Package 'freegroup'

September 25, 2018

Type Package

Title The Free Group

Version 1.1-0

Date 2018-09-14

Author Robin K. S. Hankin

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

Depends magrittr,methods,magic (>= 1.5-9), plyr

Suggests knitr, rmarkdown

VignetteBuilder knitr

Description Provides functionality for manipulating elements of the free group (juxtaposition is represented by a plus) including inversion, multiplication by a scalar, group-theoretic power operation, and Tietze forms. The package is fully vectorized.

License GPL-2

URL https://github.com/RobinHankin/freegroup.git

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

NeedsCompilation no

Repository CRAN

Date/Publication 2018-09-25 20:50:08 UTC

R topics documented:

`freegroup-package` *The Free Group*

Description

Provides functionality for manipulating elements of the free group (juxtaposition is represented by a plus) including inversion, multiplication by a scalar, group-theoretic power operation, and Tietze forms. The package is fully vectorized.

Details

The DESCRIPTION file:

Package:	freegroup
Type:	Package
Title:	The Free Group
Version:	1.1-0
Date:	2018-09-14
Author:	Robin K. S. Hankin
Maintainer:	Robin K. S. Hankin <hankin.robin@gmail.com>
Depends:	magrittr,methods,magic (>= 1.5-9), plyr
Suggests:	knitr, rmarkdown
VignetteBuilder:	knitr
Description:	Provides functionality for manipulating elements of the free group (juxtaposition is represented by a plus)
License:	GPL-2
URL:	https://github.com/RobinHankin/freegroup.git
BugReports:	https://github.com/RobinHankin/freegroup/issues

Index of help topics:

`Extract.free`	`Extract or replace parts of a free group object`
`Ops.free`	`Arithmetic Ops methods for the free group`
`abc`	`Create an alphabetical free group element`
`abelianize`	`Abelianization of free group elements`
`abs.free`	`Absolute value of a 'free' object`
`alpha`	`Single-symbol words`
`backwards`	`Write free objects backwards`
`c`	`Concatenation of free objects`
`char_to_free`	`Convert character vectors to free objects`
`cumsum`	`Cumulative sum`
`cycred`	`Cyclic reductions of a word`
`free`	`Objects of class 'free'`
`freegroup-package`	`The Free Group`
`getlet`	`Get letters of a freegroup object`
`identity`	`The identity element`
`keep`	`Keep or drop symbols`
`print.free`	`Print free objects`
`reduce`	`Reduction of a word to reduced form`
`rfree`	`Random free objects`
`size`	`Bignesses of a free object`
`subs`	`Substitute and invert symbols`
`sum`	`Repeated summation by concatenation`
`tietze`	`Tietze form for free group objects`

Author(s)

Robin K. S. Hankin

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

Examples

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

a+x

a^x

sum(a)

abelianize(a)

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

---

abc *Create an alphabetical free group element*

Description

Create a simple free group element

Usage

```
abc(n)
```

Arguments

`n`	An integer specifying the length of the word; if a vector, return the appropriate free vector

Author(s)

Robin K. S. Hankin

Examples

```
abc(8)

abc(1:26)   # compare alpha(1:26)

abc(-3:3)   # negative numbers give expected result

abc(26) ^ alpha(1:9)
```

`abelianize` *Abelianization of free group elements*

Description

Return the result of modifying a free group element under the assumption of Abelianness

Usage

```
abelianize(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.

Author(s)

Robin K. S. Hankin

Examples

```
x <- rfree(10,20,20)
abelianize(x)

p <- free(rbind(rep(1:5,4),rep(1:4,5)))
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` *Single-symbol words*

Description

Produces a vector of single-symbol words

Usage

```
alpha(v)
```

Arguments

`v`	Vector of integers

Author(s)

Robin K. S. Hankin

Examples

```
alpha(1)  # just the letter 'a'


alpha(1:26)  # the whole alphabet; compare abc(1:26)

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)
sum(abc(-5:5))


## bear in mind that the symbols used are purely for the print method:
jj <- LETTERS[1:10]
options(symbols = apply(expand.grid(jj,jj),1,paste,collapse=""))
alpha(c(66,67,68,69))   # sensible output
options(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

Function `backwards()` is distinct from `rev()`, see examples.

Author(s)

Robin K. S. Hankin

Examples

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

x <- rfree(10,5)
all(abelianize(x) == abelianize(backwards(x)))
```

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 <- rfree(10,3)
y <- rfree(10,3)
c(x,y)


## 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 robust; for example, passing anything other than lower-case letters a-z will give possibly undesirable behaviour.

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 the symbols' names, for example `options(symbols=sample(letters))`, 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("aaabacdcd")

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

as.free(letters)*7

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

```
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)
is.cyclically_reduced2(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 `is.cyclically_reduced2()` gives identical output; it uses slicker but marginally slower R idiom.

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

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

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))))]))
```

`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(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} would be

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

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 objet to `free`. Function `as.free()` is a bit more user-friendly and tries a bit harder to do the Right Thing.

Author(s)

Robin K. S. Hankin

See Also

char_to_free

Examples

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

x <- rfree(10,4)
x
x+x
x-x
x * (0:3)


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

---

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(30,4,11)

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 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(10,5,8)

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



x[1:4] %<>% keep(alpha(3))  # keep only abc in first 4 elements of x
```

`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 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.

Multiplication between a free object `a` and an integer `n` is defined as juxtaposing `n` copies of `a` and reducing. Zero and negative values of `n` work as expected.

Note

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

Author(s)

Robin K. S. Hankin

Examples

```
x <- rfree(10,2)
y <- rfree(10,2)
z <- rfree(10,9)    # more complicated than x or y




x+y
x-y

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

x+as.free(0) == x     # always true
as.free(0)+x == x     # always true
x+(y+z)  == (x+y)+z   # always true
x*5 == x+x+x+x+x      # always true

x + alpha(26)

x^alpha(26)

x*12
x*(0:9)
```

`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 codeTRUE 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(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("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(symbols=state.name)
rfree(10,4)
```

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

options(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`.

Whether the identity should be regarded as cyclically reduced is problematic. On the one hand the identity cannot be shortened by a combination of cyclic permutation followed by reduction; but on the other, I cannot exhibit a symbol at the start of the identity which can be reduced by juxtaposition with a symbol at the end (because there are no symbols). Currently it returns `NA` but I am open to suggestions.

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

Usage

```
rfree(n, size, 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 to sample from

Details

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

Author(s)

Robin K. S. Hankin

See Also

`size`

Examples

```

x <- rfree(10,2)
y <- rfree(10,30,26)

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

---

size *Bignesses of a free object*

Description

Various metrics to say 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

```
subs(a, from, to)
flip(a, turn)
```

Arguments

`a`	Object of class `free`
`from,to,turn`	Objects coerced to class `free` specifying symbols to alter

Details

Function `subs(a,from,to)` takes object `a` and transforms every symbol present in `from` into the symbol specified in `to`.

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

Author(s)

Robin K. S. Hankin

See Also

`abs`

Examples

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

`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.

Author(s)

Robin K. S. Hankin

Examples

```
x <- rfree(10,3)
y <- rfree(10,6)
z <- alpha(26)


sum(x)
abelianize(sum(x))


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

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


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 $a = 1, b = 2$, etc. Negative integers represent the inverses of the symbols.

Function `vec_to_free()` 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

```
tietze(rfree(10,3))

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