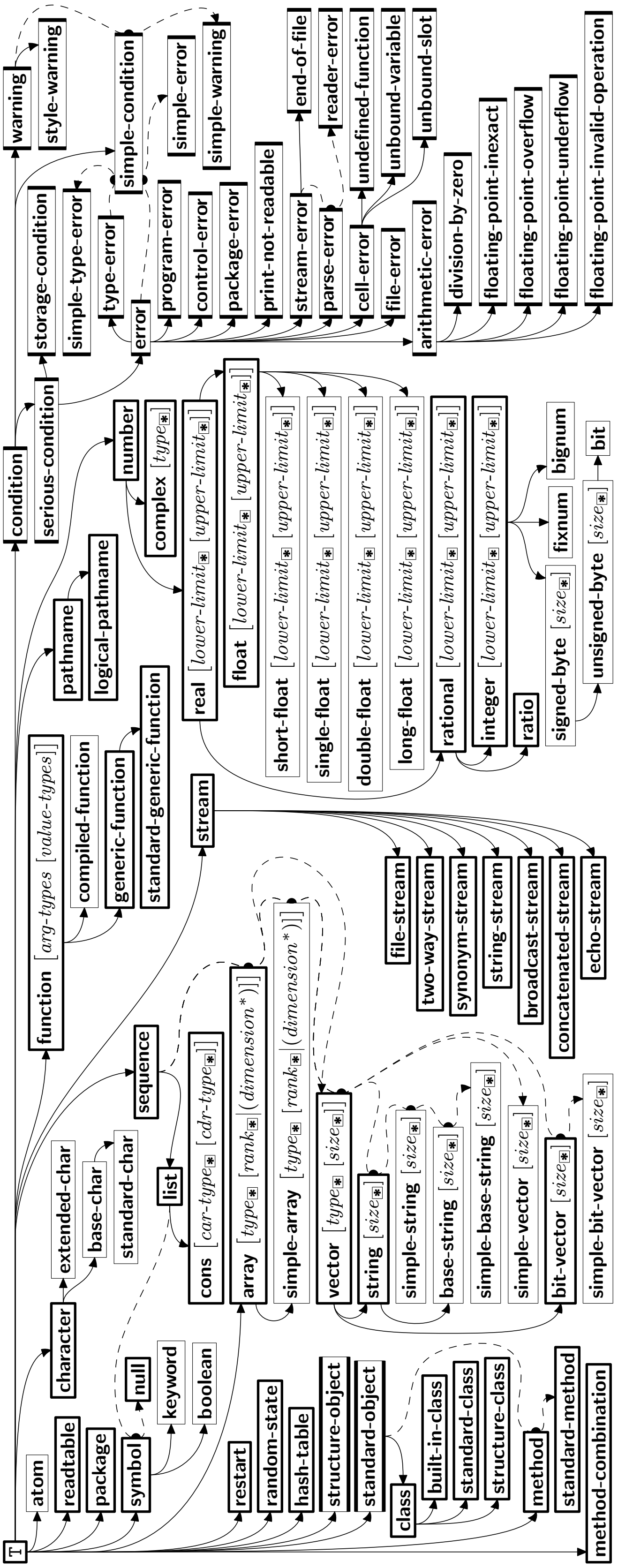

Figure 2: Precedence Order of System Classes (▭), Classes (▭), Types (▭), and Condition Types (▭).

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

($^{\text{M}}$**deftype** *foo* (*macro-λ**) (**declare** $\widehat{decl}$*)* [$\widehat{doc}$] *form*$^{\text{P}}$*)
▷ Define type *foo* 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 $^{\text{sO}}$**block** named *foo*.

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

(**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 *type*s.

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

(**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

($^{\text{Fu}}$**streamp** *foo*)
($^{\text{Fu}}$**pathnamep** *foo*) ▷ T if *foo* is of indicated type.
($^{\text{Fu}}$**readtablep** *foo*)

($^{\text{Fu}}$**input-stream-p** *stream*)
($^{\text{Fu}}$**output-stream-p** *stream*)
($^{\text{Fu}}$**interactive-stream-p** *stream*)
($^{\text{Fu}}$**open-stream-p** *stream*)
▷ Return T if *stream* is for input, for output, interactive, or open, respectively.

($^{\text{Fu}}$**pathname-match-p** *path wildcard*)
▷ T if *path* matches *wildcard*.

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

13.2 Reader

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

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

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

($^{\text{Fu}}$**read-from-string** *string* [*eof-error*$_{\boxed{\texttt{T}}}$ [*eof-val*$_{\boxed{\texttt{NIL}}}$ [{|**:start** *start*$_{\boxed{0}}$ | **:end** *end*$_{\boxed{\texttt{NIL}}}$ | **:preserve-whitespace** *bool*$_{\boxed{\texttt{NIL}}}$}]]])
▷ Return object read from string and zero-indexed position$_2$ of next character.

($\overset{\text{Fu}}{\textbf{set-pprint-dispatch}}$ *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.

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*] [,.$\widetilde{\textit{quux}}$] [*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** $\frac{n}{d}$.

$\left\{[m].n[\{\textbf{S}|\textbf{F}|\textbf{D}|\textbf{L}|\textbf{E}\}x_{\boxed{\textbf{E0}}}] \middle| m[.[n]]\{\textbf{S}|\textbf{F}|\textbf{D}|\textbf{L}|\textbf{E}\}x\right\}$

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

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

#'*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.

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

▷ 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*.

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

▷ 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*.

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

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

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

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

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

▷ 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 **~//**.

$(\overset{\text{M}}{\textbf{pprint-logical-block}}\ (\widetilde{\textit{stream}}\ \textit{list}\ \left\{\begin{vmatrix}\begin{Bmatrix}\textbf{:prefix}\ \textit{string}\\ \textbf{:per-line-prefix}\ \textit{string}\end{Bmatrix}\\ \textbf{:suffix}\ \textit{string}_{\boxed{\texttt{""}}}\end{vmatrix}\right\})$
$(\textbf{declare}\ \widehat{\textit{decl}^*})^*\ \textit{form}^{\text{P}*})$

▷ 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 $\overset{\text{Fu}}{\textbf{write}}$. Return $\underline{\texttt{NIL}}$.

$(\overset{\text{M}}{\textbf{pprint-pop}})$

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

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

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

$(\overset{\text{Fu}}{\textbf{pprint-indent}}\ \begin{Bmatrix}\textbf{:block}\\ \textbf{:current}\end{Bmatrix}\ n\ [\widetilde{\textit{stream}}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}])$

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

$(\overset{\text{M}}{\textbf{pprint-exit-if-list-exhausted}})$

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

$(\overset{\text{Fu}}{\textbf{pprint-newline}}\ \begin{Bmatrix}\textbf{:linear}\\ \textbf{:fill}\\ \textbf{:miser}\\ \textbf{:mandatory}\end{Bmatrix}\ [\widetilde{\textit{stream}}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}])$

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

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

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

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

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

$\overset{\text{var}}{\textbf{*print-circle*}}_{\boxed{\texttt{NIL}}}$

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

$\overset{\text{var}}{\textbf{*print-escape*}}_{\boxed{\texttt{T}}}$

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

$\overset{\text{var}}{\textbf{*print-gensym*}}_{\boxed{\texttt{T}}}$

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

$\overset{\text{var}}{\textbf{*print-length*}}_{\boxed{\texttt{NIL}}}$
$\overset{\text{var}}{\textbf{*print-level*}}_{\boxed{\texttt{NIL}}}$
$\overset{\text{var}}{\textbf{*print-lines*}}_{\boxed{\texttt{NIL}}}$

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

$\overset{\text{var}}{\textbf{*print-miser-width*}}$

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

$\overset{\text{var}}{\textbf{*print-pretty*}}$ ▷ If `T`, print pretty.

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

$\overset{\text{var}}{\textbf{*print-readably*}}_{\boxed{\texttt{NIL}}}$

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

$\overset{\text{var}}{\textbf{*print-right-margin*}}_{\boxed{\texttt{NIL}}}$

▷ Right margin width in ems while pretty-printing.

13.7 Pathnames and Files

$(\overset{\text{Fu}}{\textbf{make-pathname}}$

$$\left\{\left|\begin{array}{l}
\textbf{:host}\ \{host|\texttt{NIL}|\textbf{:unspecific}\}\\
\textbf{:device}\ \{device|\texttt{NIL}|\textbf{:unspecific}\}\\
\textbf{:directory}\ \left\{\begin{array}{l}\{directory|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\}\\ \left(\left\{\begin{array}{l}\textbf{:absolute}\\ \textbf{:relative}\end{array}\right\}\left\{\begin{array}{l}directory\\ \textbf{:wild}\\ \textbf{:wild-inferiors}\\ \textbf{:up}\\ \textbf{:back}\end{array}\right\}^{*}\right)\end{array}\right.\\
\textbf{:name}\ \{\textit{file-name}|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\}\\
\textbf{:type}\ \{\textit{file-type}|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\}\\
\textbf{:version}\ \{\textbf{:newest}|version|\textbf{:wild}|\texttt{NIL}|\textbf{:unspecific}\}\\
\textbf{:defaults}\ path_{\boxed{\text{host from }\overset{\text{var}}{\textbf{*default-pathname-defaults*}}}}\\
\textbf{:case}\ \{\textbf{:local}|\textbf{:common}\}_{\boxed{\textbf{:local}}}
\end{array}\right.\right\})$$

▷ 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.

$$\left(\left\{\begin{array}{l}\overset{\text{Fu}}{\textbf{pathname-host}}\\ \overset{\text{Fu}}{\textbf{pathname-device}}\\ \overset{\text{Fu}}{\textbf{pathname-directory}}\\ \overset{\text{Fu}}{\textbf{pathname-name}}\\ \overset{\text{Fu}}{\textbf{pathname-type}}\end{array}\right\}\ path\ \left[\textbf{:case}\ \left\{\begin{array}{l}\textbf{:local}\\ \textbf{:common}\end{array}\right\}_{\boxed{\textbf{:local}}}\right]\right)$$

$(\overset{\text{Fu}}{\textbf{pathname-version}}\ path)$

▷ Return pathname component.

$(\overset{\text{Fu}}{\textbf{parse-namestring}}\ foo\ \Big[host$

$\Big[\textit{default-pathname}_{\boxed{\overset{\text{var}}{\textbf{*default-pathname-defaults*}}}}$

$$\left\{\left|\begin{array}{l}\textbf{:start}\ start_{\boxed{0}}\\ \textbf{:end}\ end_{\boxed{\texttt{NIL}}}\\ \textbf{:junk-allowed}\ bool_{\boxed{\texttt{NIL}}}\end{array}\right.\right\}\Big]\Big])$$

▷ Return pathname converted from string, pathname, or stream *foo*; and $\underset{2}{\text{position}}$ where parsing stopped.

$(\overset{\text{Fu}}{\textbf{merge-pathnames}}\ pathname$

$\big[\textit{default-pathname}_{\boxed{\overset{\text{var}}{\textbf{*default-pathname-defaults*}}}}$

$[\textit{default-version}_{\boxed{\textbf{:newest}}}]\big])$

▷ Return *pathname* after filling in missing components from *default-pathname*.

$\overset{\text{var}}{\textbf{*default-pathname-defaults*}}$

▷ Pathname to use if one is needed and none supplied.

$(\overset{\text{Fu}}{\textbf{user-homedir-pathname}}\ [host])$ ▷ User's home directory.

$(\overset{\text{Fu}}{\textbf{enough-namestring}}\ path\ [\textit{root-path}_{\boxed{\overset{\text{var}}{\textbf{*default-pathname-defaults*}}}}])$

▷ Return minimal path string to sufficiently describe *path* relative to *root-path*.

$(\overset{\text{Fu}}{\textbf{namestring}}\ path)$
$(\overset{\text{Fu}}{\textbf{file-namestring}}\ path)$
$(\overset{\text{Fu}}{\textbf{directory-namestring}}\ path)$
$(\overset{\text{Fu}}{\textbf{host-namestring}}\ path)$

▷ Return string representing full pathname; name, type, and version; directory name; or host name, respectively, of *path*.

$(\overset{\text{Fu}}{\textbf{translate-pathname}}\ path\ \textit{wildcard-path-a}\ \textit{wildcard-path-b})$

▷ Translate *path* from *wildcard-path-a* into *wildcard-path-b*. Return new path.

$(\overset{\text{Fu}}{\textbf{pathname}}\ path)$ ▷ Pathname of *path*.

$(\overset{\text{Fu}}{\textbf{logical-pathname}}\ \textit{logical-path})$

▷ Logical pathname of *logical-path*. Logical pathnames are represented as all-uppercase $\texttt{\#P"}[host\texttt{:}][\texttt{:}]\left\{\left\{\begin{array}{l}\{dir|\texttt{*}\}^{+}\\ \texttt{**}\end{array}\right\}\texttt{;}\right\}^{*}$ $\{name|\texttt{*}\}^{*}\left[\texttt{.}\left\{\begin{array}{l}\{type|\texttt{*}\}^{+}\\ \texttt{LISP}\end{array}\right\}[\texttt{.}\{version|\texttt{*}|\texttt{newest}|\texttt{NEWEST}\}]\right]\texttt{"}$.

$(\overset{\text{Fu}}{\textbf{logical-pathname-translations}}\ \textit{logical-host})$

▷ List of (*from-wildcard to-wildcard*) translations for *logical-host*. **setf**able.

$\{\textbf{\textasciitilde(}\ text\ \textbf{\textasciitilde)}|\textbf{\textasciitilde:(}\ text\ \textbf{\textasciitilde)}|\textbf{\textasciitilde@(}\ text\ \textbf{\textasciitilde)}|\textbf{\textasciitilde:@(}\ text\ \textbf{\textasciitilde)}\}$

▷ **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.

$\{\textbf{\textasciitilde P}|\textbf{\textasciitilde:P}\ |\textbf{\textasciitilde@P}|\textbf{\textasciitilde:@P}\}$

▷ **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.

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

$\textbf{\textasciitilde}\ [n_{\boxed{1}}]\ \textbf{\&}$

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

$\{\textbf{\textasciitilde\_}|\textbf{\textasciitilde:\_}|\textbf{\textasciitilde@\_}|\textbf{\textasciitilde:@\_}\}$

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

$\{\textbf{\textasciitilde:}\hookleftarrow|\textbf{\textasciitilde@}\hookleftarrow|\textbf{\textasciitilde}\hookleftarrow\}$

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

$\textbf{\textasciitilde}\ [n_{\boxed{1}}]\ \textbf{|}$ ▷ **Page.** Print *n* page separators.

$\textbf{\textasciitilde}\ [n_{\boxed{1}}]\ \textbf{\textasciitilde}$ ▷ **Tilde.** Print *n* tildes.

$\textbf{\textasciitilde}\ [\textit{min-col}_{\boxed{0}}]\ \big[\textbf{,}[\textit{col-inc}_{\boxed{1}}]\ \big[\textbf{,}[\textit{min-pad}_{\boxed{0}}]\ \big[\textbf{,}\textit{pad-char}_{\boxed{\texttt{'\textvisiblespace}}}\big]\big]\big]$
$[\textbf{:}]\ [\textbf{@}]\ \textbf{<}\ \big[\textit{nl-text}\ \textbf{\textasciitilde}[\textit{spare}_{\boxed{0}}\ [\textbf{,}\textit{width}]]\textbf{:;}\big]\ \{text\ \textbf{\textasciitilde;}\}^{*}\ text$ $\textbf{\textasciitilde>}$

▷ **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.

$\textbf{\textasciitilde}\ [\textbf{:}]\ [\textbf{@}]\ \textbf{<}\ \big\{[\textit{prefix}_{\boxed{\texttt{""}}}\ \textbf{\textasciitilde;}]\big|[\textit{per-line-prefix}\ \textbf{\textasciitilde@;}]\big\}\ body\ \big[\textbf{\textasciitilde;}$ $\textit{suffix}_{\boxed{\texttt{""}}}\big]\ \textbf{\textasciitilde:}\ [\textbf{@}]\ \textbf{>}$

▷ **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.

$\{\textbf{\textasciitilde}\ [n_{\boxed{0}}]\ \textbf{i}|\textbf{\textasciitilde}\ [n_{\boxed{0}}]\ \textbf{:i}\}$

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

$\textbf{\textasciitilde}\ [c_{\boxed{1}}]\ [\textbf{,}i_{\boxed{1}}]\ [\textbf{:}]\ [\textbf{@}]\ \textbf{T}$

▷ **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.

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

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

$\textbf{\textasciitilde}\ [limit]\ [\textbf{:}]\ [\textbf{@}]\ \textbf{\{}\ text\ \textbf{\textasciitilde\}}$

▷ **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.

$\textbf{\textasciitilde}\ \big[x\ \big[\textbf{,}y\ [\textbf{,}z]\big]\big]\ \textbf{\textasciicircum}$

▷ **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.

$\textbf{\textasciitilde}\ [i]\ [\textbf{:}]\ [\textbf{@}]\ \textbf{[}\ [\{text\ \textbf{\textasciitilde;}\}^{*}\ text]\ [\textbf{\textasciitilde:;}\ default]\ \textbf{\textasciitilde]}$

▷ **Conditional Expression.** Use the zero-indexed argumenth (or *i*th 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.

~ $\left[\textit{prefix}\ \{,\textit{prefix}\}^*\right]$ [**:**] [**@**] **/**$\left[\textit{package}\ \textbf{:}[\textbf{:}]_{\boxed{\texttt{cl-user:}}}\right]$*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

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

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

(**make-concatenated-stream**$^{\text{Fu}}$ *input-stream**)
(**make-broadcast-stream**$^{\text{Fu}}$ *output-stream**)
(**make-two-way-stream**$^{\text{Fu}}$ *input-stream-part output-stream-part*)
(**make-echo-stream**$^{\text{Fu}}$ *from-input-stream to-output-stream*)
(**make-synonym-stream**$^{\text{Fu}}$ *variable-bound-to-stream*)

▷ Return <u>stream</u> of indicated type.

(**make-string-input-stream**$^{\text{Fu}}$ *string* $\left[\textit{start}_{\boxed{0}}\ [\textit{end}_{\boxed{\texttt{NIL}}}]\right]$)

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

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

▷ Return a <u>**string-stream**</u> accepting characters (available via **get-output-stream-string**$^{\text{Fu}}$).

(**concatenated-stream-streams**$^{\text{Fu}}$ *concatenated-stream*)
(**broadcast-stream-streams**$^{\text{Fu}}$ *broadcast-stream*)

▷ Return <u>list of streams</u> *concatenated-stream* still has to read from/*broadcast-stream* is broadcasting to.

(**two-way-stream-input-stream**$^{\text{Fu}}$ *two-way-stream*)
(**two-way-stream-output-stream**$^{\text{Fu}}$ *two-way-stream*)
(**echo-stream-input-stream**$^{\text{Fu}}$ *echo-stream*)
(**echo-stream-output-stream**$^{\text{Fu}}$ *echo-stream*)

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

(**synonym-stream-symbol**$^{\text{Fu}}$ *synonym-stream*)

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

(**get-output-stream-string**$^{\text{Fu}}$ $\widetilde{\textit{string-stream}}$)

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

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

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

(**file-string-length**$^{\text{Fu}}$ *stream foo*)

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

(**listen**$^{\text{Fu}}$ $[\textit{stream}_{\boxed{\overset{\text{var}}{\textbf{*standard-input*}}}}]$)

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

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

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

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

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

(**close**$^{\text{Fu}}$ $\widetilde{\textit{stream}}$ [**:abort** $\textit{bool}_{\boxed{\texttt{NIL}}}$])

▷ Close *stream*. Return <u>T</u> if *stream* had been open. If **:abort** is T, delete associated file.

(**with-open-file**$^{\text{M}}$ (*stream path open-arg**) (**declare** $\widehat{\textit{decl}}$*)* $\textit{form}^{\text{P}}$*)

▷ Use **open**$^{\text{Fu}}$ with *open-arg*s to temporarily create *stream* to *path*; return <u>values of *form*s</u>.

(**with-open-stream**$^{\text{M}}$ (*foo* $\widetilde{\textit{stream}}$) (**declare** $\widehat{\textit{decl}}$*)* $\textit{form}^{\text{P}}$*)

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

(**with-input-from-string**$^{\text{M}}$ (*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** $\widehat{\textit{decl}}$*)* $\textit{form}^{\text{P}}$*)

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

(**with-output-to-string**$^{\text{M}}$ (*foo* $[\widetilde{\textit{string}}_{\boxed{\texttt{NIL}}}$ [**:element-type** $\textit{type}_{\boxed{\textbf{character}}}$]]) (**declare** $\widehat{\textit{decl}}$*)* $\textit{form}^{\text{P}}$*)

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

(**stream-external-format**$^{\text{Fu}}$ *stream*)

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

terminal-io$^{\text{var}}$ ▷ Bidirectional stream to user terminal.

standard-input$^{\text{var}}$
standard-output$^{\text{var}}$
error-output$^{\text{var}}$

▷ Standard input stream, standard output stream, or standard error output stream, respectively.

debug-io$^{\text{var}}$
query-io$^{\text{var}}$

▷ Bidirectional streams for debugging and user interaction.

$(\overset{\text{Fu}}{\textbf{compile-file}}\ \textit{file} \left\{\begin{array}{|l}\textbf{:output-file}\ \textit{out-path}\\ \textbf{:verbose}\ \textit{bool}_{\boxed{\overset{\text{var}}{\text{*compile-verbose*}}}}\\ \textbf{:print}\ \textit{bool}_{\boxed{\overset{\text{var}}{\text{*compile-print*}}}}\\ \textbf{:external-format}\ \textit{file-format}_{\boxed{\text{:default}}}\end{array}\right\})$

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

$(\overset{\text{Fu}}{\textbf{compile-file-pathname}}\ \textit{file}\ [\textbf{:output-file}\ \textit{path}]\ [\textit{other-keyargs}])$

▷ Pathname $\overset{\text{Fu}}{\textbf{compile-file}}$ writes to if invoked with the same arguments.

$(\overset{\text{Fu}}{\textbf{load}}\ \textit{path} \left\{\begin{array}{|l}\textbf{:verbose}\ \textit{bool}_{\boxed{\overset{\text{var}}{\text{*load-verbose*}}}}\\ \textbf{:print}\ \textit{bool}_{\boxed{\overset{\text{var}}{\text{*load-print*}}}}\\ \textbf{:if-does-not-exist}\ \textit{bool}_{\boxed{\text{T}}}\\ \textbf{:external-format}\ \textit{file-format}_{\boxed{\text{:default}}}\end{array}\right\})$

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

$\left.\begin{array}{l}\overset{\text{var}}{\textbf{*compile-file}}\\ \overset{\text{var}}{\textbf{*load}}\end{array}\right\}\text{-}\left\{\begin{array}{l}\textbf{pathname*}_{\boxed{\text{NIL}}}\\ \textbf{truename*}_{\boxed{\text{NIL}}}\end{array}\right.$

▷ 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 values of *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 `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 values of *forms*.

$(\overset{\text{M}}{\textbf{with-compilation-unit}}\ ([\textbf{:override}\ \textit{bool}_{\boxed{\text{NIL}}}])\ \textit{form}^{\text{P}*})$

▷ Return values of *forms*. Warnings deferred by the compiler until end of compilation are deferred until the end of evaluation of *forms*.

$(\overset{\text{sO}}{\textbf{load-time-value}}\ \textit{form}\ [\widehat{\textit{read-only}}_{\boxed{\text{NIL}}}])$

▷ Evaluate *form* at compile time and treat its value as literal at run time.

$(\overset{\text{sO}}{\textbf{quote}}\ \widehat{\textit{foo}})$ ▷ Return unevaluated *foo*.

$(\overset{\text{gF}}{\textbf{make-load-form}}\ \textit{foo}\ [\textit{environment}])$

▷ Its methods are to return a creation form which on evaluation at $\overset{\text{Fu}}{\textbf{load}}$ time returns an object equivalent to *foo*, and an optional initialization form$_2$ 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 creation form and an initialization form$_2$ 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 macro function, or compiler macro function, respectively, if any. Return `NIL` otherwise. **setf**able.

$(\overset{\text{Fu}}{\textbf{eval}}\ \textit{arg})$

▷ Return values of value of *arg* evaluated in global environment.

$(\overset{\text{Fu}}{\textbf{load-logical-pathname-translations}}\ \textit{logical-host})$

▷ Load *logical-host*'s translations. Return `NIL` if already loaded; return `T` if successful.

$(\overset{\text{Fu}}{\textbf{translate-logical-pathname}}\ \textit{pathname})$

▷ Physical pathname corresponding to (possibly logical) *pathname*.

$(\overset{\text{Fu}}{\textbf{probe-file}}\ \textit{file})$
$(\overset{\text{Fu}}{\textbf{truename}}\ \textit{file})$

▷ Canonical name of *file*. If *file* does not exist, return `NIL`/signal **file-error**, respectively.

$(\overset{\text{Fu}}{\textbf{file-write-date}}\ \textit{file})$ ▷ Time at which *file* was last written.

$(\overset{\text{Fu}}{\textbf{file-author}}\ \textit{file})$ ▷ Return name of *file* owner.

$(\overset{\text{Fu}}{\textbf{file-length}}\ \textit{stream})$ ▷ Return length of *stream*.

$(\overset{\text{Fu}}{\textbf{rename-file}}\ \textit{foo}\ \textit{bar})$

▷ Rename file *foo* to *bar*. Unspecified components of path *bar* default to those of *foo*. Return new pathname, old physical file name$_2$, and new physical file name$_3$.

$(\overset{\text{Fu}}{\textbf{delete-file}}\ \textit{file})$ ▷ Delete *file*. Return `T`.

$(\overset{\text{Fu}}{\textbf{directory}}\ \textit{path})$ ▷ List of pathnames matching *path*.

$(\overset{\text{Fu}}{\textbf{ensure-directories-exist}}\ \textit{path}\ [\textbf{:verbose}\ \textit{bool}])$

▷ Create parts of *path* if necessary. Second return value is `T`$_2$ if something has been created.

14 Packages and Symbols

14.1 Predicates

$(\overset{\text{Fu}}{\textbf{symbolp}}\ \textit{foo})$
$(\overset{\text{Fu}}{\textbf{packagep}}\ \textit{foo})$ ▷ `T` if *foo* is of indicated type.
$(\overset{\text{Fu}}{\textbf{keywordp}}\ \textit{foo})$

14.2 Packages

:*bar*|**keyword:***bar* ▷ Keyword, evaluates to :*bar*.

package:*symbol* ▷ Exported *symbol* of *package*.

package::*symbol* ▷ Possibly unexported *symbol* of *package*.

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

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

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

▷ Create package *foo*.

$(\overset{\text{Fu}}{\textbf{rename-package}}\ \textit{package}\ \textit{new-name}\ [\textit{new-nicknames}_{\boxed{\text{NIL}}}])$

▷ Rename *package*. Return renamed package.

$(\overset{\text{M}}{\textbf{in-package}}\ \widehat{\textit{foo}})$ ▷ Make package *foo* current.

$(\left\{\begin{array}{l}\overset{\text{Fu}}{\textbf{use-package}}\\ \overset{\text{Fu}}{\textbf{unuse-package}}\end{array}\right\}\ \textit{other-packages}\ [\textit{package}_{\boxed{\overset{\text{var}}{\text{*package*}}}}])$

▷ Make exported symbols of *other-packages* available in *package*, or remove them from *package*, respectively. Return `T`.

(**package-use-list**$^{\text{Fu}}$ *package*)
(**package-used-by-list**$^{\text{Fu}}$ *package*)
▷ List of other packages used by/using *package*.

(**delete-package**$^{\text{Fu}}$ $\widetilde{package}$)
▷ Delete *package*. Return `T` if successful.

package$^{\text{var}}_{\boxed{\text{common-lisp-user}}}$ ▷ The current package.

(**list-all-packages**$^{\text{Fu}}$) ▷ List of registered packages.

(**package-name**$^{\text{Fu}}$ *package*) ▷ Name of *package*.

(**package-nicknames**$^{\text{Fu}}$ *package*) ▷ List of nicknames of *package*.

(**find-package**$^{\text{Fu}}$ *name*) ▷ Package with *name* (case-sensitive).

(**find-all-symbols**$^{\text{Fu}}$ *foo*)
▷ List of symbols *foo* from all registered packages.

($\begin{Bmatrix}\textbf{intern}^{\text{Fu}}\\ \textbf{find-symbol}^{\text{Fu}}\end{Bmatrix}$ *foo* [*package*$_{\boxed{\text{*package*}^{\text{var}}}}$])
▷ Intern or find, respectively, symbol *foo* in *package*. Second return value is one of **:internal**$_2$, **:external**$_2$, or **:inherited**$_2$ (or `NIL`$_2$ if **intern**$^{\text{Fu}}$ created a fresh symbol).

(**unintern**$^{\text{Fu}}$ *symbol* [*package*$_{\boxed{\text{*package*}^{\text{var}}}}$])
▷ Remove *symbol* from *package*, return `T` on success.

($\begin{Bmatrix}\textbf{import}^{\text{Fu}}\\ \textbf{shadowing-import}^{\text{Fu}}\end{Bmatrix}$ *symbols* [*package*$_{\boxed{\text{*package*}^{\text{var}}}}$])
▷ Make *symbols* internal to *package*. Return `T`. In case of a name conflict signal correctable **package-error** or shadow the old symbol, respectively.

(**shadow**$^{\text{Fu}}$ *symbols* [*package*$_{\boxed{\text{*package*}^{\text{var}}}}$])
▷ Make *symbols* of *package* shadow any otherwise accessible, equally named symbols from other packages. Return `T`.

(**package-shadowing-symbols**$^{\text{Fu}}$ *package*)
▷ List of symbols of *package* that shadow any otherwise accessible, equally named symbols from other packages.

(**export**$^{\text{Fu}}$ *symbols* [*package*$_{\boxed{\text{*package*}^{\text{var}}}}$])
▷ Make *symbols* external to *package*. Return `T`.

(**unexport**$^{\text{Fu}}$ *symbols* [*package*$_{\boxed{\text{*package*}^{\text{var}}}}$])
▷ Revert *symbols* to internal status. Return `T`.

($\left\{\begin{matrix}\begin{Bmatrix}\textbf{do-symbols}^{\text{M}}\\ \textbf{do-external-symbols}^{\text{M}}\end{Bmatrix} (\widehat{var}\ [package_{\boxed{\text{*package*}^{\text{var}}}}\ [result_{\boxed{\text{NIL}}}]])\\ \textbf{do-all-symbols}^{\text{M}}\ (var\ [result_{\boxed{\text{NIL}}}])\end{matrix}\right\}$
(**declare** $\widehat{decl}$*)* $\left\{\begin{vmatrix}\widehat{tag}\\ form\end{vmatrix}\right\}$*)
▷ Evaluate **tagbody**$^{\text{sO}}$-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 values of *result*. Implicitly, the whole form is a **block**$^{\text{sO}}$ named `NIL`.

(**with-package-iterator**$^{\text{M}}$ (*foo packages* [**:internal**|**:external**|**:inherited**])
(**declare** $\widehat{decl}$*)* *form*$^{\text{P}}$*)
▷ Return 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.

(**require**$^{\text{Fu}}$ *module* [*paths*$_{\boxed{\text{NIL}}}$])
▷ If not in ***modules***$^{\text{var}}$, try *paths* to load *module* from. Signal **error** if unsuccessful. Deprecated.

(**provide**$^{\text{Fu}}$ *module*)
▷ If not already there, add *module* to ***modules***$^{\text{var}}$. Deprecated.

modules$^{\text{var}}$ ▷ 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*).

(**make-symbol**$^{\text{Fu}}$ *name*)
▷ Make fresh, uninterned symbol *name*.

(**gensym**$^{\text{Fu}}$ [*s*$_{\boxed{\text{G}}}$])
▷ Return fresh, uninterned symbol **#:***sn* with *n* from ***gensym-counter***$^{\text{var}}$. Increment ***gensym-counter***$^{\text{var}}$.

(**gentemp**$^{\text{Fu}}$ [*prefix*$_{\boxed{\text{T}}}$ [*package*$_{\boxed{\text{*package*}^{\text{var}}}}$]])
▷ Intern fresh symbol in package. Deprecated.

(**copy-symbol**$^{\text{Fu}}$ *symbol* [*props*$_{\boxed{\text{NIL}}}$])
▷ Return uninterned copy of *symbol*. If *props* is `T`, give copy the same value, function and property list.

(**symbol-name**$^{\text{Fu}}$ *symbol*)
(**symbol-package**$^{\text{Fu}}$ *symbol*)
(**symbol-plist**$^{\text{Fu}}$ *symbol*)
(**symbol-value**$^{\text{Fu}}$ *symbol*)
(**symbol-function**$^{\text{Fu}}$ *symbol*)
▷ Name, package, property list, value, or function, respectively, of *symbol*. **setf**able.

($\begin{Bmatrix}\textbf{documentation}^{\text{gF}}\\ (\textbf{setf documentation}^{\text{gF}})\ new\text{-}doc\end{Bmatrix}$ *foo* $\begin{Bmatrix}\text{'variable}|\text{'function}\\ \text{'compiler-macro}\\ \text{'method-combination}\\ \text{'structure}|\text{'type}|\text{'setf}|\texttt{T}\end{Bmatrix}$)
▷ Get/set documentation string of *foo* of given type.

t$^{\text{co}}$
▷ Truth; the supertype of every type including **t**; the superclass of every class except **t**; ***terminal-io***$^{\text{var}}$.

nil$^{\text{co}}$|**()**$^{\text{co}}$
▷ Falsity; the empty list; the empty type, subtype of every type; ***standard-input***$^{\text{var}}$; ***standard-output***$^{\text{var}}$; 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

(**special-operator-p**$^{\text{Fu}}$ *foo*) ▷ `T` if *foo* is a special operator.

(**compiled-function-p**$^{\text{Fu}}$ *foo*)
▷ `T` if *foo* is of type **compiled-function**.

15.2 Compilation

(**compile**$^{\text{Fu}}$ $\left\{\begin{matrix}\texttt{NIL}\ definition\\ \begin{Bmatrix}name\\ (\textbf{setf}\ name)\end{Bmatrix}\ [definition]\end{matrix}\right\}$)
▷ Return compiled function or replace *name*'s function definition with the compiled function. Return `T`$_2$ in case of warnings or errors, and `T`$_3$ in case of warnings or errors excluding style warnings.

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.

($\begin{Bmatrix}\overset{\text{Fu}}{\textbf{macroexpand-1}}\\ \overset{\text{Fu}}{\textbf{macroexpand}}\end{Bmatrix}$ *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.

($\overset{\text{M}}{\textbf{trace}}$ $\begin{Bmatrix}\textit{function}\\ (\textbf{setf}\ \textit{function})\end{Bmatrix}^{*}$)
▷ Cause *functions* to be traced. With no arguments, return $\underline{\text{list of traced functions}}$.

($\overset{\text{M}}{\textbf{untrace}}$ $\begin{Bmatrix}\textit{function}\\ (\textbf{setf}\ \textit{function})\end{Bmatrix}^{*}$)
▷ Stop *functions*, 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 *args*.

($\overset{\text{M}}{\textbf{time}}$ *form*)
▷ Evaluate *forms* 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

$(\overset{\text{Fu}}{\textbf{proclaim}}\ decl)$
$(\overset{\text{M}}{\textbf{declaim}}\ \widehat{decl^*})$

▷ 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.

$(\textbf{dynamic-extent}\ variable^*\ (\overset{\text{sO}}{\textbf{function}}\ function)^*)$

▷ Declare lifetime of *variables* and/or *functions* to end when control leaves enclosing block.

([**type**] *type* $variable^*$)
(**ftype** *type* $function^*$)

▷ Declare *variables* or *functions* to be of *type*.

$$\left(\begin{Bmatrix}\textbf{ignorable}\\ \textbf{ignore}\end{Bmatrix}\begin{Bmatrix}var\\ (\overset{\text{sO}}{\textbf{function}}\ function)\end{Bmatrix}^*\right)$$

▷ Suppress warnings about used/unused bindings.

(**inline** $function^*$)
(**notinline** $function^*$)

▷ Tell compiler to integrate/not to integrate, respectively, called *functions* into the calling routine.

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

▷ Tell compiler how to optimize. $n = 0$ means unimportant, $n = 1$ is neutral, $n = 3$ means important.

(**special** var^*) ▷ Declare *var*s to be dynamic.

16 External Environment

$(\overset{\text{Fu}}{\textbf{get-internal-real-time}})$
$(\overset{\text{Fu}}{\textbf{get-internal-run-time}})$

▷ $\underline{\text{Current time}}$, or $\underline{\text{computing time}}$, respectively, in clock ticks.

$\overset{\text{co}}{\textbf{internal-time-units-per-second}}$

▷ Number of clock ticks per second.

$(\overset{\text{Fu}}{\textbf{encode-universal-time}}\ sec\ min\ hour\ date\ month\ year\ [zone_{\boxed{\text{curr}}}])$
$(\overset{\text{Fu}}{\textbf{get-universal-time}})$

▷ $\underline{\text{Seconds from 1900-01-01, 00:00}}$, ignoring leap seconds.

$(\overset{\text{Fu}}{\textbf{decode-universal-time}}\ universal\text{-}time\ [time\text{-}zone_{\boxed{\text{current}}}])$
$(\overset{\text{Fu}}{\textbf{get-decoded-time}})$

▷ 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 $\underset{9}{\underline{\text{zone}}}$.

$(\overset{\text{Fu}}{\textbf{room}}\ [\{\texttt{NIL}|\textbf{:default}|\texttt{T}\}])$

▷ Print information about internal storage management.

$(\overset{\text{Fu}}{\textbf{short-site-name}})$
$(\overset{\text{Fu}}{\textbf{long-site-name}})$

▷ $\underline{\text{String}}$ representing physical location of computer.

$$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{lisp-implementation}}\\ \overset{\text{Fu}}{\textbf{software}}\\ \overset{\text{Fu}}{\textbf{machine}}\end{Bmatrix}\text{-}\begin{Bmatrix}\textbf{type}\\ \textbf{version}\end{Bmatrix}\right)$$

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

$(\overset{\text{Fu}}{\textbf{machine-instance}})$ ▷ $\underline{\text{Computer name}}$.

Index

Common Lisp Quick Reference Revision 130 [2011-10-12]
Copyright © 2008, 2009, 2010, 2011 Bert Burgemeister
LaTeX source: http://clqr.boundp.org