Appendix B

PLaSM libraries

The set of predefined `PLaSM` operators is listed here, grouped by library and then alphabetically ordered. Functions are documented according to the format below. For sake of readability the preconditions are given using the same semantics of a `PLaSM` definition. The postcondition is a predicate that must be satisfied by the function output. Currently, all libraries are loaded at set-up by the interpreter. All visible symbols, i.e. those listed in this appendix, are protected and cannot be redefined by the user. It is easy to see when a symbol is protected: (a) it is colored blue by the XPLODE editor, and (b) a `false` value is returned by the interpreter when asking for the evaluation of a redefinition of some protected symbol. The user may easily change this behavior, by either preventing the loading of some libraries, or by loading them as non-protected at set-up, or by loading some needed library on request during the work session. Let us finally remember that the language is not case-sensitive.

`NAME` short description of how the function works

Pre/Post conds	`function prototype` → `type of returned value`
Example	`function usage example`

B.1 Standard

The standard library contains basic predefined combinators and functions providing backward compatibility with previous `PLaSM` versions.

`AA` applies `fun` to each element of the `args` sequence

Pre/Post conds	`(fun::isfun)(args::isseq)` → `(isseq)`
Example	`aa:sqrt:<1,4,9,16>` ≡ `<1,2,3,4>`

`ABS` returns the absolute value of `n`

Pre/Post conds	`(n::isnum)` → `(isnum)`
Example	`abs:-5` ≡ `5`

`AC` apply-in-composition. `AC:fun:seq` is equivalent to `(COMP` ∼ `AA:fun):seq`

Pre/Post conds	`(fun::isfun)(seq::isseq)` → `(isfun)`
Example	`AC:SEL:<1,2,3>` ≡ `SEL:1` ∼ `SEL:2` ∼ `SEL:3`

Geometric Programming for Computer-Aided Design Alberto Paoluzzi
© 2003 John Wiley & Sons, Ltd ISBN 0-471-89942-9

`ACOS` computes the closest to zero arc associated with a given cosine value `n`

Pre/Post conds	`(n::isnum)` ⟶ `(isnum)`
Example	`acos:1 ≡ 0`

`AL` append left. appends `elem` on the left of `seq`

Pre/Post conds	`(elem::tt; seq::isseq)` ⟶ `(isseq)`
Example	`al:<0,<1,2,3,4>> ≡ <0,1,2,3,4>`

`ALIGN` aligns a pair of polyhedral complexes along any given subset of coordinates. *(see Scripts 2.3.1 and 2.3.3)*

Pre/Post conds	`(constraints::iseqof:istriple)(pol1,pol2::ispol)` ⟶ `(ispol)`
Example	`align:<<1,min,min>,<2,min,max>>:<cuboid:<2,2>,cuboid:<1,1>>`

`AND` standard logical operation on an arbitrary sequence of logical expressions

Pre/Post conds	`(preds::isseqof:isbool)` ⟶ `(isbool)`
Example	`and:<true,eq:<1,cos:0>,lt:0:(cos:pi)> ≡ true`

`ANIMATION` is a container for animation clips and/or polyhedra and/or affine trans.

Pre/Post conds	`(clips::isseqof:isanimpolc)` ⟶ `(isanimpolc)`
Example	`see` *`Script 15.6.8`*

`APPLY` returns the result of the expression `fun:value`

Pre/Post conds	`(fun::isfun,value::tt)` ⟶ `(tt)`
Example	`apply:<cat, <<1,2>,<3,4>>> ≡ <1,2,3,4>`

`AR` append right. appends `elem` on the right of `seq`

Pre/Post conds	`(seq::isseq; elem::tt)` ⟶ `(isseq)`
Example	`ar:<<1,2,3,4>,5> ≡ <1,2,3,4,5>`

`AS` apply-in-sequence. `AS:fun:seq` is equivalent to`(CONS ∼ AA:fun):seq`

Pre/Post conds	`(fun::isfun)(seq::isseq)` ⟶ `(isfun)`
Example	`AS:SEL:<1,2,3> ≡ [SEL:1, SEL:2, SEL:3]`

`ASIN` computes the closest to zero arc associated with a given sine value `n`

Pre/Post conds	`(n::isnum)` ⟶ `(isnum)`
Example	`asin:0 ≡ 0`

`ATAN` computes the closest to zero arc associated with a given tangent value `n`

Pre/Post conds	`(n::isnum)` ⟶ `(isnum)`
Example	`atan:0 ≡ 0`

`BOTTOM` locates the second argument bottom the first, by centering their xy extents

Pre/Post conds	`(pol1, pol2 ::ispol)` ⟶ `(ispol)`
Example	`bottom:< simplex:3, cuboid:<1,1,1> >`

`BOX` generates the containment box of `pol` in the `coords` subspace

Pre/Post conds	`(coords::isseqof:isintpos)(pol::ispol)` ⟶ `(ispol)`
Example	`box:<1,2>:(simplex:4)`

`BSPIZE` converts the HPC representation to BSP and vice versa, thus producing a BSP fragmentation of a non-convex `pol`

Pre/Post conds	`(pol::ispol)` ⟶ `(ispol)`
Example	`bspize:pol`

`C` curryfies a binary function, so that, for example, `fun:<a,b>`, can be evaluated as `c:fun:a:b`

Pre/Post conds	`(fun::isfun) ⟶ (isfun)`
Example	`AA:(c:*:2)(1..10) ≡ < 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 >`

`CASE` arguments are pairs `<pred`$_i$`, fun`$_i$`>` to be tested in sequence. If `pred`$_i$`:x ≡ true`, then `fun`$_i$`:x` is evaluated; otherwise the $(i+1)$-th pair is tested

Pre/Post conds	`(conds::isseqof:(ispred ∼ s1, isfun ∼ s2)(x::t) ⟶ (isfun)`
Example	`CASE:<<LT:0,K:'1'>, <C:EQ:0,K:'2'>, <GT:0,K:'3'>>:22 ≡ '3'`

`CAT` catenates a sequence of sequences, by eliminating a level of angled parenthesis

Pre/Post conds	`(seqs::isseqof:isseq) ⟶ (isseq)`
Example	`cat:<<0>,<1,2>,<<3,4>>,<>,<5,6,7>> ≡ <0,1,2,<3,4>,5,6>`

`CATCH` is used to catch a raised exception (see `SIGNAL`)

Pre/Post conds	`(and ∼ [ispair,isseqof:isfun]) ⟶ (isfun)`

Example	`def nonzero = if:< c:eq:0, signal, id>;`
	`catch:<nonzero, k:'zero'>:0 ≡ 'zero'`
	`catch:<nonzero, k:'zero'>:10 ≡ 10`

`CEIL` returns the nearest integer greater or equal than `n`.

Pre/Post conds	`(n::isnum) ⟶ (isnum)`
Example	`ceil:2.3 ≡ 3.0`

`CHAR` maps an integer from {`1,2, ... ,255`} into the corresponding ASCII character

Pre/Post conds	`(n::(and ∼ [isint,ge:1,le:255])) ⟶ (ischar)`
Example	`char:99 ≡ 'c'`

`CHARSEQ` maps a string into a sequence of characters

Pre/Post conds	`(str::isstring) ⟶ (isseqof:ischar)`
Example	`charseq:'plasm' ≡ <'p', 'l', 'a', 's', 'm'>`

`CMAP` version of `MAP` operator used for animations

Pre/Post conds	`(fun::isfun)(pol::ispol) ⟶ (ispol)`
Example	`CMAP:[s1,s2,sin∼s1 * sin∼s2]:dom`

`COMP` composition. Returns the composition of the functions in the argument sequence

Pre/Post conds	`(funs::isseqof:isfun) ⟶ (isfun)`
Example	`comp:<sqrt,+>:<4,5> ≡ (sqrt ∼ +):<4,5> ≡ 3`

`CONS` construction. Applies a function sequence `<f`$_1$`, ... ,f`$_n$`>` to `x` producing the sequence of applications `<f`$_1$`:x, ... ,f`$_n$`:x>`. Notice the "syntactical sugar" `[ ... ]`

Pre/Post conds	`(funs::isseqof:isfun)(x::tt) ⟶ (isseq)`
Example	`cons:<+,->:<3,2> ≡ [+,-]:<3,2> ≡ <5,1>`

`COS` computes the `cos` trigonometric function

Pre/Post conds	`(n::isnum) ⟶ (isnum)`
Example	`cos:0 ≡ 1`

`COSH` computes the hyperbolic cosine function

Pre/Post conds `(n::isnum) → (isnum)`

Example `cosh:0 ≡ 1.0`

`CUBOID` dimension-independent interval generator. `dims` is the sequence of projection sizes on coordinate directions

Pre/Post conds `(dims::isseqof:isnum) → (ispol)`

Example `cuboid:<1,1,1,1> ≡ polcomplex{4,4}`

`DETERMINANT` evaluates the determinant of the `m` matrix

Pre/Post conds `(m::ismat) → (isnum)`

Example `determinant:<<4,2>,<0,2>> ≡ 8`

`DIFFERENCE` computes the difference of a set of solids of the same dimension. The operator is dimension-independent

Pre/Post conds `(seq::isseqof:ispol) → (ispol)`

Example `difference:<pol1,pol2,pol3> ≡ pol1 - pol2 - pol3`

`DIFFERENCEPR` returns the *progressive* Boolean difference of a polyhedral sequence

Pre/Post conds `(seq::isseqof:ispol) → (ispol)`

Example `differencepr:<pol1,pol2,pol3> ≡`
`STRUCT:< pol1, pol2 - pol1, pol3 - pol2 - pol1 >`

`DIM` returns the *intrinsic* dimension (number of coordinates in a *chart*) of `pol`

Pre/Post conds `(pol::ispol) → (isint)`

Example `dim:(simplex:2) ≡ 2`

`DISTL` distribute left. Returns the pair sequence with `x` and the elements of `seq`

Pre/Post conds `(x::tt,seq::isseq) → (isseqof:ispair)`

Example `distl:<x,<1,2,3>> ≡ <<x,1>,<x,2>,<x,3>>`

`DISTR` distribute right. Returns the pair sequence with the elements of `seq` and `x`

Pre/Post conds `(seq::isseq,x::tt) → (isseqof:ispair)`

Example `distr:<<1,2,3>,x> ≡ <<1,x>,<2,x>,<3,x>>`

`DIV` n-ary left-associative division. It is an alias for “/”

Pre/Post conds `(nums::isseqof:isnum) → (isnum)`

Example `/:<20> ≡ div:<20> ≡ 1/20`
`20 / 2 ≡ 20 div 2 ≡ div:<20,2> ≡ 10`
`20 / 5 / 2 ≡ /:<20,5,2> ≡ div:<20,5,2> ≡ 2`

`DOWN` locates the second argument down the first (along the x_2 coordinate). Equivalent to `align:<<1,min,min>,<2,min,max>>`

Pre/Post conds `(pol1, pol2 ::ispol) → (ispol)`

Example `down:<cuboid:<1,1,1>, cuboid:<2,2,2>>`

`DUMP` prints a face-based representation of `pol` in the listener

Pre/Post conds `(pol::ispol) → (isstring)`

Example `DUMP:(CUBOID:<1,1,1>)`

`DUMPREP` prints a `pol` representation, face-based if `rep` = 1, vertices-based if `rep` = 0

Pre/Post conds `(pol::ispol)(rep::or ∼ [c:eq:0, c:eq:1]) → (isstring)`

Example `DUMP:(CUBOID:<1,1,1>):0`

`EMBED` embeds a d-polyhedron into the subspace $x_{d+1} = \cdots = x_{d+n} = 0$ of $\mathbb{E}^{d+n}$

Pre/Post conds	(n::isintpos)(pol::ispol) ⟶ (ispol)
Example	([dim,rn] ~ embed:1 ~ cuboid):<1,1> ≡ <2,3>

EQ predicate, testing for equality of all values in the argument sequence

Pre/Post conds	(or ~ aa:(or ~ [isnum,isbool,ischar,isstring,isfun])) ⟶ (isbool)
Example	4 eq len:<1,2,3,4> ≡ eq:<4,len:<1,2,3,4>> ≡ true eq:<4, 5 - 1, 3 + 1, 2 * 2, 8 / 2> ≡ true eq:<char:56,′8′> ≡ true eq:<4> ≡ true

EXP exponential. Computes the function $\mathbb{R} \to \mathbb{R} : x \mapsto e^x$

Pre/Post conds	(x::isnum) ⟶ (isnum)
Example	exp:1 ≡ 2.718281828459045

EXPORT exports a geometric value to a VRML file

Pre/Post conds	(pol::ispol)(filename::isstring) ⟶ (ispol)
Example	VRML:(cuboid:<2,2,2>):′out.wrl′;

FALSE primitive logical value

Pre/Post conds	⟶ (isbool)
Example	and:<false,gt:0:1> ≡ false

FIRST returns the first element of the sequence given as argument.

Pre/Post conds	(seq::and ~ [isseq,not ~ isnull]) ⟶ (tt)
Example	first:<<1,2>,<3,4>,<5,6>> ≡ <1,2>

FLASH exports a 2D pol within a drawing area of width pixels, in a .swf file

Pre/Post conds	(pol::ispol)(width::isintpos)(filename::isstring) ⟶ (ispol)
Example	flash:(cuboid:<2,2>):200:′path/out.swf′

FLASHANIM exports a 2D clip to a .swf file, with a given width and framerate

Pre/Post conds	(clip::isseqof:ispol)(width::isintpos)(filename::isstring) (framerate::isintpos) ⟶ (ispol)
Example	see Script 15.3.3

FLOOR returns the nearest integer less or equal to x

Pre/Post conds	(x::isnum) ⟶ (isint)
Example	floor:pi ≡ 3

FRAME creates a *static* object rendered within the [start,end] animation time

Pre/Post conds	(pol::ispol)(start,end::isnum) ⟶ (isanimpol)
Example	FRAME:(CUBOID:<1,1,1>):<2,5>

FROMTO returns the integer sequence from m to n. Empty if m > n. Alias for ..

Pre/Post conds	(m,n::isint) ⟶ (isseqof:isint)
Example	fromto:<1,4> ≡ 1 .. 4 ≡ <1,2,3,4>

GE predicate testing if the second argument n is *greater or equal* than m

Pre/Post conds	(m::isnum)(n::isnum) ⟶ (isbool)
Example	ge:5.2:5.3 ≡ true

GT predicate testing if the second argument n is *greater than* m

Pre/Post conds (m::isnum)(n::isnum) → (isbool)
Example gt:2:pi ≡ true

HELP prints a help screen within the listener

Pre/Post conds (a::tt) → (tt)
Example help:0

ID returns the arg argument unchanged

Pre/Post conds (arg::tt) → (tt)
Example id:7 ≡ 7

IF It is applied to a *triplet* of functions, where pred is a *predicate* specifying the conditional behavior with respect to x

Pre/Post conds (pred, then, else::isfun)(x::tt) → (tt)
Example if:<gt:0, sqrt, k:0>:9 ≡ 3; if:<gt:0, sqrt, k:0>:-9 ≡ 0

INSL *insert left* combinator, allowing to apply a *binary* operator f to n arguments: insl:f:<x_1, ... ,x_{n-1},x_n> ≡ f:<insl:f:<x_1, ... ,x_{n-1}>, x_n>

Pre/Post conds (f::isfun)(args::and ~ [isseq,not~isnull]) → (tt)
Example insl:**:<2,2,3> ≡ 4**3 ≡ 64

INSR *insert right* combinator, allowing to apply a *binary* operator f to n arguments: insr:f:<x_1, ... ,x_{n-1},x_n> ≡ f:<x_1, insr:f:<x_2, ... ,x_n>>

Pre/Post conds (f::isfun)(args::and ~ [isseq,not~isnull]) → (tt)
Example insr:**:<2,2,3> ≡ 2**8 ≡ 256

INTERSECTION computes the intersection of a set of solids of the same dimension. The operator is dimension-independent

Pre/Post conds (seq::(and~[isseqof:ispol,eq~aa:dim,and~aa:(eq~[dim,rn])])) → (ispol)
Example intersection:<cuboid:<0.8,0.8>, simplex:2>

INTSTO integers to. The operator returns either the sequence 1 .. n if $0 <$ n, or the sequence -1 .. n if n < 0, or the empty sequence if n $= 0$

Pre/Post conds (n::isint) → (isseqof:isint)
Example intsto:6 ≡ <1,2,3,4,5,6>

INV matrix inversion returns the inverse matrix of m.

Pre/Post conds (m::(and~[ismat, eq~[len,len~s1]])) → (ismat)
Example inv:<<1,2>,<2,0>> ≡ <<0,1/2>,<1/2,-1/4>>

ISANIMPOL predicate that tests if arg is an animated polyhedral complex

Pre/Post conds (arg::tt) → (isbool)
Example isanimpol:(cuboid:<2,2,2>) ≡ false

ISBOOL predicate that tests if arg is a Boolean expression

Pre/Post conds (arg::tt) → (isbool)
Example isbool:(eq:<3+1,5-2>) ≡ true

ISCHAR predicate that tests if arg is a character

Pre/Post conds (arg::tt) → (isbool)
Example ischar:′a′ ≡ true

ISEMPTY predicate that tests if a geometric value is empty

Pre/Post conds `(pol::ispol) → (isbool)`

Example `isempty:(-:<cuboid:<2,2>,<cuboid:<2,2>>) ≡ true`

`ISFUN` predicate that tests if `arg` is a function

Pre/Post conds `(arg::tt) → (isbool)`

Example `isfun:cons ≡ true`

`ISINT` predicate that tests if `arg` is an integer

Pre/Post conds `(arg::tt) → (isbool)`

Example `isint:10 ≡ true`

`ISINTNEG` predicate that tests if `arg` is a negative integer

Pre/Post conds `(arg::tt) → (isbool)`

Example `isintneg:-7 ≡ true`

`ISINTPOS` predicate that tests if `arg` is a positive integer

Pre/Post conds `(arg::tt) → (isbool)`

Example `isintpos:4 ≡ true`

`ISNULL` predicate that tests if `arg` is the empty sequence

Pre/Post conds `(arg::tt) → (isbool)`

Example `isnull:<> ≡ true`

`ISNUM` predicate that tests if `arg` is a number

Pre/Post conds `(arg::tt) → (isbool)`

Example `isnum:pi ≡ true`

`ISNUMNEG` predicate that tests if `arg` is a negative number

Pre/Post conds `(arg::tt) → (isbool)`

Example `isnumneg:-12.7 ≡ true`

`ISNUMPOS` predicate that tests if `arg` is a positive number

Pre/Post conds `(arg::tt) → (isbool)`

Example `isnumpos:12.7 ≡ true`

`ISPAIR` predicate that tests if `arg` is a pair (a sequence of exactly two elements)

Pre/Post conds `(arg::tt) → (isbool)`

Example `ispair:<+,-> ≡ true`

`ISPOL` predicate that tests if `arg` is a geometric value

Pre/Post conds `(arg::tt) → (isbool)`

Example `ispol:(simplex:1) ≡ true`

`ISREAL` predicate that tests if `arg` is a real number

Pre/Post conds `(arg::tt) → (isbool)`

Example `isreal:0.45 ≡ isreal:4.5e-1 ≡ true`

`ISREALNEG` predicate that tests if `arg` is a negative real number

Pre/Post conds `(arg::tt) → (isbool)`

Example `isrealneg:-5.4 ≡ true`

`ISREALPOS` predicate that tests if `arg` is a positive real number

Pre/Post conds `(arg::tt) → (isbool)`

Example `isrealpos:pi ≡ true`

ISSEQ predicate that tests if arg is a sequence

Pre/Post conds (arg::tt) ⟶ (isbool)
Example isseq:<id,cons> ≡ true

ISSEQOF second-order predicate that tests if arg is a sequence with all elements of pred type

Pre/Post conds (pred::isfun)(arg::tt) ⟶ (isbool)
Example isseqof:isint:<2,4,5.01> ≡ false

ISSTRING predicate that tests if arg is a string

Pre/Post conds (arg::tt) ⟶ (isbool)
Example isstring:′PLaSM′ ≡ true

JOIN returns the *convex hull* of a sequence of geometric values in $\mathbb{E}^n$

Pre/Post conds (seq::or ∼ [isseqof:ispol, ispol]) ⟶ (ispol)
Example join:< (embed:1 ∼ cuboid):<1,1>, simplex:3 >

K constant functional that always returns the first argument, for any value of the second one

Pre/Post conds (a::tt)(b::tt) ⟶ (tt)
Example k:<1,2>:100 ≡ <1,2>

LAST returns the last element of the non-empty sequence argument

Pre/Post conds (sequence::and ∼ [isseq,not ∼ isnull]) ⟶ (tt)
Example last:<<1,2>,<3,4>,<5,6>> ≡ <5,6>

LE predicate that tests if the second argument n is *less or equal* than m

Pre/Post conds (m::isnum)(n::isnum) ⟶ (isbool)
Example le:2:(PI - 2) ≡ true

LEFT locates the second argument on the left of the first (along the x_1 coordinate)

Pre/Post conds (pol1, pol2 ::ispol) ⟶ (ispol)
Example left:<cuboid:<1,1,1>, cuboid:<2,2,2>>

LEN returns the *length* of the sequence given as argument

Pre/Post conds (sequence::isseq) ⟶ (isint)
Example len:<2,5,2,1> ≡ 4

LESS predicate that tests if the argument is a sequence of increasing numbers

Pre/Post conds (nums::isseqof:isnum) ⟶ (isbool)
Example less:<1,2,3> ≡ true

LESSEQ predicate that tests if the argument is a sequence of non-decreasing numbers

Pre/Post conds (nums::isseqof:isnum) ⟶ (isbool)
Example lesseq:<1,2,2,3> ≡ true

LIFT combining form with semantics lift:f:<$f_1, \dots, f_n$> ≡ f ∼ [$f_1, \dots, f_n$]

Pre/Post conds (f::isfun)(funs::isseqof:isfun) ⟶ (isfun)
Example lift:+:<sin,cos> ≡ + ∼ [sin,cos]

LIST returns the sequence containing arg. Alias for [id]

Pre/Post conds (arg::tt) ⟶ (isseq)
Example list:4 ≡ <4>

`LN` *natural logarithm* $\log_e$ of a positive real `x`

Pre/Post conds	`(x::isrealpos)` ⟶ `(isreal)`
Example	`DEF e = (+ ~ aa:(c:/:1.0 ~ fact)):(0..20); ln:1 = 0; ln:e = 1;`

`LOAD` loads a *script* file within the run-time `PLaSM` environment

Pre/Post conds	`(filename::isstring))` ⟶ (*side effect*)
Example	`load:'~/Documents/example.psm'`

`LOADLIB` loads the *library* file passed as argument. Let us use no file extension

Pre/Post conds	`(filename::isstring)` ⟶ (*side effect*)
Example	`loadlib:'psmlib/curves'`

`LOOP` generates `times` repetitions of an animation

Pre/Post conds	`(times::isintpos)(anim::isanimpolc)` ⟶ `(isanimpol)`
Example	`def movie = loop:10:(animation:<clip1, clip2>);`

`LT` predicate that tests if the second argument m is *less than* n

Pre/Post conds	`(n::isnum)(m::isnum)` ⟶ `(isbool)`
Example	`lt:5:2` ≡ `true`

`MAP` simplicial mapping. It maps a (possibly `CONS`ed) sequence of coordinate `funs` over a polyhedral `domain`. A simplicial decomposition is automatically generated

Pre/Post conds	`(funs::or ~ [isseqof:isfun, isfun])(domain::ispol)` ⟶ `(ispol)`
Example	`map:<cos ~ s1,sin ~ s1>:((quote ~ #:32):(2*pi/32))`

`MAT` generates a tensor (bijective transformation function) from its invertible matrix with first row and column homogeneous. Dimension independent operator

Pre/Post conds	`(m::issqrmat)` ⟶ `(isfun)`

Example	`def rot2d = mat ~ mathom ~ [[cos,- ~ sin],[sin,cos]];`
	`rot2d:(pi/4):(cuboid:<1,1>)`

`MAX` returns the maximum values achieved by `pol` on `coords` coordinates

Pre/Post conds	`(coords::isseqof:isintpos)(pol::ispol)` ⟶ `(isseqof:isnum)`
Example	`max:<1,3>:(cuboid:<2,4,6>)` ≡ `<2.0,6.0>`

`MED` returns the medium values achieved by `pol` on `coords` coordinates

Pre/Post conds	`(coords::isseqof:isintpos)(pol::ispol)` ⟶ `(isseqof:isnum)`
Example	`med:<1,3>:(cuboid:<2,4,6>)` ≡ `<1.0,3.0>`

`MERGE` merging of two ordered sequences `seqs` using the binary predicate `pred`

Pre/Post conds	`(pred::isfun)(seqs::and ~ [isseq,not ~ isnull])` ⟶ `(isseq)`
Example	`merge:less:<<1,3,4,5>,<2,4,6,8>>` ≡ `< 1,2,3,4,4,5,6,8 >`

`MIN` returns the minimum values achieved by `pol` on `coords` coordinates

Pre/Post conds	`(coords::isseqof:isintpos)(pol::ispol)` ⟶ `(isseqof:isnum)`
Example	`min:<1,2>:(cuboid:<2,4,6>)` ≡ `<0.0,0.0>`

`MKPOL` is a mapping from triples of number sequences to polyhedral complexes: `mkpol:< verts, cells, pols >`, where `verts` are *points* in $\mathbb{E}^d$ (given as sequences of coordinates); `cells` are convex *cells* (given as sequences of point indices); `pols` are *polyhedra* (given as sequences of cell indices). Each cell is the convex hull of its vertices, each polyhedron is the set union of its cells

Pre/Post conds `(verts::ismatof:isreal;`
`cells,pols::AND ∼AA:(isseqof:isintpos)) ⟶ (ispol)`
Example `mkpol:<<<0,0>,<0,1>,<1,1>,<1,0>>, <<1,2,3,4>>,<<1>>>`

`MOVE` basic primitive for configuration space (cs) sampling animation. Is applied to: (a) geometry generator function of real parameters (*degrees of freedom*); (b) sequence of cs points; (c) increasing sequence of *time* values, s.t. `len:cspoints ≡ len:timepoints`

Pre/Post conds `(geometry::isfun)(cspoints::or ∼ [iseqof:isreal,ismatof:isreal])`
`(timepoints::isseqof:isrealpos) ⟶ (isanimpol)`
Example `def obj(x,a::isreal) = (t:1:x ∼ r:<1,2>:a):(cuboid:<1,1>);`
`def clip = move:obj:<<0,0>,<5,pi>,<5,0>>:<0,1,2>;`

`NEQ` predicate, testing the non-equality of all values in the argument sequence

Pre/Post conds `(or ∼ aa:(or ∼ [isnum,isbool,ischar,isstring,isfun]))`
`⟶ (isbool)`
Example `neq:<4, 5 - 1, 3 + 1, 2 * 2, 8 / 2> ≡ false`

`NOT` standard unary logical operation on logical values. Actually, it considers every `PLaSM` value as `true` but `<>`, thus returning, e.g., `not:′z′ ≡ false` and texttnot:¡¿ ≡ true

Pre/Post conds `(a::tt) ⟶ (isbool)`
Example `not:false ≡ true`

`OPEN` restores a geometric object from a `.xml` file (see `SAVE`)

Pre/Post conds `(filename::isstring) ⟶ (ispol)`
Example `def cube = open:′path/cube.xml′;`

`OR` standard logical operation between arguments with logical values

Pre/Post conds `(preds::isseqof:isbool) ⟶ (isbool)`
Example `or:<false,(not ∼ eq):<1,2>> ≡ true`

`ORD` maps an ascii character into its ordinal value, i.e. its index in the ascii table

Pre/Post conds `(c::ischar) ⟶ (and ∼ [isintpos,le:255])`
Example `ord:′c′ ≡ 99, ord:′\t′ ≡ 9, ord:′␣′ ≡ 32`

`PI` constant value. `PLaSM` denotation of π

Pre/Post conds `⟶ (isnum)`
Example `pi ≡ 3.14159265358979`

`PRINT` returns `arg` and prints its value in the listener. It may be used to debugging

Pre/Post conds `(arg::tt) ⟶ (tt)`
Example `(@1 ∼ print ∼ embed:1 ∼ print ∼ simplex):2`

`QUOTE` transforms non-empty sequences of non-zero reals into 1D polyhedra. Positive numbers produce solid segments; negative numbers are used as translations

Pre/Post conds `(nums::and ∼ [isseqof:isnum, and ∼ aa:(c:neq:0)]) ⟶ (ispol)`
Example `quote:<2,-10,1,1,-10,2>`

`R` dimension-independent rotation tensor. `coords` are the indices of the coordinate *pair* affected by the transformation. The rotation `angle` is given in radians

Pre/Post conds `(coords::and ∼ [ispair, isseqof:isintpos])`
`(angle::isnum)(pol::or ∼ [ispol,isanimpol])`
`→ (or ∼ [ispol,isanimpol])`
Example `r:<1,2>:(pi/4):(cuboid:<10,10,10>)`

`RAISE` this combinating form is used to overload operators over both numbers and functions. In fact `RAISE:f:seq ≡ IF:<IsSeqOf:IsFun, LIFT:f, f>:seq`

Pre/Post conds `(f::isfun)(args::isseq) → (isfun)`
Example `raise:+:<+,*> ≡ + ∼ [+,*]`

`RANGE` returns the integer sequence (possibly reversed) from `m` to `n`

Pre/Post conds `(m,n::isint) → (isseq)`
Example `range:<5,-1> ≡ <5,4,3,2,1,0,-1>`

`REVERSE` returns a sequence in reverse order

Pre/Post conds `(seq::isseq) → (isseq)`
Example `reverse:<<1,2>,<3,4>,<5,6>> ≡ <<5,6>,<3,4>,<1,2>>`

`RIGHT` locates the second argument on the right of the first (along the x_1 coordinate)

Pre/Post conds `(pol1, pol2 ::ispol) → (ispol)`
Example `right:<cuboid:<1,1,1>, cuboid:<2,2,2>>`

`RN` returns the *embedding dimension*, i.e. the number of coordinates of points

Pre/Post conds `(pol::ispol) → (isintpos)`
Example `(rn ∼ embed:2 ∼ simplex):3 ≡ 5`

`S` dimension-independent scaling tensor. `coords` are the indices of coordinates affected by the transformation

Pre/Post conds `(coords::or ∼ [isintpos, isseqof:isintpos]) (params::or ∼`
`[isnum, isseqof:isnum]) (pol::or ∼ [ispol,isanimpol])`
`→ (or ∼ [ispol,isanimpol])`
Example `s:<1,2>:<0.5,-1.5>:(cuboid:<10,10,10>)`

`SAVE` stores a geometric value into an XML file

Pre/Post conds `(pol::ispol)(filename::isstring) → (ispol)`
Example `save:(cuboid:<1,1,1>):′/path/cube.xml′`

`SEL` returns the *i*-th element of `seq` sequence. An exception is raised if `i > len:seq`

Pre/Post conds `(i::isintpos)(seq::isseq) → (tt)`
Example `sel:2:<<1,2>,<3,4>,<5,6>> ≡ <3,4>`

`SHIFT` shifts the beginning of the animation `clip` of `t` seconds

Pre/Post conds `(t::isnum)(clip::isanimpolc) → (isanimpol)`
Example `shift:10:clip`

`SHOWPROP` returns the sequence of `<property,value>` pairs associated with `obj`

Pre/Post conds `(obj::ispol) → (isseqof:ispair)`
Example `showprop:(cuboid:<1,1> color red) ≡ <<'RGBcolor',<1,0,0>>>`

`SIGN` returns either `1` if `x` is positive, or `-1` if `x` is negative, or `0` if `x` is zero

Pre/Post conds `(x::isnum) → (isint)`
Example `sign:-4.5 ≡ -1`

`SIGNAL` raises an *exception*, to be captured by the `CATCH` primitive

Pre/Post conds `(value::tt)` → `(exception)`

Example `def nonzero = if:<c:neq:0, id, signal>;`
`nonzero:0` ≡ `plasm exception: 0` (message in the listener)
`catch:<nonzero, k:'nonzero'>:0` ≡ `'nonzero'`

`SIMPLEX` generator of the simplex $\sigma^d \equiv \mathrm{conv}\,(\{\boldsymbol{e}_i\} \cup \{\mathbf{0}\}) \subset \mathrm{I\!R}^d, 1 \leq i \leq d$

Pre/Post conds `(d::isnat)` → `(ispol)`

Example `simplex:5`

`SIN` computes the `sin` trigonometric function. The argument is in radians

Pre/Post conds `(alpha::isnum)` → `(isnum)`

Example `sin:(pi/2)` ≡ `1.0`

`SINH` computes the hyperbolic sine of `x`

Pre/Post conds `(x::isnum)` → `(isnum)`

Example `sinh:0` ≡ `0.0`

`SIZE` return the size of the `pol` projection/s on the specified coordinate direction/s

Pre/Post conds `(coords::or` ∼ `[isintpos,isseqof:isintpos])(pol::ispol)`
→ `(or` ∼ `[isrealpos, isseqof:isrealpos])`

Example `(size:2` ∼ `cuboid):<2,4,6>` ≡ `4.0`

`SQRT` square root operator. Negative arguments are allowed

Pre/Post conds `(x::isnum)` → `(isnum)`

Example `sqrt:64` ≡ `8; sqrt:-64` ≡ `0+8i`

`STRING` maps a sequence of characters into a string

Pre/Post conds `(chars::isseqof:ischar)` → `(isstring)`

Example `string:<'c', 'a', 'd'>` ≡ `'cad'`

`STRUCT` constructor of hierarchical assemblies

Pre/Post conds `(args::isseqof:(or` ∼ `[ispol, isanimpol, isfun]))`
→ `(or` ∼ `[ispol, isanimpol])`

Example `struct:<cuboid:<2,2>, t:1:3:, simplex:2>`

`SVG` exporter of a 2D geometric value `pol` into a canvas of `width` pixels in a `.svg` file

Pre/Post conds `(pol::ispol)(width::isnum)(filename::isstring)` → `(ispol)`

Example `svg:(cuboid:<1,1>):250:'out.svg'`

`T` dimension-independent translation tensor. `coords` are the indices of the coordinates affected by the transformation

Pre/Post conds `(coords::or` ∼ `[isintpos,isseqof:isintpos])`
`(params::or` ∼ `[isnum,isseqof:isnum])`
`(pol::or` ∼ `[ispol,isanimpol])` → `(or` ∼ `[ispol,isanimpol])`

Example `t:<1,2>:<-5,-5>:(cuboid:<10,10>)`

`TAIL` returns the non-empty argument sequence but its first element

Pre/Post conds `(seq::and` ∼ `[isseq,not` ∼ `isnull])` → `(isseq)`

Example `tail:<<1,2>,<3,4>,<5,6>>` ≡ `<<3,4>,<5,6>>`

`TAN` computes the `tan` trigonometric function. The argument is in radians

Pre/Post conds `(alpha::isnum)` → `(isnum)`

Example `tan:(pi/4)` ≡ `1`

`TANH` computes the hyperbolic tangent of the argument

Pre/Post conds `(x::isnum)` → `(isnum)`

Example `tanh:0` ≡ `0`

`TIME` returns information about the execution time of the function argument

Pre/Post conds `(f::isfun)` → `(tt)`

Example `time:cuboid:<1,1,1>`

`TOP` locates the second argument over the first (z dir), by centering their xy extents

Pre/Post conds `(pol1, pol2 ::ispol)` → `(ispol)`

Example `top:<cuboid:<1,1,0.5> color red, cuboid:<1,1,0.5> color blue>`

`TRANS` transposes a sequence of sequences of the same length. The elements may be of arbitrary type

Pre/Post conds `(seq::ismat)` → `(ismat)`

Example `trans:<<1,2>,<3,4>,<5,6>>` ≡ `<<1,3,5>,<2,4,6>>`

`TREE` recursively applies a binary function `f` to a sequence of arguments `arg`

Pre/Post conds `(f::isfun)(args::and` ∼ `[isseq,not` ∼ `isnull])` → `(tt)`

Example
```
def bigger (a,b::isreal) = if:< greater, s1, s2 >:<a,b>;
def biggest (seq::isseqof:isnum) = tree:bigger:seq;
biggest:<8,2,4,2,3,11,-5> ≡ 11
```

`TRUE` a truth value. Primitive `PLaSM` value

Pre/Post conds → `(isbool)`

Example `and:<true, gt:1:0>` ≡ `false`

`TT` constant predicate that returns `true` for every argument. Alias for `k:true`

Pre/Post conds `(arg::tt)` → `(isbool)`

Example `tt:cons` ≡ `true; tt:1000` ≡ `true; tt:'aaa'` ≡ `true;`

`UKPOL` UnmaKe POLyhedron. Inverse operator of `MKPOL` (see). Returns `pol` represented as a *triplet* of vertices, convex and polyhedral cells

Pre/Post conds `(pol::ispol)` → `(isseqof:isseq)`

Example `ukpol:(cuboid:<1,1>)` ≡ `<<<0.0, 1.0>, <1.0, 1.0>, <0.0, 0.0>, <1.0, 0.0>>, <<1, 2, 3, 4>>, <<1>>>`

`UKPOLF` unmake polyhedron *by faces*. Returns the internal representation by faces as a triplet `<covectors, cells, pols>`. Covectors are normalized

Pre/Post conds `(pol::ispol)` → `(iseqof:isseq)`

Example `ukpolf:(cuboid:<1,1>)` ≡ `<<<1.0, 0.0, 0.0>, <-0.7071, 0.0, 0.7071>, < 0.0, 1.0, 0.0>, <0.0, -0.7071, 0.7071>>, <<1, 2, 3, 4>>, <<1>>>`

`UNION` of a set of solids of the same dimension. More expensive than the `+` operator, but produces a well defined cellular result

Pre/Post conds `(args::isseqof:ispol)` → `(ispol)`

Example `(@1` ∼ `union` ∼ `[id, t:<1,2>:<0.5,0.5>]` ∼ `cuboid):<1,1,1>`

`UP` locates the second argument over the first (along the x_2 coordinate)

Pre/Post conds `(pol1, pol2 ::ispol)` → `(ispol)`

Example `up:<cuboid:<1,1,1>, cuboid:<2,2,2>>`

`VRML` exports a geometric value into a `vrml` file with suffix `.wrl`

Pre/Post conds `(pol::ispol)(filename::isstring) → (ispol)`

Example `vrml:(cuboid:<2,2,2>):′out.wrl′;`

`WARP` time scaling operator used for animations

Pre/Post conds `(s::isnum)(anim::isanimpol) → (isanimpol)`

Example `(shift:10 ∼ warp:-1):clip`

`WITH` binary operator used to dynamically annotate a geometric value with pairs `<property,values>`, where `property` is a string

Pre/Post conds `(pol::ispol; prop_val:: and ∼ [ispair, isstring ∼ s1, tt ∼ s2])`
`→ (tt)`

Example `cuboid:<1,1> with < ′RGBcolor′,<1,0,0> >`

`XOR` Boolean XOR (union minus intersection) of a sequence of geometric values

Pre/Post conds `(args::isseqof:ispol) → (ispol)`

Example `xor:<cuboid:<3,3,3>, t:<1,2>:<0.5,0.5>:(cuboid:<3,3,3>)>`

`-` *n*-ary difference operator between (a) numbers, (b) functions, (c) matrices and (d) geometric values

Pre/Post conds `(args::lift:or:(AA:isseqof:<isnum, isfun, ismat, ispol>))`
`→ (or ∼ [isnum,isfun,ismat,ispol])`

Example `2 - 3.5 - 1 ≡ -:< 2, 3.5, 1 > ≡ 0.5`
`(sin - cos):PI ≡ (- ∼ [sin,cos]):PI ≡ 1.0`
`idnt:2 - <<1,1>,<1,1>> ≡ <<0,-1>,<-1,0>>`
`(id - t:<1,2>:<0.5,0.5>):(cuboid:<3,3,3>) ≡ PolComplex<3,3>`

`#` repetition operator. Returns a sequence with `n` repetitions of `arg`

Pre/Post conds `(n::isintpos)(arg::tt) → (isseq)`

Example `#:4:true ≡ <true,true,true,true>`

`##` sequence repetition operator. `##:n:seq` is equivalent to `(cat ∼ #:n):seq`

Pre/Post conds `(n::isintpos)(seq::isseqof:tt) → (isseq)`

Example `##:3:<1,2> ≡ cat:(#:3:<1,2>) ≡ <1,2,1,2,1,2>`

`&` *n*-ary Boolean intersection operator

Pre/Post conds `(seq::isseqof:ispol) → (ispol)`

Example `&:<cuboid:<0.8,0.8,0.8>, simplex:3>`

`&&` binary intersection of extrusions. The `args` are properly embedded into `coords` subspaces, indefinitely extruded and pair-wise intersected

Pre/Post conds `(coords::isseqof:isint)(args::isseqof: ispol) → (ispol)`

Example

`*` *n*-ary product operator between (a) numbers, (b) functions, (c) matrices and (d) geometric values

Pre/Post conds `(args::lift:or:(AA:isseqof:<isnum, isfun, ismat, ispol>))`
`→ (or ∼ [isnum,isfun,ismat,ispol])`

Example `*:<20,5,2> ≡ 200`
`(sin * cos):PI ≡ (* ∼ [sin,cos]):PI ≡ 0.0`
`<<4,2>,<2,1>> * <<1,1>,<0,2>> ≡ <<4,8>,<2,4>>`
`simplex:2 * Q:1 ≡ PolComplex{3,3}`

`**` power raising. Mathematical operator

Pre/Post conds `(base,exp::isnum)` → `(isnum)`
Example `**:<2,3>` ≡ `8.0; 81 ** (1/2)` ≡ `9.0`

`..` generator of the integer sequence from `m` to `n`. Alias for `fromto`

Pre/Post conds `(m,n::isint)` → `(isseqof:isint)`
Example `-1 .. 4` ≡ `<-1,0,1,2,3,4>`

`/` *n*-ary division operator between numbers and functions

Pre/Post conds `(args::lift:or:(AA:isseqof:<isnum, isfun, ispol>))`
→ `(or ∼ [isnum,isfun,ispol])`
Example `/:<20,5,2>` ≡ `2`

^ evaluates the Boolean XOR of a sequence of geometric values. It is less time-consuming than the `xor` operator, but returns a “weak” complex

Pre/Post conds `(seq::isseqof:ispol)` → `(ispol)`
Example `(@1 ∼ ^ ∼ [id, t:<1,2>:<0.5,0.5>] ∼ cuboid):<3,3,0.5>`

∼ function *composition* operator. Alias for *n*-ary `COMP`

Pre/Post conds `(funs::isseqof:isfun)` → `(isfun)`
Example `(sqrt ∼ +):<4,5>` ≡ `3`

`+` *n*-ary addition operator between (a) numbers, (b) functions, (c) matrices and (d) geometric values (as union)

Pre/Post conds `(args::lift:or:(AA:isseqof:<isnum, isfun, ismat, ispol>))`
→ `(or ∼ [isnum,isfun,ismat,ispol])`
Example `+:<5,2,1>` ≡ `8`
`(sin + cos):PI` ≡ `(+ ∼ [sin,cos]):PI` ≡ `-1.0`
`<<4,2>,<2,1>> + <<1,1>,<0,2>>` ≡ `<<5,3>,<2,3>>`
`cuboid:<3,3,3> + t:<1,2>:<0.5,0.5>:cuboid:<3,3,3>`

`@n` returns the *n*-dimensional skeleton of a complex

Pre/Post conds `(pol::ispol)` → `(ispol)`
Example `@1:(cuboid:<0.8,0.8,0.8> & simplex:3)` ≡ `PolComplex{1,3}`

B.2 `animation` Library

`Curve2cspath` Transforms a 2D point sequence into a CS path (3 DOFs)

Pre/Post conds `(curve::isseqof:isfun)` → `(isfun)`
Example `(AA:(Curve2CSPath:trajectory ∼ [ID]) ∼ Sampling):20;`

`Inarcs` returns the inward arcs of a given node in a graph

Pre/Post conds `(node::isint)(graph::isseqOf:IsTriple)` → `(IsSeqOf:IsPair)`
Example `inarcs:7:<<0,1,2>,<1,2,5>,<2,3,3>,<3,4,4>,<1,5,0>,<6,2,0>,`
`<2,7,0>,<8,3,0>,<5,6,10>,<6,7,5>,<7,8,2>>` ≡ `<<2,0>,<6,5>>`

`Outarcs` returns the outward arcs of a given node in a graph

Pre/Post conds `(node::isint)(graph::isseqOf:IsTriple)` → `(IsSeqOf:IsPair)`
Example `outarcs:7:<<0,1,2>,<1,2,5>,<2,3,3>,<3,4,4>,<1,5,0>,<6,2,0>,`
`<2,7,0>,<8,3,0>,<5,6,10>,<6,7,5>,<7,8,2>>` ≡ `<<8,2>>`

`Tmax` computes the maximum spanning time of a given node in a graph

Pre/Post conds	`(graph::isseqof:istriple)(node::isint)` → `(isint)`
Example	`See p. 672`

`Tmin` computes the minimum spanning time of a given node in a graph

Pre/Post conds	`(graph::isseqof:istriple)(node::isint)` → `(isint)`
Example	`See p. 672`

B.3 `colors` Library

The `colors` library makes large use of the recent OO extension of `PLaSM` described in [MMPP02]. In such a context *objects* are values belonging to classes; *classes* are sets generated by a `CLASS` constructor function; this one automatically generates a predicate `in`*classname* to test set-membership of objects.

`Appearance` the appearance property of `pol` is set by its `mat` material and `fulltex`

Pre/Post conds	`(pol::ispol; mat::isbasematerial; fulltex::isfulltexture)` → `(ispol)`
Example	`appearance:<pol,mat,fulltex>` ≡ `pol material mat texture fulltex`

`Basecamera` full detail definition according to the VRML specs of *camera* node

Pre/Post conds	`(position, orientation::or` ∼ `[isvect, isnull]; fieldofview::or` ∼ `[isreal, isnull]; description::isstring)` → `(isbasecamera)`
Example	`Basecamera:<<3,0,0>, <0,1,0,PI/2>, pi/4, ′x axis camera′>`

`Basedirlight` specialization of `GenericLight` with `type` ≡ 1 and various defaults

Pre/Post conds	`(dirappearance, dirgeometry::isseq)` → `(isbasedirlight)`
Example	see `psmlib/colors.psm`

`Basematerial` full detail definition according to the VRML specs of *material* node

Pre/Post conds	`(diffuse, specular::isrgbcolor; ambient::isinto:<0, 1>;`
	`emissive::isrgbcolor; shininess, transparency::isinto:<0, 1>)`
	→ `(isbasematerial)`

Example	`basematerial:<rgbcolor:<1,0.85,0.85>,black,0.2,black,0.2,0.0>`

`Basepointlight` specialization of `GenericLight` with `type` ≡ 0 and defaults

Pre/Post conds	`(pointappearance, pointgeometry::isseq)` → `(isbasepointlight)`
Example	see `psmlib/colors.psm`

`Basespotlight` specialization of `GenericLight` with `type` ≡ 0 and defaults

Pre/Post conds	`(spotappearance, spotgeometry::isseq)` → `(isbasespotlight)`
Example	see `psmlib/colors.psm`

`Basetexture` specialization of `Fulltexture` with no texture transformation

Pre/Post conds	`(url::isstring; repeats, repeatt::isbool)` → `(isbasetexture)`
Example	see `psmlib/colors.psm`

`Black` plasm *object* of class `rgbcolor` and value `<0,0,0>`

Pre/Post conds	→ `(isrgbcolor)`
Example	`cuboid:<1,1,1> color black`

`Blue` plasm *object* of class `rgbcolor` and value `<0,0,1>`

Pre/Post conds → `(isrgbcolor)`
Example `cuboid:<1,1,1> color blue`

`Brown` plasm *object* of class `rgbcolor` and value `<3/5,2/5,1/5>`

Pre/Post conds → `(isrgbcolor)`
Example `cuboid:<1,1,1> color brown`

`Camera` used to associate a `camera` to `pol`, to be inserted in a hierarchical graph

Pre/Post conds `(pol::ispol; camera::isbasecamera)` → `(ispol)`
Example `MK:<0,0,0> CAMERA BaseCamera:< prp, <0,0,1,0>, PI/4, string >`

`Color` returns `pol` annotated with `col` value for ′`rgbcolor`′ property

Pre/Post conds `(pol::ispol; col::isrgbcolor)` → `(ispol)`
Example `cuboid:<1,1,1> color yellow`

`Crease` smooths `pol` by annotating it with `angle` value for ′`VRMLcrease`′ property

Pre/Post conds `(pol::ispol; angle::isreal)` → `(ispol)`
Example `sphere:1:<12,24> crease (pi/2)`

`Cyan` plasm *object* of class `rgbcolor` and value `<0,1,1>`

Pre/Post conds → `(isrgbcolor)`
Example `cuboid:<1,1,1> color cyan`

`Fulltexture` generator of texture objects, including 2D texture transformations

Pre/Post conds `(url::isstring; repeats, repeatt::isbool; center::ispoint;`
`rotation::isreal; scale, translation::isvect)` → `(isfulltexture)`
Example `fulltexture:<′img/glass.jpg′,true,true,<0,0>,0,<1,1>,<0,0>>`

`Genericlight` used to switch between point, directional and spot lights

Pre/Post conds `(type::isinto:<0, 2>; appearance, geometry::isgenericlightgeometry)`
→ `(isgenericlight)`
Example see `examples/color/lights.psm`

`Genericlightappearance` returns objects embodying common params of light types

Pre/Post conds `(color::or` ∼ `[isrgbcolor, isnull]; intensity, ambient::or` ∼
`[isreal, isnull]; ison::or` ∼ `[isbool, isnull])`
→ `(isgenericlightappearance)`
Example `genericlightappearance:<magenta, 1, 0.4, true>`

`Genericlightgeometry` returns objects with common params of light geometries

Pre/Post conds `(location, direction, attenuation::or` ∼ `[isvect, isnull];`
`radius, beamwidth, cutoffangle::or` ∼ `[isreal, isnull])`
→ `(isgenericlightgeometry)`
Example `GenericLightGeometry:<<0,0,0>,<1,0,0>,<1,0,0>,10,PI/4,PI/6>`

`Gray` plasm *object* of class `rgbcolor` and value `<1/2,1/2,1/2>`

Pre/Post conds → `(isrgbcolor)`
Example `cuboid:<1,1,1> color gray`

`Green` plasm *object* of class `rgbcolor` and value `<0,1,0>`

Pre/Post conds → `(isrgbcolor)`
Example `cuboid:<1,1,1> color green`

`Isinto` predicate to test set-membership of `x` into the [`lower`,`upper`] interval

Pre/Post conds `(lower,upper::isnum)(x::isnum)` → `(isbool)`
Example `isinto:<0,1>:0.5` ≡ `true`

`Light` is used to apply a `genericlight` object to `pol` complex

Pre/Post conds `(pol::ispol; light::isgenericlight)` → `(islight)`
Example `(sqr ~ q ~ #:10):1 light spot:<red, <10,15,20>,<0,0,-1>>`

`Magenta` plasm *object* of class `rgbcolor` and value `<1,0,1>`

Pre/Post conds → `(isrgbcolor)`
Example `cuboid:<1,1,1> color magenta`

`Material` annotates `pol` with `mat` object value for `'VRMLmaterial'` property

Pre/Post conds `(pol::ispol; mat::isbasematerial)` → `(ispol)`
Example `cuboid:<1,1,1> material Transparentmaterial:<green, 0.4>`

`Orange` plasm *object* of class `rgbcolor` and value `<1,1/2,0>`

Pre/Post conds → `(isrgbcolor)`
Example `cuboid:<1,1,1> color orange`

`Purple` plasm *object* of class `rgbcolor` and value `<1/2,0,1/2>`

Pre/Post conds → `(isrgbcolor)`
Example `cuboid:<1,1,1> color purple`

`Red` plasm *object* of class `rgbcolor` and value `<1,0,0>`

Pre/Post conds → `(isrgbcolor)`
Example `cuboid:<1,1,1> color red`

`Simplecamera` specialization of `BaseCamera` using defaults for common params

Pre/Post conds `(position::or ~ [isvect, isnull]; description::isstring)`
→ `(issimplecamera)`
Example `(@1~cuboid):<1,1,1> camera simplecamera:<<0.5,0.5,2.5>,'cam'>`

`Simplematerial` specialization of `basematerial` using defaults for common params

Pre/Post conds `(color::isrgbcolor)` → `(issimplematerial)`
Example `circle:1:<32,1> material simplematerial:blue`

`Simpletexture` specialization of `basetexture` with no repetitions

Pre/Post conds `(url::isstring)` → `(issimpletexture)`
Example `cuboid:<2,3> texture simpletexture:'path/monnalisa.jpg'`

`Spot` function that returns a plasm *object* of class `basespotlight`

Pre/Post conds `(color,location,orientation::tt)` → `(isbasespotlight)`
Example `(sqr ~ q ~ #:10):1 light spot:<red, <10,15,20>,<0,0,-1>>`

`Texture` annotates `pol` with `tex` value for the `'VRMLtexture'` property

Pre/Post conds `(pol::ispol; tex::isfulltexture)` → `(ispol)`
Example `cuboid:<2,3> texture simpletexture:'path/monnalisa.jpg'`

`Transparentmaterial` specialization of `basematerial` with default values

Pre/Post conds `(color::isrgbcolor; transparency::isinto:<0,1>)` →
`(istransparentmaterial)`
Example `ndimsphere:3 material transparentmaterial:<red, 0.7>`

`White` plasm *object* of class `rgbcolor` and value `<1,1,1>`

Pre/Post conds → `(isrgbcolor)`
Example `cuboid:<1,1,1> color white`

`Yellow` plasm *object* of class `rgbcolor` and value `<1,1,0>`

Pre/Post conds → `(isrgbcolor)`
Example `cuboid:<1,1,1> color yellow`

B.4 curves Library

`Basehermite` returns the graph of the cubic Hermite basis polynomials

Pre/Post conds `(domain::ispol)` → `(ispol)`
Example `basehermite:(intervals:1:20)`

`Beziercurve` generator of coordinate functions of Bézier curves of arbitrary degree.
Alias for `Bezier:S1`

Pre/Post conds `(controlpoints::ismat)` → `(isseqof:isfun)`
Example `beziercurve:<<0,4,1>,<7,5,-1>,<8,5,1>,<12,4,0>>`

`Bezierstripe` generator of a 2D stripe generated by a Bézier curve of any degree

Pre/Post conds `(controlpoints::ismat; width::isreal;n::isintpos)` → `(ispol)`
Example `Bezierstripe:<<<0,0>,<7,5>,<8,5>,<12,4>>,1,20>`

`Curve2mapvect` coerces a vector function into a sequence of real maps

Pre/Post conds `(curve::isfun)` → `(isseqof:isfun)`
Example `curve2mapvect:[cos ∼ s1, sin ∼ s1]`

`Derbernsteinbase` derivative of the Bernstein/Bézier basis polynomials of degree `n`

Pre/Post conds `(n::isintpos)` → `(isseqof:isfun)`
Example `derbernsteinbase:2`

`Derbernstein` derivative of Bernstein polynomial of degree n and index i, $0 \leq i \leq n$

Pre/Post conds `(n::isint)(i::isint)` → `(isfun)`
Example `derbernstein:3:0`

`Derbezier` generator of coordinate functions of the derivative of a Bézier curve

Pre/Post conds `(controlpoints::ismat)` → `(isseqof:isfun)`
Example `derbezier:<<0,0>,<7,5>,<8,5>,<12,4>>`

`Hermite` generator of the coordinate functions of a cubic Hermite curve

Pre/Post conds `(handles::ismat)` → `(isseqof:isfun)`
Example `MAP:(Hermite:<<0,0>,<1,1>,<-3,0>,<3,0>>):(Intervals:1:20)`

`Norm2` generator of the coordinate functions of the normal unit field to a 2D curve

Pre/Post conds `(curve::and ∼ [ispair,isseqof:isfun])`
→ `(and ∼ [ispair,isseqof:isfun])`
Example `(norm2 ∼ derbezier):<<0,0>,<1,1>,<-3,0>,<3,0>>`

`Rationalbezier` rational Bézier curves of arbitrary degree (weights on last coord)

Pre/Post conds `(controlpoints::ismat)` ⟶ `(isseqof:isfun)`
Example `MAP:(RationalBezier:<<1,0,1>,[id,id,id]:(SQRT:2/2),<0,1,1>>):(Intervals:1:12)`

`Rationalblend` linear comb. of `basis` with `controlpoints`, and normalization

Pre/Post conds `(basis::isseqof:isfun)(controlpoints::ismat)` ⟶ `(isseqof:isfun)`
Example `rationalblend:(bernsteinbasis:s1:degree):controlpoints`

`Rationalize` division of coordinate functions by the last element, then dropped out

Pre/Post conds `(coords::isseqof:isfun)` ⟶ `(isseqof:isfun)`
Example `rationalize:(blend:(bernsteinbasis:s1:2):<<1,1,1>,<-3,0,1>,<3,0,1>>)`

`Rev` reversing parametrization operator $[a, b] \mapsto [b, a]$

Pre/Post conds `(a,b::isreal)` ⟶ `(isfun)`
Example `map:([cos,sin] ∼ rev:<0,pi> ∼ s1):(intervals:pi:24)`

B.5 `derivatives` Library

`Binormal` returns the coordinate functions of binormal vector function to a `curve`

Pre/Post conds `(curve::isseqof:isfun)` ⟶ `(isseqof:isfun)`
Example `binormal:(beziercurve:<<-1,2,1>,<0,1.2,3>,<0,2,-1>,<3,2,2>>)`

`Curl` returns the curl of a smooth vector field `f` computed at `x` point

Pre/Post conds `(f::isseqof:isfun)(x::ispoint)` ⟶ `(isvect)`
Example `curl:<sin∼s1,cos∼s2,s1*s3>:<0,pi,pi/6>` ≡ `<0.0,-0.52359,0.0>`

`Curvature` computes the scalar curvature function of the input curve

Pre/Post conds `(curve::isseqof:isfun)(a::ispoint)` ⟶ `(isfun)`
Example `MAP:<s1, curvature:<cos ∼ s1, sin ∼ s1>>:(intervals:(2*pi):24);`

`Divergence` returns the trace of Jacobian matrix of vector field `f`, evaluated at `x`

Pre/Post conds `(f::isseqof:isfun)(x::isseqof:isreal)` ⟶ `(isnum)`
Example
```
def g = < sin ∼ s1, cos ∼ s2, s1 * s3 >;
divergence:< s1 ∼ curl:g, s2 ∼ curl:g, s3 ∼ curl:g >:
<0.5,110.5,1> ≡ 0.0
```

`Dp` partial derivative in the i-th coordinate direction of the real function `f` of several variables, at a point `x`

Pre/Post conds `(i::isIntPos)(f::IsFun)(x::IsPoint)` ⟶ `(isfun)`
Example `dp:2:(sin ∼ s1 * sin ∼ s2):<pi/3, pi/6>:<1>` ≡ `0.75`

`Ds` i-th partial derivative of a vector function `f` of several variables

Pre/Post conds `(i::isintpos)(f::isseqof:isfun)` ⟶ `(isseqof:isfun)`
Example `MAP:(DS:1:<s1,s2,sin∼s1,sin∼s2>):((sqr ∼ intervals:pi):12)` ≡ `PolComplex<1,4>`

`D` derivative operator for scalar and vector functions of one or more variables

Pre/Post conds (f::or∼[isfun,isseqof:isfun])(u::or∼[isnum,isseqof:isnum]) → (or∼[isnum,isseqof:isnum])
Example d:sin:pi ≡ -1
CONS:(d:(beziercurve:<<-2,0>,<1,3>,<2,1>>):<1>):<0.5>≡<1,-2>

Gausscurvature returns the Gauss curvature of vector field f at point x

Pre/Post conds (f::isseqof:isfun)(x::ispoint) → (isnum)
Example gausscurvature:< s1, s2, sin∼s1 * sin∼s2 >:<0,0> ≡ -1.0

Grad gradient (linear map) of a scalar function f of several variables at point a

Pre/Post conds (f::isfun)(a::ispoint) → (isseqof:isfun)
Example cons:(grad:(sin∼s1*sin∼s2):<pi/3,pi/-2>):<1,1> ≡ <-0.5,0>

Gradient gradient (vector) of a scalar field point a

Pre/Post conds (f::isfun)(a::ispoint) → (isvect)
Example Gradient:(s1*s1 - s2*s2):<0.25,0.3> ≡ <0.5,-0.6>

Jacobian returns the Jacobian matrix at point a of a vector field f

Pre/Post conds (f::isseqof:isfun)(a::ispoint) → (ismat)
Example Jacobian:<(s1*s1 - s2*s2)/K:2, (s1*s1 + s2*s2)/K:2>:<0.25,0.3>
≡ <<0.25,-0.3>,<0.25,0.3>>

Normalmap normal vector field map

Pre/Post conds (f::isseqof:isfun; dom::ispol) → (ispol)
Example normalmap:<s1,s2,sin ∼ s1*sin ∼ s2>:((sqr ∼ intervals:pi):5)

N normal field operator, i.e. the normalized vector product of the (tangent) fields generators DS:1 and DS:2

Pre/Post conds (f::isseqof:isfun) → (isseqof:isfun)
Example (cons∼n):<s1,s2,sin∼s1*sin∼s2>:<0,0> ≡ <0,0,1.0>

Principalnormal intrinsic vector for a curve given by coordinate functions

Pre/Post conds (curve::isseqof:isfun)(a::ispoint) → (isfun)
Example MAP:(principalnormal:<cos ∼ s1, sin ∼ s1>):(intervals:(pi):12)

Tangent intrinsic vector for a curve given by coordinate functions

Pre/Post conds (curve::isseqof:isfun)(a::ispoint) → (isfun)
Example MAP:((tangent ∼ bezier:s1):<<0,0,0>,<1,0,0>,<1,1,0>,<1,1,1>>):
(intervals:1:20)

X *i*-th partial derivative of a scalar function f of several variables at point x

Pre/Post conds (i::isintpos)(f::isfun)(x::ispoint) → (isnum)
Example cons:(aa:(x:2):<s1,s2,sin ∼ s1*sin ∼ s2>):<0,0> ≡ <0,1.0,0>

B.6 drawtree Library

Drawtree returns a 2D complex giving a picture of the input hierarchical structure

Pre/Post conds (levels::isseqof:isseq) → (ispol)
Example drawtree:<<<′1′>>,<<′2′,′3′,′4′,′5′>>, <<′6′,′7′>,<>,<′8′,′9′,′10′>,<′11′>>>

B.7 `flash` Library

`Acolor` annotates the `pol` parameter with the `color` value, of `rgba` type

Pre/Post conds	`(pol::ispol; color::isrgbacolor) → (ispol)`
Example	`cuboid:<1,1> acolor rgbacolor:<0,1,0,0.5>`

`Actor` returns an animation level starting at time (`timestop - len:framelist`)

Pre/Post conds	`(framelist::isseq)(timestop::isintpos) → (isseqof:ispol)`
Example	

`Fillcolor` defines the `rgba` color to fill a 2D geometric object `pol`

Pre/Post conds	`(pol::ispol; col::isrgbacolor) → (ispol)`
Example	`cuboid:<1,1> fillcolor RGBAcolor:<1,0,0,1>`

`Frame` displays the `obj` object within the $[t_1, t_2]$ time interval

Pre/Post conds	`(obj::ispol)(t1::isintpos)(t2::isintpos)`
	`→ (isseqof:ispol ∼ S1)`

Example	`frame:(cuboid:<1,1>):1:32`

`Linecolor` used to define the color of 1-skeleton of a 2D geometric object `pol`

Pre/Post conds	`(pol::ispol; col::isrgbacolor) → (ispol)`
Example	`cuboid:<1,1> linecolor rgbacolor:<0,0.1,1,0.8>`

`Linesize` used to define the drawing size of 1-skeleton of a 2D object `pol`

Pre/Post conds	`(pol::ispol; pixelsize::isint) → (ispol)`
Example	`out fillcolor rgbacolor:< 0,1,1,0.5 > linecolor rgbacolor:< 0,0,1,1 > linesize 1`

B.8 `general` Library

`Alias` to return the data value paired with an integer `key` in an associative `table`

Pre/Post conds	`(key::isint)(table::isseqof:ispair) → (tt)`
Example	`alias:2:<<-1,35>,<2,1..3>,<5,41>,<7,43>,<18,44>> ≡ <1,2,3>`

`Assoc` returns the pair whose key has smallest distance from the input key. Pairs are maintained in increasing key order

Pre/Post conds	`(key::isint) → (ispair)`
Example	`alias:2:<<-1,35>,<2,1..3>,<5,41>,<7,43>,<18,44>> ≡ <2,1..3>`

`Bigger` is a binary operator that returns the greater of arguments

Pre/Post conds	`(pair::and ∼ [ispair, lift:or:(AA:isseqof:<isnum, ischar, isstring>)])`
	`→ (or ∼ [isnum,ischar,isstring])`

Example	`bigger:<-122,22E2> ≡ 2200.0`
	`bigger:<′John′,′Robert′> ≡ ′Robert′`

`Biggest` binary operator that returns the greatest of `args` values

Pre/Post conds	`(args::lift:or:(AA:isseqof:<isnum, isfun, ismat, ispol>))`
	`→ (or ∼ [isnum,ischar,isstring])`

Example	`biggest:<′fred′,′wilma′,′barney′,′lucy′> ≡ ′wilma′`

`Cart` returns the Cartesian product of two sequences

Pre/Post conds `(a,b::isseqof::tt)` → `(isseqof:ispair)`
Example `cart:<<1,2,3>,<′a′,′b′>>` ≡
`<<1,′a′>,<1,′b′>,<2,′a′>,<2,′b′>,<3,′a′>,<3,′b′>>`

`Choose` is a generator of binomial numbers

Pre/Post conds `(n,k::isnat)` → `(isintpos)`
Example `6 choose 2` ≡ `15`

`Fact` is a generator of the function $n \mapsto n!$

Pre/Post conds `(n::isnat)` → `(isintpos)`
Example `fact:5` ≡ `120`

`Filter` used for filtering a `sequence` according to a `predicate` on elements

Pre/Post conds `(predicate::isfun)(sequence::isseq)` → `(isseq)`
Example `filter:(LE:0):<-101,23,0,-37.02,0.1,84>` ≡ `<23,0.1,84>`

`In` predicate to test the set-membership of `element` ∈ `set`

Pre/Post conds `(set::isseq)(element::tt)` → `(isbool)`
Example `in:<′a′,′e′,′i′,′o′,′u′>:′z′` ≡ `false`

`Iseven` predicate to test if `n` is an even number

Pre/Post conds `(n::isint)` → `(isbool)`
Example `iseven:13` ≡ `false`

`Isge` binary predicate to test if `b` ≥ `a` in some suitable ordering

Pre/Post conds `(a,b::tt)` → `(isbool)`
Example `isge:<′Fred′, ′Wilma′>` ≡ `true`

`Isgt` binary predicate to test if `b` > `a` in some suitable ordering

Pre/Post conds `(a,b::tt)` → `(isbool)`
Example `isgt:<′Fred′, ′Wilma′>` ≡ `true`

`Isle` binary predicate to test if `b` ≤ `a` in some suitable ordering

Pre/Post conds `(a,b::tt)` → `(isbool)`
Example `isge:<′Fred′, ′Wilma′>` ≡ `false`

`Islt` binary predicate to test if `b` < `a` in some suitable ordering

Pre/Post conds `(a,b::tt)` → `(isbool)`
Example `isge:<′Fred′, ′Wilma′>` ≡ `false`

`Isnat` unary predicate to test if a number `n` is a natural number. A natural number is any of the numbers $0, 1, 2, 3, \ldots$

Pre/Post conds `(n::isnum)` → `(isbool)`
Example `isnat:-1233` ≡ `false`

`Isodd` predicate to test if `n` is an odd number

Pre/Post conds `(n::isint)` → `(isbool)`
Example `isodd:13` ≡ `true`

`Mean` computes the arithmetic mean of a sequence `seq` of numbers

Pre/Post conds `(seq:isseqof:isnum)` → `(isnum)`
Example `mean:<10,22,5,16,4>` ≡ `57/5`

`Mk` returns a 0D polyhedron starting from the coordinates of a point `x` $\in \mathbb{E}^d$, $d \geq 1$

Pre/Post conds	`(x:ispoint)` → `(and ∼ [ispol,c:eq:0 ∼ dim])`
Example	`(c:eq:0 ∼ dim):(mk:<1,0,0,0>)` ≡ `true`

`Mod` binary operator that returns the remainder of the division of `a` by `b`

Pre/Post conds	`(a,b::isnum)` → `(isnum)`
Example	`mod:<13.5,9.2>` ≡ `4.3`

`Pascaltriangle` returns the first $n + 1$ rows of the Pascal triangle of binomial numbers

Pre/Post conds	`(n::isnat)` → `and ∼ aa:(isseqof:isintpos)`
Example	`pascalTriangle:3` ≡ `<<1>,<1,1>,<1,2,1>,<1,3,3,1>>`

`Permutations` returns the set of permutations of elements of the input `seq`

Pre/Post conds	`(seq::isseqof:tt)` → `(and ∼ aa:(isseqof:tt))`

Example	`permutations:<1,2,3>` ≡
	`<<1,2,3>,<1,3,2>,<2,1,3>,<2,3,1>,<3,1,2>,<3,2,1>>`
	`permutations:<'a','b'>` ≡ `<<'a','b'>,<'b','a'>>`

`Powerset` returns the powerset $2^{\texttt{set}}$ of the input `set`

Pre/Post conds	`(set::isseqof:tt)` → `(and ∼ aa:(isseqof:tt))`
Example	`powerSet:<1,2,3>` ≡ `<<1,2,3>,<1,2>,<1,3>,<1>,<2,3>,<2>,<3>,<>>`

`Progressivesum` operator to compute the map $\{a_i \in \texttt{Num}\} \mapsto \{b_i = \sum_{j=1}^{i} a_j\}$

Pre/Post conds	`(input::isseqof:isnum)` → `(isseqof:isnum)`
Example	`ProgressiveSum:<1,3,5,7,9,11>` ≡ `<1,4,9,16,25,36>`

`Q` generalized alias for `QUOTE`, that is applicable to either numbers or sequences

Pre/Post conds	`(params::and∼[or∼[isnum,isseqof:isnum],and∼ aa:(c:neq:0)])` → `(and∼[ispol,c:eq:<1,1>∼[dim,rn]])`
Example	`ispol:(q:1)` ≡ `true; (ispol ∼ q ∼ ##:10):<1,-2>` ≡ `true`

`Rtail` returns the input `seq`, but the last element

Pre/Post conds	`(seq::isseqof:tt)` → `(isseqof:tt)`
Example	`rtail:<'a','b','c','e'>` ≡ `<'a','b','c'>`

`Setand` set intersection between the argument sequences

Pre/Post conds	`(set_a, set_b::isseqof:tt)` → `(isseqof:tt)`
Example	`<id,11,'Lucy',12,'Bart'> setand <'Bart','Homer',11,id>` ≡ `<id,11,'Bart'>`

`Setdiff` set difference between the argument sequences

Pre/Post conds	`(set_a, set_b::isseqof:tt)` → `(isseqof:tt)`
Example	`<id,11,'Lucy',12,'Bart'> setdiff <'Bart','Homer',11,id>` ≡ `<'Lucy',12>`

`Setor` set union between the argument sequences

Pre/Post conds	`(set_a, set_b::isseqof:tt)` → `(isseqof:tt)`
Example	`<id,11,'Lucy',12,'Bart'> setor <'Bart','Homer',11,id>` ≡ `<'Lucy',12,'Bart','Homer',11,id>`

`Setxor` symmetric difference (XOR) between the argument sequences

Pre/Post conds `(set_a, set_b::isseqof:tt)` ⟶ `(isseqof:tt)`
Example `<id,11,'Lucy',12,'Bart'> setxor <'Bart','Homer',11,id>` ≡
`<'Lucy',12,'Homer'>`

`Sort` merge-sort on numbers, characters and strings, with order depending on `pred`
Pre/Post conds `(pred::isfun)(seq::isseqof:tt)` ⟶ `(isseqof:tt)`
Example `sort:isgt:<'fred','wilma','barney','lucy'>` ≡
`<'barney','fred','lucy','wilma'>`
`sort:greater:<8,2,4,2,3,11,-5>` ≡ `<11,8,4,3,2,2,-5>`

`Smaller` *binary* operator that returns the smaller argument (in a proper ordering!)
Pre/Post conds `(args::or~[ispairof:isnum,ispairof:isstring])`
⟶ `(or ~ [isnum,isstring])`
Example `smaller:<-122,22E2>` ≡ `-122`
`smaller:<'John','Robert'>` ≡ `'John'`

`Smallest` returns the smallest element of the `args` input sequence
Pre/Post conds `(args::or~[isseqof:isnum,isseqof:isstring])`
⟶ `(or ~ [isnum,isstring])`
Example `smallest:<'fred','wilma','barney','lucy'>` ≡ `'barney'`

`Sqr` unary operator that returns the *square* of the `arg` argument
Pre/Post conds `(arg::or ~ [isnum, isfun])` ⟶ `(or ~ [isnum, isfun])`
Example `sqr:sin:(PI/2)` ≡ `(sin * sin):(PI/2)` ≡ `1.0`
`sqr:4` ≡ `16`

`Uk` UnmaKe. Returns the point in $\mathbb{E}^d$ corresponding to a 0D geometric object
Pre/Post conds `(arg::and ~ [ispol,c:eq:0 ~ dim])` ⟶ `(ispoint)`
Example `(uk ~ embed:2 ~ mk):<1,1,1>` ≡ `<1.0,1.0,1.0,0.0,0.0>`

B.9 `myfont` Library

`Fontcolor` applies the `col` parameter to the polyhedral objects in `myfont` font
Pre/Post conds `(col::isrgbcolor)` ⟶ `(iseqof:ispol)`
Example `fontcolor:red`

`Fontheight` constant value, giving the height of characters in `myfont`. Default is 6
Pre/Post conds ⟶ `(isnum)`
Example `s:<1,2>:< textwidth/fontwidth, textheight/fontheight >`

`Fontspacing` constant value, giving the spacing of character boxes in `myfont`. Default is 2
Pre/Post conds ⟶ `(isnum)`
Example `t:1:(fontwidth + fontspacing)`

`Fontwidth` constant value, giving the width of characters in `myfont`
Pre/Post conds ⟶ `(isnum)`
Example `s:<1,2>:< textwidth/fontwidth, textheight/fontheight >`

`Myfont` is the name of the internal data structure where the character shapes are stored as geometric values. The drawable ASCII subset is $[32, 126]$
Pre/Post conds ⟶ `(isseqof:ispol)`
Example `sel:(ord:'a' - 31):myfont` ≡ `PolComplex<1,2>`

B.10 `operations` Library

`Depth_sort` returns a depth-sort ordering of the 2-faces of a polyhedral `scene`	
Pre/Post conds	`(scene::ispol) → (isseqof:ispol)`
Example	`(depth_sort ~ @2 ~ r:<1,2>:(pi/6) ~ cuboid):<1,1,1>`
`Depth_test` is the Newell's binary predicate used to compare two 2-faces	
Pre/Post conds	`(a,b::and ~ [ispol,c:eq:<2,3> ~ [dim,rn]]) → (isbool)`
Example	`(depth_test ~ [t:3:1, id] ~ embed:1 ~ simplex):2`
`Explode` 3D "explosion" operator of the `scene` parameter	
Pre/Post conds	`(sx,sy,sz::isreal) (scene::isseqof:ispol) → (isseqof:ispol)`

Example	`def hole = ((id - s:<1,2>:<0.5,0.5>) ~ mxmy ~ cuboid):<2,2,2>;`
	`(struct ~ explode:<1,1,1.5> ~ extract_polygons):hole`

`Extract_bodies` returns the 3D cells from the `scene` parameter	
Pre/Post conds	`(scene::and ~ [ispol,ge:3 ~ dim]) → (isseqof:ispol)`
Example	`extract_bodies:(q:<1,-1,1,-1,1> * q:1 * q:10)`
`Extract_polygons` returns the 2D cells from the `scene` parameter	
Pre/Post conds	`(scene::and ~ [ispol,ge:2 ~ dim]) → (isseqof:ispol)`
Example	`extract_polygons:(q:<1,-1,1,-1,1> * q:1 * q:10)`
`Extract_wires` returns the 1D cells from the `scene` parameter	
Pre/Post conds	`(scene::and ~ [ispol,ge:1 ~ dim]) → (isseqof:ispol)`
Example	`extract_wires:(q:<1,-1,1,-1,1> * q:1 * q:10)`
`Extrude` with `h` displacement, the *n*-th convex cell in a `pol` complex	
Pre/Post conds	`(n::isintpos; pol::ispol; h::isrealpos) → (ispol)`
Example	`extrude:<2,q:<1,-1,1,-1,1> * q:1,10>`
`Extrusion` *generalized* operator, with `h` steps and `alpha` angle, of `pol` parameter	
Pre/Post conds	`(alpha::isreal)(h::isint)(pol::ispol) → (ispol)`
Example	`extrusion:(pi/18):1:(q:1 * q:1)`
`Ex` *right* extrusion, with `x2 - x1` height and `x1` starting	
Pre/Post conds	`(x1,x2::isreal)(pol::ispol) → (ispol)`
Example	`ex:<0.5,1>:(q:1 * q:1)`
`Lex` *linear* extrusion, with `x2 - x1` height and shearing, and `x1` starting	
Pre/Post conds	`(x1,x2::isreal)(pol::ispol) → (ispol)`
Example	`lex:<0.5,1>:(q:1 * q:1)`
`Lxmy` *left x, middle y* alignment operator. Moves the origin of the local frame	
Pre/Post conds	`(ispol) → (ispol)`
Example	`lxmy:(cuboid:<5,5>)`
`Mirror` returns the `obj` parameter reflected on the *d*-th coordinate direction	
Pre/Post conds	`(d::isintpos)(obj::ispol) → (ispol)`
Example	`(@1 ~ struct ~ [id, mirror:1] ~ simplex):2`
`Minkowski` sum of `p` complex with the zonotope defined by `vects` sequence	

Pre/Post conds `(vects::isseqof:isvect)(p::ispol) ⟶ (ispol)`

Example `minkowski:<<-1/2, SQRT:2/-2>,<-1/2, SQRT:2/2>,<1, 0>>: ((@1 ∼ cuboid):<5,5>)`

`Multextrude` a polyhedral complex, by associating the facets of `p` with the `h` heights

Pre/Post conds `(p::ispol) (h::isseqof:isreal) ⟶ (ispol)`

Example `multextrude:(q:<1,-1,1,-1,1> * q:1):<1.0,2.0,3.0>`

`Mxby` *middle x, bottom y* alignment operator. Moves the origin of the local frame

Pre/Post conds `(ispol) ⟶ (ispol)`

Example `mxby:(cuboid:<5,5>)`

`Mxmy` *middle x, middle y* alignment operator. Moves the origin of the local frame

Pre/Post conds `(ispol) ⟶ (ispol)`

Example `mxmy:(cuboid:<5,5>)`

`Mxty` *middle x, top y* alignment operator. Moves the origin of the local frame

Pre/Post conds `(ispol) ⟶ (ispol)`

Example `mxty:(cuboid:<5,5>)`

`Offset` geometric operator. Implemented as the composition of suitable extrusions, followed by projection

Pre/Post conds `(v::isvect)(pol::ispol) ⟶ (ispol)`

Example `offSet:<0.1,0.2,0.1>:((@1 ∼ cuboid):<1,1,1>)`

`Optimize` is used to flatten the internal HPC data structure. The annotations of parts with properties are lost. Alias for `mkpol ∼ ukpol`

Pre/Post conds `(ispol) ⟶ (ispol)`

Example `(optimize ∼ struct ∼ [id, t:1:1, t:2:1]):(simplex:2)`

`Planemapping` plane mapping through three points `p0`, `p1` and `p2`

Pre/Post conds `(p0,p1,p2::ispoint) ⟶ (ispol)`

Example `map:(planemapping:<<0,0,0>,<1,0,0>,<1,1,1>>):(cuboid:<1,1>)`

`Polar` generator of the polar set of a n-dimensional convex

Pre/Post conds `(ispol) ⟶ (ispol)`

Example `(polar ∼ simplex):4 ≡ polcomplex<4,4>`

`Presort` executes the preliminary z-ordering when depth-sorting a polygon sequence

Pre/Post conds `(pols::isseqof:(c:eq:<2,3> ∼ [dim,rn])) ⟶ (isseqof:ispol)`

Example `(presort ∼ [t:3:1, id] ∼ embed:1 ∼ simplex):2`

`Project` projection operator, that removes the last `m` coordinates of `pol`

Pre/Post conds `(m::isintpos)(pol::ispol) ⟶ (ispol)`

Example `(Project:1 ∼ @1 ∼ R:<1,4>:(PI/6) ∼ R:<1,3>:(PI/7)): (cuboid:<1,1,1,1>);`

`Rxmy` *right x, middle y* alignment operator. Moves the origin of the local frame

Pre/Post conds `(ispol) ⟶ (ispol)`

Example `rxmy:(cuboid:<5,5>)`

`Schlegel2d` returns 2D Schlegel diagrams of 3-polytopes, projected from $(0, 0, d)$

Pre/Post conds `(d::isreal)(pol::ispol) ⟶ (ispol)`

Example `(@1 ∼ schlegel2D:0.2 ∼ T:3:2.5 ∼ CUBOID):<1,1,1>`

`Schlegel3d` returns 3D Schlegel diagrams of 4-polytopes, projected from $(0, 0, 0, d)$

Pre/Post conds `(d::isreal)(pol::ispol) → (ispol)`

Example `(schlegel3d:0.2 ∼ @1 ∼ t:<1,2,3,4>:<-1,-1,-1,1> ∼ cuboid):<2,2,2,2>`

`Sex` *screw* extrusion of `pol`, with `h` steps, `x2 - x1` total angle, and `x1` starting angle

Pre/Post conds `(x1,x2::isreal)(h::isintpos)(pol::ispol) → (ispol)`

Example `sex:<0,pi>:12:(q:1 * q:1)`

`Solidify` mapping boundary to interior; multidimensional operator

Pre/Post conds `(and ∼ [ispol, c:eq:1 ∼ (rn - dim)]) → (ispol)`

Example `Solidify ∼ STRUCT ∼ AA:polyline`

`Splitcells` extracts the convex d-cells of the d-dimensional `scene`

Pre/Post conds `(scene::ispol) → (isseqof:ispol)`

Example `(struct ∼ explode:<1,1,1.5> ∼ splitcells ∼ @2):hole`

`Splitpols` extracts the polyhedral d-cells of the d-dimensional `scene`

Pre/Post conds `(scene::ispol) → (isseqof:ispol)`

Example `(struct ∼ explode:<1.5,1.5,1.5> ∼ splitpols ∼ @2):hole`

`Sweep` returns the point set swept by `pol` when moved by a `v` displacement

Pre/Post conds `(v::isvect)(pol::ispol) → (ispol)`

Example `sweep:<10,0>:(circle:1:<24,1>)`

B.11 `primitives` Library

`Displaygraph` graph generator for `f` : $\mathbb{R} \to \mathbb{R}$, where `n` is a marker index

Pre/Post conds `(n::isint)(f::isfun)(sample::isseqof:isnum) → (ispol)`

Example `(displaygraph:1:sin ∼ c:al:0 ∼ progressivesum ∼ #:32):(pi/16)`

`Isclosedshape` predicate to test if the arg `shape` is either closed or not

Pre/Post conds `(arg::isshape) → (isbool)`

Example `isclosedshape:<<5,3,-2.5,-2.5,-3>,<-2,4,-2,2,-2>> ≡ true`

`Iscloseto` predicate to test is the `arg` distance from `x` is less than `1e-4`

Pre/Post conds `(x::isnum)(arg::isnum) → (isbool)`

Example `iscloseto:0:0.001 ≡ false; iscloseto:0:1e-6 ≡ true`

`Isorthoshape` predicate to test if the arg `shape` is made by orthogonal segments

Pre/Post conds `(arg::isshape) → (isbool)`

Example `isorthoshape:<<5,3,-2.5,-2.5,-3>,<-2,4,-2,2,-2>> ≡ false`

`Isshape` predicate to test if `arg` is a *shape* (see Section 7.3)

Pre/Post conds `(arg::and ∼ [ispair,ismat]) → (isbool)`

Example `isshape:<<5,3,-2.5,-2.5,-3>,<-2,4,-2,2,-2>> ≡ true`

`Mapshapes` returns a sampling of segment between two shapes, made compatible

Pre/Post conds `(p,q::isshape) → (isseqof:isshape)`

Example `mapShapes:< <<5,0,-5>,<0,5,-5>>, <<0,1,0,-2,0,3,0,-4,0,5>,<-1,0,2,0,-3,0,4,0,-5,0>> >`

`Markersize` constant value used to define the marker size. Default value is 0.05

Pre/Post conds → `(isnum)`
Example `DEF MarkerSize = 0.10`

`Mesh` returns a d-dimensional mesh with hyperparallelepiped cells

Pre/Post conds `(seqs::and ∼ aa:(isseqof:isnum))` → `(ispol)`
Example `(@1 ∼ mesh):<<1,2,1,2,1>,<1,2,1,2,1,2,1>>`

`Points2shape` transforms a 2D point `seq` into a *shape* instance

Pre/Post conds `(seq::and ∼ [ismat, ispair ∼ trans])` → `(isshape)`
Example `(points2shape):<<0,0>,<3,0>,<2,4>,<1,2>> ≡ <<3,-1,-1>,<0,4,-2>>`

`Polypoint` point primitive generator

Pre/Post conds `(points::ismat)` → `(ispol)`
Example `(join ∼ polypoint):<<0,-0.23>,<20,0>,<5.77,11>,<20,-10>>`

`Polyline` generator of 1D connected complexes from the `points` sequence

Pre/Post conds `(points::ismat)` → `(ispol)`
Example `polyline:<<1,0,-5.1>,<1,1.2,0>,<0,2,-2>,<-1,-1.25,4>>`

`Polymarker` returns a complex of *markers* generated at specified `points`

Pre/Post conds `(markertype::isintpos)(points::ismat)` → `(ispol)`
Example `polymarker:3:`
`((aa:[id,sin] ∼ c:al:0 ∼ progressivesum ∼ #:24):(pi/12))`

`Quadmesh` generator of a mesh of quadrilaterals from an array of `points`

Pre/Post conds `(points::ismatof:ispoint)` → `(ispol)`
Example `quadmesh:< <<0,0>,<1,0>,<2,0>>, <<0,1>,<1,1>,<2,1>>,`
`<<0,2>,<1,2>,<2,2>> >`

`Shape2points` operator to return a point sequence from the `arg` shape

Pre/Post conds `(arg::isshape)` → `(isseqof:ispoint)`
Example `shape2points:<<1,2,3>,<0,1,0>> ≡ <<0,0>,<1,0>,<3,1>,<6,1>>`

`Shape2pol` operator to return a polyhedral complex from the `arg` shape

Pre/Post conds `(arg::isshape)` → `(ispol)`
Example `shape2pol:<<1,2,3>,<0,1,0>> ≡ polcomplex<1,2>`

`Shapeclosed` mapping from a d-shape to a $(d+1)$-shape, that adds a final tangent vector to close the `arg` shape

Pre/Post conds `(arg::isshape)` → `(isshape)`
Example `shapeclosed:<<1,2,3>,<0,1,0>> ≡ <<1,2,3,-6>,<0,1,0,-1>>`

`Shapecomb` operator to linearly combine the input shapes, returning `ap` + `bq`

Pre/Post conds `(a,b::isreal; p,q::isshape)` → `(isshape)`
Example `shapecomb:<0.5,0.5,<<1,0,1>,<2,-1,3>>,<<0,2,2>,<-0.5,-1,0>>>`

`Shapediff` difference operator between `p` and `q` shapes

Pre/Post conds `(p,q::isshape)` → `(isshape)`
Example `<<1,0,1>,<2,-1,3>> shapediff <<0,2,2>,<-0.5,-1,0>>`

`Shapedist` Euclidean distance computation between `p` and `q` shapes

Pre/Post conds `(p,q::isshape)` → `(isnum)`
Example `<<1,0,1>,<2,-1,3>> shapedist <<0,2,2>,<-0.5,-1,0>> ≡ 4.60977`

`Shapeinbetweening` returns the polyhedral complex of `n` shapes on the s
Pre/Post conds `(tx::isreal)(n::isint)(p,q::isshape)` → `(ispol)`
Example `ShapeInBetweening:0:4<<<1,0,1>,<2,-1,3>>,<<0,2,2>,<-0.5,-1,0>>>`

`Shapeinf` returns the inferior shape of the `p` input shape
Pre/Post conds `(p::isshape)` → `(isshape)`
Example `(shape2pol ∼ shapeinf):<<5,3,-2.5,-2.5,2.5>,<0,4,-2,2,-2>>`

`Shapejoin` joins two shapes and returns a shape value
Pre/Post conds `(p,q::isshape)` → `(isshape)`
Example `shapejoin:<<<1,0,1>,<2,-1,3>>,<<0,2,2>,<-0.5,-1,0>>>`

`Shapelen` returns the sum of lengths of tangent vectors of `p`
Pre/Post conds `(p::isshape)` → `(isnum)`
Example `shapelen:<<1,0,1>,<2,-1,3>> ≡ 6.39834563766817`

`Shapenormal` returns a shape whose tangent vectors are normal to those of `p`
Pre/Post conds `(p::isshape)` → `(isshape)`
Example `(struct ∼ aa:shape2pol ∼ [id,shapenormal]):<<1,0,1>,<2,-1,3>>`

`Shapenorm` returns the Euclidean norm of p as a vector in $\mathbb{R}^{2n}$
Pre/Post conds `(p::isshape)` → `(isnum)`
Example `shapenorm:<<1,0,1>,<2,-1,3>> ≡ 4`

`Shapeprod` product of the `p` (shape) vector times the `alpha` scalar
Pre/Post conds `(alpha::isreal; p::isshape)` → `(isshape)`
Example `shapeprod:<3,<<1,0,1>,<2,-1,3>>> ≡ <<3,0,3>,<6,-3,9>>`

`Shaperot` rotation of angle α of the `p` shape
Pre/Post conds `(alpha::isreal)(p::isshape)` → `(isshape)`
Example `shaperot:(pi/6):<<1,0,1>,<2,-1,3>>`

`Shapesum` addition operation between `p` and `q` shapes in their vector space
Pre/Post conds `(p,q::isshape)` → `(isshape)`
Example `<<1,0,1>,<2,-1,3>> shapesum <<0,2,2>,<-0.5,-1,0>> ≡`
`<<1,2,3>,<1.5,-2,3>>`

`Shapesup` returns the superior shape of the `p` input shape
Pre/Post conds `(p::isshape)` → `(isshape)`
Example `(shape2pol ∼ shapesup):<<5,3,-2.5,-2.5,2.5>,<0,4,-2,2,-2>>`

`Shapezero` returns the neutral (zero) element of the vector space of n-shapes
Pre/Post conds `(n::isint)` → `(isshape)`
Example `shapezero:4 ≡ <<0,0,0,0>,<0,0,0,0>>`

`Star` 2D `star` primitive with `n` tips
Pre/Post conds `(n::isintpos)` → `(ispol)`
Example `(struct ∼ [@1 * k:(q:0.5), embed:1] ∼ star):5 ≡`
`polcomplex<2,3>`

`Trianglefan` multidimensional primitive with the first element of `verts` as pivot
Pre/Post conds `(verts::isseqof:ispoint)` → `(ispol)`
Example `trianglefan:<<0,0,0>,<1,0,0>,<1,0,4>,<0,0,4>,<0,1,4>,<0,1,0>>`

`Trianglestripe` multidimensional primitive giving a complex of oriented triangles
Pre/Post conds `(verts::isseqof:ispoint)` → `(ispol)`
Example `triangleStripe:<<0,3>,<1,2>,<3,3>,<2,2>,<3,0>,<2,1>,<0,0>,`
`<1,1>,<0,3>,<1,2>>`

B.12 `shapes` Library

`Circle`	returns on approx. with `m`×`n` quads/triangles of the 2D circle of `r` radius
Pre/Post conds	`(r::isreal)(m,n::isintpos)` → `(ispol)`
Example	`circle:1:<24,1>`
`Circumference`	approx. with `m` segments of the 2D circle boundary of unit radius
Pre/Post conds	`(m::isintpos)` → `(ispol)`
Example	`circumference:36`
`Cone`	approx. with `m` facets of the 3D cone with r radius and `h` height
Pre/Post conds	`(r, h::isreal)(n::isint)` → `(ispol)`
Example	`Cone:<1,2>:24`
`Convexhull`	multidimensional operator returning the convex hull of `points` $\subset \mathbb{E}^d$
Pre/Post conds	`(points::ismat)` → `(ispol)`
Example	`convexhull:<<0,0,0,0>,<1,0,0,0>,<0,1,0,0>,<0,0,1,0>,<0,0,0,1>>`
`Crosspolytope`	returns the d-dimensional `crossPolytope`
Pre/Post conds	`(d::isintpos)` → `(ispol)`
Example	`crossPolytope:3`
`Cube`	generator of the 3D hexahedron of given `side`, with a vertex on the origin
Pre/Post conds	`(side::isrealpos)` → `(ispol)`
Example	`mxmy:(cube:2)`
`Dsphere`	generator of d-sphere of unit radius, with boundary facets of π`/m` resolution
Pre/Post conds	`(d::isnat)(m::isintpos)` → `(ispol)`
Example	`dsphere:3:24 material transparentmaterial:<red,0.7>`
`Dodecahedron`	constant value inscribed in the unit sphere
Pre/Post conds	→ `(ispol)`
Example	`VRML:dodecahedron:′path/out.wrl′`
`Ellipse`	approx. with $4 \times$ `m` segments of the ellipse boundary of `a,b` radiuses
Pre/Post conds	`(a,b::isreal)(m::isintpos)` → `(ispol)`
Example	`ellipse:<1/2,1>:8 * quote:<1/2>`
`Finitecone`	d-dimensional cone with given basis and apex in $(0,\ldots,0) \in \mathbb{E}^d$
Pre/Post conds	`(basis::ispol)` → `(ispol)`
Example	`finitecone:((t:<1,2,3>:<1,2,3> ∼ cuboid):<1,1,1>)`
`Fractalsimplex`	generator of recursive d-simplex with n levels
Pre/Post conds	`(d::isintpos)(n::isintpos)` → `(ispol)`
Example	`fractalsimplex:3:5`
`Hexahedron`	constant value. 3D cube inscribed in the standard unit sphere
Pre/Post conds	→ `(ispol)`
Example	`VRML:hexahedron:′path/out.wrl′`
`Icosahedron`	constant value. 3D icosahedron inscribed in the standard unit sphere
Pre/Post conds	→ `(ispol)`
Example	`VRML:icosahedron:′path/out.wrl′`

`Intervals` constructor of a uniform partition of 1D interval $[0, a]$ with `m` segments

Pre/Post conds `(a::isrealpos)(m::isintpos)`
`→ (and ∼ [ispol,c:eq:<1,1> ∼ [dim,rn]])`
Example `intervals:(2*pi):24`

`Ispolytope` predicate testing if `arg` is a polytope (bounded polyhedron) or not

Pre/Post conds `(arg::ispol) → (isbool)`
Example `ispolytope:(cuboid:<1,1,1,1>) ≡ true`

`Issimplex` predicate testing if `arg` is either a simplex or not

Pre/Post conds `(arg::ispol) → (isbool)`
Example `issimplex:(simplex:3) ≡ true`

`Mkframe` constant geometric value, returning a model of the 3D reference frame

Pre/Post conds `→ (ispol)`
Example `struct:<mkframe, cuboid:<1,1,1>>`

`Mkvector` constructor of a 3D model of vector `p2 - p1`, with `p1,p2` $\in \mathbb{E}^3$

Pre/Post conds `(p1::ispoint)(p2::ispoint) → (ispol)`
Example `mkvector:<1,0,0>:<1,1,1>`

`Mkversork` constant geometric value. Returns a 3D model of unit vector $\boldsymbol{e}_3 \in \mathbb{E}^3$

Pre/Post conds `→ (ispol)`
Example `struct:<mkversork, cuboid:<1,1,1>>`

`Ngon` constructor of 2D regular polygons with `n` sides

Pre/Post conds `(n::and ∼ [isintpos, ge:3]) → (ispol)`
Example `(struct ∼ cat):(aa:ngon:(3..8) distr t:1:2.5)`

`Octahedron` constant value. 3D Octahedron inscribed in the standard unit sphere

Pre/Post conds `→ (ispol)`
Example `VRML:octahedron:′path/out.wrl′`

`Permutahedron` generator of the d-dimensional permutahedron

Pre/Post conds `(d::isintpos) → (ispol)`
Example `permutahedron:3`

`Plane` generator of the 2-flat passing through 3 points in $\mathbb{E}^3$

Pre/Post conds `(point0, point1, point2::ispoint) → (ispol)`
Example `(s3 ∼ plane):<<0,0,0>,<1,0,0>,<1,1,1>>`

`Prism` generator of the $(d+1)$-prism with given `height` and d-dimensional `basis`

Pre/Post conds `(height::isrealpos)(basis::ispol) → (ispol)`
Example `prism:1:(crosspolytope:2)`

`Pyramid` complex of $(d+1)$-pyramids of `h` height, associated with the `basis` d-cells

Pre/Post conds `(h::isreal)(basis::ispol) → (ispol)`
Example `(struct ∼ aa:(pyramid:1) ∼ splitcells): (q:<3,3,3>*q:<3,3,3>)`

`Ring` difference of 2D circles with radiuses `r1`, `r2`, approximated with `m`×`n` steps

Pre/Post conds `(r1,r2::isrealpos)(m,n::isintpos) → (ispol)`
Example `(@1 ∼ Ring:<0.5,1>):<24,2>`

`Segment` scaled segment through two d-points `a` and `b`, with coefficient `sx`

Pre/Post conds `(sx::isreal)(a,b::ispoint)` ⟶ `(ispol)`
Example `segment:2:<<0,0,0>,<1,1,1>>`

`Simplexpile` extrusion operator for the *d*-simplex

Pre/Post conds `(cell::issimplex)` ⟶ `(ispol)`
Example `(struct ∼ [@1 ∼ simplexpile, id] ∼ simplex):2`

`Sphere` generator of 3D sphere of `r` radius, approximated with `m`×`n` facets

Pre/Post conds `(r::isrealpos)(m,n::isintpos)` ⟶ `(ispol)`
Example `Sphere:1:<12,24>`

`Tetrahedron` constant value. 3D regular tetrahedron, inscribed in the unit sphere

Pre/Post conds ⟶ `(ispol)`
Example `VRML:tetrahedron:′path/out.wrl′`

`Torus` generator of 3D torus with radiuses `r1,r2`, approximated with `m`×`n` facets

Pre/Post conds `(r1,r2::isreal) (n,m::isintpos)` ⟶ `(ispol)`
Example `torus:<1,3>:<12,24>` ≡ `PolComplex<2,3>`

`Truncone` 3D truncated cone, with `h` height, `r1,r2` radiuses and `n` lateral facets

Pre/Post conds `(r1,r2,h::isrealpos)(n::isintpos)` ⟶ `(ispol)`
Example `truncone:<2,1,2>:24`

`Tube` 3D empty tube with `h` height, `r1,r2` radiuses and 2 × `n` lateral facets

Pre/Post conds `(r1,r2,h::isreal)(n::isint)` ⟶ `(ispol)`
Example `tube:<0.8,1,2>:24`

B.13 `splines` Library

`Blend` generator of the *coordinate functions* of a specific spline curve

Pre/Post conds `(basis::isseqof:isfun) (controlpoints::ismat)` ⟶ `(isseqof:isfun)`
Example `blend:(bsplinebasis:4:<0,0,0,0,1,2,3,4,4,4,4>):`
`<<0.1,0>,<2,0>,<6,1.5>,<6,4>,<2,5.5>,<2,6>,<3.5,6.5>>`

`Bsplinebasis` non-uniform B-spline basis generator with assigned `order` and `knots`

Pre/Post conds `(order::isnat) (knots::isseqof:isreal)` ⟶ `(isseqof:isfun)`
Example `bsplinebasis:4:<0,0,0,0,1,2,3,4,4,4,4>`

`Bspline` non-uniform B-spline curve of assigned `degree`, `knots` and `points`

Pre/Post conds `(dom::and ∼ [ispol,c:eq:<1,1> ∼ [dim,rn]])(degree::isnat)`
`(knots::isseqof:isreal)(points::ismat)` ⟶ `(ispol)`
Example `bspline:(intervals:1:10):3:<0,0,0,0,1,2,3,4,4,4,4>:`
`<<0,0>,<-1,2>,<1,4>,<2,3>,<1,1>,<1,2>,<2.5,1>>`

`Cubiccardinalbasis` constant value. Cubic cardinal polynomial basis

Pre/Post conds ⟶ `(isseqof:isfun)`
Example `blend:cubiccardinalbasis:<<-1,0>,<-1,2>,<1,4>,<2,3>,<-4,2>>`

`Cubiccardinal` generator of the function argument to the `spline` operator, independent on the control points

Pre/Post conds `(segmentdomain::ispol)` → `(isfun)`
Example `spline:(cubiccardinal:(intervals:1:10)):`
`<<-3,6>,<-4,2>,<-3,-1>,<-1,1>,<1.5,1.5>,<3,4>>`

`Cubicubsplinebasis` constant value. Cubic uniform b-spline polynomial basis

Pre/Post conds → `(isseqof:isfun)`
Example `blend:Cubicubsplinebasis:<<-1,0>,<-1,2>,<1,4>,<2,3>,<-4,2>>`

`Cubicubspline` generator of the function argument to the `spline` operator, independent on the control points

Pre/Post conds `(segmentdomain::ispol)` → `(isfun)`
Example `spline:(cubicubspline:(intervals:1:10)):`
`<<-3,6>,<-4,2>,<-3,-1>,<-1,1>,<1.5,1.5>,<3,4>>`

`Deboor` generator of a non-uniform b-spline basis polynomial

Pre/Post conds `(knots::isseqof:isreal)` → `(isfun)`
Example `map:[s1,deboor:<2,3,4,5>]:(intervals:5:50)`

`Displaynubspline` returns a non-uniform b-spline, with control polygon and joints

Pre/Post conds `(degree::isnat; knots::isseq ; points::isseq)` → `(ispol)`
Example `displaynubspline:< 2,<0,0,0,1,2,3,4,5,5,5>,`
`<<0.1,0>,<2,0>,<6,1.5>,<6,4>,<2,5.5>,<2,6>,<3.5,6.5>> >`

`Displaynurbspline` returns a NURB spline, with control polygon and joints

Pre/Post conds `(degree::isnat; knots::isseq ; points::isseq)` → `(ispol)`
Example `displaynurbspline:< 2,<0,0,0,1,2,3,4,5,5,5>, <<0.1,0,1>,`
`<2,0,1>,<6,1.5,1>,<6,4,1>,<2,5.5,1>,<2,6,1>,<3.5,6.5,1>> >`

`Joints` is used to apply a marker to each sampled point of the spline curve

Pre/Post conds `(thespline::isfun)` → `(isfun)`
Example `joints:cubiccardinal:<<-3,6>,<-4,2>,<-3,-1>,<-1,1>,<1.5,1.5>>`

`Nubsplineknots` returns the 0D complex of joints between nub-spline segments

Pre/Post conds `(degree::isnat)(knots::isseq)(points::isseq)` → `(ispol)`
Example `(polymarker:2∼s1∼ukpol∼nubsplineknots:2:<0,0,0,1,2,3,4,4,4>):`
`<<0.1,0>,<2,0>,<6,1.5>,<6,4>,<2,5.5>,<2,6>>`

`Nubspline` non-uniform B-spline curve of assigned `degree`, `knots` and `points`

Pre/Post conds `(degree::isnat)(knots::isseqof:isreal)(points::ismat)`→`(ispol)`
Example `nubspline:2:<0,0,0,1,2,3,4,5,5,5>:`
`<<0,0>,<-1,2>,<1,4>,<2,3>,<1,1>,<1,2>,<2.5,1>>`

`Nurbsplineknots` returns the 0D complex of joints between NURB spline segments

Pre/Post conds `(degree::isnat)(knots::isseq)(points::isseq)` → `(ispol)`
Example `(polymarker:2∼s1∼ukpol∼nurbsplineknots:2:<0,0,0,1,2,3,4,4,4>):`
`<<0.1,0,1>,<2,0,1>,<6,1.5,1>,<6,4,1>,<2,5.5,1>,<2,6,1>>`

`Nurbspline` NURB spline curve of assigned `degree`, `knots` and `points`

Pre/Post conds `(degree::isnat)(knots::isseqof:isreal)(points::ismat)`→`(ispol)`
Example `nubspline:2:<0,0,0,1,2,3,4,5,5,5>:`
`<<0,0,1>,<-1,2,1>,<1,4,1>,<2,3,1>,<1,1,1>,<1,2,1>,<2.5,1,1>>`

`Rationalbspline` NURB spline curve of assigned `degree`, `knots` and `points`

Pre/Post conds (dom::and ∼ [ispol,c:eq:<1,1> ∼ [dim,rn]])(degree::isnat)
(knots::isseqof:isreal)(points::ismat) → (ispol)
Example rationalbspline:(intervals:1:11):3:<0,0,0,0,1,2,3,4,4,4,4>:
<<0,0,1>,<-1,2,1>,<1,4,1>,<2,3,1>,<1,1,1>,<1,2,1>,<2.5,1,1>>

Splinesampling constant number of subintervals in the partition of unit interval
Pre/Post conds → (isnum)
Example intervals:1:splinesampling

Spline generator of uniform splines starting from a curve generator function
Pre/Post conds (curve::isfun) → (isfun)
Example spline:(cubicubspline:(intervals:1:splinesampling)):
<<-3,6>,<-4,2>,<-3,-1>,<-1,1>,<1.5,1.5>,<3,4>>

B.14 strings Library

Nat2string returns a binary representation of n, i.e. a string of binary digits
Pre/Post conds (n::isnat) → (isstring)
Example nat2string:19 ≡ ′10011′

Stringtokens returns a sequence of tokens from the input string, given a set of separators
Pre/Post conds (separators::isseqof:isstring)(input::isstring)
→ (isseqof:isstring)
Example StringTokens:<′⊔′,′and′,′,′>:′Fred, Wilma, Barney and Lucy′
≡ <′Fred′,′Wilma′,′Barney′,′Lucy′>

B.15 surfaces Library

Beziermanifold generator of Bézier d-manifolds in $\mathbb{E}^n$, for any dimensions/degrees
Pre/Post conds (degrees::isseqof:isnat)(controlpoints::isseq)
→ (isseqof:isfun)
Example see Script 12.4.1

Beziersurface generator of Bézier surfaces of arbitrary degree
Pre/Post conds (controlpoints::ismatof:ispoint) → (isseqof:isfun)
Example MAP:(BezierSurface:pointArray):((sqr ∼ intervals:1):10)

Bilinearsurface generator of coord functions of a bilinear surface in $\mathbb{E}^n$
Pre/Post conds (controlpoints::ismatof:ispoint) → (isseqof:isfun)
Example def mapping = bilinearsurface:
<<<0,0,0>,<2,-4,2>>,<<0,3,1>,<4,0,0>>>;
map:mapping:((sqr∼intervals:1):10)

Biquadraticsurface generator of coord functions of a biquadratic surface in $\mathbb{E}^n$
Pre/Post conds (controlpoints::ismatof:ispoint) → (isseqof:isfun)
Example biquadraticSurface:< <<0,0,0>, <2,0,1>,<3,1,1>>,
<<1,3,-1>,<3,2,0>,<4,2,0>>, <<0,9,0>, <2,5,1>,<3,3,2>> >;
map:mapping:((sqr∼intervals:1):10)

`Conicalsurface` generalized cone, with apex `a` and curve `beta` crossing all the rules

Pre/Post conds `(a::isseqof:isreal; beta::isseqof:isfun)` → `(isseqof:isfun)`

Example `map:(conicalsurface:<<0,0,1>,beta>):((sqr∼intervals:1):10)`

`Cylindricalsurface` generalized cylinder, with direction `a` and curve `beta` crossing all the rules

Pre/Post conds `(a::isseqof:isreal; beta::isseqof:isfun)` → `(isseqof:isfun)`

Example `map:(cylindricalsurface:<<0,0,1>,beta>):((sqr∼intervals:1):10)`

`Hermitesurface` generator of coord functions of the *bicubic* Hermite surface

Pre/Post conds `(controlpoints::ismatof:ispoint)` → `(isseqof:isfun)`

Example `map:(hermitesurface:<` 4 × 4 matrix of points `>):domain2d`

`Profileprodsurface` returns the coord functions of a profile product surface

Pre/Post conds `(profile, section::isseqof:isfun)` → `(isseqof:isfun)`

Example `map:(profileprodsurface:< alpha, beta >):domain2d`

`Rotationalsurface` generates a surface by rotation of `profilecurve`. The opening angle of the rotational patch depends on the 2*nd* domain parameter

Pre/Post conds `(profilecurve::isseqof:isfun)` → `(isseqof:isfun)`

Example
```
map:(rotationalsurface:(bezier:s1:<<0,0>,<8,5>,<0,10>>)):
(intervals:1:12 * intervals:(2*pi):24)
```

`Ruledsurface` surface from profile `alpha`(u) and tangent vectors `beta`(u)

Pre/Post conds `(alpha,beta::isseqof:isfun)` → `(isseqof:isfun)`

Example `map:(ruledsurface:< c1, c2 vectdiff c1 >):domain2d`

`Tensorprodsurface` tensor product surface generator

Pre/Post conds `(ubasis,vbasis::isseqof:isfun)(points::ismatof:ispoint)` → `(isseqof:isfun)`

Example
```
(tensorprodsurface:< bernsteinbasis:s1:3,
bernsteinbasis:s1:3>:controlpoints)
```

`Thinsolid` thin solid generated by a `surface`

Pre/Post conds `(surface::isseqof:isfun)` → `(isseqof:isfun)`

Example `def solidmapping = (thinsolid ∼ coonspatch):<su0,su1,s0v,s1v>`

B.16 `text` Library

`Rotatedtext` returns a 1D geometric text rotated by `alpha` radians

Pre/Post conds `(alpha::isreal)` → `(ispol)`

Example `rotatedtext:(pi/4):'Hello Plasm!'`

`Solidifier` operator to return an offset 3D geometric value for the `arg` string

Pre/Post conds `(arg::isstring)` → `(ispol)`

Example `solidifier:'Hello, PLaSM World !'`

`Textwithattributes` returns a 1D geometric text string with specified attributes

Pre/Post conds `(TextAlignment::IsString; TextAngle, TextWidth, TextHeight, TextSpacing::IsReal)(arg::isstring)` → `(ispol)`

Example `TextWithAttributes:<'center',0,1,1,0.5>:'Hello, PLaSM World !'`

`Text` returns some geometric text with default value for attributes

Pre/Post conds `(arg::isstring)` → `(ispol)`

Example `text:'Hello, PLaSM World !'`

B.17 `transfinite` Library

`Bernsteinbasis` returns the Bernstein/Bézier polynomial basis of degree `n`	
Pre/Post conds	`(u::isfun)(n::isint)` ⟶ `(isseqof:isfun)`
Example	`bernsteinbasis:s1:3`
`Bernstein` generator of the i-th Bernstein polynomial function of degree n	
Pre/Post conds	`(u::isfun)(n::isint)(i::isint)` ⟶ `(isfun)`
Example	`bernstein:s1:3:2`
`Bezier` transfinite Bézier mapping of arbitrary dimension/degree	
Pre/Post conds	`(u::isfun) (controldata::isseq)` ⟶ `(isseqof:isfun)`

Example	`map(bezier:s1:<<10,0,0>,<10,5,3>,<10,10,0>>):dom1d`
	`map(bezier:s2:<c1,c2,c3,c4>):dom2d`
	`map(bezier:s3:<sur1,sur2,sur3,sur4>):dom3d`

`Coonspatch` Coons$'$ patch interpolating four boundary curves `su0`, `su1`, `s0v`, `s1v`	
Pre/Post conds	`(su0,su1,s0v,s1v::isseqof:isfun)` ⟶ `(isseqof:isfun)`
Example	`MAP:(CoonsPatch:<Su0,Su1,S0v,S1v>):((sqr ∼ Intervals:1):10)`
`Cubichermite` transfinite cubic Hermite d-manifold generator	
Pre/Post conds	`(u::isfun) (p1,p2,t1,t2::isseq)` ⟶ `(isseqof:isfun)`
Example	`cubichermite:s2:< c1,v2,<0,0,1>,<0,0,-1> >`
`Hermitebasis` returns the transfinite cubic Hermite basis	
Pre/Post conds	`(u::isfun)` ⟶ `(isseqof:isfun)`
Example	`hermitebasis:s1`

B.18 `vectors` Library

`Convexcoords` returns the convex coords of a point `x` w.r.t. a simplex `p`	
Pre/Post conds	`(p::issimplex)(x::ispoint)` ⟶ `(ispoint)`
Example	`convexcoords:(simplex:3):<1/3,1/3,1/3>` ≡ `<0.`$\overline{3}$`,0.`$\overline{3}$`,0.`$\overline{3}$`,0.0>`
`Dirproject` directional projection of `v` vector in `e` direction	
Pre/Post conds	`(e::isvect)(v::isvect)` ⟶ `(isvect)`
Example	`dirproject:<1,1,0,0>:<10,15,20,25>` ≡ `<12.5,12.5,0,0>`
`Idnt` identity matrix constructor	
Pre/Post conds	`(n::isintpos)` ⟶ `(ismat)`
Example	`idnt:4` ≡ `<<1,0,0,0>,<0,1,0,0>,<0,0,1,0>,<0,0,0,1>`
`Innerprod` inner product of vectors in $\mathbb{R}^n$	
Pre/Post conds	`(v,w::isvect)` ⟶ `(isnum)`
Example	`innerprod:<<11,12,13>,<4,5,6>>` ≡ `182`
`Isfunvect` predicate to test if `arg` is a sequence of functions or not	
Pre/Post conds	`(arg::tt)` ⟶ `(isbool)`
Example	`isfunvect:<id,k,sin>` ≡ `true`
`Ismat` predicate to test if `arg` is a matrix (of either numbers or functions) or not	

Pre/Post conds `(arg:tt)` → `(isbool)`
Example `ismat:<<1.0,2.0,3.0>,<4.0,5.0,6.0>,<7.0,8.0,9.0>>` ≡ `true`

`Ismatof` to test if `arg` is a matrix of elements satisfying the `istype` predicate

Pre/Post conds `(istype::isfun)(arg:tt)` → `(isbool)`
Example `ismatof:ispoint:<<<0,0,0>,<2,0,1>>,<<1,3,-1>,<3,2,0>>>` ≡ `true`

`Ispointseq` predicate to test if `arg` is a sequence of points in some $\mathbb{E}^d$

Pre/Post conds `(arg:tt)` → `(isbool)`
Example `isPointSeq:<<0,0,0>,<2,0,1>,<1,3,-1.5>,<3,2,0>>` ≡ `true`

`Ispoint` predicate to test if `arg` is a point in some $\mathbb{E}^d$

Pre/Post conds `(arg:tt)` → `(isbool)`
Example `ispoint:<0,0,0,1>` ≡ `true`

`Isrealvect` predicate to test if `arg` is a vector in some $\mathbb{R}^d$

Pre/Post conds `(arg:tt)` → `(isbool)`
Example `isrealvect:<0,0,0,1>` ≡ `true`

`Issqrmat` predicate to test if `arg` is a square matrix in some $\mathcal{M}_d^d$

Pre/Post conds `(arg:tt)` → `(isbool)`
Example `issqrmat:<<<0,0,0>,<2,0,1>>,<<1,3,-1>,<3,2,0>>>` ≡ `true`

`Isvect` predicate to test if `arg` is a vector in some $\mathcal{V}^d$ (of either numbers or functions)

Pre/Post conds `(arg:tt)` → `(isbool)`
Example `isvect:<0,0,0,1>` ≡ `true`
`isvect:(beziercurve:<<0,0,0>,<1,0,0>,<1,1,1>,<0,1,0>>)` ≡ `true`

`Iszero` predicate to test if `arg` is the **0** element in some $\mathbb{R}^d$

Pre/Post conds `(arg:tt)` → `(isbool)`
Example `iszero:<0,0,0,0>` ≡ `true`

`Matdotprod` binary inner product of matrix `pair` ≡ ¡a,b¿ in some $\mathbb{R}_n^m$

Pre/Post conds `(pair::ismatof:isvect ∼ trans)` → `(isnum)`
Example `<<1,2>,<3,4>,<5,6>> matdotprod <<10,20>,<30,40>,<50,60>>` ≡ `910`

`Mathom` matrix homogenization, i.e. adding of a unit *first* row and column

Pre/Post conds `(m::issqrmat)` → `(issqrmat)`
Example `mathom:<<10,20>,<30,40>>` ≡ `<<1,0,0>,<0,10,20>,<0,30,40>>`

`Meanpoint` returns the point with middle coordinates from a `points` sequence

Pre/Post conds `(points::ispointseq)` → `(ispoint)`
Example `Meanpoint:<<0,2,0>,<3,0,10>,<10,4,0>,<1,10,2>>` ≡ `<7/2,4,3>`

`Mixedprod` returns the mixed product $\boldsymbol{a} \times \boldsymbol{b} \cdot \boldsymbol{c}$ of three vectors in $\mathbb{R}^3$

Pre/Post conds `(a,b,c::and ∼ [isvect, c:eq:3 ∼ len])` → `(isnum)`
Example `mixedprod:<<1,1,1>,<2,0,2>,<0,3,0>>` ≡ `0`

`Orthoproject` orthogonal projection of `v` vector in `e` direction

Pre/Post conds `(e::isvect)(v::isvect)` → `(isvect)`
Example `orthoproject:<1,1,0,0>:<10,15,20,25>` ≡ `<-2.5,2.5,20,25>`

`Ortho` orthogonal component of a square `matrix`

Pre/Post conds	`(matrix::issqrmat)` → `(issqrmat)`
Example	`Ortho:<<0,1,0>,<0,0,2>,<1,1,1>>` ≡ `<<0,1/2,1/2>,<1/2,0,3/2>,<1/2,3/2,1>>`

`Pivotop` pivoting operation on the (i,j) element of `mat` in some $\mathbb{R}^m_n$

Pre/Post conds	`(i,j::isintpos)(mat::ismat)` → `(ismat)`
Example	`(PivotOp:<2,2> * ID):<<1,2,0>,<0,-1,2>,<1,1,1>>` ≡ `<<1,0,4>,<0,1,-2>,<1,0,3>>`

`Rotn` rotation in $\mathbb{E}^3$ of α angle about an arbitrary axis $\boldsymbol{n}$ for the origin

Pre/Post conds	`(alpha::isreal; n::isvect)` → `(isfun)`
Example	`rotn:<pi/4, <1,1,1>>:(cuboid:<1,1,1>)`

`Scalarmatprod` product of a scalar `a` times a matrix `mat`

Pre/Post conds	`(a::isnum; mat::ismat)` → `(ismat)`
Example	`9 ScalarMatProd IDNT:3` ≡ `<<9,0,0>,<0,9,0>,<0,0,9>>`

`Scalarvectprod` product of a scalar `a` times a vector `v`

Pre/Post conds	`(arg::ispair)` → `(isvect)`
Example	`10 ScalarVectProd <1,2>` ≡ `<1,2> ScalarVectProd 10` ≡ `<10,20>`

`Skew` skew component of a square `matrix`

Pre/Post conds	`(matrix::issqrmat)` → `(issqrmat)`
Example	`skew:<<0,1,0>,<0,0,2>,<1,1,1>>` ≡ `<<0,1/2,-1/2>,<-1/2,0,1/2>,<1/2,-1/2,0>>`

`Trace` returns the trace of the input `matrix`

Pre/Post conds	`(matrix::issqrmat)` → `(isnum)`
Example	`trace:<<1,2,3>,<4,5,6>,<7,8,9>>` ≡ `15`

`Unitvect` returns the unit vector of $\mathbb{R}^n$ parallel to $\boldsymbol{v} \in \mathbb{R}^n$

Pre/Post conds	`(v::isvect)` → `(isvect)`
Example	`unitvect:<10,20,30>` ≡ `<0.2672612419, 0.534522483, 0.801783725>`

`Vectdiff` difference of vectors $\boldsymbol{v}, \boldsymbol{w}$ in a vector space $\mathcal{V}^d$ (of numbers or functions)

Pre/Post conds	`(v,w::isvect)` → `(isvect)`

Example	`vectdiff:<<11,12,13>,<4,5,6>>` ≡ `<7,7,7>`
	`beziercurve:<<0,0>,<1,0>,<1,1>,<0,1>> vectdiff <k:1,k:1>`

`Vectnorm` Euclidean norm of the vector `v`

Pre/Post conds	`(v::isvect)` → `(isnum)`
Example	`(vectnorm ∼ unitvect):<10,20,30>` ≡ `0.9999999999999999`

`Vectprod` vector product of vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^3$

Pre/Post conds	`(u,v::isvect)` → `(isvect)`
Example	`vectProd:<<1,0,0>,<1,1,0>>` ≡ `<0,0,1>`

`Vectsum` addition of vectors $\boldsymbol{v}, \boldsymbol{w}$ in a vector space $\mathcal{V}^d$ (of numbers or functions)

Pre/Post conds	`(v,w::isvect)` → `(isvect)`

Example	`vectsum:<<11,12,13>,<4,5,6>>` ≡ `<15,17,19>`
	`beziercurve:<<0,0>,<1,0>,<1,1>,<0,1>> vectsum <k:1,k:1>`

`Vect2dtoangle` maps a vector $\boldsymbol{v} \in \mathbb{E}^2$ to its signed angle with the x-axis

Pre/Post conds	`(v::isvect)` → `(isnum)`
Example	`vect2dtoangle:<1,1>` ≡ `vect2dtoangle:<2,2>` ≡ `0.78539816339745`

B.19 `viewmodels` Library

`Axialcameras` for VRML exporting. Centered on the reference frame axes	
Pre/Post conds	`(scene::ispol)` → `(ispol)`
Example	`Axialcameras:(cuboid:<1,1,1>)`
`Cabinet` object; standard view model for parallel oblique projection	
Pre/Post conds	→ `(isviewmodel)`
Example	`projection:parallel:cabinet:(cuboid:<1,1,1>)`
`Centeredcameras` for VRML exporting. Centered on the `scene` containment box	
Pre/Post conds	`(scene::ispol)` → `(ispol)`
Example	`Axialcameras:(cuboid:<1,1,1>)`
`Centralcavalier` object; standard view model for parallel oblique projection	
Pre/Post conds	→ `(isviewmodel)`
Example	`projection:parallel:centralcavalier:(cuboid:<1,1,1>)`
`Dimetric` object; standard view model for parallel orthogonal projection	
Pre/Post conds	→ `(isviewmodel)`
Example	`projection:parallel:dimetric:(cuboid:<1,1,1>)`
`Isometric` object; standard view model for parallel orthogonal projection	
Pre/Post conds	→ `(isviewmodel)`
Example	`projection:parallel:isometric:(cuboid:<1,1,1>)`
`Leftcavalier` object; standard view model for parallel oblique projection	
Pre/Post conds	→ `(isviewmodel)`
Example	`projection:parallel:leftcavalier:(cuboid:<1,1,1>)`
`Onepoint` object; standard view model for perspective projection	
Pre/Post conds	→ `(isviewmodel)`
Example	`projection:perspective:onepoint:(cuboid:<1,1,1>)`
`Orthox` object; standard view model for parallel orthographic projection	
Pre/Post conds	→ `(isviewmodel)`
Example	`projection:parallel:orthox:(cuboid:<1,1,1>)`
`Orthoy` object; standard view model for parallel orthographic projection	
Pre/Post conds	→ `(isviewmodel)`
Example	`projection:parallel:orthoy:(cuboid:<1,1,1>)`
`Orthoz` object; standard view model for parallel orthographic projection	
Pre/Post conds	→ `(isviewmodel)`
Example	`projection:parallel:orthoy:(cuboid:<1,1,1>)`
`Parallel` projection class, determining the type of 3D pipeline	
Pre/Post conds	`(vrp, vpn, vup, prp, window::IsSeq; front, back::IsReal)` → `(isfun)`
Example	`projection:parallel:orthoy:(cuboid:<1,1,1>)`
`Perspective` projection class, determining the type of 3D pipeline	

Pre/Post conds `(vrp, vpn, vup, prp, window::IsSeq; front, back::IsReal)` → `(isfun)`

Example `projection:perspective:threepoints:(cuboid:<1,1,1>)`

`Projection` top-level user interface operator

Pre/Post conds `(type::or` ∼ `[isparallel, isperspective])(view::isviewmodel)` `(scene::ispol)` → `(ispol)`

Example `projection:parallel:orthoy` ≡ `An-Anonymous-Function :` $\mathbb{E}^3 \to \mathbb{E}^2$

`Rightcavalier` object; standard view model for parallel oblique projection

Pre/Post conds → `(isviewmodel)`

Example `projection:parallel:rightcavalier:(cuboid:<1,1,1>)`

`Threepoints` object; standard view model for perspective projection

Pre/Post conds → `(isviewmodel)`

Example `projection:perspective:threepoints:(cuboid:<1,1,1>)`

`Trimetric` object; standard view model for parallel orthogonal projection

Pre/Post conds → `(isviewmodel)`

Example `projection:parallel:trimetric:(cuboid:<1,1,1>)`

`Twopoints` object; standard view model for perspective projection

Pre/Post conds → `(isviewmodel)`

Example `projection:perspective:twopoints:(cuboid:<1,1,1>)`

`Xcavalier` object; standard view model for parallel oblique projection

Pre/Post conds → `(isviewmodel)`

Example `projection:parallel:xcavalier:(cuboid:<1,1,1>)`

`Ycavalier` object; standard view model for parallel oblique projection

Pre/Post conds → `(isviewmodel)`

Example `projection:parallel:ycavalier:(cuboid:<1,1,1>)`