Package 'garray'

November 5, 2018

Type Package

Title Generalized Array Arithmetic for Ragged Arrays with Named Margins

Version 1.1.2

Author Qingsheng Huang

Maintainer Qingsheng Huang <huangqqss@126.com>

Depends R (>= 3.4.0)

Suggests parallel (>= 3.4.0)

Description Organize a so-called ragged array as generalized arrays, which is simply an array with sub-dimensions denoting the subdivision of dimensions (grouping of members within dimensions).
By the margins (names of dimensions and sub-dimensions) in generalized arrays, operators and utility functions provided in this package automatically match the margins,
doing map-reduce style parallel computation
along margins. Generalized arrays are also cooperative to R's native functions that work on simple arrays.

License GPL-3

Encoding UTF-8

RoxygenNote 6.1.0

NeedsCompilation no

Repository CRAN

Date/Publication 2018-11-05 19:20:08 UTC

R topics documented:

`abind` *Combine generalized arrays*

Description

Combine generalized arrays, similar to the manner cbind and rbind work. Put a sequence of generalized arrays and get a single generalized array of the same or one more margins.

Usage

```
abind(..., margins = character(), along = character())
```

Arguments

`...`	arrays, or a list of several arrays. Margins of these arrays should be the same.
`margins`	Resulting margins. If includes a totally new margin (not used by any arrays in `...`), then `along` is neglected and the new margin setting `along`. If no new name and `length(along)==0L`, the last `margins` become along. If `0L==length(margins)`, `along` will become last dimensions of output.
`along`	The dimension along which to bind the arrays. The arrays may have different lengths along that dimension, and are bind along it, with addition subdimension '*.bind.from' indicating the composition of the `along` dimension. Some arrays may not have that margin, then the dimension of these arrays expand to 1. If `along` is a totally new margin, it is created. In such case, all arrays should have the same dimension.

Details

Combine sdim correctly. Saving or dropping of subdimensions follow a few rules: subdimensions of the margin with bound along are dropped; of the other margins are save unless the names of subdimensions are the same; subdimensions of the same names are dropped except the first one.

Examples

```
a <- garray(1:24, c(4,6),
dimnames=list(X=1:4, Y=letters[1:6]),
  sdim=list(XX=c(x1=3,x2=1), YY=c(y1=1,y2=2)))
b <- garray(1:6/10,6,dimnames=list(Y=letters[1:6]))
ab <- abind(a=a, b=b, along="X")
#abind(a, b, margins=c("X","Y")) # Error
ab2 <- abind(a=a, b=b, margins=c("X","Y"), along="X")
aa <- abind(a=a, a=a, along="Z")
ab3 <- abind(a, b, along="X")
```

---

amap *Mapping matching dimension of arrays with a function*

Description

Generalized and smart mapply()/Map()/outer()/sweep() for data mapping. Matching is checked automatically.

Usage

```
amap(FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, VECTORIZED = NA)

## S3 method for class 'garray'
Ops(e1, e2)
```

Arguments

FUN	Known vectorized function is recognized if passed in as character. Other functions will be vectorized by `.mapply()`.
...	Arrays with margins (names of dimnames) and maybe with sdim. Orders of their margins can be different, but the extent along a margin is matched. Unmatched margins are broadcasting like outer(). Scalar (length 1 vector) do not contribute margins and not broadcast here (they will broadcast by `.mapply()` later).
MoreArgs	a list of other arguments to 'FUN', no matching of margins.
SIMPLIFY	logical, attempt to reduce the result to exclude recursive structure (no list hierachy but plain generalized array).
VECTORIZED	Whether FUN is vectorized will affect the behaviours. Some combination of FUN and VECTORIZED is not simply slowing down, but produces meaningless results or even stop (e.g., cumsum). TRUE - call FUN once with arrays being reorganize on dimensions; FALSE - call FUN many times (via `.mapply`), with each cell of arrays.
e1, e2	Generalized arrays, being operands.

Value

The dimensions is deduced from inputs.

Examples

```
a <- garray(1:24, c(4,6,2), dimnames=list(X=1:4, Y=letters[1:6], Z=NULL),
sdim=list(XX=c(x1=3,x2=1), YY=c(y1=1,y2=2)))
b <- garray(1:6/10,6,dimnames=list(Y=letters[1:6]))
c <- garray(1:4/100,c(X=4))
d <- garray(1:4/1000,c(Y=4))
e <- garray(1:2/1000,c(X=2))
f <- garray(0,c(Z=2))
g <- garray(0,c(ZZ=2))
m1 <- amap(psummary,c,a,b,      0.0001, VECTORIZED=FALSE)
m2 <- amap(sum,     c,a,b,      0.0001, VECTORIZED=FALSE)
m3 <-               c+a+b+      0.0001
n1 <- amap(sum,     c,a,b,d,    0.0001, VECTORIZED=FALSE)
n2 <- amap(sum,     c,a,b,e,    0.0001, VECTORIZED=FALSE)
n3 <- amap(sum,     c,a,b,e,f,  0.0001, VECTORIZED=FALSE)
p1 <- amap(sum,     c,a,b,e,f,g,0.0001, VECTORIZED=FALSE)
q1 <- amap(sum, c,a,b,e,f,g,0.0001, SIMPLIFY=FALSE, VECTORIZED=FALSE)
q2 <- amap(c,   c,a,b,e,f,g,0.0001, SIMPLIFY=FALSE, VECTORIZED=FALSE)
q3 <- amap(list,c,a,b,e,f,g,0.0001, SIMPLIFY=FALSE, VECTORIZED=FALSE)
m1==m2
m2==m3
m2==aperm(m3, 3:1)
```

`amult` *Generalized array multiplication.*

Description

Default to Einstein summation convention, without explicitly subscripts.

Usage

```
amult(X, Y, FUN = "*", SUM = "sum", BY = NULL, MoreArgs = NULL,
  ..., SIMPLIFY = TRUE, VECTORIZED = TRUE)

X %X% Y
```

Arguments

`X, Y`	Generalized arrays that can be multiplied.
`FUN`	The 'multiply' function.
`SUM`	The 'reduce' function.
`BY`	margins excluded from summary by SUM.
`MoreArgs, SIMPLIFY, VECTORIZED`	Argument used by 'amap()'.
`...`	Argument used by 'areduce()'.

Details

Margins shared by X and Y are parallelly mapped by FUN, and then reduced by SUM (inner product like %*%); margins in BY and shared by X and Y are simply mapped by FUN but excluded from reducing (parallel product like *); other margins are extended repeatly (outer product like %o%). Shared margins not to be mapped have to be renamed (like outer product). For special FUN and SUM, fast algorithms are implemented.

Examples

```
a <- garray(1:24, c(4,6), list(X=LETTERS[1:4], Y=letters[1:6]),
sdim=list(XX=c(x1=3,x2=1), YY=c(y1=1,y2=2)))
b <- garray(1:20, c(Z=5, X=4))
c <- garray(1:120, c(X=4,Y=6,Z=5))
m1 <- amult(a, b)
m2 <- amult(a, b, `*`, sum)
m3 <- amult(b, a)
all.equal(m1, m2)
all.equal(m1, m3)
all.equal(m1, t(m3))
n1 <- amult(a, c, `*`, sum)
n2 <- a%X%c
all.equal(n1, n2)
amult(garray(1:5,margins="I"), garray(1:8,margins="J"))
amult(garray(1:8,c(I=2,J=4)), garray(1:9,c(K=3,L=3)))
```

---

aperm.garray *General array transposition*

Description

Restore garray attributes that discarded by aperm.default(). Cannot permute between subdimension (means to promote subdimension of equal length into regular dim and reduce dim into subdimension), sorry.

Usage

```
## S3 method for class 'garray'
aperm(a, perm = NULL, ...)
```

Arguments

a	A generalized array to be transposed.
perm	Desired margins after permutation, integer or character.
...	Useless, potential arguments inherited from the S3 generic.

areduce *Generalized and smart apply()/Reduce()/tapply() for data folding.*

Description

Generalized and smart apply()/Reduce()/tapply() for data folding.

Usage

```
areduce(FUN, X, MARGIN, ..., SIMPLIFY = TRUE, SAFE = FALSE)
```

Arguments

FUN	Usually a summary function (like `all` and `sum`).
X	A garray, with margins (names of dimnames) and maybe with sdim.
MARGIN	Some margins of X and names of sdim. MARGIN=character() means to reduce all margins (over no margin). In such case, areduce() is not needed actually.
...	Further arguments to 'FUN', no matching of margins.
SIMPLIFY	TRUE - simplifies the result list to vector of atomic if possible, and triggers warning and not simplifies if impossible; FALSE - not simplifies for non speed-up function, and issues warning (and have to simplify) for speed-up function; NA - simplifies but no warning if impossible.
SAFE	TRUE - use safe but slow implementation, in which data splited from the array are reorganized into small arrays (as are being subset by `[]`) and passed to FUN (other attributes are dropped, however); FALSE - faster, data are passed to FUN as dimension-less vectors.

Value

A matrix (similar to return of apply() or tapply()), with the trailing margins the same as MARGIN, while the leading margins depend on FUN and SIMPLIFY. If FUN returns a scalar or SIMPLIFY=FALSE, then no leading margins. In MARGIN, subdimension is replaced with superdims.

Examples

```
a <- garray(1:24, c(4,6),
dimnames=list(X=LETTERS[1:4], Y=letters[1:6]),
sdim=list(XX=c(x1=3,x2=1), YY=c(y1=1,y2=2)))
x1 <- areduce("sum", a, c("X"))
x2 <- areduce(`sum`, a, c("X"))
stopifnot(garray(c(66,72,78,84), margins="X")==x1, x2==x1)
yy1 <- areduce("sum", a, c("YY"))
yy2 <- areduce(`sum`, a, c("YY"))
stopifnot(garray(c(10,68,58,164), margins="Y")==yy1, yy2==yy1)
xyy1 <- areduce("sum", a, c("X","YY"))
xyy2 <- areduce(`sum`, a, c("X","YY"))
stopifnot(xyy1==xyy2)
```

```
xxyy1 <- areduce("sum", a, c("XX","YY"))
xxyy2 <- areduce(`sum`, a, c("XX","YY"))
stopifnot(garray(c(6,4,48,20,42,16,120,44), c(X=2,Y=4))==xxyy1)
stopifnot(xxyy2==xxyy1)
b <- garray(1:24, c(3,4,2),
dimnames=list(X=LETTERS[1:3], Y=letters[1:4], Z=NULL),
sdim=list(XX=c(x1=2,x2=1), YY=c(y1=1,y2=1)))
xxyyz1 <- areduce("sum", b, c("XX","YY","Z"))
xxyyz2 <- areduce(`sum`, b, c("XX","YY","Z"))
stopifnot(xxyyz1==xxyyz2)
xyz1 <- areduce(identity, b, c("XX","YY","Z"), SIMPLIFY=FALSE)
xyz2 <- areduce("c",      b, c("XX","YY","Z"), SIMPLIFY=FALSE)
xy1 <- areduce(identity, b, c("XX","YY"), SIMPLIFY=FALSE, SAFE=TRUE)
stopifnot(identical(dimnames(xy1[2,3][[1]]), list(X="C",Y="c",Z=NULL)))
# garray of lists, cannot use `xyz1==xyz2` etc to compare.
```

---

as.data.frame.garray *Coerce to a Data Frame*

Description

Convert a 2D generalized array into a data.frame, making `print()` work correctly.

Usage

```
## S3 method for class 'garray'
as.data.frame(x, row.names = NULL, optional = FALSE,
  col.names = NULL, ..., stringsAsFactors = FALSE)
```

Arguments

`x`	A generalized array object.
`row.names, optional, stringsAsFactors, ...`	See the same arguments in ?as.data.frame.
`col.names`	'NULL' or a character vector giving the column names.

---

awipe *Generalized array's sweep() for data cleaning.*

Description

Return a generalized array, by wiping out a summary statistic.

Usage

```
awipe(X, FUN = "-", STATS = "mean", MARGIN = NULL, MoreArgs = NULL,
  ..., SIMPLIFY = TRUE, VECTORIZED = NA)
```

Arguments

X	A generalized array.
FUN	The wiping function.
STATS	Numeric array or function.
MARGIN	NULL - STATS is an array; character - STATS is a function, and by X being reduced along MARGIN, X is wiped. Length 0 character vector means reducing along no margin, resulting in a scalar (in this case, for example, `areduce(sum, X)` is the same as `sum(X)`.
MoreArgs, SIMPLIFY, VECTORIZED	Argument used by 'amap()'.
...	Argument used by 'areduce()'.

Examples

```
a <- garray(1:24, c(4,6), list(X=LETTERS[1:4], Y=letters[1:6]),
sdim=list(XX=c(x1=3,x2=1), YY=c(y1=1,y2=2)))
m1 <- awipe(a, MARGIN="XX")
m2 <- awipe(a, `-`, mean, "XX")
```

---

dim.garray *Dimensions of a generalized array*

Description

Retrieve or set the dimension of a generalized array.

Usage

```
## S3 method for class 'garray'
dim(x)

## S3 replacement method for class 'garray'
dim(x) <- value
```

Arguments

x	An generalized array.
value	An integer (can be coerced from double numeric) vector, with names.

Details

The functions `dim` and `dim<-` are internal generic primitive functions. Here `dim.garray` and `dim<-.garray` are methods for 'garray's, which returns and setting with the named dimensions (margins). The two function is usually used as, for example, `dim(arr)` and `dim(arr) <- c(A=3,B=2)`. Native R saves the names of dim but seldom uses it. However, it is undocumented and not stable because some functions discard it (like: t()) . This package will totally neglect it but keeps the margins in dimnames.

garray *Generalized and smart array*

Description

Creates or tests for generalized arrays.

Usage

```
garray(data, dim = NULL, dimnames = NULL, margins = NULL,
  sdim = attr(data, "sdim", exact = TRUE))

garray.array(x, sdim)

as.garray(x, ...)

## S3 method for class 'garray'
as.garray(x, ...)

## Default S3 method:
as.garray(x, ...)

is.garray(x)

is.garray.duck(x)

is.scalar(x)
```

Arguments

data	Usually a simple array, and can be a vector without dimensions.
dim	An integer vector giving the maximal indices in each dimension.
dimnames	A list (or it will be ignored) with for each dimension one component, either 'NULL' or a character vector.
margins	Override the names of dim and of dimnames.
sdim	Optional, a named list of numeric vectors indicating the subdivision of some of the dimensions. The value vill become, after validated, the attribute sdim. See 'Details' and '?sdim'.
x	An R object.
...	Additional arguments to be passed to or from methods.

Details

Generalized arrays are generalized because they handle dimensions and subdimensions that are ragged; and they are also smart because they automatically match dimensions by margins (names of

dimnames). Margins is implemented similar to R's native class "table", i.e., use names of dimnames to store the margins

Attribute sdim denotes subdimensions, which are the subdivision of dimensions or grouping of members of a dimension, for organizing a ragged array. It is a named list of numeric vectors, each of which indicates the lengths of subdivision groups within a dimension. Every name of the list prefixed with a margin of the generalized array. By the matching of sdim names and dim names, utility functions figure out which dimensions the sub dimensions reside in. Sum of a vector of the list usually equals to the extent of the corresponding dimension. If they are not equal and the extent can not be divided exactly by the sum, the subdimension is invalid and will be dropped. If the extent can be divided exactly by the sum, the subdimension is still valid but non-canonical. Non-canonical subdimension can be provided to `garray()` and `sdim<-` as argument, and the two functions canonicalize it. Other utility functions cannot handle non-canonical subdimension, thus manually constructing objects of garray class is permitted but dangerous. Values of each vector of the list denotes the repeating times of subdimension residing in the coresponding dimension (called superdim). More than 1 subdimension reside in the same superdim is allowed. This feature allows dividing a subdimension further, organizing the subdims into hierachy.

By definition and for S3 dispatching, `class(.)="garray"` is required, but simple arrays with proper margins actually work correctly with most functionalities of this package. For the sake of compatibility and reducing warning message, `is.garray.duck()` tests whether the array has proper margins.

A still problem is that attributes in R are fragile, even indexing will drop most attributes. Utility functions and methods for dispatching for 'garray' implemented in this package guaranttee to save the margins (names of dimnames) and subdimension (attr(*,'sdim')).

Functions

- `garray.array`: A simple and faster version of garray(), mainly for internal usage. Note that garray() is not generic function, thus garray.array() will never be called by dispatching.
- `is.garray`: `is.garray` do simple validation, no check for validity of sdim because it is too expensive. Operation of sdim by this package is always guaranteed the validity.
- `is.garray.duck`: `is.garray.duck` do duck-typing validation, ignoring the class
- `is.scalar`: Test whether the vector or array is actually a scalar (`length(x)==1L`).

Examples

```
a1 <- garray(1:27, c(A=3,B=9), sdim=list(AA=c(a=2,b=1),BB=c(a=3)))
a2 <- garray(1:27, c(A=3,B=9), sdim=list(AA=c(a=2,b=1),BB=c(a=4)))
```

`margins` *The margins and dimensions of a generalized array object*

Description

Margins means the names of dimnames of an array. `margins<-` and `remargins` are for renaming, but `margins<-` ignores the names of `value` while `remargins` according to the names of `value` renames the margins. Doing so, `remargins` may also keep sdim. `margins<-` always removes sdim. For `remargins` the length of value can be shorter than that of the margins if the value has names.

Usage

```
margins(x)

margins(x) <- value

remargins(x, value)
```

Arguments

x	A generalized array.
value	A character vector will become the margins (names of dimnames) of the generalized array. `margins<-` ignores the names of `value` while `remargins` according to the names of `value` renames the margins. For `remargins` the length of value can be shorter than that of the margins if the value has names.

---

print.garray *Print Values*

Description

Print out a generalized array and returns it *invisibly*.

Usage

```
## S3 method for class 'garray'
print(x, ...)
```

Arguments

x	A generalized array object.
...	Additional arguments to be passed to or from methods.

---

psummary *Parallel summary, inspired by pmax() and pmin().*

Description

Functions of Summary group are all, any, max, min, prod, range, and sum, which reduce a vector into a scalar (except range), thus the name of psummary(). Of course, other FUN can be passed-in, but functions like range() that returns a non-scalar vector result in unpredictable return. For arguments of different size, pmin() and pmax() make fractional recycling and issue warning, but psummary() error since as.data.frame() do not fractionally recycle.

Usage

```
psummary(...)

## S3 method for class 'garray'
psummary(...)

## Default S3 method:
psummary(...)
```

Arguments

`...`	Usually in the form `psummary(x, y, z, FUN=sum, na.rm=TRUE)`, alternaitvely `psummary(list(x, y, z), FUN=sum, na.rm=TRUE)`.

---

read.ctable *Read a complex table and return array in basic storagemode.*

Description

A complex table has several row and colume headers, some of which indicate the hierarchy of the dimensions (thus the returned array may have more dimensions than row and column), some of which are real row.names and col.names that will be turn into dimnames, and some of which are additional attributes. Cells of non-header are all in a same format (like double). The original header are preserved as attributes. The dimnames strictly save the layout of the matrix, thus row.names and col.names should be carefully chosen for matching the actual dimension. Sometime aperm() is needed after this function. Sparse table, where dimnames is not complete (unbalanced) and needs non-fractionally recycling, is not supported.

Usage

```
read.ctable(file, header, row.names, col.names, ...,
  storagemode = "double")
```

Arguments

`file`	The name of the file to be read in, see ('file' in ?read.table).
`header`	hierarchy structure of the header, a list of length 2, assigning the index of row and col header, affecting parsing of the input table;
`row.names, col.names`	a list, whose elements can be a character vector assigning the name of one dimension directly, or a integer scalar which is the index (icol,irow) of row,colheader that will be extracted; since the row and the col headers can have hierarchy in the input table and the hierarchy will be organized into addisional dimensions, rownames indicate the names of dimensions that are from the row of the input table, while colnames, the col; in the input table, the header can be in two pattern: AAA and AOO; in the AAA pattern, the names are repeated for

	the cells that are in the same index in high dimension, while in the AOO pattern, the names appear once at the first and the other cells that are in the same index in high dimension are left blank; in the output array, row,col.names are not necessary `dimnames[[1]]` and `dimnames[[2]]`; if all elements of the list are a integer scalar, then the list can also be coded as a integer vector (since they are the same for lapply);
`...`	Further arguments to be passed to 'read.table'.
`storagemode`	The storagemode of return matrix, usually 'double'.

`sdim` *Subdimensions of an array*

Description

Retireve or set the subdimension of an array.

Usage

```
sdim(x)

sdim(x, warn = TRUE) <- value
```

Arguments

`x`	A generalized array.
`warn`	Whether issue warning when some of subdimensions are invalid and get dropped.
`value`	A named list of numeric vectors indicating the subdivision of some of the dimensions. The value vill become, after validated, the attribute sdim. See '?garray'.

Details

Validation of subdimension is expensive because its consistency with dim need checking. Thus most of functions do not validate it. Operations of subdimension with functions discussed here are guaranteed to be always keeping the consistency.

Value

The subdimensions (a non-empty list) or NULL

`[.garray` *Indexing for the garray class*

Description

Indexing along margins as usual `[.array`, and along subdim.

Usage

```
## S3 method for class 'garray'
...[drop=TRUE]
#`[.garray`(..., drop=TRUE)
#x[i]
#x[i,j,...,drop=TRUE]
#x[m]
#x[l]
#x[M=i,N=j,...]

#\method{[}{garray}(...) <- value
#`[<-.garray`(..., value)
#x[i] <- value
#x[i,j,...] <- value
#x[m] <- value
#x[l] <- value
#x[M=i,N=j,...] <- value
```

Arguments

`drop`	Whether indeces where 1==dim are removed. Different from R's native `[`, a garray will become a garray or scalar, never a vector.
`value`	An array or a scalar.
`x`	A generalized array from which elements are extracted or replaced.
`i, j, m, l, M, N, ...`	In addition to the native styles (`i`, `j`, etc.) accepted by `[`, can be: 1. a matrix `m` with column names, which (`colnames(m)`) is a permutation of margins of the array. 2. an list `l <- list(i,j,...)`, can be unnamed or named, where NULL means to select all; 3. arguments with names (`M`, `N`, etc), where `NULL` and missing means to select all. These extensions make indexing 3 times slower than native indexing. Since it is hard to assign MissingArg in list(), at the moment MissingArg is only safe in native R subsettting style. Using NULL to select all like MissingArg is actually not consistent in semantic of other uses of NULL. As shown by what `c()` returns, NULL is a generalized form of `logical(0)`, `integer(0)`, and `character(0)`,

all of which means to select none when indexing. So take care of NULL if indexing with variables.

Examples

```
mm <- matrix(c(1:3,1), 2, 2, dimnames=list(NULL, c("B","A")))
a <- garray(1:27, c(A=3,B=9), sdim=list(AA=c(a=2,b=1),BB=c(a=3)))
b <- a[mm]
c1 <- a[B=1:2,A=NULL]
c2 <- a[B=1:2,A=]
c3 <- a[B=1:2]
c4 <- a[list(B=1:2)]
c5 <- a[list(B=1:2,A=NULL)]
c6 <- a[list(NULL,1:2)]
d1 <- a[,] ; d1[B=1:2,A=NULL]       <- c1*10
d2 <- a[,] ; d2[B=1:2,A=]           <- c1*10
d3 <- a[,] ; d3[B=1:2]              <- c1*10
d4 <- a[,] ; d4[list(B=1:2)]        <- c1*10
d5 <- a[,] ; d5[list(B=1:2,A=NULL)] <- c1*10
d6 <- a[,] ; d6[B=1:2,A=NULL] <- 1
d7 <- a[,] ; d7[mm] <- 1000
d8 <- a[,] ; d8[mm] <- 1:2*1000
e1 <- a[AA=1,drop=FALSE]
e11 <- a[AA=c(1,1),drop=FALSE]
e2 <- a[AA="b",drop=FALSE]
ebb <- a[AA=c("b","b"),drop=FALSE]
e3 <- a[,] ; e3[AA="b"] <- e2*10
e33 <- a[,] ; e33[AA=c("b","b")] <- c(e2*0.1, e2*100)
# Work in the same manner of `e33[c(3,3),] <- c(e2*0.1, e2*100)`.
e4 <- a[A=c(TRUE,FALSE,FALSE),drop=FALSE]
e5 <- a[A=TRUE,drop=FALSE]
e6 <- a[B=c(TRUE,FALSE,FALSE),drop=FALSE]
e7 <- a[AA=TRUE,drop=FALSE]
e8 <- a[AA=c(TRUE,FALSE),drop=FALSE]
```

%+% *Function composition operator*

Description

Composite functions a and b into a(b(...)).

Usage

```
a %+% b
```

Arguments

a	A function that can be called with one argument.
b	A function that can be called with one or more argument, and result of b() can be passed to a().

Value

A new function, whose arguments are what `b()` can accept, and whose result is what `a()` can return.

Examples

```
lse <- log%+%sum%+%exp
lse(1:10)
#logsumexp(1:10) # actual logsumexp() is more sophistic
log(sum(exp(1:10)))
sum <- sd
lse(1:10) # lse() is fixed at definition
log(sum(exp(1:10)))
(log%+%sum%+%exp)(1:10) # now is (log%+%sd%+%exp)
```

Index