Package 'freegroup'

March 20, 2025

Type Package

Title The Free Group

Version 1.2-0

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

Description The free group in R; juxtaposition is represented by a plus. Includes inversion, multiplication by a scalar, group-theoretic power operation, and Tietze forms. To cite the package in publications please use Hankin (2022) <doi:10.48550/ARXIV.2212.05883>.

Depends methods, plyr, R (>= 4.1.0)

Suggests knitr, rmarkdown, permutations, testthat, covr

LazyData yes

Imports freealg (>= 1.0-4), magic (>= 1.5-9), magrittr

VignetteBuilder knitr

License GPL-2

URL https://github.com/RobinHankin/freegroup, https://robinhankin.github.io/freegroup/

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

NeedsCompilation no

Author Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Repository CRAN

Date/Publication 2025-03-20 11:40:02 UTC

Contents

`freegroup-package` *The Free Group*

Description

The free group in R; juxtaposition is represented by a plus. Includes inversion, multiplication by a scalar, group-theoretic power operation, and Tietze forms. To cite the package in publications please use Hankin (2022) <doi:10.48550/ARXIV.2212.05883>.

Details

The DESCRIPTION file:

Package:	freegroup
Type:	Package
Title:	The Free Group
Version:	1.2-0
Authors@R:	c(person(c("Robin", "K. S. "), "Hankin", role=c("aut","cre"), email="hankin.robin@gmail.com", comment
Maintainer:	Robin K. S. Hankin <hankin.robin@gmail.com>
Description:	The free group in R; juxtaposition is represented by a plus. Includes inversion, multiplication by a scalar, g
Depends:	methods, plyr, R (>= 4.1.0)
Suggests:	knitr, rmarkdown, permutations, testthat, covr

LazyData:	yes
Imports:	freealg (>= 1.0-4), magic (>= 1.5-9), magrittr
VignetteBuilder:	knitr
License:	GPL-2
URL:	https://github.com/RobinHankin/freegroup, https://robinhankin.github.io/freegroup/
BugReports:	https://github.com/RobinHankin/freegroup/issues
Author:	Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Index of help topics:

Author(s)

Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

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

Examples

```
p <- rfree(10,6,3)
x <- as.free('x')
```

```
p+x

p^x

sum(p)

abelianize(p)

subsu(p,"ab","z")
subs(p,a='z')


discard(p+x,'a')
```

`abelianize` *Abelianization of free group elements*

Description

Function `abelianize()` returns a word that is equivalent to its argument under assumption of Abelianness. The symbols are placed in alphabetical order.

Usage

```
abelianize(x)
is.abelian(x)
```

Arguments

`x`	An object of class `free`

Details

Abelianizing a free group element means that the symbols can commute past one another. Abelianization is vectorized.

Function `is.abelian()` is trivial: it just checks to see whether argument `x` has its symbols in alphabetical order. It might have been better to call this `abelianized()`.

Package **frab** presents extensive R-centric functionality for dealing with the free Abelian group. It is much more efficient than this package for Abelian operations, and contains bespoke methods for working with a range of applications such as tables of counts.

Author(s)

Robin K. S. Hankin

Examples

```
x <- as.free("aabAA")
x
abelianize(x)

x <- rfree(10,10,2)
x
abelianize(x)

abelianize(.[rfree(),rfree()])


p <- free(rbind(rep(1:5,4),rep(1:4,5)))
p
abelianize(p)
```

`abs.free` *Absolute value of a* `free` *object*

Description

Replaces every term's power with its absolute value

Usage

```
## S3 method for class 'free'
abs(x)
```

Arguments

`x`	Object of class `free`

Details

Replaces every term's power with its absolute value

Note

The function's name is motivated by the inequality in the examples section.

Author(s)

Robin K. S. Hankin

See Also

`subs`

Examples

```
abs(abc(-5:5))

a <- rfree(10,4,7)
b <- rfree(10,4,7)

a
abs(a)

## following should all be TRUE:
all(size(abs(a+b))  <=  size(abs(a) + abs(b)))
all(total(abs(a+b)) <=  total(abs(a) + abs(b)))
all(number(abs(a+b)) <= number(abs(a) + abs(b)))

all(size(a+b)   <= size(abs(a) + abs(b)))
all(total(a+b)  <= total(abs(a) + abs(b)))
all(number(a+b) <= number(abs(a) + abs(b)))
```

---

alpha *Alphabetical free group elements*

Description

Produces simple vectors of free group elements based on the alphabet

Usage

```
alpha(v)
abc(v)
```

Arguments

v	Vector of integers

Details

Function `alpha()` takes an integer `i` and returns the letter `i` of the alphabet. Thus `alpha(3)` returns `c`. The function is vectorised: `alpha(1:3)` returns `a b c`.

Function `abc()` takes an integer `i` and returns letters 1 to `i` of the alphabet. Thus `abc(4)` returns `a.b.c.d`. The function is vectorised.

Remember that "letters of the alphabet" is just a phrase: above it refers to the default print method which can be changed, see the examples.

Author(s)

Robin K. S. Hankin

Examples

```
alpha(5)  # just the single letter 'e'
abc(5)    # product of a,b,c,d,e

alpha(1:26)  # the whole alphabet; c

all(alpha(1:26) == as.free(letters))  # should be TRUE

z <- alpha(26)  # variable 'z' is symbol 26, aka 'z'.
abc(1:10) ^ z

abc(-5:5)
alpha(-5:5)
sum(abc(-5:5))


## bear in mind that the symbols used are purely for the print method:
jj <- LETTERS[1:10]
options(freegroup_symbols = apply(expand.grid(jj,jj),1,paste,collapse=""))
alpha(c(66,67,68,69))   # sensible output
options(freegroup_symbols=NULL)   # restore to symbols to default letters
alpha(c(66,67,68,69))   # print method not very helpful now
```

---

backwards *Write free objects backwards*

Description

Write free objects in reverse order

Usage

```
backwards(x)
```

Arguments

x	Object of class `free`

Note

For each element of a `free` object, function `backwards()` writes the symbols in reverse order. It is distinct from `rev()`, see examples.

Function backwards is an involution: it is its own inverse.

Author(s)

Robin K. S. Hankin

Examples

```
abc(1:5)
backwards(abc(1:5))
rev(abc(1:5))

x <- rfree(10,5)
backwards(backwards(x)) == x  # involution
all(abelianize(x) == abelianize(backwards(x))) # should be TRUE
```

---

c *Concatenation of free objects*

Description

Concatenate free objects together

Usage

```
## S3 method for class 'free'
c(...)
## S3 method for class 'free'
rep(x, ...)
```

Arguments

...	In the method for c(), objects to be concatenated. Should all be of the same type
x	In the method for rep(), a free object

Author(s)

Robin K. S. Hankin

Examples

```
(x <- abc(1:3))
(y <- alpha(22:25))

c(x,y,x,x)


## NB: compare
rep(x,2)
x*2
```

`char_to_free` *Convert character vectors to free objects*

Description

Convert character vectors to free objects

Usage

```
char_to_matrix(x)
```

Arguments

`x`	A character vector

Details

Function `char_to_matrix()` gives very basic conversion between character vectors and free objects. Current functionality is limited to strings like "`aaabaacd`", which would give a^3ba^2cd. It would be nice to take a string like "`a^3b^(-3)`" but this is not yet implemented.

Function `char_to_free()` is a vectorized version that coerces output to `free`.

Note

The function is not particularly robust; for example, passing anything other than letters a-z or A-Z will give possibly undesirable behaviour.

Upper-case letters A-Z are interpreted by `char_to_matrix()` as the inverse of their corresponding lower-case equivalents. This behaviour is inherited by `char_to_free()` and `as.free()`, so that `as.free("A") == inverse(as.free("a"))`.

Function `char_to_free()` is consistent with the default print options (which are that the symbols are the lowercase letters a-z). If you change the symbols' names, for example `options(freegroup_symbols=sample(lette` then things can get confusing. The print method does not change the internal representation of a `free` object, which is a list of integer matrices.

Author(s)

Robin K. S. Hankin

See Also

`print.free`

Examples

```
char_to_matrix("aaabcABC")

rfree(10,3) + as.free('xxxxxxxxxxx')

as.free(letters)*7

all(is.id(as.free(letters) + as.free(LETTERS)))


as.free('')  # identity element
```

---

cumsum *Cumulative sum*

Description

Cumulative sum of free vectors

Usage

```
## S3 method for class 'free'
cumsum(x)
```

Arguments

x	Vector of class free

Author(s)

Robin K. S. Hankin

See Also

sum

Examples

```
abc(1:6)
cumsum(abc(1:6))

x <- rfree(10,2)
cumsum(c(x,-rev(x)))
```

cycred *Cyclic reductions of a word*

Description

Functionality to cyclically reduce words and detect conjugacy

Usage

```
is.cyclically_reduced(a)
as.cyclically_reduced(a)
cyclically_reduce(a)
cyclically_reduce_tietze(p)
is.conjugate_single(u,v)
x %~% y
## S3 method for class 'free'
is.conjugate(x,y)
allconj(x)
```

Arguments

a, x, y	An object of class `free`
p, u, v	Integer vector corresponding to Tietze form of a word

Details

A `free` object is *cyclically reduced* iff every cyclic permutation of the word is reduced. A reduced word is cyclically reduced iff the first letter is not the inverse of the last one. A reduced word is cyclically reduced if the first and last symbol differ (irrespective of power) or, if identical, have powers of opposite sign. For example, `abac` and `abca` are cyclically reduced but `abca^{-1}` is not. Function `is.cyclically_reduced()` tests for this.

Function `as.cyclically_reduced()` takes a vector of free objects and returns the elementwise cyclically reduced equivalents. Function `cyclically_reduce()` is a synonym with better (English) grammar.

The identity is cyclically reduced: it cannot be shortened by a combination of cyclic permutation followed by reduction. This ensures that `is.cyclically_reduced(as.cyclically_reduced(x))` is always TRUE. Also, it is clear that the identity should be conjugate to itself.

Two words a, b are *conjugate* if there exists a x such that $ax = xb$ (or equivalently $a = x^{-1}bx$). This is detected by function `is.conjugate()`. Functions `is_conjugate_single()` and `cyclically_reduce_tietze()` are lower-level helper functions.

Function `allconj()` returns all cyclically reduced words conjugate to its argument.

Author(s)

Robin K. S. Hankin

See Also

reduce

Examples

```
(x <- abc(1:9) - abc(9:1))
as.cyclically_reduced(x)

 a <- rfree(1000,3)
 all(size(as.cyclically_reduced(a)) <= size(a))
 all(total(as.cyclically_reduced(a)) <= total(a))
 all(number(as.cyclically_reduced(a)) <= number(a))



 x <- rfree(1000,2)
 y <- as.free('ab')
 table(conjugate = (x%~%y), equal = (x==y))  # note zero at top right

 allconj(as.free('aaaaab'))
 allconj(sum(abc(seq_len(3))))



 x <- rfree(1,10,8,8)
 all(is.id(allconj(x) + allconj(-x)[shift(rev(seq_len(total(x))))]))
```

---

donames *Names attributes of free group elements*

Description

Get and set names of free group elements and arithmetic operations

Usage

```
donames(f,e1,e2)
```

Arguments

f	A vector, typically of class free
e1, e2	Objects of class free, possibly with names

Details

Function `donames()` is a low-level helper function that ensures that the result of arithmetic operations such as `+` and `^` have the correct names attributes. The behaviour is inherited from that of `base::`+``.

Author(s)

Robin K. S. Hankin

See Also

`Ops.free`

Examples

```
x <- rfree(9,4)
x
names(x) <- letters[1:9]
x

z <- as.free('z')
x + x
x^z
z^x

n <- 1:9
names(n) <- LETTERS[1:9]

x*n
n*x  # note different names
```

---

dot-class *Class "dot"*

Description

The dot object is defined in the **freealg** package, and imported here, so that idiom like `.[x,y]` returns the commutator, that is, `x^-1 y^-1 xy`.

Arguments

`x`	Object of any class
`i, j`	elements to commute
`...`	Further arguments to `dot_error()`, currently ignored

Value

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

Author(s)

Robin K. S. Hankin

Examples

```
.[as.free("x"), as.free("y")]

.[abc(1:6),"z"]

x <- rfree()
y <- rfree()
z <- rfree()

.[x,y] == -x-y+x+y   # should be TRUE

abelianize(.[x,y])

## Jacobi identity _not_ satisfied with this definition:
is.id(.[x,.[y,z]] + .[y,.[z,x]] + .[z,.[x,y]])

## But the Hall-Witt identity is:
all(is.id(.[.[x,-y],z]^y + .[.[y,-z],x]^z + .[.[z,-x],y]^x))
```

`Extract` *Extract or replace parts of a free group object*

Description

Extract or replace subsets of free objects

Arguments

`x`	Object of class `free`
`index`	elements to extract or replace
`value`	replacement value

Details

These methods (should) work as expected: an object of class `free` is a list but standard extraction techniques should work.

Examples

```
(x <- rfree(20,8,8))

x[5:6]
x[1:2]  <- -x[11:12]

x[1:5] <- keep(x[1:5],1:3)
```

`free` *Objects of class* `free`

Description

Generate, and test for, objects of class `free`

Usage

```
free(x)
as.free(x)
is.free(x)
list_to_free(x)
```

Arguments

`x`	Function `free()` needs either a two-row matrix, or a list of two-row matrices; function `as.free()` attempts to coerce different types of argument before passing to `free()` (possibly via `list_to_free()`)

Details

The basic structure of an element of the free group is a two-row matrix. The top row is the symbols (1=a, 2=b, 3=c, etc) and the bottom row is the corresponding power. Thus $a^2ba^{-1}c^9$ would be

```
> rbind(c(1,2,1,3),c(2,1,-1,9))
     [,1] [,2] [,3] [,4]
[1,]    1    2    1    3
[2,]    2    1   -1    9
>
```

Function `free()` needs either a two-row matrix or a list of two-row matrices. It is the only place in the package that sets the class of an object to `free`. Function `as.free()` is a bit more user-friendly and tries a bit harder to do the Right Thing.

The package uses `setOldClass("free")` for the `dot` methods.

Author(s)

Robin K. S. Hankin

See Also

char_to_free

Examples

```
free(rbind(1:5,5:1))

x <- rfree(10,3)
x
x+x
x-x
x[1:5]*(1:5)


as.free(c(4,3,2,2,2))
as.free("aaaabccccaaaaa")
as.free(c("a","A","abAAA"))
```

---

getlet *Get letters of a freegroup object*

Description

Get the symbols in a freegroup object

Usage

```
getlet(x)
```

Arguments

x	Object of class free

Note

By default, return a list with elements corresponding to the elements of x. But, if object x is of length 1, a vector is returned. The result is sorted for convenience.

Author(s)

Robin K. S. Hankin

Examples

```
(x <- rfree(6,7,3))

getlet(x)

as.free(getlet(x))

identical(as.free(getlet(abc(1:26))), abc(1:26))
```

`identity` *The identity element*

Description

Create and test for the identity element

Usage

```
is.id(x)
id(n)
## S3 method for class 'free'
is.id(x)
```

Arguments

`x`	Object of class `free`
`n`	Strictly positive integer

Details

Function `id()` returns a vector of `n` free objects, all of which are the identity element. Do not ask what happens if $n = 0$.

Function `is.id()` returns a Boolean indicating whether an element is the identity or not. The identity can also be generated using `as.free(0)`.

Author(s)

Robin K. S. Hankin

Examples

```
id()
as.free(0)   # convenient R idiom for creating the identity

x <- rfree(10,3)
stopifnot(all(x == x + as.free(0)))
stopifnot(all(is.id(x-x)))
```

keep *Keep or drop symbols*

Description

Keep or drop symbols

Usage

```
keep(a, yes)
discard(a, no)
```

Arguments

a	Object of class free
yes, no	Specification of symbols to either keep (yes) or discard (no), coerced to a free object

Note

Function keep() needs an explicit return() to prevent it from returning invisibly.

The functions are vectorised in the first argument but not the second.

The second argument—the symbols to keep or discard—is formally a vector of nonnegative integers, but the functions coerce it to a free object. The symbols kept or dropped are the union of the symbols in the elements of the vector. Function discard() was formerly known as drop() but this conflicted with base::drop().

These functions have nothing in common with APL's take() and drop().

Author(s)

Robin K. S. Hankin

Examples

```
(x <- rfree(20,5,8))

keep(x,abc(4))           # keep only symbols a,b,c,d
discard(x,as.free('cde'))   # drop symbols c,d,e

keep(x,alpha(3))  # keep only abc
```

`nielsen` *Outer automorphisms of the free group*

Description

Vectorized functionality to implement outer automorphisms of the free group

Usage

```
permsymb_single_X(X,f)
permsymb_single_f(X,f)
permsymb_vec(X,f)
permsymb(X,f)
autosub_lowlevel(M,e,S)
autosub(X,e,S,automorphism_warning=TRUE)
```

Arguments

`X`, `S`	Object of class `free`
`f`	Permutation function
`M`	Single free group element, in two-row matrix form
`e`	Single element to substitute
`automorphism_warning`	Boolean, with default `TRUE` meaning to give a warning if the requested substitution is not an automorphism and `FALSE` meaning not to give the warning

Details

In 1924, Nielsen showed that the automorphism group of the free group with basis $[x_1, \ldots, x_n]$ is generated by the following four elementary Nielsen transformations:

1. switch x_1 and x_2
2. Cyclically permute $x_1, x_2, \ldots, x_n$ to $x_2, \ldots, x_n, x_1$
3. Replace x_1 with x_1^{-1}
4. Replace x_1 with $x_1 x_2$.

The functions documented here give vectorized methods to effect such outer automorphisms, using the **permutations** package.

Operations 1 and 2 above generate the symmetric group S_n and such automorphisms are effected by function `permsymb()`. Operation 3 is carried out by by `flip()` and operation 4 by `subsymb()`.

Functions `permsymb_single_X()`, `permsymb_single_f()`, `permsymb_vec()` and `subsymb_lowlevel()` are low-level helper functions that are not really suited for the end user; use `permsymb()`, `(flip)` and `subsymb()` instead.

Note

Function `permsymb()` is intended to work nicely with the **permutations** package; see `inst/outer.Rmd` for some illustrations. The function is not perfect.

Author(s)

Robin K. S. Hankin

References

Wikipedia contributors. (2018, October 29). "Automorphism group of a free group". In *Wikipedia, The Free Encyclopedia*. Retrieved 19:58, January 10, 2019, from https://en.wikipedia.org/w/index.php?title=Automorphism_group_of_a_free_group&oldid=866270661

See Also

`flip`

Examples

```
P <- as.free(c("abc","aba","cc","ca"))
autosub(P,"c",as.free("xyz"))

flip(P,"c")
flip(P,"ac")
```

`Ops.free` *Arithmetic Ops methods for the free group*

Description

Allows arithmetic operators to be used for manipulation of free group elements such as addition, multiplication, powers, etc

Usage

```
## S3 method for class 'free'
Ops(e1, e2)
free_equal(e1,e2)
free_power(e1,e2)
free_repeat(e1,n)
juxtapose(e1,e2)
## S3 method for class 'free'
inverse(e1)
## S3 method for class 'matrix'
inverse(e1)
```

Arguments

e1, e2	Objects of class free
n	An integer, possibly non-positive

Details

The function Ops.free() passes binary arithmetic operators ("+", "-", "*", "^", and "==") to the appropriate specialist function.

There are two non-trivial basic operations: juxtaposition, denoted "a+b", and inversion, denoted "-a". Note that juxtaposition is noncommutative and a+b will not, in general, be equal to b+a.

All operations return a reduced word.

The caret, as in a^b, denotes group-theoretic exponentiation (-b+a+b); the notation is motivated by the identities x^(yz) == (x^y)^z and (xy)^z == x^z * y^z, as in the permutations package.

As an experimental feature the package now accepts multiplicative notation, so these identities manifest in package idiom as written. However, this renders distributivity incorrect so that x*(y + z) and x*y + x*z are not equal, in general [distributivity manifests as x*c(y, z) == c(x*y, x*z)].

Multiplication between a free object a and an integer n (as in a*n or n*a) is defined as juxtaposing n copies of a and reducing. Zero and negative values of n work as expected.

Comparing a free object with a numeric does not make sense and idiom such as rfree() > 4 will return an error. Comparing a free object with another free object might be desirable [specifically, lexicographic ordering], but is not currently implemented.

Note

The package uses additive notation but multiplicative notation might have been better.

Author(s)

Robin K. S. Hankin

Examples

```
x <- as.free(c("a","ab","aaab","abacc"))
y <- as.free(c("aa","BA","Bab","aaaaa"))
x
y


x + x
x + y
x + as.free("xyz")

x+y == y+x    # not equal in  general

x*5 == x+x+x+x+x      # always true

x + alpha(26)
```

```
x^y
```

`primitive` *Primitive elements of the free algebra*

Description

A *primitive word* is one that is not of the form `a^m` for any $m > 1$. The concept is used in Lyndon and Schutzenberger 1962.

Usage

```
is.primitive(x)
is.power(d,n)
```

Arguments

`x`	Freegroup object, coerced to Tietze form
`d`	Numeric vector
`n`	Integer

Details

Function `is.primitive()` returns a boolean vector indicating whether the elements of its argument are primitive.

Function `is.power()` is a lower-level helper function. `is.power(d,n)` determines whether `d` is an `n`-th power (that is, `d` may be written as `n` copies of some numeric vector).

Thus `is_power(c(4,5,7,4,5,7,4,5,7,4,5,7),4)` returns `TRUE` because its primary argument is indeed a fourth power (of `c(4,5,7)`).

Value

Returns a boolean.

Author(s)

Robin K. S. Hankin. The code for finding the factors of an integer was (somewhat more than) inspired by the **numbers** package.

References

R. C. Lyndon and M. P. Schutzenberger 1962. "The equation $a^M = b^N c^P$ in a free group". Michigan Mathematical Journal, 9(4): 289–298.

Examples

```
is.primitive(as.free(c("a","aaaa", "aaaab", "aabaab", "aabcaabcaabcaabc")))

is.power(c(7,8,4,7,8,4,7,8,4,7,8,4),4)


table(is.primitive(rfree(100)))


## primitive with >1 symbols is rare:
x <- rfree(1000)
x[!is.primitive(x) & number(x)>1]
##  try x <- rfree(10000) [but this takes a long time]
```

`print` *Print free objects*

Description

Print methods for free objects

Usage

```
## S3 method for class 'free'
print(x,...)
as.character_free(m,latex=getOption("latex"))
```

Arguments

`x`	Object of class `free` in the print method
`m`	A two-row matrix in function `as.character_free()`
`latex`	Boolean, with `TRUE` meaning to print latex-friendly output including curly braces, and default `NULL` option meaning to give a nicer-looking output that latex would typeset incorrectly
`...`	Further arguments, currently ignored

Note

The print method does not change the internal representation of a `free` object, which is a list of integer matrices.

The default print method uses multiplicative notation (powers) which is inconsistent with the juxtaposition method "+".

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

The default print method uses lowercase letters a-z, but it is possible to override this using `options("freegroup_symbols" = foo)`, where `foo` is a character vector. This is desirable if you have more than 26 symbols, because unallocated symbols appear as `NA`.

The package will allow the user to set `options("freegroup_symbols")` to unhelpful things like `rep("a",20)` without complaining (but don't actually do it, you crazy fool).

Author(s)

Robin K. S. Hankin

See Also

`char_to_free`

Examples

```
## default symbols:

abc(26)
rfree(1,10)


# if we need more than 26:
options(freegroup_symbols=state.name)
rfree(10,4)

# or even:
jj <- letters[1:10]
options(freegroup_symbols=apply(expand.grid(jj,jj),1,paste,collapse=""))
rfree(10,10,100,4)

options(freegroup_symbols=NULL)  #  NULL is interpreted as letters a-z
rfree(10,4)            #  back to normal
```

---

reduce *Reduction of a word to reduced form*

Description

Given a word, remove redundant zero-power terms, and consolidate adjacent like terms into a single power

Usage

```
reduce(a)
is_reduced(a)
remove_zero_powers(a)
consolidate(a)
is_proper(a)
```

Arguments

a	An object of class free

Details

A word is *reduced* if no symbol appears next to its own inverse and no symbol has zero power. The essence of the package is to reduce a word into a reduced form. Thus $a^2b^{-1}ba$ will transformed into a^3.

In the package, reduction happens automatically at creation, in function free().

Apart from is_proper(), the functions all take a free object, but the meat of the function operates on a single two-row matrix.

Reduction is carried out by repeatedly consolidating adjacent terms of identical symbol (function consolidate()), and removing zero power terms (function remove_zero_power()) until the word is in reduced form (function is_reduced()).

Function is_proper() checks to see whether a matrix is suitably formed for passing to reduce().

A free object is *cyclically reduced* iff every cyclic permutation of the word is reduced. A reduced word is cyclically reduced iff the first letter is not the inverse of the last one. A reduced word is cyclically reduced if the first and last symbol differ (irrespective of power) or, if identical, have powers of opposite sign. For example, abac and abca are cyclically reduced but abca^{-1} is not. Function is.cyclically.reduced() tests for this, documented at cycred.Rd.

Author(s)

Robin K. S. Hankin

See Also

cycred

Examples

```
## create a matrix:
(M <- rbind(c(1,2,3,3,2,3,2,1),c(1,2,3,-3,5,0,7,0)))

## call the print method (note non-reduced form):
as.character_free(M)

## show the effect of reduce():
as.character_free(reduce(M))

## free() calls reduce() automatically:
free(M)
```

rfree *Random free objects*

Description

Creates a vector of random free objects. Intended as a quick "get you going" example of free group objects

Usage

```
rfree(n = 7, size = 4, number = size, powers = seq(from = -size, to = size))
rfreee(n = 30, size = 8, number = size, powers = seq(from = -size, to = size))
rfreeee(n = 40, size = 25, number = size, powers = seq(from = -size, to = size))
```

Arguments

n	Length of random vector to generate
size	Maximum length of each element
number	How many distinct letters to sample from
powers	Powers in resulting polynomial. An integer n is interpreted (via sample()) as seq_len(n)

Details

The auxiliary arguments specify the general complexity of the returned object with small meaning simpler.

Functions rfreee() and rfreeee() give, by default, successively more complicated expressions.

Author(s)

Robin K. S. Hankin

See Also

size

Examples

```
rfree()

abelianize(rfree())

rfree(10,2)
rfree(10,30,26)

rfree(powers=5)
rfree(powers=5:6)
```

```
rfree(20,2)^alpha(26)
```

`shift` *Permute elements of a vector in a cycle*

Description

Given a vector, permute the elements with a cyclic permutation

Usage

```
shift(x,i=1)
```

Arguments

`x`	Vector
`i`	Integer, number of places to permute. Negative values mean to count from the end

Details

This function is that of the **magic** package, where it is motivated and discussed.

Value

Returns a vector

Author(s)

Robin K. S. Hankin

Examples

```
shift(1:9)
shift(1:9,-1)

shift(1:9,2)
```

`size` *Bignesses of a free object*

Description

Various metrics to describe how "big" a free object is

Usage

```
size(a)
total(a)
number(a)
bigness(a)
```

Arguments

`a`	Vector of free group objects

Details

- The *size* of an object is the number of pure powers in it (this is the number of columns of the matrix representation of the word)
- The *total* of an object is the sum of the absolute values of its powers
- The *number* of an object is the number of distinct symbols in it

Thus `size(a^2ba)=3`, `total(a^2ba)=4`, and `number(a^2ba)=2`.

Function `bigness()` is a convenience wrapper that returns all three bigness measures.

Value

These functions return an integer vector.

Note

I would like to thank Murray Jorgensen for his insightful comments which inspired this functionality.

Author(s)

Robin K. S. Hankin

See Also

`abs`

Examples

```
(a <- rfree(20,6,4))
size(a)
total(a)
number(a)

a <- rfree(20,6,4)
b <- rfree(20,6,4)

## Following should all be TRUE
size(a+b)   <= size(a)  + size(b)
total(a+b)  <= total(a) + total(b)
number(a+b) <= number(a)+ number(b)

bigness(rfree(10,3,3))
bigness(allconj(rfree(1,6,1)))
```

---

subs *Substitute and invert symbols*

Description

Substitute and invert specific symbols in a free object

Usage

```
subsu(X, from, to)
subs(X, ...)
flip(X, turn)
```

Arguments

X	Object of class free
from, to, turn	Objects coerced to class free specifying symbols to alter. These arguments are coerced to symbols using getlet(as.free())
...	Named arguments for substitution

Details

Function subsu(X,from,to) takes object X and transforms every symbol present in from into the symbol specified in to.

Function flip(X,turn) takes object X and replaces every symbol present in turn with its inverse.

Function discard(), documented at keep.Rd, effectively substitutes a symbol with the identity element (thereby discarding it).

Experimental function subs() is modelled on similar functionality in the **freealg** package and makes idiom such as subs(X,a='z') work as expected (viz, taking each instance of symbol a and replacing it with x).

Note

Functions subs() and subsu() substitute for particular symbols, not free group elements. In particular, be careful with uppercase (inverse) symbols; because the power is discarded, substituting with x is the same as substituting for X. This behaviour might change in the future.

Author(s)

Robin K. S. Hankin

See Also

abs,discard

Examples

```
subsu(abc(1:10),abc(5),'z')
flip(abc(1:10),abc(5))


o <- rfree(30,5,10)

# Following tests should all be TRUE:
size(flip(o,'a'))   == size(o)
number(flip(o,'a')) == number(o)
total(flip(o,'a'))  == total(o)

size(subsu(o,'a','b'))   <= size(o)
number(subsu(o,'a','b')) <= number(o)
total(subsu(o,'a','b'))  <= total(o)


frog <- rfree()
subs(frog,a='x')
```

---

sum *Repeated summation by concatenation*

Description

Concatenates its arguments to give a single free object

Usage

```
## S3 method for class 'free'
sum(..., na.rm = FALSE)
```

Arguments

...	Objects of class free, to be summed
na.rm	Boolean, indicating whether to ignore NA entries (currently ignored)

Details

Concatenates its arguments and gives a single element of the free group. It works nicely with rev(), see the examples.

Note

The package uses additive notation, but it is easy to forget this and wonder why idiom like prod(rfree()) does not work as desired. Of course, the package using additive notation means that one probably wants sum(rfree()).

Author(s)

Robin K. S. Hankin

Examples

```
(x <- rfree(10,3))
sum(x)
abelianize(sum(x))

(y <- rfree(10,6))

sum(x,y)
sum(x,y) == sum(sum(x),sum(y))
x+y  # not the same!

sum(x,-x)
sum(x,rev(-x))

z <- alpha(26)
stopifnot(sum(x^z) == sum(x)^z)
```

---

tietze *Tietze form for free group objects*

Description

Translate an object of class free to and from Tietze form

Usage

```
## S3 method for class 'free'
tietze(x)
## S3 method for class 'matrix'
tietze(x)
vec_to_matrix(x)
```

Arguments

x	Object to be converted

Details

The Tietze form for a word is a list of integers corresponding to the symbols of the word; typically $a = 1, b = 2, c = 3, d = 4$, etc. Negative integers represent the inverses of the symbols. Thus `c^4.d^-2.a.c` becomes `3 3 3 3 -4 -4 1 3`.

Function `vec_to_matrix()` is a low-level helper function that returns a two-row integer matrix. If given `0` or `NULL`, it returns a two-row, zero-column matrix.

Author(s)

Robin K. S. Hankin

Examples

```
(x <- rfree(10,3))
tietze(x)

vec_to_matrix(c(1,3,-1,-1,-1,2))

as.free(list(c(1,1,8),c(2,-4,-4)))

all(as.free(tietze(abc(1:30))) == abc(1:30))
```

Index