Package 'freealg'

May 13, 2022

Type Package

Title The Free Algebra

Version 1.0-7

Maintainer Robin K. S. Hankin <hankin.robin@gmail.com>

Depends R (>= 3.5.0), methods

Description The free algebra in R; multivariate polynomials with non-commuting indeterminates.

License GPL (>= 2)

LazyData yes

Imports Rcpp (>= 1.0-7), partitions (>= 1.9-22), mathjaxr, disordR (>= 0.0-8)

LinkingTo Rcpp

SystemRequirements C++11

Suggests knitr,testthat,magrittr,markdown,rmarkdown

VignetteBuilder knitr

URL https://github.com/RobinHankin/freealg

BugReports https://github.com/RobinHankin/freealg/issues

RdMacros mathjaxr

R topics documented:

freealg-package *The Free Algebra*

Description

The free algebra in R; multivariate polynomials with non-commuting indeterminates.

Details

The DESCRIPTION file:

Package:	freealg
Type:	Package
Title:	The Free Algebra
Version:	1.0-7
Authors@R:	person(given=c("Robin", "K. S."), family="Hankin", role = c("aut","cre"), email="hankin.robin@
Maintainer:	Robin K. S. Hankin <hankin.robin@gmail.com>
Depends:	R (>= 3.5.0), methods
Description:	The free algebra in R; multivariate polynomials with non-commuting indeterminates.
License:	GPL (>= 2)
LazyData:	yes
Imports:	Rcpp (>= 1.0-7), partitions (>= 1.9-22), mathjaxr, disordR (>= 0.0-8)
LinkingTo:	Rcpp
SystemRequirements:	C++11
Suggests:	knitr,testthat,magrittr,markdown,rmarkdown
VignetteBuilder:	knitr
URL:	https://github.com/RobinHankin/freealg
BugReports:	https://github.com/RobinHankin/freealg/issues
RdMacros:	mathjaxr
Author:	Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Index of help topics:

```
Ops.freealg             Arithmetic Ops methods for the the free algebra
accessor                Accessor methods for freealg objects
adjoint                 The adjoint map
constant                The constant term
deriv                   Differentiation of 'freealg' objects
dot-class               Class "dot"
freealg                 The free algebra
freealg-package         The Free Algebra
grade                   The grade (or degree) of terms in a 'freealg'
                        object
horner                  Horner's method
linear                  A simple free algebra object
nterms                  Number of terms in a freealg object
pepper                  Combine variables in every possible order
print.freealg           Print freealg objects
rfalg                   Random free algebra objects
```

subs	Substitution
zero	The zero algebraic object

Author(s)

NA

Maintainer: Robin K. S. Hankin <hankin.robin@gmail.com>

Examples

```
a <- as.freealg("x+xyx")
b <- as.freealg("4x +XyX")  # upper-case interpreted as inverse

a+b
stopifnot(a+b==b+a)   # should be TRUE

a*b ==b*a # FALSE; noncommutative algebra

as.freealg("1+X+xy")^3

rfalg()
rfalg()^2
```

`accessor` *Accessor methods for freealg objects*

Description

Accessor methods for free algebra objects

Usage

```
words(x)
coeffs(x)
coeffs(x) <- value
```

Arguments

x	Object of class `freealg`
value	Numeric vector of length 1

Details

Access or set the different parts of an `freealg` object. The constant term is technically a coefficient but is documented under `constant.Rd`.

Note

There is an extended discussion of this issue in the `mvp` object at `accessor.Rd`.

Author(s)

Robin K. S. Hankin

See Also

`constant`

Examples

```
a <- rfalg()
a
coeffs(a)
words(a)  # Note hash is identical to that of coeffs(a)

coeffs(a) <- 7   # replacement methods work
a
coeffs(a)  #
```

`adjoint` *The adjoint map*

Description

The adjoint ad_X of X is a map from a Lie group G to the endomorphism group of G defined by

$$\mathrm{ad}_X(Y) = [X, Y]$$

Usage

```
ad(x)
```

Arguments

`x`	Object nominally of class `freealg` but other classes accepted where they make sense

Details

details here

Note

Vignette `adjoint` gives more description

Author(s)

Robin K. S. Hankin

Examples

```
x <- rfalg()
y <- rfalg()

f <- ad(x)
f(y)


f(f(f(y))) # [x,[x,[x,y]]]
```

---

constant *The constant term*

Description

Get and set the constant term of a `freealg` object

Usage

```
## S3 method for class 'freealg'
constant(x)
## S3 method for class 'numeric'
constant(x)
## S3 replacement method for class 'freealg'
constant(x) <- value
is.constant(x)
```

Arguments

`x`	Object of class `freealg`
`value`	Scalar value for the constant

Details

The constant term in a free algebra object is the coefficient of the empty term. In a `freealg` object, the map including `{} -> v` implies that `v` is the constant.

If `x` is a `freealg` object, `constant(x)` returns the value of the constant in the multivariate polynomial; if `x` is numeric, it returns a constant `freealg` object with value `x`.

Function `is.constant()` returns `TRUE` if its argument has no variables and `FALSE` otherwise.

Setting the coefficients of the empty `freealg` returns the zero (empty) object.

Author(s)

Robin K. S. Hankin

Examples

```
p <- as.freealg("1+X+Y+xy")

constant(p)
constant(p^5)

constant(p) <- 1000
p
```

`deriv` *Differentiation of* `freealg` *objects*

Description

Differentiation of `freealg` objects

Usage

```
## S3 method for class 'freealg'
deriv(expr, r, ...)
```

Arguments

`expr`	Object of class `freealg`
`r`	Integer vector. Elements denote variables to differentiate with respect to. If `r` is a character vector, it is interpreted as `a=1,b=2,...,z=26`; if of length 1, "`aab`" is interpreted as `c("a","a","b")`
`...`	Further arguments, currently ignored

Details

Experimental function `deriv(S,v)` returns $\frac{\partial^r S}{\partial v_1 \partial v_2 \ldots \partial v_r}$. The Leibniz product rule

$$(u \cdot v)' = uv' + u'v$$

operates even if (as here) u, v do not commute. For example, if we wish to differentiate $aaba$ with respect to a, we would write $f(a) = aaba$ and then

$$f(a + \delta a) = (a + \delta a)(a + \delta a)b(a + \delta a)$$

and working to first order we have

$$f(a + \delta a) - f(a) = (\delta a)aba + a(\delta a)ba + aab(\delta a).$$

In the package:

```
> deriv(as.freealg("aaba"),"a")
free algebra element algebraically equal to
+ 1*aab(da) + 1*a(da)ba + 1*(da)aba
```

A term of a `freealg` object can include negative values which correspond to negative powers of variables. Thus:

```
> deriv(as.freealg("AAAA"),"a")
free algebra element algebraically equal to
- 1*AAAA(da)A - 1*AAA(da)AA - 1*AA(da)AAA - 1*A(da)AAAA
```

(see also the examples). Vector `r` may include negative integers which mean to differentiate with respect to the inverse of the variable:

```
> deriv(as.freealg("3abcbCC"),"C")
free algebra element algebraically equal to
+ 3*abcbC(dC) + 3*abcb(dC)C - 3*abc(dC)cbCC
```

It is possible to perform repeated differentiation by passing a suitable value of `r`. For $\frac{\partial^2}{\partial a \partial c}$:

```
> deriv(as.freealg("aaabAcx"),"ac")
free algebra element algebraically equal to
- 1*aaabA(da)A(dc)x + 1*aa(da)bA(dc)x + 1*a(da)abA(dc)x + 1*(da)aabA(dc)x
```

The infinitesimal indeterminates (“da” etc) are represented by `SHRT_MAX+r`, where `r` is the integer for the symbol, and `SHRT_MAX` is the maximum short integer. This includes negative `r`. So the maximum number for any symbol is `SHRT_MAX`. Inverse elements such as `A`, being represented by negative integers, have differentials that are `SHRT_MAX-r`.

Function `deriv()` calls helper function `lowlevel_diffn()` which is documented at `Ops.freealg.Rd`.

A vignette illustrating this concept and furnishing numerical verification of the code in the context of matrix algebra is given at `inst/freealg_matrix.Rmd`.

Author(s)

Robin K. S. Hankin

Examples

```
deriv(as.freealg("4*aaaabaacAc"),1)

x <- rfalg()
deriv(x,1:3)

y <- rfalg(7,7,17,TRUE)

deriv(y,1:5)-deriv(y,sample(1:5)) # should be zero
```

dot-class *Class "dot"*

Description

The dot object is defined so that idiom like `.[x,y]` returns the commutator, that is, `xy-yx` or the Lie bracket $[x, y]$. It would have been nice to use `[x,y]` (that is, without the dot) but although this is syntactically consistent, it cannot be done in R.

The "meat" of the package is:

```
setClass("dot", slots = c(ignore='numeric'))
`.` <- new("dot")
setMethod("[",signature(x="dot",i="ANY",j="ANY"),function(x,i,j,drop){i*j-j*i})
```

The package code includes other bits and pieces such as informative error messages for idiom such as `.[]`. The package defines a matrix method for the dot object. This is because "`*`" returns (incorrectly, in my view) the elementwise product, not the matrix product.

The Jacobi identity, satisfied by any associative algebra, is

$$[x, [y, z]] + [y, [z, x]] + [z, [x, y]] = 0$$

and the left hand side is returned by `jacobi()`, which should be zero (for some definition of "zero").

Function `ad()` returns the adjoint operator. The `adjoint` vignette provides details and examples of the adjoint operator.

The dot object is generated by running script `inst/dot.Rmd`, which includes some further discussion and technical documentation, and creates file `dot.rda` which resides in the `data/` directory.

Value

Always returns an object of the same class as `xy`.

Slots

`ignore`: Object of class `"numeric"`, just a formal placeholder

Methods

`[` `signature(x = "dot", i = "ANY", j = "ANY")`: ...

`[` `signature(x = "dot", i = "ANY", j = "missing")`: ...

`[` `signature(x = "dot", i = "function", j = "function")`: ...

`[` `signature(x = "dot", i = "matrix", j = "matrix")`: ...

`[` `signature(x = "dot", i = "missing", j = "ANY")`: ...

`[` `signature(x = "dot", i = "missing", j = "missing")`: ...

Author(s)

Robin K. S. Hankin

Examples

```
.[as.freealg("x"),as.freealg("y")]
.[as.freealg("x"),as.freealg("y+2z")]
.[as.freealg("x+y+2xYx"),as.freealg("x+y+2xYx")]


x <- rfalg()
y <- rfalg()
z <- rfalg()

jacobi(x,y,z) # Jacobi identity
.[x,.[y,z]] + .[y,.[z,x]] + .[z,.[x,y]]  # Jacobi, expanded


f <- ad(x)
f(y)


rM <- function(...){matrix(sample(1:9,9),3,3)} # a random matrix

M <- rM()
N <- rM()
O <- rM()

.[M,N]
jacobi(M,N,O)
```

`freealg` *The free algebra*

Description

Create, test for, and coerce to, `freealg` objects

Usage

```
freealg(words, coeffs)
is_ok_free(words,coeffs)
is.freealg(x)
as.freealg(x,...)
char_to_freealg(ch)
natural_char_to_freealg(string)
string_to_freealg(string)
vector_to_free(v,coeffs)
```

Arguments

`words`	Terms of the algebra object, eg `c(1,2,-1,-3,-2)` corresponds to `abACB` because $a = 1$, $b = 2$ etc; uppercase, or negative number, means inverse
`coeffs`	Numeric vector corresponding to the coefficients of each element of the `word` list

`string`	Character string
`ch`	Character vector
`v`	Vector of integers
`x`	Object possibly of class `freealg`
`...`	Further arguments, passed to the methods

Details

Function `freealg()` is the formal creation mechanism for `freealg` objects. However, it is not very user-friendly; it is better to use `as.freealg()` in day-to-day use.

Function `is_ok_freealg()` checks for consistency of its arguments.

A `freealg` object is a two-element list. The first element is a list of integer vectors representing the indices and the second is a numeric vector of coefficients. Thus, for example:

```
> as.freealg("a+4bd+3abbbbc")
free algebra element algebraically equal to
 + 1*a + 3*abbbbc + 4*bd
> dput(as.freealg("a+4bd+3abbbbc"))
structure(list(indices = list(1L, c(1L, 2L, 2L, 2L, 2L, 3L),
    c(2L, 4L)), coeffs = c(1, 3, 4)), class = "freealg")
```

Observe that the order of the terms is not preserved and indeed is undefined (implementation-specific). Zero entries are stripped out.

Character strings may be coerced to `freealg` objects; `as.freealg()` calls `natural_char_to_freealg()`, which is user-friendly. Functions `char_to_freealg()` and `string_to_freealg()` are low-level helper functions. These functions assume that upper-case letters are the multiplicative inverses of the lower-case equivalents; so for example `as.freealg("aA")` and `as.freealg(aBcCbA)` evaluate to one. This can be confusing with the default print method.

Even though individual symbols have multiplicative inverses, a general element of the free algebra will not have a multiplicative inverse. For example, `1+x` does not have an inverse. The free algebra is not a division algebra, in general.

Note

Internally, the package uses signed integers and as such can have `.Machine$integer.max` different symbols; on my machine this is 2147483647. Of course the print method cannot deal with this as it only has 26 symbols for letters a-z (and A-Z for the inverses), but the objects themselves do not care about the print method. Note also that the experimental calculus facility (as per `deriv()`) reserves numbers in the range `SHRT_MAX` $\pm r$ for infinitesimals, where `r` is the integer for a symbol. This system might change in the future.

Author(s)

Robin K. S. Hankin

Examples

```
freealg(list(1:2, 2:1,numeric(0),1:6),1:4)

freealg(sapply(1:5,seq_len),1:5)
freealg(replicate(5,sample(-5:5,rgeom(1,1/5),replace=TRUE)),1:5)


as.freealg("1+xaX")^5
```

`grade` *The grade (or degree) of terms in a* `freealg` *object*

Description

The free algebra $\mathcal{B}$ is a *graded* algebra: that is, for each integer $n \geq 0$ there is a homogeneous subspace $\mathcal{B}_n$ with $\mathcal{B}_0 = \mathcal{R}$ and

$$\mathcal{B} = \bigoplus_{n=0}^{\infty} \mathcal{B}_n, \quad \text{and} \quad \mathcal{B}_n\mathcal{B}_m \subseteq \mathcal{B}_{n+m} \quad \text{for all } m, n \geq 0.$$

The elements of $\cup_{n\geq 0}\mathcal{B}_n$ are called *homogeneous* and those of $\mathcal{B}_n$ are called homogenous of degree (or grade) n.

The grade of a term is the number of symbols in it. Thus the grade of `xxx` and `4xxy` is 3; the grade of a constant is zero. Because the terms are stored in an implementation-specific way, the grade of a multi-term object is a `disord` object.

Usage

```
grades(x)
grade(x,n)
grade(x,n) <- value
```

Arguments

`x`	Freealg object
`n`	Integer vector
`value`	Replacement value, a numeric vector

Details

`grades(x)` returns the grade (number of symbols) in each term of a `freealg` object `x`.

`grade(x,n)` returns the freealg object comprising terms with grade `n` (which may be a vector). Note that this function is considerably less efficient than `clifford::grade()`.

`grade(x,n) <- value` sets the coefficients of terms with grade `n`. For `value`, a length-one numeric vector is accepted (notably zero, which kills terms of grade `n`) and also a `freealg` object comprising terms of grade coden.

Value

Returns a disord object

Note

A similar concept *grade* is discussed in the **clifford** package

Author(s)

Robin K. S. Hankin

References

H. Munthe-Kaas and B. Owren 1999. “Computations in a free Lie algebra”, *Phil. Trans. R. Soc. Lond. A*, 357:957–981 (theorem 3.8)

Examples

```
X <- as.freealg("1 -x + 5*y + 6*x*y -8*x*x*x*x*x*y*x")
X
grades(X)

a <- rfalg(30)
a
grades(a)
grade(a,2)
grade(a,2) <- 0 # kill all grade-2 terms
a

grade(a,1) <- grade(a,1) * 888
a
```

`horner` *Horner’s method*

Description

Horner’s method for multivariate polynomials

Usage

```
horner(P,v)
```

Arguments

`P`	Free algebra polynomial
`v`	Numeric vector of coefficients

Details

This function is (almost) the same as `mvp::horner()`.

Given a polynomial

$$p(x) = a_0 + a_1 + a_2x^2 + \cdots + a_nx^n$$

it is possible to express $p(x)$ in the algebraically equivalent form

$$p(x) = a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + xa_n\right)\cdots\right)\right)$$

which is much more efficient for evaluation, as it requires only n multiplications and n additions, and this is optimal. Function `horner()` will take a `freealg` object for its first argument.

Author(s)

Robin K. S. Hankin

Examples

```
horner("x",  1:4)  # note constant term is 1.

horner("x+y",1:3) # note presence of xy and yx terms

horner("1+x+xyX",1:3)
```

`linear` *A simple free algebra object*

Description

Create simple free algebra objects including linear expressions, for example

```
> linear(1:3)
free algebra element algebraically equal to
+ 1*a + 2*b + 3*c
> linear(1:3,power=5)
free algebra element algebraically equal to
+ 1*aaaaa + 2*bbbbb + 3*ccccc
>
```

Usage

```
linear(x,power=1)
```

Arguments

`x`	Numeric vector of terms
`power`	Integer vector of powers

Note

Many of the functions documented at `mvp::special.Rd` do not make sense in the context of the free algebra. Function `mvp::product()`, for example, imposes an order on the expansion.

Function `constant()` is documented at `constant.Rd`, but is listed below for convenience.

Author(s)

Robin K. S. Hankin

See Also

constant, zero

Examples

```
linear(1:3)
linear(1:3,power=5)
linear(1:3,power=3:1)
```

---

nterms *Number of terms in a freealg object*

Description

Number of terms in a freealg object; number of coefficients

Usage

```
nterms(x)
```

Arguments

x	Freealg object

Value

Returns a non-negative integer

Author(s)

Robin K. S. Hankin

Examples

```
(a <- freealg(list(1:3,4:7,1:10),1:3))
nterms(a)
nterms(a+1)
nterms(a*0)
```

`Ops.freealg` *Arithmetic Ops methods for the the free algebra*

Description

Arithmetic operators for manipulation of freealg objects such as addition, multiplication, powers, etc

Usage

```
## S3 method for class 'freealg'
Ops(e1, e2)
free_negative(S)
free_power_scalar(S,n)
free_eq_free(e1,e2)
free_plus_numeric(S,x)
free_plus_free(e1,e2)
lowlevel_simplify(words,coeffs)
lowlevel_free_prod(words1,coeffs1,words2,coeffs2)
lowlevel_free_sum(words1,coeffs1,words2,coeffs2)
lowlevel_free_power(words,coeffs,n)
lowlevel_diffn(words,coeffs,r)
lowlevel_subs(words1, coeffs1, words2, coeffs2, r)
inv(S)
```

Arguments

`S,e1,e2`	Objects of class `freealg`
`n`	Integer, possibly non-positive
`r`	Integer vector indicating variables to differentiate with respect to
`x`	Scalar value
`words,words1,words2`	A list of words, that is, a list of integer vectors representing the variables in each term
`coeffs,coeffs1,coeffs2`	Numeric vector representing the coefficients of each word

Details

The function `Ops.freealg()` passes binary arithmetic operators (“+”, “-”, “*”, “^”, and “==”) to the appropriate specialist function.

The caret, as in `a^n`, denotes arithmetic exponentiation, as in `x^3==x*x*x`. The only comparison operators are equality and inequality; `x==y` is defined as `is.zero(x-y)`.

Functions `lowlevel_foo()` are low-level functions that interface directly with the `C` routines in the `src/` directory and are not intended for the end-user.

Function `inv()` is defined only for freealg objects with a single term. If `x` has a single term we have `inv(x)*x=x*inv(x)=1`. There is no corresponding division in the package because `a/b` may be either `a*inv(b)` or `inv(b)*a`.

Author(s)

Robin K. S. Hankin

Examples

```
rfalg()
as.freealg("1+x+xy+yx")  # variables are non-commutative
as.freealg("x") * as.freealg("X") # upper-case letters are lower-case inverses

constant(as.freealg("x+y+X+Y")^6)  # OEIS sequence A035610

inv(as.freealg("2aaabAAAAx"))
```

---

pepper *Combine variables in every possible order*

Description

Given a list of variables, construct every term comprising only those variables; function `pepper()` returns a free algebra object equal to the sum of these terms.

The function is named for a query from an exam question set by Sarah Marshall in which she asked how many ways there are to arrange the letters of word "pepper", the answer being $\binom{6}{1\,2\,3} = \frac{6!}{1!2!3!} = 60$.

Function `multiset()` in the `partitions` package gives related functionality.

Usage

```
pepper(v)
```

Arguments

`v`	Variables to combine. If a character string, coerce to variable numbers

Author(s)

Robin K. S. Hankin

See Also

`linear`

Examples

```
pepper(c(1,2,2,2,3))
pepper("pepper")
```

`print` *Print freealg objects*

Description

Print methods for free algebra objects

Usage

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

Arguments

`x`	Object of class `freealg` in the print method
`...`	Further arguments, currently ignored

Note

The print method does not change the internal representation of a `freealg` object, which is a two-element list, the first of which is a list of integer vectors representing words, and the second is a numeric vector of coefficients.

The print method has special dispensation for length-zero freealg objects but these are not handled entirely consistently.

The print method is sensitive to the value of `getOption("usecaret")`, defaulting to "no". The default is to use uppercase letters to represent multiplicative inverses, but if `TRUE`, inverses are indicated using "`^-1`". This becomes cumbersome for powers above the first. For example, the default notation for aba^{-2} is `abAA` but becomes `aba^-1a^-1` if `usecaret` is `TRUE`.

Integers exceeding `SHRT_MAX` are reserved for infinitesimals, which are printed as "da"; see the note at `deriv.Rd` for details.

Author(s)

Robin K. S. Hankin

See Also

`freealg`,`deriv`

Examples

```
rfalg()

x <- rfalg(inc=TRUE)
x                         # default
options("usecaret" = TRUE)  # use caret
x
options("usecaret" = FALSE) # back to the default
x
```

rfalg *Random free algebra objects*

Description

Random elements of the free algebra, intended as quick "get you going" examples of `freealg` objects

Usage

```
rfalg(n=7, distinct=3, maxsize=4, include.negative=FALSE)
```

Arguments

`n`	Number of terms to generate
`distinct`	Number of distinct symbols to use
`maxsize`	Maximum number of symbols in any word
`include.negative`	Boolean, with default `FALSE` meaning to use only positive symbols (lower-case letters) and `TRUE` meaning to use upper-case letters as well, corresponding to the inverse of the lower-case symbols

Details

What you see is what you get, basically. A term such as `aaBaAbaC` will be simplified to `aaaC`.

Author(s)

Robin K. S. Hankin

Examples

```
rfalg()
rfalg()^3

constant(rfalg())
```

subs *Substitution*

Description

Substitute symbols in a `freealg` object for numbers or other `freealg` objects

Usage

```
subs(S, ...)
subsu(S1,S2,r)
```

Arguments

`S,S1,S2`	Objects of class `freealg`
`r`	Integer specifying symbol to substitute ($a = 1, b = 2$ etc)
`...`	named arguments corresponding to variables to substitute

Details

Function `subs()` substitutes variables for `freealg` objects (coerced if necessary) using a natural R idiom. Observe that this type of substitution is sensitive to order:

```
> subs("ax",a="1+x",x="1+a")
free algebra element algebraically equal to
 + 2 + 3*a + 1*aa

> subs("ax",x="1+a",a="1+x")
free algebra element algebraically equal to
 + 2 + 3*x + 1*xx
```

Functions `subsu()` is a lower-level formal function, not really intended for the end-user. Function `subsu()` takes `S1` and substitutes occurrences of symbol `r` with `S2`.

No equivalent to `mvp::subvec()` is currently implemented.

Value

Returns a `freealg` object.

Author(s)

Robin K. S. Hankin

Examples

```
subs("abccc",b="1+3x")
subs("aaaa",a="1+x")  # binomial

subs("abA",b=31)

subs("1+a",a="A")   # can substitute for an inverse
subs("A",a="1+x")   # inverses are not substituted for


## Sequential substitution works:

subs("abccc",b="1+3x",x="1+d+2e")
subs(rfalg(),a=rfalg())
```

`zero` *The zero algebraic object*

Description

Test for a `freealg` object's being zero

Usage

```
is.zero(x)
```

Arguments

`x`	Object of class `freealg`

Details

Function `is.zero()` returns `TRUE` if `x` is indeed the zero free algebra object. It is defined as `length(coeffs(x))==0` for reasons of efficiency, but conceptually it returns `x==constant(0)`.

(Use `constant(0)` to create the zero object).

Author(s)

Robin K. S. Hankin

See Also

`constant`

Examples

```
stopifnot(is.zero(constant(0)))
```

Index