Package ‘mathml’

January 17, 2023

Type Package

Title Translate R Expressions to 'MathML' and 'LaTeX'/'MathJax'

Version 0.5

Date 2023-01-16

Maintainer Matthias Gondan <Matthias.Gondan-Rochon@uibk.ac.at>

Description Translate R expressions to 'MathML' or 'MathJax' so that they can be rendered in 'rmarkdown' documents and shiny apps.

License FreeBSD

Depends rolog (>= 0.9.10)

Encoding UTF-8

Suggests rmarkdown, knitr, testthat

VignetteBuilder knitr, rmarkdown

RoxygenNote 7.2.3

Config/testthat/edition 3

NeedsCompilation no

Author Matthias Gondan [aut, cre, cph] (Universität Innsbruck),
Irene Alfarone [ctb] (Universität Innsbruck)

Repository CRAN

Date/Publication 2023-01-17 17:30:08 UTC

R topics documented:

add *add*

Description

This is a function that allows the user to highlight the mistakes, in particular an extra element in a list

Usage

```
add(expr)
```

Arguments

`expr`	expression

Value

expr , e.g., highlights a + b from a + b

add_left *add_left*

Description

This is a function that allows the user to highlight the mistakes, in particular the redundancies in the left-hand side of the expression.

Usage

```
add_left(expr)
```

Arguments

expr	expression

Value

expr e.g., highlights a + from a + b

add_right *add_right*

Description

This is a function that allows the user to highlight the mistakes, in particular the redundancies in the right-hand side of the expression.

Usage

```
add_right(expr)
```

Arguments

expr	expression

Value

expr , e.g., highlights + b from a + b

cal *Calligraphic font*

Description

Calligraphic font

Usage

```
cal(x)
```

Arguments

`x`	an R symbol. This function is used to render the content in calligraphic font in MathJax. In MathML, script font is used.

Value

The function cal is a wrapper for the identity function.

See Also

`identity()`

Examples

```
mathjax(quote(K %in% cal(K)))
```

canonical *Canonicalize an R call: Reorder the function arguments*

Description

Canonicalize an R call: Reorder the function arguments

Usage

```
canonical(term = quote(`%in%`(table = Table, x = X)), drop = TRUE)
```

Arguments

`term`	an R call.
`drop`	whether to drop the argument names or not

Value

The R function, with arguments rearranged

Examples

```
canonical(term=quote(`%in%`(table=Table, x=X)))
```

denote *denote This is a function that allows the user to insert abbreviations in the formula, explain them and make the needed computations*

Description

denote This is a function that allows the user to insert abbreviations in the formula, explain them and make the needed computations

Usage

```
denote(abbr, expr, info)
```

Arguments

abbr	Abbreviation used in the text to refer to the calculation, for example 's_p' for the pooled variance.
expr	Expression: calculations to be made in order to obtain the value to which the abbreviation refers to.
info	Information: Explanation of the formula used to provide the value of the abbreviation. e.g. 'the pooled variance'

Value

expr e.g., x denotes a^2 + b

dfrac *Division displayed as large fraction*

Description

Division displayed as large fraction

Usage

```
dfrac(e1, e2)
```

Arguments

`e1`	numerator
`e2`	denominator

Value

e1 / e2

See Also

`frac()`, `over()`

`dot` *Multiplication*

Description

Multiplication

Usage

```
dot(e1, e2)

nodot(e1, e2)

times(e1, e2)
```

Arguments

`e1`	numerator
`e2`	denominator

Value

e1 * e2

`fname` *Return function body*

Description

Return function body

Usage

```
fname(fname, body)
```

Arguments

`fname`	not clear
`body`	not clear

Value

body

`frac` *Division displayed as fraction*

Description

Division displayed as fraction

Usage

```
frac(e1, e2)
```

Arguments

`e1`	numerator
`e2`	denominator

Value

e1 / e2

hook *Hook for custom symbols*

Description

Hook for custom symbols

Usage

```
hook(term, display, quote = TRUE, as.rolog = TRUE)
```

Arguments

`term`	an R call or symbol/number. This is the expression to replace.
`display`	an R call or symbol/number. This is shown instead of *term*.
`quote`	(default is TRUE) indicates that *term* and *display* should be quoted.
`as.rolog`	(default is TRUE) indicates that simplified quasi-quotoation is to be used.

Value

TRUE on success

Examples

```
hook(t0, subscript(t, 0))
mathml(quote(t0))

hook(term=quote(t0), display=quote(subscript(t, 0)), quote=FALSE)
mathml(quote(t0))
```

instead *instead*

Description

This is a function that allows the user to highlight the mistakes, in particular adds a curly bracket under the wrong term and it provides the correct solutions.

Usage

```
instead(inst, of)
```

Arguments

`inst`	instead
`of`	of

Value

inst , e.g. a + c instead of a + b

`mathjax` *Mathjax output*

Description

Mathjax output

Usage

```
mathjax(
  term = quote((a + b)^2L == a^2L + 2L * a * b + b^2L),
  flags = NULL,
  env = globalenv()
)
```

Arguments

`term`	an R call or symbol/number. This function translates *term* into a LaTeX/MathJax string.
`flags`	(default NULL) list of flags that control the translation
`env`	(default globalenv()) The R environment in which r_eval is being executed (see vignette for details, "Ringing back to R").

Details

In some functions, the Prolog code may ring back R, for example, to find the names of function arguments. For example (see vignette), when rendering the call `integrate(g, lower=0L, upper=Inf)` as Int_0^Inf g(x) dx, Prolog needs to know that the function g is a function of x. The Prolog rule then searches for the formalArgs of g in the environment *env*.

Value

A string with the MathJax representation or *term*.

See Also

`mathml()`

Examples

```
mathjax(term=quote((a + b)^2L == a^2L + 2L*a*b + b^2L))
```

mathml *MathML output*

Description

MathML output

Usage

```
mathml(
  term = quote((a + b)^2L == a^2L + 2L * a * b + b^2L),
  flags = NULL,
  env = globalenv()
)
```

Arguments

term	an R call or symbol/number. This function translates *term* into a MathML string.
flags	(default NULL) list of flags that control the translation
env	(default globalenv()) The R environment in which r_eval is being executed.

Details

In some functions, the Prolog code may ring back R, for example, to find the names of function arguments. For example (see vignette), when rendering the call `integrate(g, lower=0L, upper=Inf)` as Int_0^Inf g(x) dx, Prolog needs to know that the function g is a function of x. The Prolog rule then searches for the formalArgs of g in the environment *env*.

Value

A string with the MathML representation or *term*.

See Also

`mathjax()`

Examples

```
mathml(term=quote((a + b)^2L == a^2L + 2L*a*b + b^2L))
```

`mathml_preproc` *Map R operators to their respective Prolog counterparts*

Description

Map R operators to their respective Prolog counterparts

Usage

```
mathml_preproc(query = quote(5%%2))
```

Arguments

`query`	an R call or symbol/number. This function translates components of *query* into their respective counterparts from Prolog

Value

The translated query

See Also

`mathjax()`, `mathml()`

Examples

```
mathml_preproc(quote(5 %% 2))
```

`name` *Add a name attribute to an element (most often, an R function)*

Description

Add a name attribute to an element (most often, an R function)

Usage

```
name(x, name)
```

Arguments

`x`	an R object, e.g., an R function
`name`	the name of the object/function

Value

The object with the name attribute

Examples

```
f <- function(x) {sin(x)}
mathjax(call("integrate", name(f, "sin"), 0L, 2L*pi))
```

---

omit *omit*

Description

This is a function that allows the user to highlight the mistakes, in particular the omission of an element from a list.

Usage

```
omit(expr)
```

Arguments

`expr`	expression

Value

NULL e.g., remove a + b from a + b

---

omit_left *omit_left This is a function that allows the user to highlight the mistakes, in particular the omissions in the left-hand side of the expression*

Description

omit_left This is a function that allows the user to highlight the mistakes, in particular the omissions in the left-hand side of the expression

Usage

```
omit_left(expr)
```

Arguments

`expr`	The expression, e.g. a + b

Value

substitute(expr)[[3]], e.g., b from a + b

`omit_right` *omit_right This is a function that allows the user to highlight the mistakes, in particular the omissions in the right-hand side of the expression*

Description

omit_right This is a function that allows the user to highlight the mistakes, in particular the omissions in the right-hand side of the expression

Usage

```
omit_right(expr)
```

Arguments

`expr`	expression

Value

substitute(expr)[[2]], e.g., a from a + b

`over` *Division displayed as fraction*

Description

Division displayed as fraction

Usage

```
over(e1, e2)
```

Arguments

`e1`	numerator
`e2`	denominator

Value

e1 / e2

subscript *Subscript. On the R side, this function is a wrapper of identity, but allows for decorations.*

Description

Subscript. On the R side, this function is a wrapper of identity, but allows for decorations.

Usage

```
subscript(fun = quote(x), sub = quote(i))
```

Arguments

`fun`	an R call or symbol, e.g. sum(x). This is the return value of the function.
`sub`	an R symbol or call, e.g., i

Value

The function over is a wrapper for the identity function, returning *fun*

See Also

`identity()`

Examples

```
mathjax(quote(subscript(sub=i, fun=x)))
```

subsupscript *Subsupscript. This is a wrapper for the identity function, but decorates the result with a sub- and a superscript.*

Description

Subsupscript. This is a wrapper for the identity function, but decorates the result with a sub- and a superscript.

Usage

```
subsupscript(fun = quote(sum(x[i])), sub = quote((i = 1)), sup = quote(N))
```

Arguments

`fun` an R call or symbol, e.g. `sum(x[i])`. This is the return value.

`sub` an R symbol, e.g., `i=1`

`sup` an R symbol, e.g., `N`

Value

The function over is a wrapper for the identity function, returning *fun*

See Also

`identity()`

Examples

```
N <- 10
i <- 1:N
x <- rnorm(N)
mathjax(call("subsupscript", fun=sum(x[i]), sub=quote(`=`(i, 1L)), sup=quote(N)))
```

`superscript` *Superscript. This is a wrapper for the identity function, but decorates the result with a superscript.*

Description

Superscript. This is a wrapper for the identity function, but decorates the result with a superscript.

Usage

```
superscript(fun = quote(A), sup = "*")
```

Arguments

`fun` an R call or symbol, e.g. x. This is the return value of the function.

`sup` an R symbol, e.g., "*"

Value

The function over is a wrapper for the identity function, returning *fun*

See Also

`identity()`

Examples

```
mathjax(quote(superscript(fun=A, sup="*")))
```

`%.%` *Product x * y, shown as x dot y*

Description

Product x * y, shown as x dot y

Usage

```
x %.% y
```

Arguments

x	first factor
y	second factor

Value

x * y

`%dbldown%` *Down double arrow, displayed as x dArr y*

Description

Down double arrow, displayed as x dArr y

Usage

```
x %dbldown% y
```

Arguments

x	first element
y	second element

Value

x=y ,it produces a downward double arrow

`%dblup%` *Up double arrow, displayed as x uArr y*

Description

Up double arrow, displayed as x uArr y

Usage

```
x %dblup% y
```

Arguments

`x`	first element
`y`	second element

Value

x=y ,it produces a upward double arrow

`%down%` *Down arrow, presented as x downarrow y*

Description

Down arrow, presented as x downarrow y

Usage

```
x %down% y
```

Arguments

`x`	first element
`y`	second element

Value

x=y , it produces a downward arrow

`%==%` *Equivalence, shown as x == y*

Description

Equivalence, shown as x == y

Usage

```
x %==% y
```

Arguments

x	first argument
y	second argument

Value

x=y , e.g., a = b

`%=>%` *Left double arrow, displayed as x <= y*

Description

Left double arrow, displayed as x <= y

Usage

```
x %=>% y
```

Arguments

x	first element
y	second element

Value

x=y , it produces a left doublearrow

`%=~%` *Congruence, shown as x =~ y*

Description

Congruence, shown as x =~ y

Usage

```
x %=~% y
```

Arguments

x	first argument
y	second argument

Value

x=y , e.g., a cong b

`%->%` *Right arrow, presented as x -> y*

Description

Right arrow, presented as x -> y

Usage

```
x %->% y
```

Arguments

x	first element
y	second element

Value

x=y , it produces a right arrow

`%<=%` *Right double arrow, displayed as x => y*

Description

Right double arrow, displayed as x => y

Usage

```
x %<=% y
```

Arguments

`x`	first element
`y`	second element

Value

x=y , it produces a right double arrow

`%<=>%` *If and only if condition, displayed as x <=> y*

Description

If and only if condition, displayed as x <=> y

Usage

```
x %<=>% y
```

Arguments

`x`	first element
`y`	second element

Value

x=y , it produces a double arrow double-sided

`%+-%` *Plus Minus, it shows x and calculates x +- y*

Description

Plus Minus, it shows x and calculates x +- y

Usage

```
x %+-% y
```

Arguments

`x`	first term
`y`	second term

Value

c(x - y, x + y) x plus min y

`%prop%` *Proportional, shown as x prop y*

Description

Proportional, shown as x prop y

Usage

```
x %prop% y
```

Arguments

`x`	first argument
`y`	second argument

Value

x=y e.g, x prop y

`%<-%` *Left arrow, presented as x <- y*

Description

Left arrow, presented as x <- y

Usage

```
x %<-% y
```

Arguments

x	first element
y	second element

Value

x=y , it produces a left arrow

`%<->%` *Double sided arrow, presented as x <-> y*

Description

Double sided arrow, presented as x <-> y

Usage

```
x %<->% y
```

Arguments

x	first element
y	second element

Value

x=y ,it produces a double sided arrow

`%~~%` *Approximate equality, shown as x ~~ y*

Description

Approximate equality, shown as x ~~ y

Usage

```
x %~~% y
```

Arguments

x	first argument
y	second argument

Value

The result of isTRUE(all.equal(x, y))

`%up%` *Up arrow, presented as x up y*

Description

Up arrow, presented as x up y

Usage

```
x %up% y
```

Arguments

x	first element
y	second element

Value

x=y , it produces an upward arrow

Index