Quick Reference

Common lisp

Bert Burgemeister

Common Lisp Quick Reference Revision 94 [2009-02-07]

LaTeX source: http://clqr.berlios.de

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.

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

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

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

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

$foo^{\overset{\text{P}}{*}}$ ▷ foo^* is evaluated as in $\overset{\text{sO}}{\textbf{progn}}$; see p. 20.

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

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

1 Numbers

1.1 Predicates

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

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

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

▷ Return $\underline{\texttt{T}}$ if *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

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

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

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

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

($\left\{\begin{matrix}\overset{\text{M}}{\textbf{incf}}\\ \overset{\text{M}}{\textbf{decf}}\end{matrix}\right\}$ $\widetilde{place}$ [*delta*$_{\boxed{1}}$])

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

($\overset{\text{Fu}}{\textbf{exp}}$ *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 *integers*. (**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.

$(\overset{\text{Fu}}{\textbf{sinh}}\ a)$
$(\overset{\text{Fu}}{\textbf{cosh}}\ a)$ ▷ $\underline{\sinh a}$, $\underline{\cosh a}$, or $\underline{\tanh a}$, respectively.
$(\overset{\text{Fu}}{\textbf{tanh}}\ a)$

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

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

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

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

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

▷ Return $\underline{n/d}$ (**integer** or **float**, respectively) truncated towards $-\infty$, $+\infty$, 0, or rounded, respectively; and $\underset{2}{\underline{\text{remainder}}}$.

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

▷ 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.

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

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

$(\overset{\text{Fu}}{\textbf{signum}}\ n)$
▷ 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 $\underline{|n|}$.

$(\overset{\text{Fu}}{\textbf{rational}}\ real)$
$(\overset{\text{Fu}}{\textbf{rationalize}}\ real)$
▷ Convert *real* to rational. Assume complete/limited accuracy for *real*.

$(\overset{\text{Fu}}{\textbf{float}}\ real\ [prototype_{\boxed{\textbf{single-float}}}])$
▷ Convert *real* into float with type of *prototype*.

Index

1.3 Logic Functions

Negative integers are used in two's complement representation.

($\overset{\text{Fu}}{\textbf{boole}}$ *operation int-a int-b*)
▷ Return $\underline{\text{value}}$ of bitwise logical *operation*. *operation*s are

$\overset{\text{co}}{\textbf{boole-1}}$	▷ $\underline{int\text{-}a}$.
$\overset{\text{co}}{\textbf{boole-2}}$	▷ $\underline{int\text{-}b}$.
$\overset{\text{co}}{\textbf{boole-c1}}$	▷ $\underline{\neg int\text{-}a}$.
$\overset{\text{co}}{\textbf{boole-c2}}$	▷ $\underline{\neg int\text{-}b}$.
$\overset{\text{co}}{\textbf{boole-set}}$	▷ $\underline{\text{All bits set}}$.
$\overset{\text{co}}{\textbf{boole-clr}}$	▷ $\underline{\text{All bits zero}}$.
$\overset{\text{co}}{\textbf{boole-eqv}}$	▷ $\underline{int\text{-}a \equiv int\text{-}b}$.
$\overset{\text{co}}{\textbf{boole-and}}$	▷ $\underline{int\text{-}a \wedge int\text{-}b}$.
$\overset{\text{co}}{\textbf{boole-andc1}}$	▷ $\underline{\neg int\text{-}a \wedge int\text{-}b}$.
$\overset{\text{co}}{\textbf{boole-andc2}}$	▷ $\underline{int\text{-}a \wedge \neg int\text{-}b}$.
$\overset{\text{co}}{\textbf{boole-nand}}$	▷ $\underline{\neg(int\text{-}a \wedge int\text{-}b)}$.
$\overset{\text{co}}{\textbf{boole-ior}}$	▷ $\underline{int\text{-}a \vee int\text{-}b}$.
$\overset{\text{co}}{\textbf{boole-orc1}}$	▷ $\underline{\neg int\text{-}a \vee int\text{-}b}$.
$\overset{\text{co}}{\textbf{boole-orc2}}$	▷ $\underline{int\text{-}a \vee \neg int\text{-}b}$.
$\overset{\text{co}}{\textbf{boole-xor}}$	▷ $\underline{\neg(int\text{-}a \equiv int\text{-}b)}$.
$\overset{\text{co}}{\textbf{boole-nor}}$	▷ $\underline{\neg(int\text{-}a \vee int\text{-}b)}$.

($\overset{\text{Fu}}{\textbf{lognot}}$ *integer*) ▷ $\underline{\neg integer}$.

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

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

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

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

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

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

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

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

($\overset{\text{Fu}}{\textbf{logbitp}}$ *i integer*)
▷ $\underline{\texttt{T}}$ if zero-indexed *i*th bit of *integer* is set.

($\overset{\text{Fu}}{\textbf{logtest}}$ *int-a int-b*)
▷ Return $\underline{\texttt{T}}$ if there is any bit set in *int-a* which is set in *int-b* as well.

($\overset{\text{Fu}}{\textbf{logcount}}$ *int*)
▷ $\underline{\text{Number of 1 bits}}$ in *int* ≥ 0, $\underline{\text{number of 0 bits}}$ in *int* < 0.

($\overset{\text{Fu}}{\textbf{ash}}$ *integer count*)
▷ Return copy of $\underline{integer}$ arithmetically shifted left by *count* adding zeros at the right, or, for *count* < 0, shifted right discarding bits.

($\overset{\text{Fu}}{\textbf{mask-field}}$ *byte-spec integer*)
▷ Return copy of $\underline{integer}$ with all bits unset but those denoted by *byte-spec*. **setf**able.

1.4 Integer Functions

($^{\text{Fu}}$**integer-length** *integer*)
▷ Number of bits necessary to represent *integer*.

($^{\text{Fu}}$**ldb-test** *byte-spec integer*)
▷ Return T if any bit specified by *byte-spec* in *integer* is set.

($^{\text{Fu}}$**ldb** *byte-spec integer*)
▷ Extract byte denoted by *byte-spec* from *integer*. **setf**able.

({$^{\text{Fu}}$**deposit-field** | $^{\text{Fu}}$**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 ($^{\text{Fu}}$**byte-size** *byte-spec*) bits of *int-a*, respectively.

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

($^{\text{Fu}}$**byte-size** *byte-spec*)
($^{\text{Fu}}$**byte-position** *byte-spec*)
▷ Size or position, respectively, of *byte-spec*.

1.5 Implementation-Dependent

{$^{\text{co}}$**short-float** | $^{\text{co}}$**single-float** | $^{\text{co}}$**double-float** | $^{\text{co}}$**long-float**}-{**epsilon** | **negative-epsilon**}
▷ Smallest possible number making a difference when added or subtracted, respectively.

{$^{\text{co}}$**least-negative** | $^{\text{co}}$**least-negative-normalized** | $^{\text{co}}$**least-positive** | $^{\text{co}}$**least-positive-normalized**}-{**short-float** | **single-float** | **double-float** | **long-float**}
▷ Available numbers closest to −0 or +0, respectively.

{$^{\text{co}}$**most-negative** | $^{\text{co}}$**most-positive**}-{**short-float** | **single-float** | **double-float** | **long-float** | **fixnum**}
▷ Available numbers closest to −∞ or +∞, respectively.

($^{\text{Fu}}$**decode-float** *n*)
($^{\text{Fu}}$**integer-decode-float** *n*)
▷ Return significand, $\underline{\text{exponent}}_2$, and $\underline{\text{sign}}_3$ 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*$_{\boxed{\text{NIL}}}$])
▷ Type of most specialized **complex** number able to hold parts of type *foo*.

2 Characters

($^{\text{Fu}}$**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.

15.4 Declarations

($^{\text{Fu}}$**proclaim** *decl*)
($^{\text{M}}$**declaim** $\widehat{decl^*}$)
▷ Globally make declaration(s) *decl*. *decl* can be: **declaration**, **type**, **ftype**, **inline**, **notinline**, **optimize**, or **special**. See below.

(**declare** $\widehat{decl^*}$)
▷ Inside certain forms, locally make declarations *decl**. *decl* can be: **dynamic-extent**, **type**, **ftype**, **ignorable**, **ignore**, **inline**, **notinline**, **optimize**, or **special**. See below.

(**declaration** foo*)
▷ Make *foo*s names of declarations.

(**dynamic-extent** *variable** ($^{\text{sO}}$**function** *function*)*)
▷ Declare lifetime of *variable*s and/or *function*s to end when control leaves enclosing block.

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

({**ignorable** | **ignore**} {*var* | ($^{\text{sO}}$**function** *function*)}*)
▷ Suppress warnings about used/unused bindings.

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

(**optimize** {**compilation-speed**|(**compilation-speed** $n_{\boxed{3}}$) | **debug**|(**debug** $n_{\boxed{3}}$) | **safety**|(**safety** $n_{\boxed{3}}$) | **space**|(**space** $n_{\boxed{3}}$) | **speed**|(**speed** $n_{\boxed{3}}$)})
▷ Tell compiler how to optimize. $n = 0$ means unimportant, $n = 1$ is neutral, $n = 3$ means important.

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

16 External Environment

($^{\text{Fu}}$**get-internal-real-time**)
($^{\text{Fu}}$**get-internal-run-time**)
▷ Current time, or computing time, respectively, in clock ticks.

$^{\text{co}}$**internal-time-units-per-second**
▷ Number of clock ticks per second.

($^{\text{Fu}}$**encode-universal-time** *sec min hour date month year* [*zone*$_{\boxed{\text{curr}}}$])
($^{\text{Fu}}$**get-universal-time**)
▷ Seconds from 1900-01-01, 00:00.

($^{\text{Fu}}$**decode-universal-time** *universal-time* [*time-zone*$_{\boxed{\text{current}}}$])
($^{\text{Fu}}$**get-decoded-time**)
▷ Return second, $\underline{\text{minute}}_2$, $\underline{\text{hour}}_3$, $\underline{\text{date}}_4$, $\underline{\text{month}}_5$, $\underline{\text{year}}_6$, $\underline{\text{day}}_7$, $\underline{\text{daylight-p}}_8$, and $\underline{\text{zone}}_9$.

($^{\text{Fu}}$**room** [{NIL|**:default**|T}])
▷ Print information about internal storage management.

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

({$^{\text{Fu}}$**lisp-implementation** | $^{\text{Fu}}$**software** | $^{\text{Fu}}$**machine**}-{**type** | **version**})
▷ Name or version of implementation, operating system, or hardware, respectively.

($^{\text{Fu}}$**machine-instance**) ▷ Computer name.

($\overset{\text{Fu}}{\textbf{eval}}$ *arg*)
▷ Return values of value of *arg* evaluated in global environment.

15.3 REPL and Debugging

$\overset{\text{var}}{+}|\overset{\text{var}}{++}|\overset{\text{var}}{+++}$
$\overset{\text{var}}{*}\ |\ \overset{\text{var}}{**}\ |\ \overset{\text{var}}{***}$
$\overset{\text{var}}{/}\ |\ \overset{\text{var}}{//}\ |\ \overset{\text{var}}{///}$
▷ Last, penultimate, or antepenultimate form evaluated in the REPL, or their respective primary value, or a list of their respective values.

$\overset{\text{var}}{-}$ ▷ 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*$_{\boxed{\texttt{NIL}}}$])
▷ List of interned symbols containing *string*.

($\overset{\text{Fu}}{\textbf{dribble}}$ [*path*])
▷ Save a record of interactive session to file at *path*. Without *path*, close that file.

($\overset{\text{Fu}}{\textbf{ed}}$ [*file-or-function*$_{\boxed{\texttt{NIL}}}$]) ▷ Invoke editor if possible.

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

$\overset{\text{var}}{\textbf{*macroexpand-hook*}}$
▷ Function of arguments expansion function, macro form, and environment called by $\overset{\text{Fu}}{\textbf{macroexpand-1}}$ to generate macro expansions.

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

($\overset{\text{M}}{\textbf{untrace}}$ $\left\{\begin{array}{l}\textit{function}\\ (\textbf{setf}\ \textit{function})\end{array}\right\}^{*}$)
▷ Stop *function*s, or each currently traced function, from being traced.

$\overset{\text{var}}{\textbf{*trace-output*}}$
▷ Stream $\overset{\text{M}}{\textbf{trace}}$ and $\overset{\text{M}}{\textbf{time}}$ print their output on.

($\overset{\text{M}}{\textbf{step}}$ *form*)
▷ Step through evaluation of *form*. Return values of *form*.

($\overset{\text{Fu}}{\textbf{break}}$ [*control arg**])
▷ Jump directly into debugger; return NIL. See p. 35, $\overset{\text{Fu}}{\textbf{format}}$, for *control* and *arg*s.

($\overset{\text{M}}{\textbf{time}}$ *form*)
▷ Evaluate *form*s and print timing information to $\overset{\text{var}}{\textbf{*trace-output*}}$. Return values of *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 NIL.

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

($\overset{\text{Fu}}{\textbf{digit-char-p}}$ *character* [*radix*$_{\boxed{10}}$])
▷ Return its weight if *character* is a digit, or NIL otherwise.

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

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

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

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

($\overset{\text{Fu}}{\textbf{char-upcase}}$ *character*)
($\overset{\text{Fu}}{\textbf{char-downcase}}$ *character*)
▷ Return corresponding uppercase/lowercase character, respectively.

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

($\overset{\text{Fu}}{\textbf{char-name}}$ *character*)
▷ Name of *character* if there is one, or NIL.

($\overset{\text{Fu}}{\textbf{name-char}}$ *name*)
▷ Character with *name* if there is one, or NIL.

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

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

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

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

3 Strings

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

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

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

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

▷ If *foo* is lexicographically not equal, greater, not less, less, or not greater, respectively, then return character number from beginning of *foo* where they begin to differ. Otherwise return `NIL`.

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

▷ If *foo* is lexicographically not equal, greater, not less, less, or not greater, respectively, ignoring case, then return character number from beginning of *foo* where they begin to differ. Otherwise return `NIL`.

($\overset{\text{Fu}}{\textbf{string}}$ *x*)

▷ Convert *x* (**symbol**, **string**, or **character**) into a string.

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

▷ Return string of length *size*.

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

▷ Return *string* (not modified or modified, respectively) with first letter of every word turned into uppercase, letters all uppercase, or letters all lowercase, respectively.

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

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

($\overset{\text{Fu}}{\textbf{char}}$ *string i*)
($\overset{\text{Fu}}{\textbf{schar}}$ *string i*)

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

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

▷ Return integer parsed from *string* and $\underset{2}{\text{index}}$ of parse end.

4 Conses

4.1 Predicates

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

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

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

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

$$\left(\overset{\text{Fu}}{\textbf{member}}\ \mathit{foo}\ \mathit{list}\ \left\{\left|\begin{array}{l}\left\{\begin{array}{l}\textbf{:test}\ \mathit{function}_{\boxed{\textbf{eql}}}\\\textbf{:test-not}\ \mathit{function}\end{array}\right.\\\textbf{:key}\ \mathit{function}\end{array}\right.\right\}\right)$$

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

15.2 Compilation

$$\left(\overset{\text{Fu}}{\textbf{compile}}\ \left\{\begin{array}{l}\texttt{NIL}\ \mathit{definition}\\\left\{\begin{array}{l}\mathit{name}\\(\textbf{setf}\ \mathit{name})\end{array}\right\}\ [\mathit{definition}]\end{array}\right\}\right)$$

▷ Return compiled function or replace *name*'s function definition with the compiled function. Return $\underset{2}{\texttt{T}}$ in case of warnings or errors, and $\underset{3}{\texttt{T}}$ in case of warnings or errors excluding style warnings.

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

▷ Write compiled contents of *file* to *out-path*. Return true output path or `NIL`, $\underset{2}{\texttt{T}}$ in case of warnings or errors, $\underset{3}{\texttt{T}}$ in case of warnings or errors excluding style warnings.

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

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

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

▷ Load source file or compiled file into Lisp environment. Return $\underline{\texttt{T}}$ if successful.

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

▷ Input file used by $\overset{\text{Fu}}{\textbf{compile-file}}$/by $\overset{\text{Fu}}{\textbf{load}}$.

$$\left.\begin{array}{l}\overset{\text{var}}{\textbf{*compile}}\\\overset{\text{var}}{\textbf{*load}}\end{array}\right\}\text{-}\left\{\begin{array}{l}\textbf{print*}\\\textbf{verbose*}\end{array}\right.$$

▷ Defaults used by $\overset{\text{Fu}}{\textbf{compile-file}}$/by $\overset{\text{Fu}}{\textbf{load}}$.

$$\left(\overset{\text{sO}}{\textbf{eval-when}}\ \left(\left\{\left|\begin{array}{l}\{\textbf{:compile-toplevel}|\textbf{compile}\}\\\{\textbf{:load-toplevel}|\textbf{load}\}\\\{\textbf{:execute}|\textbf{eval}\}\end{array}\right.\right\}\right)\ \mathit{form}^{P_*}\right)$$

▷ 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.)

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

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

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

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

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

($\overset{\text{gF}}{\textbf{make-load-form}}$ *foo* [*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 $\underset{2}{\text{initialization form}}$ which on evaluation performs some initialization of the object.

$$\left(\overset{\text{Fu}}{\textbf{make-load-form-saving-slots}}\ \mathit{foo}\ \left\{\left|\begin{array}{l}\textbf{:slot-names}\ \mathit{slots}_{\boxed{\text{all local slots}}}\\\textbf{:environment}\ \mathit{environment}\end{array}\right.\right\}\right)$$

▷ Return a creation form and an $\underset{2}{\text{initialization form}}$ which on evaluation construct an object equivalent to *foo* with *slots* initialized with the corresponding values from *foo*.

($\overset{\text{Fu}}{\textbf{macro-function}}$ *symbol* [*environment*])

$$\left(\overset{\text{Fu}}{\textbf{compiler-macro-function}}\ \left\{\begin{array}{l}\mathit{name}\\(\textbf{setf}\ \mathit{name})\end{array}\right\}\ [\mathit{environment}]\right)$$

▷ Return specified macro function, or compiler macro function, respectively, if any. Return `NIL` otherwise. **setf**able.

(**require**$^{\text{Fu}}$ *module* [*path-list*$_{\boxed{\texttt{NIL}}}$])
▷ If not in ***modules***$^{\text{var}}$, try paths in *path-list* 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*$_{\boxed{\textbf{*package*}^{\text{var}}}}$]])
▷ Intern fresh symbol in package. Deprecated.

(**copy-symbol**$^{\text{Fu}}$ *symbol* [*props*$_{\boxed{\texttt{NIL}}}$])
▷ 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.

$\left(\begin{cases}\textbf{documentation}^{\text{gF}}\\ \textbf{(setf documentation}^{\text{gF}}\textbf{)}\ \textit{new-doc}\end{cases}\right.$ *foo* {**'variable**|**'function**|**'compiler-macro**|**'method-combination**|**'structure**|**'type**|**'setf**|`T`})
▷ 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**.

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

(**subsetp**$^{\text{Fu}}$ *list-a list-b* $\left\{\left|\begin{cases}\textbf{:test}\ \textit{function}_{\boxed{\textbf{eql}}}\\ \textbf{:test-not}\ \textit{function}\end{cases}\right.\right.$ | **:key** *function*})
▷ Return `T` if *list-a* is a subset of *list-b*.

4.2 Lists

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

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

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

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

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

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

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

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

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

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

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

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

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

({**rplaca**$^{\text{Fu}}$ | **rplacd**$^{\text{Fu}}$} $\widetilde{\textit{cons}}$ *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* $\left\{\left|\begin{cases}\textbf{:test}\ \textit{function}_{\boxed{\textbf{eql}}}\\ \textbf{:test-not}\ \textit{function}\end{cases}\right.\right.$ | **:key** *function*})
▷ Return *list* if *foo* is already member of *list*. If not, return (**cons**$^{\text{Fu}}$ *foo list*).

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

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

(**pushnew**$^{\text{M}}$ *foo* $\widetilde{\textit{place}}$ $\left\{\left|\begin{cases}\textbf{:test}\ \textit{function}_{\boxed{\textbf{eql}}}\\ \textbf{:test-not}\ \textit{function}\end{cases}\right.\right.$ | **:key** *function*})
▷ Set *place* to (**adjoin**$^{\text{Fu}}$ *foo place*).

(**append**$^{\text{Fu}}$ [*list** *foo*])
(**nconc**$^{\text{Fu}}$ [$\widetilde{\textit{list}}$* *foo*])
▷ Return concatenated list. *foo* can be of any type.

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

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

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

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

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

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

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

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

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

4.3 Association Lists

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

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

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

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

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

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

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

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

4.4 Trees

$\left(\overset{\text{Fu}}{\textbf{tree-equal}} \textit{ foo bar } \begin{Bmatrix}\textbf{:test } \textit{test}_{\boxed{\textbf{eql}}}\\ \textbf{:test-not } \textit{test}\end{Bmatrix}\right)$

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

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

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

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

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

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

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

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

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

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

($\overset{\text{Fu}}{\textbf{package-use-list}}$ *package*)
($\overset{\text{Fu}}{\textbf{package-used-by-list}}$ *package*)

▷ List of other packages used by/using *package*.

($\overset{\text{Fu}}{\textbf{delete-package}}$ $\widetilde{\textit{package}}$)

▷ Delete *package*. Return T if successful.

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

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

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

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

($\overset{\text{Fu}}{\textbf{find-package}}$ *name*)

▷ Package object with *name* (case-sensitive).

($\overset{\text{Fu}}{\textbf{find-all-symbols}}$ *name*)

▷ Return list of symbols with *name* from all registered packages.

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{intern}}\\ \overset{\text{Fu}}{\textbf{find-symbol}}\end{Bmatrix} \textit{foo } [\textit{package}_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}]\right)$

▷ Intern or find, respectively, symbol *foo* in *package*. Second return value is one of $\underset{2}{\textbf{:internal}}$, $\underset{2}{\textbf{:external}}$, or $\underset{2}{\textbf{:inherited}}$ (or $\underset{2}{\texttt{NIL}}$ if $\overset{\text{Fu}}{\textbf{intern}}$ created a fresh symbol).

($\overset{\text{Fu}}{\textbf{unintern}}$ *symbol* [*package*$_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$])

▷ Remove *symbol* from *package*, return T on success.

$\left(\begin{Bmatrix}\overset{\text{Fu}}{\textbf{import}}\\ \overset{\text{Fu}}{\textbf{shadowing-import}}\end{Bmatrix} \textit{symbols } [\textit{package}_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}]\right)$

▷ Make *symbols* internal to *package*. Return T. In case of a name conflict signal correctable **package-error** or shadow the old symbol, respectively.

($\overset{\text{Fu}}{\textbf{shadow}}$ *symbols* [*package*$_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$])

▷ Add *symbols* to shadowing list of *package* making equally named inherited symbols shadowed. Return T.

($\overset{\text{Fu}}{\textbf{package-shadowing-symbols}}$ *package*)

▷ List of shadowing symbols of *package*.

($\overset{\text{Fu}}{\textbf{export}}$ *symbols* [*package*$_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$])

▷ Make *symbols* external to *package*. Return T.

($\overset{\text{Fu}}{\textbf{unexport}}$ *symbols* [*package*$_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$])

▷ Revert *symbols* to internal status. Return T.

$\left(\left\{\begin{array}{l}\begin{Bmatrix}\overset{\text{M}}{\textbf{do-symbols}}\\ \overset{\text{M}}{\textbf{do-external-symbols}}\end{Bmatrix} (\widehat{\textit{var}}\ [\textit{package}_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}\ [\textit{result}_{\boxed{\texttt{NIL}}}]])\\ \overset{\text{M}}{\textbf{do-all-symbols}}\ (\textit{var}\ [\textit{result}_{\boxed{\texttt{NIL}}}])\end{array}\right\}\right.$

$(\textbf{declare } \widehat{\textit{decl}^{*}})^{*} \left\{\left|\begin{array}{l}\textit{tag}\\ \textit{form}\end{array}\right.\right\}^{*}\Big)$

▷ Evaluate $\overset{\text{sO}}{\textbf{tagbody}}$-like body with *var* successively bound to every symbol from *package*, to every external symbol from *package*, or to every symbol from all registered packages, respectively. Return values of *result*. Implicitly, the whole form is a $\overset{\text{sO}}{\textbf{block}}$ named NIL.

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

▷ 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.

(**check-type** *place type* [*string*])
▷ Return NIL and signal correctable **type-error** if *place* is not of *type*.

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

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

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

($^{\text{M}}$**deftype** *foo* ($\mathit{macro}\text{-}\lambda^*$) (**declare** $\widehat{\mathit{decl}^*}$)* [$\widehat{\mathit{doc}}$] $\mathit{form}^{\text{P}*}$)
▷ Define type *foo* which when referenced as (*foo* $\widehat{\mathit{arg}}^*$) applies expanded *forms* to *args* returning the new type. For ($\mathit{macro}\text{-}\lambda^*$) see p. 19 but with default value of * instead of NIL. *forms* are enclosed in an implicit $^{\text{sO}}$**block** *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** $\mathit{type}^*_{\boxed{\texttt{T}}}$) ▷ Type specifier for intersection of *type*s.

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

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

14 Packages and Symbols

14.1 Predicates

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

14.2 Packages

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

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

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

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

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

($^{\text{Fu}}$**rename-package** *package new-name* [$\mathit{new\text{-}nicknames}_{\boxed{\texttt{NIL}}}$])
▷ Rename *package*. Return renamed package.

($^{\text{M}}$**in-package** $\widehat{\mathit{foo}}$) ▷ Make package *foo* current.

4.5 Sets

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

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

($^{\text{Fu}}$**adjustable-array-p** *array*)
($^{\text{Fu}}$**array-has-fill-pointer-p** *array*)
▷ Return T if *array* is adjustable/has a fill pointer, respectively.

($^{\text{Fu}}$**array-in-bounds-p** *array* [*subscripts*])
▷ Return T if *subscripts* are in *array*'s bounds.

5.2 Array Functions

$$\left(\left\{\begin{array}{l}\overset{\text{Fu}}{\textbf{make-array}}\ \mathit{dimensions}\ [\textbf{:adjustable}\ \mathit{bool}_{\boxed{\texttt{NIL}}}]\\ \overset{\text{Fu}}{\textbf{adjust-array}}\ \widetilde{\mathit{array}}\ \mathit{dimensions}\end{array}\right\}\right.$$
$$\left.\left\{\begin{array}{|l}\textbf{:element-type}\ \mathit{type}_{\boxed{\texttt{T}}}\\ \textbf{:fill-pointer}\ \{\mathit{num}|\mathit{bool}\}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{|l}\textbf{:initial-element}\ \mathit{obj}\\ \textbf{:initial-contents}\ \mathit{sequence}\\ \textbf{:displaced-to}\ \mathit{array}_{\boxed{\texttt{NIL}}}\ [\textbf{:displaced-index-offset}\ i_{\boxed{0}}]\end{array}\right.\end{array}\right\}\right)$$
▷ Return fresh, or readjust, respectively, vector or array of *dimensions*.

($^{\text{Fu}}$**aref** *array* [*subscripts*])
▷ Return array element pointed to by *subscripts*. **setf**able.

($^{\text{Fu}}$**row-major-aref** *array i*)
▷ Return *i*th element of *array* in row-major order. **setf**able.

($^{\text{Fu}}$**array-row-major-index** *array* [*subscripts*])
▷ Index in row-major order of the element denoted by *subscripts*.

($^{\text{Fu}}$**array-dimensions** *array*)
▷ List containing the lengths of *array*'s dimensions.

($^{\text{Fu}}$**array-dimension** *array i*)
▷ Length of *i*th dimension of *array*.

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

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

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

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

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

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

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

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

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 *i*th element of *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.

6 Sequences

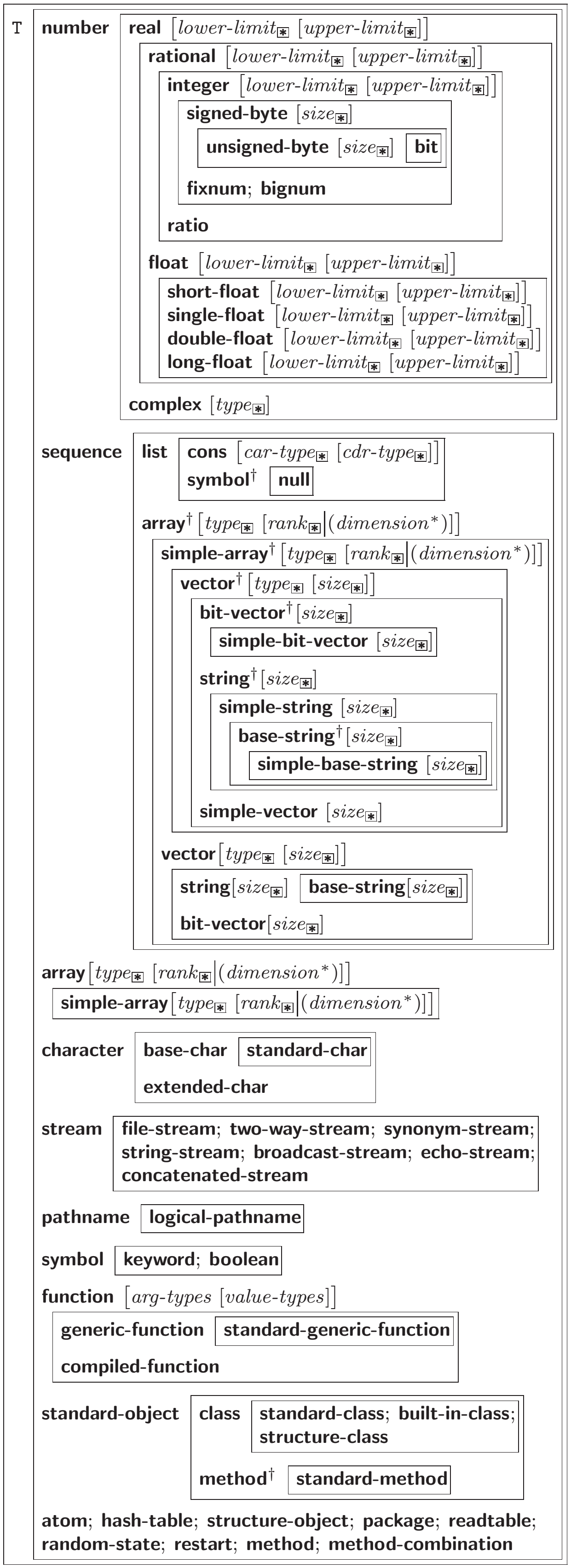
6.1 Sequence Predicates

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

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

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

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


T
number
real [lower-limit* [upper-limit*]]
rational [lower-limit* [upper-limit*]]
integer [lower-limit* [upper-limit*]]
signed-byte [size*]
unsigned-byte [size*]
bit
fixnum; bignum
ratio
float [lower-limit* [upper-limit*]]
short-float [lower-limit* [upper-limit*]]
single-float [lower-limit* [upper-limit*]]
double-float [lower-limit* [upper-limit*]]
long-float [lower-limit* [upper-limit*]]
complex [type*]
sequence
list
cons [car-type* [cdr-type*]]
symbol†
null
array† [type* [rank*|(dimension*)]]
simple-array† [type* [rank*|(dimension*)]]
vector† [type* [size*]]
bit-vector† [size*]
simple-bit-vector [size*]
string† [size*]
simple-string [size*]
base-string† [size*]
simple-base-string [size*]
simple-vector [size*]
vector [type* [size*]]
string[size*]
base-string[size*]
bit-vector[size*]
array [type* [rank*|(dimension*)]]
simple-array [type* [rank*|(dimension*)]]
character
base-char
standard-char
extended-char
stream
file-stream; two-way-stream; synonym-stream; string-stream; broadcast-stream; echo-stream; concatenated-stream
pathname
logical-pathname
symbol
keyword; boolean
function [arg-types [value-types]]
generic-function
standard-generic-function
compiled-function
standard-object
class
standard-class; built-in-class; structure-class
method†
standard-method
atom; hash-table; structure-object; package; readtable; random-state; restart; method; method-combination

†For supertypes of this type look for the instance without a †.

As a type argument, ∗ means no restriction.

Figure 3: Data Types.

($\overset{\text{Fu}}{\textbf{translate-logical-pathname}}$ *path*)
 ▷ Physical $\underline{\text{pathname}}$ of *path*.

($\overset{\text{Fu}}{\textbf{probe-file}}$ *file*)
($\overset{\text{Fu}}{\textbf{truename}}$ *file*)
 ▷ $\underline{\text{Canonical name}}$ of *file*. If *file* does not exist, return $\underline{\texttt{NIL}}$/signal **file-error**, respectively.

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

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

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

$$(\overset{\text{Fu}}{\textbf{file-position}}\ \textit{stream}\ [\left\{\begin{array}{l}\textbf{:start}\\ \textbf{:end}\\ \textit{position}\end{array}\right\}])$$
 ▷ Return $\underline{\text{position within stream}}$, or set it to $\underline{\textit{position}}$ and return $\underline{\texttt{T}}$ on success.

($\overset{\text{Fu}}{\textbf{file-string-length}}$ *stream foo*)
 ▷ $\underline{\text{Length}}$ *foo* would have in *stream*.

($\overset{\text{Fu}}{\textbf{rename-file}}$ *foo bar*)
 ▷ Rename file *foo* to *bar*. Unspecified parts of path *bar* default to those of *foo*. Return $\underline{\text{new pathname}}$, $\underline{\text{old file name}}_2$, and $\underline{\text{new file name}}_3$.

($\overset{\text{Fu}}{\textbf{delete-file}}$ *file*) ▷ Delete *file*, return $\underline{\texttt{T}}$.

($\overset{\text{Fu}}{\textbf{directory}}$ *path*) ▷ Return $\underline{\text{list of pathnames}}$.

($\overset{\text{Fu}}{\textbf{ensure-directories-exist}}$ *path* [**:verbose** *bool*])
 ▷ Create parts of $\underline{\textit{path}}$ if necessary. Second return value is $\underline{\texttt{T}}_2$ if something has been created.

($\overset{\text{M}}{\textbf{with-open-file}}$ (*stream path open-arg**) (**declare** $\widehat{decl}$*)* $form^{\text{P}*}$)
 ▷ Use $\overset{\text{Fu}}{\textbf{open}}$ with *open-arg*s to temporarily create *stream* to *path*; return $\underline{\text{values of } \textit{forms}}$.

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

13 Types and Classes

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

($\overset{\text{Fu}}{\textbf{typep}}$ *foo type* [$environment_{\boxed{\texttt{NIL}}}$])
 ▷ Return $\underline{\texttt{T}}$ if *foo* is of *type*.

($\overset{\text{Fu}}{\textbf{subtypep}}$ *type-a type-b* [*environment*])
 ▷ Return $\underline{\texttt{T}}$ if *type-a* is a recognizable subtype of *type-b*, and $\underline{\texttt{NIL}}_2$ if the relationship could not be determined.

($\overset{\text{sO}}{\textbf{the}}$ $\widehat{type}$ *form*)
 ▷ Return $\underline{\text{values}}$ of *form* which are declared to be of *type*.

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

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

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

($\overset{\text{Fu}}{\textbf{type-of}}$ *foo*) ▷ $\underline{\text{Type of } \textit{foo}}$.

$$(\overset{\text{Fu}}{\textbf{mismatch}}\ \textit{sequence-a}\ \textit{sequence-b}\ \left\{\begin{array}{l}\textbf{:from-end}\ \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\textbf{eql}}}\\ \textbf{:test-not}\ \textit{function}\end{array}\right.\\ \textbf{:start1}\ \textit{start-a}_{\boxed{0}}\\ \textbf{:start2}\ \textit{start-b}_{\boxed{0}}\\ \textbf{:end1}\ \textit{end-a}_{\boxed{\texttt{NIL}}}\\ \textbf{:end2}\ \textit{end-b}_{\boxed{\texttt{NIL}}}\\ \textbf{:key}\ \textit{function}\end{array}\right\})$$
 ▷ Return $\underline{\text{position in } \textit{sequence-a}}$ where *sequence-a* and *sequence-b* begin to mismatch. Return $\underline{\texttt{NIL}}$ if they match entirely.

6.2 Sequence Functions

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

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

($\overset{\text{Fu}}{\textbf{merge}}$ *type* $\widetilde{sequence}$*-a* $\widetilde{sequence}$*-b test* [**:key** $function_{\boxed{\texttt{NIL}}}$])
 ▷ Return $\underline{\text{interleaved sequence}}$ of *type*. Merged sequence will be sorted if both *sequence-a* and *sequence-b* are sorted.

$$(\overset{\text{Fu}}{\textbf{fill}}\ \widetilde{sequence}\ \textit{foo}\ \left\{\begin{array}{l}\textbf{:start}\ \textit{start}_{\boxed{0}}\\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}\end{array}\right\})$$
 ▷ Return $\underline{\textit{sequence}}$ after setting elements between *start* and *end* to *foo*.

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

$$(\overset{\text{Fu}}{\textbf{count}}\ \textit{foo}\ \textit{sequence}\ \left\{\begin{array}{l}\textbf{:from-end}\ \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test}\ \textit{function}_{\boxed{\textbf{eql}}}\\ \textbf{:test-not}\ \textit{function}\end{array}\right.\\ \textbf{:start}\ \textit{start}_{\boxed{0}}\\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key}\ \textit{function}\end{array}\right\})$$
 ▷ Return $\underline{\text{number of } \textit{foos}}$ in *sequence* which satisfy tests.

$$(\left\{\begin{array}{l}\overset{\text{Fu}}{\textbf{count-if}}\\ \overset{\text{Fu}}{\textbf{count-if-not}}\end{array}\right\}\ \textit{test}\ \textit{sequence}\ \left\{\begin{array}{l}\textbf{:from-end}\ \textit{bool}_{\boxed{\texttt{NIL}}}\\ \textbf{:start}\ \textit{start}_{\boxed{0}}\\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key}\ \textit{function}\end{array}\right\})$$
 ▷ Return $\underline{\text{number of elements}}$ in *sequence* which satisfy *test*.

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

($\overset{\text{Fu}}{\textbf{subseq}}$ *sequence start* [$end_{\boxed{\texttt{NIL}}}$])
 ▷ Return $\underline{\text{subsequence of } \textit{sequence}}$ between *start* and *end*. **setf**able.

$$(\left\{\begin{array}{l}\overset{\text{Fu}}{\textbf{sort}}\\ \overset{\text{Fu}}{\textbf{stable-sort}}\end{array}\right\}\ \widetilde{sequence}\ \textit{test}\ [\textbf{:key}\ \textit{function}])$$
 ▷ Return $\underline{\textit{sequence} \text{ sorted}}$. Order of elements considered equal is not guaranteed/retained, respectively.

($\overset{\text{Fu}}{\textbf{reverse}}$ *sequence*)
($\overset{\text{Fu}}{\textbf{nreverse}}$ $\widetilde{sequence}$) ▷ Return $\underline{\textit{sequence} \text{ in reverse order}}$.

$$(\left\{\begin{array}{l}\overset{\text{Fu}}{\textbf{find}}\\ \overset{\text{Fu}}{\textbf{position}}\end{array}\right\}\ \textit{foo}\ \textit{sequence}\ \left\{\begin{array}{l}\textbf{:from-end}\ \textit{bool}_{\boxed{\texttt{NIL}}}\\ \left\{\begin{array}{l}\textbf{:test}\ \textit{test}_{\boxed{\textbf{eql}}}\\ \textbf{:test-not}\ \textit{test}\end{array}\right.\\ \textbf{:start}\ \textit{start}_{\boxed{0}}\\ \textbf{:end}\ \textit{end}_{\boxed{\texttt{NIL}}}\\ \textbf{:key}\ \textit{function}\end{array}\right\})$$
 ▷ Return $\underline{\text{first element}}$ in *sequence* which satisfies *test*, or its $\underline{\text{position}}$ relative to the begin of *sequence*, respectively.

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

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

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

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

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

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

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

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

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

▷ Make copy of *sequence* without duplicates.

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

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

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

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

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

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

($\overset{\text{Fu}}{\textbf{map}}$ *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`.

$\overset{\text{var}}{\textbf{*standard-input*}}$
$\overset{\text{var}}{\textbf{*standard-output*}}$
$\overset{\text{var}}{\textbf{*error-output*}}$

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

$\overset{\text{var}}{\textbf{*debug-io*}}$
$\overset{\text{var}}{\textbf{*query-io*}}$

▷ Bidirectional streams for debugging and user interaction.

12.7 Files

$$\left(\overset{\text{Fu}}{\textbf{make-pathname}}\left\{\begin{array}{|l}\textbf{:host }\textit{host}\\\textbf{:device }\textit{dev}\\\textbf{:directory }\textit{dir}\\\textbf{:name }\textit{name}\\\textbf{:type }\textit{type}\\\textbf{:version }\textit{ver}\\\textbf{:defaults }\textit{path}\\\textbf{:case }\{\textbf{:local}|\textbf{:common}\}_{\boxed{\textbf{:local}}}\end{array}\right\}\right)$$

▷ Construct pathname.

($\overset{\text{Fu}}{\textbf{merge-pathnames}}$ *pathname* $\left[\textit{default-pathname}_{\boxed{\overset{\text{var}}{\textbf{*default-pathname-defaults*}}}}\left[\textit{default-version}_{\boxed{\textbf{:newest}}}\right]\right]$)

▷ Return *pathname* after filling in missing parts from defaults.

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

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

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

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

▷ 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*.

$$\left(\overset{\text{Fu}}{\textbf{parse-namestring}}\ \textit{foo}\ \left[\textit{host}\ \left[\textit{default-pathname}_{\boxed{\overset{\text{var}}{\textbf{*default-pathname-defaults*}}}}\ \left[\left\{\begin{array}{|l}\textbf{:start }\textit{start}_{\boxed{0}}\\\textbf{:end }\textit{end}_{\boxed{\texttt{NIL}}}\\\textbf{:junk-allowed }\textit{bool}_{\boxed{\texttt{NIL}}}\end{array}\right\}\right]\right]\right]\right)$$

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

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

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

▷ Return pathname component.

($\overset{\text{Fu}}{\textbf{logical-pathname}}$ *path*) ▷ Logical name of *path*.

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

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

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

▷ *host*'s list of translations. **setf**able.

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

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

(**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 stream of indicated type.

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

▷ Return a **string-stream** supplying the characters from *string*.

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

▷ Return a **string-stream** 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 list of streams *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 source stream or sink stream of *two-way-stream*/ *echo-stream*, respectively.

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

▷ Return symbol of *synonym-stream*.

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

▷ Clear and return as a string characters on *string-stream*.

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

▷ T if there is a character in input *stream*.

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

▷ Clear input from *stream*, return NIL.

({**clear-output**$^{\text{Fu}}$ | **force-output**$^{\text{Fu}}$ | **finish-output**$^{\text{Fu}}$} [$\widetilde{\textit{stream}}$$_{\boxed{\text{*standard-output*}^{\text{var}}}}$])

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

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

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

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

▷ Evaluate *forms* with *foo* locally bound to *stream*. Return values of *forms*.

(**with-input-from-string**$^{\text{M}}$ (*foo string* {| **:index** $\widetilde{\textit{index}}$ | **:start** *start*$_{\boxed{0}}$ | **:end** *end*$_{\boxed{\text{NIL}}}$}) (**declare** $\widehat{\textit{decl}}$*)* *form*$^{\text{P}}$*)

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

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

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

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

▷ External file format designator.

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

(**map-into**$^{\text{Fu}}$ $\widetilde{\textit{result-sequence}}$ *function sequence**)

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

(**reduce**$^{\text{Fu}}$ *function sequence* {| **:initial-value** *foo*$_{\boxed{\text{NIL}}}$ | **:from-end** *bool*$_{\boxed{\text{NIL}}}$ | **:start** *start*$_{\boxed{0}}$ | **:end** *end*$_{\boxed{\text{NIL}}}$ | **:key** *function*})

▷ 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.

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

▷ Return 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 10 and 17.

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

(**make-hash-table**$^{\text{Fu}}$ {| **:test** {**eq**$^{\text{Fu}}$|**eql**$^{\text{Fu}}$|**equal**$^{\text{Fu}}$|**equalp**$^{\text{Fu}}$}$_{\boxed{\text{eql}}}$ | **:size** *int* | **:rehash-size** *num* | **:rehash-threshold** *num*})

▷ Make a hash table.

(**gethash**$^{\text{Fu}}$ *key hash-table* [*default*$_{\boxed{\text{NIL}}}$])

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

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

▷ Number of entries in *hash-table*.

(**remhash**$^{\text{Fu}}$ *key* $\widetilde{\textit{hash-table}}$)

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

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

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

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

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

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

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

▷ Test function used in *hash-table*.

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

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

(**sxhash**$^{\text{Fu}}$ *foo*)

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

8 Structures

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

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

▷ Define structure type <u>*foo*</u> together with functions `MAKE-`*foo*, `COPY-`*foo* and (unless **:type** without **:named** is used) *foo*`-P`; and **setf**able accessors *foo-slot*. Instances of type *foo* can be created by (`MAKE-`*foo* {**:***slot value*}*) or, if *ord-λ* (see p. 17) is given, by (*maker arg** {**:***key value*}*). In the latter case, *args* 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.

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

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

9 Control Structure

9.1 Predicates

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

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

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

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

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

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

▷ <u>T</u> if *foo* and *bar* are identical; or are the same **character** ignoring case; or are **number**s of the same value ignoring type; or are equivalent **pathname**s; or are **cons**es or **array**s of the same shape with $\overset{\text{Fu}}{\textbf{equalp}}$ elements; or are structures of the same type with $\overset{\text{Fu}}{\textbf{equalp}}$ elements; or are **hash-table**s of the same size with the same **:test** function, the same keys in terms of **:test** function, and **equalp** elements.

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

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

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

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

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

▷ 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.

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

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

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

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

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

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

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

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

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

▷ The *text*s are format control subclauses the zero-indexed argument (or the *i*th if given) of which is chosen. With **:**, the argument is boolean and takes first *text* for `NIL` and second *text* for `T`. With **@**, the argument is boolean and if `T`, takes the only *text* and remains to be read; no *text* is chosen and the argument is used up if it is `NIL`.

~[**@**]**?**

▷ Process two arguments as $\overset{\text{Fu}}{\textbf{format}}$ string and argument list. With **@**, take one argument as $\overset{\text{Fu}}{\textbf{format}}$ string and use then the rest oft the original arguments.

~[*prefix* {**,** *prefix*}*][**:**][**@**]**/***function***/**

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

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

▷ 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.

12.6 Streams

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

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

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

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

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

▷ 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**}

▷ Print integer argument as number (decimal, binary, octal, or hexadecimal, respectively). With **:** group digits *comma-interval* each; with **@**, always prepend a sign.

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

▷ Print argument as fixed-format floating-point number. With **@**, always prepend a sign.

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

▷ 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.

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

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

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

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

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

▷ Convert to lowercase, convert first letter of each word to uppercase, capitalize first word and convert the rest to lowercase, or convert to uppercase, respectively.

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

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

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

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

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

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

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

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

▷ Act like **pprint-logical-block** using *body* as $^{\text{M}}$**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.

~[**:**][**@**]↩

▷ (Tilde-newline.) Ignore newline and following whitespace. With **:**, ignore only newline; with **@**, ignore only following whitespace.

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

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

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

▷ 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.

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

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

9.2 Variables

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

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

($^{\text{M}}$**defvar** $\widehat{\textit{foo}}$ [*form* [$\widehat{\textit{doc}}$]])

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

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

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

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

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

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

($^{\text{M}}$**multiple-value-setq** *vars form*)

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

($^{\text{M}}$**shiftf** $\widetilde{\textit{place}}^{+}$ *foo*)

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

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

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

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

($^{\text{Fu}}$**get** *symbol key* [*default*$_{\boxed{\texttt{NIL}}}$])
($^{\text{Fu}}$**getf** *place key* [*default*$_{\boxed{\texttt{NIL}}}$])

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

($^{\text{Fu}}$**get-properties** *property-list keys*)

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

($^{\text{Fu}}$**remprop** $\widetilde{\textit{symbol}}$ *key*)
($^{\text{M}}$**remf** $\widetilde{\textit{place}}$ *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

$$(\textit{var}^{*}\ \left[\textbf{\&optional}\ \left\{\begin{array}{l}\textit{var}\\ (\textit{var}\ [\textit{init}_{\boxed{\texttt{NIL}}}\ [\textit{supplied-p}]])\end{array}\right\}^{*}\right]\ [\textbf{\&rest}\ \textit{var}]$$

$$\left[\textbf{\&key}\ \left\{\begin{array}{l}\textit{var}\\ \left(\left\{\begin{array}{l}\textit{var}\\ (\textit{:key}\ \textit{var})\end{array}\right\}\ [\textit{init}_{\boxed{\texttt{NIL}}}\ [\textit{supplied-p}]]\right)\end{array}\right\}^{*}\ [\textbf{\&allow-other-keys}]\right]$$

$$\left[\textbf{\&aux}\ \left\{\begin{array}{l}\textit{var}\\ (\textit{var}\ [\textit{init}_{\boxed{\texttt{NIL}}}])\end{array}\right\}^{*}\right]).$$

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

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

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

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

▷ Evaluate *forms* with locally defined functions *foo*. 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*.

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

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

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

▷ Return values of *function* called on *args*. Last *arg* must be a list. **setf**able if *function* is one of $\overset{\text{Fu}}{\textbf{aref}}$, $\overset{\text{Fu}}{\textbf{bit}}$, and $\overset{\text{Fu}}{\textbf{sbit}}$.

($\overset{\text{Fu}}{\textbf{funcall}}$ *function* arg*)

▷ Return values of *function* called with *args*.

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

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

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

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

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

($\overset{\text{Fu}}{\textbf{multiple-value-list}}$ *form*)

▷ Return in a list values of *form*.

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

▷ Zero-indexed *n*th return value of *form*.

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

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

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

▷ Return function of any number of arguments returning *foo*.

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

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

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

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

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

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

$\overset{\text{var}}{\textbf{*print-array*}}$ ▷ 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.

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

▷ 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.

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

▷ Width below which a compact pretty-printing style is used.

$\overset{\text{var}}{\textbf{*print-pretty*}}$ ▷ 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.

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

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

▷ Return highest priority *function* associated with type of *foo* and $\underset{2}{\texttt{T}}$ 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 copy of *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.

12.5 Format

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

▷ Return 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 *arg*s. 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 NIL. If first argument is NIL, return formatted output.

~$[min\text{-}col_{\boxed{0}}]$ $\big[,[col\text{-}inc_{\boxed{1}}]$ $\big[,[min\text{-}pad_{\boxed{0}}]$ $\big[,{}'pad\text{-}char_{\boxed{\text{␣}}}]\big]\big]$

[**:**][**@**]{**A**|**S**}

▷ 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.

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

($\overset{\text{Fu}}{\textbf{write-sequence}}$ *sequence* $\widetilde{stream}$ $\left\{\begin{array}{|l}\textbf{:start}\ start_{\boxed{0}}\\ \textbf{:end}\ end_{\boxed{\text{NIL}}}\end{array}\right\}$)
▷ Write elements of $\underline{sequence}$ to *stream*.

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

▷ Print *foo* to *stream* and return $\underline{foo}$, 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.)

($\overset{\text{Fu}}{\textbf{pprint-fill}}$ $\widetilde{stream}$ *foo* $\left[parenthesis_{\boxed{\text{T}}}\ [noop]\right]$)
($\overset{\text{Fu}}{\textbf{pprint-tabular}}$ $\widetilde{stream}$ *foo* $\left[parenthesis_{\boxed{\text{T}}}\ [noop\ [n_{\boxed{16}}]]\right]$)
($\overset{\text{Fu}}{\textbf{pprint-linear}}$$\widetilde{stream}$ *foo* $\left[parenthesis_{\boxed{\text{T}}}\ [noop]\right]$)
▷ Print *foo* to *stream*. If *foo* is a list, print as many elements per line as possible; do the same in a table with a column width of *n* ems; or print either all elements on one line or each on its own line, respectively. Return $\underline{\text{NIL}}$. Usable with $\overset{\text{Fu}}{\textbf{format}}$ directive ~//.

($\overset{\text{M}}{\textbf{pprint-logical-block}}$ ($\widetilde{stream}$ *list* $\left\{\begin{array}{|l}\left\{\begin{array}{l}\textbf{:prefix}\ string\\ \textbf{:per-line-prefix}\ string\end{array}\right\}\\ \textbf{:suffix}\ string_{\boxed{""}}\end{array}\right\}$)
(**declare** $\widehat{decl^*}$)* $form^{\text{P}}{}^*$)
▷ Evaluate *forms*, which should print *list*, with *stream* locally bound to a pretty printing stream which outputs to the original *stream*. If *list* is in fact not a list, it is printed by $\overset{\text{Fu}}{\textbf{write}}$. Return $\underline{\text{NIL}}$.

($\overset{\text{M}}{\textbf{pprint-pop}}$)
▷ Take $\underline{\text{next element}}$ off *list*. If there is no remaining list in *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*.

($\overset{\text{Fu}}{\textbf{pprint-tab}}$ $\left\{\begin{array}{l}\textbf{:line}\\ \textbf{:line-relative}\\ \textbf{:section}\\ \textbf{:section-relative}\end{array}\right\}$ *c i* $[\widetilde{stream}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}]$)
▷ Move cursor forward to column number $c + ki$, $k \geq 0$ being as small as possible.

($\overset{\text{Fu}}{\textbf{pprint-indent}}$ $\left\{\begin{array}{l}\textbf{:block}\\ \textbf{:current}\end{array}\right\}$ *n* $[\widetilde{stream}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}]$)
▷ Specify indentation for innermost logical block relative to leftmost position/to current position. Return $\underline{\text{NIL}}$.

($\overset{\text{M}}{\textbf{pprint-exit-if-list-exhausted}}$)
▷ If *list* is empty, terminate logical block. Return $\underline{\text{NIL}}$ otherwise.

($\overset{\text{Fu}}{\textbf{pprint-newline}}$ $\left\{\begin{array}{l}\textbf{:linear}\\ \textbf{:fill}\\ \textbf{:miser}\\ \textbf{:mandatory}\end{array}\right\}$ $[\widetilde{stream}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}]$)
▷ Print a conditional newline if *stream* is a pretty printing stream. Return $\underline{\text{NIL}}$.

9.4 Macros

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

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

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

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

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

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

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

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

One toplevel [*E*] may be replaced by **&environment** *var* where *var* carries the lexical compilation environment. *supplied-p* is T if there is a corresponding argument.

($\left\{\begin{array}{l}\overset{\text{M}}{\textbf{defmacro}}\\ \overset{\text{Fu}}{\textbf{define-compiler-macro}}\end{array}\right\}$ $\left\{\begin{array}{l}foo\\ (\textbf{setf}\ foo)\end{array}\right\}$ (*macro*-λ^*) (**declare** $\widehat{decl^*}$)*
$[\widehat{doc}]$ $form^{\text{P}}{}^*$)
▷ Define macro $\underline{foo}$ which on evaluation as (*foo tree*) applies expanded *forms* to arguments from *tree* which corresponds to *tree*-shaped *macro*-λs. *forms* are enclosed in an implicit $\overset{\text{sO}}{\textbf{block}}$ *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*-λ^*) (**declare** $\widehat{local\text{-}decl^*}$)* $[\widehat{doc}]$
$macro\text{-}form^{\text{P}}{}^*$)*) (**declare** $\widehat{decl^*}$)* $form^{\text{P}}{}^*$)
▷ Evaluate $\underline{forms}$ with locally defined mutually invisible macros *foo* which are enclosed in implicit $\overset{\text{sO}}{\textbf{block}}$s of the same name.

($\overset{\text{sO}}{\textbf{symbol-macrolet}}$ ((*foo expansion-form*)*) (**declare** $\widehat{decl^*}$)* $form^{\text{P}}{}^*$)
▷ Evaluate $\underline{forms}$ with locally defined symbol macros *foo*.

($\overset{\text{M}}{\textbf{defsetf}}$ $\widehat{function}$ $\left\{\begin{array}{l}\widehat{updater}\ [\widehat{doc}]\\ (setf\text{-}\lambda^*)\ (s\text{-}var^*)\ (\textbf{declare}\ \widehat{decl^*})^*\ [\widehat{doc}]\ form^{\text{P}}{}^*\end{array}\right\}$)
where defsetf lambda list (*setf*-λ^*) has the form
$(var^*\ [\textbf{\&optional}\left\{\begin{array}{l}var\\ (var\ [init_{\boxed{\text{NIL}}}\ [supplied\text{-}p]])\end{array}\right\}^*]\ [\textbf{\&rest}\ var]$
$[\textbf{\&key}\left\{\begin{array}{l}var\\ \left(\left\{\begin{array}{l}var\\ (\textbf{:}key\ var)\end{array}\right\}\ [init_{\boxed{\text{NIL}}}\ [supplied\text{-}p]]\right)\end{array}\right\}^*$
$[\textbf{\&allow-other-keys}]]\ [\textbf{\&environment}\ var])$
▷ Specify how to **setf** a place accessed by $\underline{function}$. Short form: (**setf** (*function arg**) *value-form*) is replaced by (*updater arg* value-form*). 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*-λ^*) (**declare** $\widehat{decl^*}$)* $[\widehat{doc}]$
$form^{\text{P}}{}^*$)

▷ Specify how to **setf** a place accessed by $\underline{\textit{function}}$. On invocation of (**setf** ($\textit{function arg}^*$) *value-form*), $\textit{form}^*$ 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 $\textit{macro-}\lambda^*$ are bound to corresponding *args*. *forms* are enclosed in an implicit $\overset{\text{sO}}{\textbf{block}}$ *function*.

($\overset{\text{Fu}}{\textbf{get-setf-expansion}}$ *place* [$\textit{environment}_{\boxed{\text{NIL}}}$])
▷ Return lists of temporary variables $\underline{\textit{arg-vars}}$ and of corresponding $\underset{2}{\underline{\textit{args}}}$ as given with *place*, list $\underset{3}{\underline{\textit{newval-vars}}}$ with temporary variables corresponding to the new values, and $\underset{4}{\underline{\textit{set-form}}}$ and $\underset{5}{\underline{\textit{get-form}}}$ specifying in terms of *arg-vars* and *newval-vars* how to **setf** and how to read *place*.

($\overset{\text{M}}{\textbf{define-modify-macro}}$ *foo* ([**&optional** $\begin{Bmatrix} var \\ (var\ [init_{\boxed{\text{NIL}}}\ [supplied\text{-}p]]) \end{Bmatrix}^*$]
[**&rest** *var*]) *function* [$\widehat{doc}$])
▷ Define macro $\underline{\textit{foo}}$ able to modify a place. On invocation of ($\textit{foo place arg}^*$), 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.

9.5 Control Flow

($\overset{\text{sO}}{\textbf{if}}$ *test then* [$\textit{else}_{\boxed{\text{NIL}}}$])
▷ Return values of $\underline{\textit{then}}$ if *test* returns `T`; return values of $\underline{\textit{else}}$ otherwise.

($\overset{\text{M}}{\textbf{cond}}$ ($\textit{test then}^{\overset{\text{P}}{*}}_{\underline{\textit{test}}}$)*)
▷ Return the $\underline{\text{values}}$ of the first $\textit{then}^*$ whose *test* returns `T`; return $\underline{\texttt{NIL}}$ if all *test*s return `NIL`.

($\begin{Bmatrix} \overset{\text{M}}{\textbf{when}} \\ \overset{\text{M}}{\textbf{unless}} \end{Bmatrix}$ $\textit{test foo}^{\overset{\text{P}}{*}}$)
▷ 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* ($\widehat{keys}\ foo^{\overset{\text{P}}{*}}$)* [($\begin{Bmatrix} \textbf{otherwise} \\ \texttt{T} \end{Bmatrix}$ $bar^{\overset{\text{P}}{*}}$)$_{\boxed{\text{NIL}}}$])
▷ Return the $\underline{\text{values}}$ of the first $\textit{foo}^*$ one of whose *keys* is **eql** *test*. Return $\underline{\text{values of } \textit{bars}}$ if no element of *keys* matches.

({$\overset{\text{M}}{\textbf{ecase}}$|$\overset{\text{M}}{\textbf{ccase}}$} *test* ($\widehat{keys}\ foo^{\overset{\text{P}}{*}}$)*)
▷ Return the $\underline{\text{values}}$ of the first $\textit{foo}^*$ one of whose *keys* is **eql** *test*. Signal non-correctable/correctable **type-error** and return $\underline{\texttt{NIL}}$ if no element of *keys* matches.

($\overset{\text{M}}{\textbf{and}}$ $\textit{form}^*_{\boxed{\text{T}}}$)
▷ Evaluate *forms* from left to right. Immediately return $\underline{\texttt{NIL}}$ if one *form*'s value is `NIL`. Return $\underline{\text{values of last } \textit{form}}$ otherwise.

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

($\overset{\text{sO}}{\textbf{progn}}$ $\textit{form}^*_{\boxed{\text{NIL}}}$)
▷ Evaluate *forms* sequentially. Return $\underline{\text{values of last } \textit{form}}$.

($\begin{Bmatrix} \overset{\text{M}}{\textbf{prog}} \\ \overset{\text{M}}{\textbf{prog*}} \end{Bmatrix}$ ($\begin{Bmatrix} \left| \begin{matrix} var \\ (var\ [value_{\boxed{\text{NIL}}}]) \end{matrix} \right. \end{Bmatrix}^*$) (**declare** $\widehat{decl}^*$)* $\begin{Bmatrix} \widehat{tag} \\ form \end{Bmatrix}^*$)
▷ Evaluate $\overset{\text{sO}}{\textbf{tagbody}}$-like body with *vars* locally bound (in parallel or sequentially, respectively) to *values*. Return $\underline{\texttt{NIL}}$ or $\underline{\text{explicitly } \overset{\text{M}}{\textbf{return}}\text{ed values}}$. Implicitly, the whole form is a $\overset{\text{sO}}{\textbf{block}}$ named `NIL`.

($\overset{\text{sO}}{\textbf{multiple-value-prog1}}$ $\textit{form-r form}^*$)
($\overset{\text{M}}{\textbf{prog1}}$ $\textit{form-r form}^*$)
($\overset{\text{M}}{\textbf{prog2}}$ $\textit{form-a form-r form}^*$)
▷ Evaluate forms in order. Return $\underline{\text{values/1st value}}$, respectively, of *form-r*.

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

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

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

#P*string* ▷ A pathname.

#:*foo* ▷ Uninterned symbol *foo*.

#.*form* ▷ Read-time value of *form*.

$\overset{\text{var}}{\textbf{*read-eval*}}_{\boxed{\text{T}}}$ ▷ If `NIL`, a **reader-error** is signalled by **#.**.

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

#*int***#** ▷ Object labelled *int*.

#< ▷ 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**} $\textit{feature}^*$), 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.

12.4 Printer

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

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

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

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

($\overset{\text{Fu}}{\textbf{terpri}}$ [$\widetilde{stream}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}$])
▷ Output a newline to *stream*. Return $\underline{\texttt{NIL}}$.

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

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

($\begin{Bmatrix} \overset{\text{Fu}}{\textbf{write-string}} \\ \overset{\text{Fu}}{\textbf{write-line}} \end{Bmatrix}$ *string* [$\widetilde{stream}_{\boxed{\overset{\text{var}}{\textbf{*standard-output*}}}}$ [$\begin{Bmatrix} \left| \begin{matrix} \textbf{:start}\ start_{\boxed{0}} \\ \textbf{:end}\ end_{\boxed{\text{NIL}}} \end{matrix} \right. \end{Bmatrix}$]])
▷ Write $\underline{\textit{string}}$ to *stream* without/with a trailing newline.

($\overset{\text{Fu}}{\textbf{readtable-case}}$ *readtable*)$_{\boxed{\textbf{:upcase}}}$
▷ Case sensitivity attribute (one of **:upcase**, **:downcase**, **:preserve**, **:invert**) of *readtable*. **setf**able.

($\overset{\text{Fu}}{\textbf{copy-readtable}}$ $\left[\mathit{from\text{-}readtable}_{\boxed{\overset{\text{var}}{\textbf{*readtable*}}}}\ [\widetilde{\mathit{to\text{-}readtable}}_{\boxed{\texttt{NIL}}}]\right]$)
▷ Return copy of *from-readtable*.

($\overset{\text{Fu}}{\textbf{set-syntax-from-char}}$ *to-char from-char* $\left[\widetilde{\mathit{to\text{-}readtable}}_{\boxed{\overset{\text{var}}{\textbf{*readtable*}}}}\right.$ $\left.[\mathit{from\text{-}readtable}_{\boxed{\text{standard readtable}}}]\right]$)
▷ Copy syntax of *from-char* to *to-readtable*. Return `T`.

$\overset{\text{var}}{\textbf{*readtable*}}$ ▷ Current readtable.

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

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

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

($\overset{\text{Fu}}{\textbf{set-macro-character}}$ *char function* $\left[\mathit{non\text{-}term\text{-}p}_{\boxed{\texttt{NIL}}}\ [\widetilde{rt}_{\boxed{\overset{\text{var}}{\textbf{*readtable*}}}}]\right]$)
▷ Make *char* a macro character associated with *function*. Return `T`.

($\overset{\text{Fu}}{\textbf{get-macro-character}}$ *char* $[rt_{\boxed{\overset{\text{var}}{\textbf{*readtable*}}}}]$)
▷ Reader macro function associated with *char*, and $\underset{2}{\texttt{T}}$ if *char* is a non-terminating macro character.

($\overset{\text{Fu}}{\textbf{make-dispatch-macro-character}}$ *char* $\left[\mathit{non\text{-}term\text{-}p}_{\boxed{\texttt{NIL}}}\ [rt_{\boxed{\overset{\text{var}}{\textbf{*readtable*}}}}]\right]$)
▷ Make *char* a dispatching macro character. Return `T`.

($\overset{\text{Fu}}{\textbf{set-dispatch-macro-character}}$ *char sub-char function* $[\widetilde{rt}_{\boxed{\overset{\text{var}}{\textbf{*readtable*}}}}]$)
▷ Make *function* a dispatch function of *char* followed by *sub-char*. Return `T`.

($\overset{\text{Fu}}{\textbf{get-dispatch-macro-character}}$ *char sub-char* $[rt_{\boxed{\overset{\text{var}}{\textbf{*readtable*}}}}]$)
▷ Dispatch function associated with *char* followed by *sub-char*.

12.3 Macro Characters and Escapes

#| *multi-line-comment** |#
; *one-line-comment**
▷ Comments. There are 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*	▷ Regarding line on which it appears.

(▷ Initiate reading of a list.

" ▷ Begin and end of a string.

'*foo* ▷ ($\overset{\text{sO}}{\textbf{quote}}$ *foo*); *foo* unevaluated

`([*foo*] [**,***bar*] [**,@***baz*] [**,.**$\widehat{quux}$] [*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; **#O**; **#X**; **#***n***R** ▷ Number of radix 2, 8, 16, or *n*.

#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.

($\overset{\text{sO}}{\textbf{progv}}$ *symbols values* $\mathit{form}^{\overset{\text{P}}{*}}$)
▷ Evaluate *forms* with *symbols* dynamically bound to *values* or `NIL`. Return values of *forms*.

($\overset{\text{sO}}{\textbf{unwind-protect}}$ *protected cleanup**)
▷ Evaluate *protected* and then, no matter how control leaves *protected*, *cleanups*. Return values of *protected*.

($\overset{\text{M}}{\textbf{destructuring-bind}}$ *destruct-λ bar* (**declare** $\widehat{decl}$*)* $\mathit{form}^{\overset{\text{P}}{*}}$)
▷ Evaluate *forms* with variables from tree *destruct-λ* bound to corresponding elements of tree *bar*, and return their values. *destruct-λ* resembles *macro-λ* (section 9.4), but without any **&environment** clause.

($\overset{\text{M}}{\textbf{multiple-value-bind}}$ ($\widehat{var}$*) *values-form* (**declare** $\widehat{decl}$*)* $\mathit{body\text{-}form}^{\overset{\text{P}}{*}}$)
▷ Evaluate *body-forms* with *vars* lexically bound to the return values of *values-form*. Return values of *body-forms*.

$$\left(\begin{Bmatrix}\overset{\text{sO}}{\textbf{let}}\\ \overset{\text{sO}}{\textbf{let*}}\end{Bmatrix}\left(\begin{Bmatrix}\left|\mathit{name}\right.\\ \left|(\mathit{name}\ [\mathit{value}_{\boxed{\texttt{NIL}}}])\right.\end{Bmatrix}^{*}\right)\ (\textbf{declare}\ \widehat{decl}^{*})^{*}\ \mathit{form}^{\overset{\text{P}}{*}}\right)$$
▷ Evaluate *forms* with *names* lexically bound (in parallel or sequentially, respectively) to *values*. Return values of *forms*.

($\overset{\text{sO}}{\textbf{locally}}$ (**declare** $\widehat{decl}$*)* $\mathit{form}^{\overset{\text{P}}{*}}$)
▷ Evaluate *forms* in a lexical environment with declarations *decl* in effect. Return values of *forms*.

($\overset{\text{sO}}{\textbf{block}}$ *name* $\mathit{form}^{\overset{\text{P}}{*}}$)
▷ Evaluate *forms* with lexical scope and dynamic extent, and return their values unless interrupted by $\overset{\text{sO}}{\textbf{return-from}}$.

($\overset{\text{sO}}{\textbf{return-from}}$ *foo* $[\mathit{result}_{\boxed{\texttt{NIL}}}]$)
($\overset{\text{M}}{\textbf{return}}$ $[\mathit{result}_{\boxed{\texttt{NIL}}}]$)
▷ Have nearest enclosing $\overset{\text{sO}}{\textbf{block}}$ named *foo*/named `NIL`, respectively, return with values of *result*.

($\overset{\text{sO}}{\textbf{tagbody}}$ $\{\widehat{tag}|\mathit{form}\}^{*}$)
▷ Evaluate *forms*. *tags* (symbols or integers) have lexical scope and dynamic extent, and are targets for $\overset{\text{sO}}{\textbf{go}}$. Return `NIL`.

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

($\overset{\text{sO}}{\textbf{catch}}$ *tag* $\mathit{form}^{\overset{\text{P}}{*}}$)
▷ Evaluate *forms* and return their values unless interrupted by $\overset{\text{sO}}{\textbf{throw}}$.

($\overset{\text{sO}}{\textbf{throw}}$ *tag 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

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

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

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

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

9.7 Loop Facility

($^{\text{M}}$**loop** *form**)

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

($^{\text{M}}$**loop** *form**)

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

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

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

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

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

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

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

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

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

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

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

{**upfrom**|**from**|**downfrom**} *start*

▷ Start stepping with *start*

{**upto**|**downto**|**to**|**below**|**above**} *form*

▷ Specify *form* as the end value for stepping.

{**in**|**on**} *list*

▷ Bind *var* to successive elements/tails, respectively, of *list*.

by {$step_{\boxed{1}}$|$function_{\boxed{\overset{\text{Fu}}{\textbf{cdr}}}}$}

▷ 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* in the first iteration to *foo* and later to *bar*.

across *vector*

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

being {**the**|**each**}

▷ Iterate over a hash table or a package.

{**hash-key**|**hash-keys**} {**of**|**in**} *hash-table* [**using** (**hash-value** *value*)]

▷ Bind *var* successively to the keys of *hash-table*; bind *value* to corresponding values.

{**hash-value**|**hash-values**} {**of**|**in**} *hash-table* [**using** (**hash-key** *key*)]

▷ Bind *var* successively to the values of *hash-table*; bind *key* to corresponding keys.

{**symbol**|**symbols**|**present-symbol**|**present-symbols**| **external-symbol**|**external-symbols**} [{**of**|**in**} $package_{\boxed{\overset{\text{var}}{\textbf{*package*}}}}$]

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

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

▷ Evaluate *forms* in every iteration.

it

▷ Value of *test* form of an enclosing **if**, **when**, or **unless** clause.

($^{\text{Fu}}$**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.)

12.2 Reader

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

▷ Ask user a question and return T or NIL depending on their answer. See p. 35, $^{\text{Fu}}$**format**, for *control* and *args*.

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

▷ Evaluate *forms* with standard behaviour of reader and printer. Return values of *forms*.

($\left\{\begin{matrix}\overset{\text{Fu}}{\textbf{read}}\\\overset{\text{Fu}}{\textbf{read-preserving-whitespace}}\end{matrix}\right\}$ [$\widetilde{stream}_{\boxed{\overset{\text{var}}{\textbf{*standard-input*}}}}$ [$eof\text{-}err_{\boxed{\texttt{T}}}$ [$eof\text{-}val_{\boxed{\texttt{NIL}}}$ [$recursive_{\boxed{\texttt{NIL}}}$]]]])

▷ Read printed representation of object.

($^{\text{Fu}}$**read-from-string** *string* [$eof\text{-}error_{\boxed{\texttt{T}}}$ [$eof\text{-}val_{\boxed{\texttt{NIL}}}$ [$\left\{\begin{matrix}|\textbf{:start}\ start_{\boxed{0}}\\|\textbf{:end}\ end_{\boxed{\texttt{NIL}}}\\|\textbf{:preserve-whitespace}\ bool_{\boxed{\texttt{NIL}}}\end{matrix}\right\}$]]])

▷ Return object read from string and zero-indexed $\underset{2}{\text{position}}$ of next character.

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

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

($^{\text{Fu}}$**read-char** [$\widetilde{stream}_{\boxed{\overset{\text{var}}{\textbf{*standard-input*}}}}$ [$eof\text{-}err_{\boxed{\texttt{T}}}$ [$eof\text{-}val_{\boxed{\texttt{NIL}}}$ [$recursive_{\boxed{\texttt{NIL}}}$]]]])

▷ Return next character from *stream*.

($^{\text{Fu}}$**read-char-no-hang** [$\widetilde{stream}_{\boxed{\overset{\text{var}}{\textbf{*standard-input*}}}}$ [$eof\text{-}error_{\boxed{\texttt{T}}}$ [$eof\text{-}val_{\boxed{\texttt{NIL}}}$ [$recursive_{\boxed{\texttt{NIL}}}$]]]])

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

($^{\text{Fu}}$**peek-char** [$mode_{\boxed{\texttt{NIL}}}$ [$\widetilde{stream}_{\boxed{\overset{\text{var}}{\textbf{*standard-input*}}}}$ [$eof\text{-}error_{\boxed{\texttt{T}}}$ [$eof\text{-}val_{\boxed{\texttt{NIL}}}$ [$recursive_{\boxed{\texttt{NIL}}}$]]]]])

▷ 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* [$\widetilde{stream}_{\boxed{\overset{\text{var}}{\textbf{*standard-input*}}}}$])

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

($^{\text{Fu}}$**read-byte** $\widetilde{stream}$ [$eof\text{-}err_{\boxed{\texttt{T}}}$ [$eof\text{-}val_{\boxed{\texttt{NIL}}}$]])

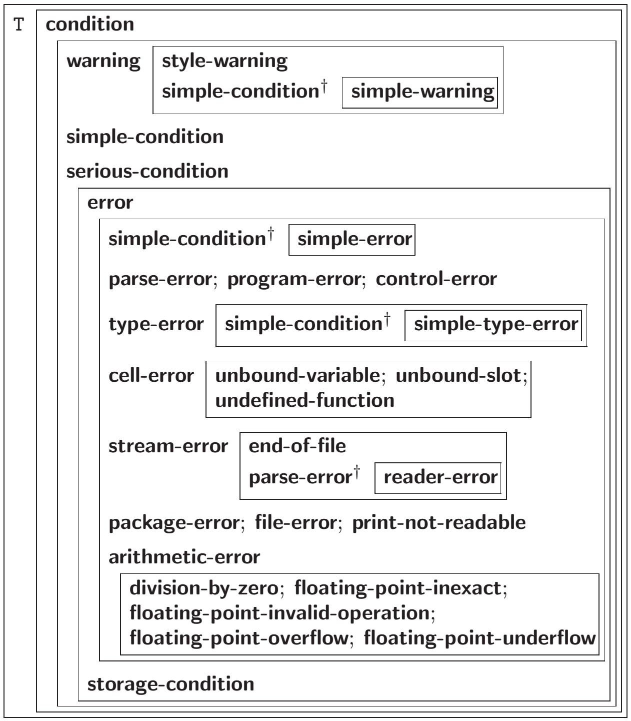
▷ Read next byte from binary *stream*.

($^{\text{Fu}}$**read-line** [$\widetilde{stream}_{\boxed{\overset{\text{var}}{\textbf{*standard-input*}}}}$ [$eof\text{-}err_{\boxed{\texttt{T}}}$ [$eof\text{-}val_{\boxed{\texttt{NIL}}}$ [$recursive_{\boxed{\texttt{NIL}}}$]]]])

▷ 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** $end_{\boxed{\texttt{NIL}}}$])

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



†For supertypes of this type look for the instance without a †.

Figure 2: Condition Types.

($^{\text{Fu}}$**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 Input/Output

12.1 Predicates

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

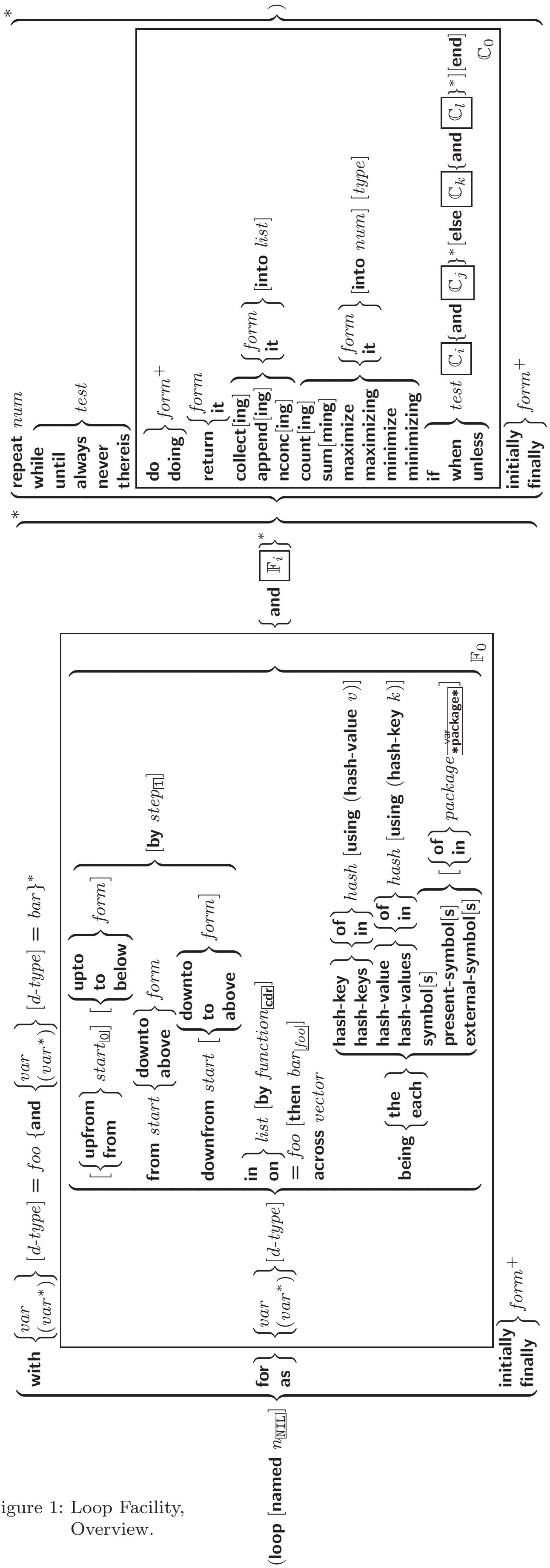

(loop [named $n_{\boxed{\text{NIL}}}$]
with {var | (var*)} [d-type] = foo {and {var | (var*)} [d-type] = bar}*
for | as {var | (var*)} [d-type]
[{upfrom | from} start$_{\boxed{0}}$] [{upto | to | below} form]
from start {downto | above} form
downfrom start [{downto | to | above} form]
[by step$_{\boxed{1}}$]
in | on list [by function$_{\boxed{\text{cdr}}}$]
= foo [then bar$_{\boxed{foo}}$]
across vector
being {the | each} {hash-key | hash-keys} {of | in} hash [using (hash-value v)]
{hash-value | hash-values} {of | in} hash [using (hash-key k)]
{symbol[s] | present-symbol[s] | external-symbol[s]} [{of | in} package$_{\boxed{\text{*package*}}}$]
$\mathbb{F}_0$
{and $\boxed{\mathbb{F}_i}$}*
repeat num
while | until | always | never | thereis test
do | doing form$^+$
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]
if | when | unless test $\boxed{\mathbb{C}_i}$ {and $\boxed{\mathbb{C}_j}$}* [else $\boxed{\mathbb{C}_k}$ {and $\boxed{\mathbb{C}_l}$}*] [end]
$\mathbb{C}_0$
initially | finally form$^+$
)*

Figure 1: Loop Facility, Overview.

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.

{**if**|**when**|**unless**} *test i-form* {**and** *j-form*}* [**else** *k-form* {**and** *l-form*}*] [**end**]
▷ If *test* returns `T`, `T`, or `NIL`, respectively, evaluate *i-form* and *j-forms*; otherwise, evaluate *k-form* and *l-forms*. Inside *i-form* and *k-form*, the value of *test* is accessible by **it**.

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`.

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

10 CLOS

10.1 Classes

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

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

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

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

($\overset{\text{M}}{\textbf{handler-case}}$ *test* (*type* ([*var*]) (**declare** $\widehat{decl}$*)* *condition-form*$^{\text{P}*}$)*
[(**:no-error** (*ord-λ**) (**declare** $\widehat{decl}$*)* *form*$^{\text{P}*}$)])
▷ If, on evaluation of *test*, a condition of *type* is signalled, evaluate matching *condition-forms* with *var* bound to the condition and return $\underline{\text{their values}}$. Without a condition, bind *ord-λ*s to values of *test* and return $\underline{\text{values of } \textit{forms}}$ or, without a **:no-error** clause, return $\underline{\text{values of } \textit{test}}$. See p. 17 for (*ord-λ**).

($\overset{\text{M}}{\textbf{handler-bind}}$ ((*condition-type handler-function*)*) *form*$^{\text{P}*}$)
▷ Return $\underline{\text{values of } \textit{forms}}$ after evaluating them with *condition-types* dynamically bound to their respective *handler-functions* of argument condition.

($\overset{\text{M}}{\textbf{with-simple-restart}}$ (*restart control arg**) *form*$^{\text{P}*}$)
▷ Return $\underline{\text{values of } \textit{forms}}$ unless *restart* is called during their evaluation. In this case, describe restart using $\overset{\text{Fu}}{\textbf{format}}$ *control* and *args* (see p. 35) and return $\underline{\texttt{NIL}}$ and $\underset{2}{\underline{\texttt{T}}}$.

$$(\overset{\text{M}}{\textbf{restart-case}}\ form\ (foo\ (ord\text{-}\lambda^*) \left\{\begin{array}{|l} \textbf{:interactive}\ arg\text{-}function \\ \textbf{:report} \left\{\begin{array}{l} report\text{-}function \\ string_{\boxed{\texttt{"foo"}}} \end{array}\right\} \\ \textbf{:test}\ test\text{-}function_{\boxed{\texttt{T}}} \end{array}\right\}$$

(**declare** $\widehat{decl}$*)* *restart-form*$^{\text{P}*}$)*)
▷ Evaluate *form* with dynamically established restarts *foo*. Return $\underline{\text{values of } \textit{form}}$ or, if by (**invoke-restarts** *foo arg**) one restart *foo* is called, use *string* or *report-function* (of a stream) to print a description of restart *foo* and return the $\underline{\text{values of its } \textit{restart-forms}}$. *arg-function* supplies appropriate *args* if *foo* is called by **invoke-restart-interactively**. If (*test-function condition*) returns `T`, *foo* is made visible under *condition*. For (*ord-λ**) see p. 17.

($\overset{\text{M}}{\textbf{restart-bind}}$ ((*restart restart-function*

$$\left\{\begin{array}{|l} \textbf{:interactive-function}\ function \\ \textbf{:report-function}\ function \\ \textbf{:test-function}\ function \end{array}\right\})^*)\ form^{\text{P}*})$$

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

$$(\left\{\begin{array}{l} \overset{\text{Fu}}{\textbf{compute-restarts}} \\ \overset{\text{Fu}}{\textbf{find-restart}}\ name \end{array}\right\} [condition])$$

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

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

$$(\left\{\begin{array}{l} \overset{\text{Fu}}{\textbf{abort}} \\ \overset{\text{Fu}}{\textbf{muffle-warning}} \\ \overset{\text{Fu}}{\textbf{continue}} \\ \overset{\text{Fu}}{\textbf{store-value}}\ value \\ \overset{\text{Fu}}{\textbf{use-value}}\ value \end{array}\right\} [condition_{\boxed{\texttt{NIL}}}])$$

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

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

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

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

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

11 Conditions and Errors

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

▷ Define, as a subtype of *parent-types*, condition type *foo*. In new conditions, a *slot*'s value defaults to *form* unless set via **:***initarg-name*, and is accessible by function *reader* and by generic function *writer*. With **:allocation :class**, *slot* is shared by all conditions of type *foo*. A condition is reported by *string* or by *report-function* of arguments condition and stream.

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

▷ Return new condition of *type*.

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

▷ Unless handled, signal as **condition**, **warning** or **error**, respectively, *condition* or a new condition of *type* or, with $\overset{\text{Fu}}{\textbf{format}}$ *control* and *args* (see p. 35), **simple-condition**, **simple-warning**, or **simple-error**, respectively. From $\overset{\text{Fu}}{\textbf{signal}}$ and $\overset{\text{Fu}}{\textbf{warn}}$, return `NIL`.

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

▷ Unless handled, signal as correctable **error** *condition* or a new condition of *type* or, with $\overset{\text{Fu}}{\textbf{format}}$ *control* and *args* (see p. 35), **simple-error**. In the debugger, use $\overset{\text{Fu}}{\textbf{format}}$ arguments *continue-control* and *continue-args* to tag the continue option. Return `NIL`.

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

▷ Return values of *forms* or, in case of **error**s, `NIL` and the $\underset{2}{\text{condition}}$.

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

▷ Invoke debugger with *condition*.

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

▷ 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. 35), **error**. When using the debugger's continue option, *places* can be altered before re-evaluation of *test*. Return `NIL`.

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

▷ Define, as a subclass of *superclasses*, class *foo*. In new instances, a *slot*'s value defaults to *form* unless set via **:***initarg-name* and is accessible by *reader-function* and *writer-function*. With **:allocation :class**, *slot* is shared by all instances of class *foo*.

$(\overset{\text{Fu}}{\textbf{find-class}}\ symbol\ \big[errorp_{\boxed{\text{T}}}\ [environment]\big])$

▷ Return class named *symbol*. **setf**able.

$(\overset{\text{gF}}{\textbf{make-instance}}\ class\ \{\textbf{:}initarg\ value\}^*\ other\text{-}keyarg^*)$

▷ Make new instance of *class*.

$(\overset{\text{gF}}{\textbf{reinitialize-instance}}\ instance\ \{\textbf{:}initarg\ value\}^*\ other\text{-}keyarg^*)$

▷ Change local slots of instance according to *initargs*.

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

$(\overset{\text{Fu}}{\textbf{slot-makunbound}}\ instance\ slot)$

▷ Make *slot* in instance unbound.

$(\begin{Bmatrix} \overset{\text{M}}{\textbf{with-slots}}\ (\{\widehat{slot}|(\widehat{var}\ \widehat{slot})\}^*) \\ \overset{\text{M}}{\textbf{with-accessors}}\ ((\widehat{var}\ \widehat{accessor})^*) \end{Bmatrix}\ instance\ (\textbf{declare}\ \widehat{decl}^*)^*$
$form^{\text{P}}{}^*)$

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

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

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

$(\overset{\text{gF}}{\textbf{change-class}}\ \widetilde{instance}\ new\text{-}class\ \{\textbf{:}initarg\ value\}^*\ other\text{-}keyarg^*)$

▷ Change class of instance to *new-class*.

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

$(\begin{Bmatrix} \overset{\text{gF}}{\textbf{initialize-instance}}\ (instance) \\ \overset{\text{gF}}{\textbf{update-instance-for-different-class}}\ previous\ current \end{Bmatrix}$
$\{\textbf{:}initarg\ value\}^*\ other\text{-}keyarg^*)$

▷ Its primary method sets slots on behalf of $\overset{\text{gF}}{\textbf{make-instance}}$/of $\overset{\text{gF}}{\textbf{change-class}}$ by means of $\overset{\text{gF}}{\textbf{shared-initialize}}$.

$(\overset{\text{gF}}{\textbf{update-instance-for-redefined-class}}\ instances\ added\text{-}slots$
$discarded\text{-}slots\ property\text{-}list\ \{\textbf{:}initarg\ value\}^*\ other\text{-}keyarg^*)$

▷ Its primary method sets slots on behalf of $\overset{\text{gF}}{\textbf{make-instances-obsolete}}$ by means of $\overset{\text{gF}}{\textbf{shared-initialize}}$.

$(\overset{\text{gF}}{\textbf{allocate-instance}}\ class\ \{\textbf{:}initarg\ value\}^*\ other\text{-}keyarg^*)$

▷ Return uninitialized instance of *class*. Called by $\overset{\text{gF}}{\textbf{make-instance}}$.

$(\overset{\text{gF}}{\textbf{shared-initialize}}\ instance \begin{Bmatrix} slots \\ \texttt{T} \end{Bmatrix} \{\textbf{:}initarg\ value\}^*\ other\text{-}keyarg^*)$

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

$(\overset{\text{gF}}{\textbf{slot-missing}}\ class\ object\ slot \begin{Bmatrix} \textbf{setf} \\ \textbf{slot-boundp} \\ \textbf{slot-makunbound} \\ \textbf{slot-value} \end{Bmatrix} [value])$

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

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

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

10.2 Generic Functions

($^{\text{Fu}}$**next-method-p**) ▷ T if enclosing method has a next method.

($^{\text{M}}$**defgeneric** $\left\{\begin{array}{l}\textit{foo}\\ (\textbf{setf}\ \textit{foo})\end{array}\right\}$ (*required-var** [**&optional** $\left\{\begin{array}{l}\textit{var}\\ (\textit{var})\end{array}\right\}^*$] [**&rest** *var*] [**&key** $\left\{\begin{array}{l}\textit{var}\\ (\textit{var}\,|\,(\textit{:key var}))\end{array}\right\}^*$ [**&allow-other-keys**]])

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

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

($^{\text{Fu}}$**ensure-generic-function** $\left\{\begin{array}{l}\textit{foo}\\ (\textbf{setf}\ \textit{foo})\end{array}\right\}$

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

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

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

($\left\{\begin{array}{l}\textit{var}\\ (\textit{spec-var}\ \left\{\begin{array}{l}\textit{class}\\ (\textbf{eql}\ \textit{bar})\end{array}\right\})\end{array}\right\}^*$ [**&optional**

$\left\{\begin{array}{l}\textit{var}\\ (\textit{var}\ [\textit{init}\ [\textit{supplied-p}]])\end{array}\right\}^*$] [**&rest** *var*] [**&key**

$\left\{\begin{array}{l}\textit{var}\\ (\left\{\begin{array}{l}\textit{var}\\ (\textit{:key var})\end{array}\right\}[\textit{init}\ [\textit{supplied-p}]])\end{array}\right\}^*$ [**&allow-other-keys**]]

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

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

($\left\{\begin{array}{l}^{\text{gF}}\textbf{add-method}\\ ^{\text{gF}}\textbf{remove-method}\end{array}\right\}$ *generic-function method*)

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

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

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

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

▷ List of methods 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**.

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

▷ Signal **error** on applicable method with invalid qualifiers, or on method combination. For *control* and *arg*s see **format**, p. 35.

($^{\text{gF}}$**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 $\underset{2}{\text{T}}$ 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*

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

▷ 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, have generic function applied to *gen-arg** return with the values of (*c-type* {*primary-method gen-arg**}*), leftmost *primary-method* being the most specific. In $^{\text{M}}$**defmethod**, primary methods are denoted by the *qualifier c-type*.

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

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

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

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

▷ Long form. Define new **method-combination** *c-type*. A call to a generic function using *c-type* will be equivalent to a call to the forms returned by *body** with *ord-λ** bound to *c-arg** (cf. $^{\text{M}}$**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 *qualifiers* match. Methods can be called via $^{\text{M}}$**call-method**. Lambda lists (*ord-λ**) and (*method-combination-λ**) according to *ord-λ* on p. 17, the latter enhanced by an optional **&whole** argument.