Package 'formula.tools'

September 21, 2017

Type Package

Title Programmatic Utilities for Manipulating Formulas, Expressions, Calls, Assignments and Other R Objects

Version 1.6.8

Date 2017-09-21

Description These utilities facilitate the programmatic manipulations of formulas, expressions, calls, assignments and other R language objects. These objects all share the same structure: a left-hand side, operator and right-hand side. This packages provides methods for accessing and modifying this structures as well as extracting and replacing names and symbols from these objects.

Depends R (>= 3.0.0)

Imports operator.tools(>= 1.4.0), utils, methods

Suggests magrittr, testthat

License GPL-2 | file LICENSE

URL https://github.com/decisionpatterns/formula.tools

BugReports https://github.com/decisionpatterns/formula.tools/issues

LazyLoad yes

LazyData yes

Collate 'as.character.formula.R' 'catcont.R' 'env.R' 'get.vars.R' 'invert.R' 'is.one.sided.R' 'is.two.sided.R' 'lhs.vars.R' 'op.type.R' 'parts.R' 'parts.lhs.get.R' 'parts.lhs.set.R' 'parts.op.get.R' 'parts.op.set.R' 'parts.rhs.get.R' 'parts.rhs.set.R' 'rhs.vars.R' 'terms.R' 'zzz.R'

RoxygenNote 6.0.1.9000

Repository CRAN

NeedsCompilation no

Author Christopher Brown [aut, cre], Decision Patterns [cph]

Maintainer Christopher Brown <chris.brown@decisionpatterns.com>

Date/Publication 2017-09-21 19:29:49 UTC

R topics documented:

`.invert.single` *Invert multiple elements of a multiple element object*

Description

Invert multiple elements of a multiple element object

Usage

```
.invert.single(x)

.invert.plural(x)
```

Arguments

x	object to invert from

See Also

.invert.single

`as.character.formula` *Converts a formula to character*

Description

Convers a formula to character representaion

Usage

```
## S3 method for class 'formula'
as.character(x, ...)
```

Arguments

x	formula object
...	further arguments passed to or from other methods.

Details

Coerces formula to a character by deparsing.

Value

A character vector

Author(s)

Christopher Brown

See Also

deparse

Examples

```
  as.character( y ~ mx +  b )

## The function is currently defined as
function(x)
  Reduce( paste, deparse(x) )
```

---

env *Get the environment*

Description

Get the environment

Usage

```
env(x)

## S3 method for class 'formula'
env(x)
```

Arguments

x	object to get environoment from

Details

S3 returns the environment associated with an object.

For a formula object, ‘env‘ returns the environment in the ‘.Environment‘ attribute.

Value

Environment

Examples

```
env( lhs ~ rhs )
```

`formula.parts` *Manipulate the component parts of formulas, expressions, calls, name/symbols and list and vectors of such objects.*

Description

`lhs`, `rhs`, `op`, and `op.type` retrieve the various parts of R formulas, calls, expressions, names/symbols. These functions were designed to greatly facilitate symbolic manupulation using native R objects. Also provided are methods to handle list of these objects.

Usage

```
lhs(x, ...)

## S4 method for signature 'call'
lhs(x)

## S4 method for signature 'formula'
lhs(x)



## S4 method for signature 'expression'
lhs(x, ...)

## S4 method for signature 'list'
lhs(x, ...)

lhs(x) <- value

## S4 replacement method for signature 'call'
lhs(x) <- value
```

```
## S4 replacement method for signature 'formula'
lhs(x) <- value



.replace.lhs.plural(x, value)

## S4 replacement method for signature 'expression'
lhs(x) <- value

## S4 replacement method for signature 'list'
lhs(x) <- value

op(x)

## S4 method for signature 'formula'
op(x)

## S4 method for signature 'call'
op(x)

## S4 method for signature 'name'
op(x)

## S4 method for signature 'expression'
op(x)

## S4 method for signature 'list'
op(x)



op(x) <- value

## S4 replacement method for signature 'call'
op(x) <- value

## S4 replacement method for signature 'formula'
op(x) <- value



.replace.op.plural(x, value)

## S4 replacement method for signature 'expression'
op(x) <- value

## S4 replacement method for signature 'list'
```

```
op(x) <- value

rhs(x, ...)

.rhs.singular(x)

## S4 method for signature 'call'
rhs(x)

## S4 method for signature 'formula'
rhs(x)



## S4 method for signature 'expression'
rhs(x, ...)

## S4 method for signature 'list'
rhs(x, ...)

rhs(x) <- value

.replace.rhs.singular(x, value)

## S4 replacement method for signature 'call'
rhs(x) <- value

## S4 replacement method for signature 'formula'
rhs(x) <- value



.replace.rhs.plural(x, value)

## S4 replacement method for signature 'expression'
rhs(x) <- value

## S4 replacement method for signature 'list'
rhs(x) <- value
```

Arguments

x	object from where to get/set the lhs/rhs
...	arguments passed to additional methods
value	the value to set for the lhs/rhs

Details

lhs retrieves the left-hand side rhs retrieves the right-hand side op retrieves the operation op.type

returns the type operator

There are also functions `lhs.vars` and `rhs.vars`. Like `all.vars` , these functions interpret the variables on the left-hand and right-hand sides respectively.

These are simple functions for extracting the left-hand side, right-hand side, operator and operator type from formulas, expressions, calls, names/symbols and list containing these objects. lhs, rhs are only defined for formulas and calls (and list and expressions) that are defined with either one of the relational or tilde ('~') operators. If the object does not contain one of these operators, it will fail with a warning.

The defined operator types are defined by the operator.tools package: See `operators` and `setOperator`

The `lhs.vars` and `rhs.vars` methods, return the variables used on the lhs and rhs, respectively. If special formula variables are used, such as '.', a data.frame or environment must also be provided such that the variable list may be properly infered.

Value

Value depends on the argument.

Note

Methods for the non-standard "<-" class exist and are not included in the usage documentation because CRAN does not support S4 documentation for this class.

Author(s)

Christopher Brown

See Also

terms, all.vars, all.names, `operators`

Examples

```
# FORMULA
f <- A + B ~ C + D
lhs(f)
lhs(f) <- quote( E / F )

rhs(f)
rhs(f) <- quote( G + H )
op(f)
op(rhs(f))
op( quote(A) )  # NULL:
op.type(f)

# ONE-SIDED FORMULA
f <- ~ A   #
lhs(f)     # NULL
rhs(f)     # A
```

```
# EXPRESSION
e <- expression( A + B == C + D )
lhs(e)
rhs(e)
op(e)
op.type(e)


# CALL
c <- quote( A + B > C + D )
lhs(c)
lhs(c) <- quote(E)
rhs(c)

op(c)
op.type(c)

# ASSIGNMENT
a  <- quote( A <- B )
lhs(a)
rhs(a)
op(a)
op.type(a)
```

`get.vars` *Get variable (names) from various R objects*

Description

`get.vars` extracts variable names from various R objects such as formulas, expressions, calls, symbols, etc. It is very similar to `all.vars` except that all symbols, etc. are interpolated to the names of variables.

Usage

```
get.vars(x, data = NULL, ...)

## S4 method for signature 'formula,ANY'
get.vars(x, data = NULL, ...)

## S4 method for signature 'call,ANY'
get.vars(x, data = NULL, ...)

## S4 method for signature 'expression,missing'
get.vars(x, data = NULL, ...)

## S4 method for signature 'name,ANY'
```

```
get.vars(x, data = NULL, ...)

## S4 method for signature 'ANY,ANY'
get.vars(x, data = NULL, ...)

## S4 method for signature '`NULL`,ANY'
get.vars(x, data = NULL, ...)

lhs.vars(x, ...)

.lhs.vars(x, ..., data = NULL)

## S4 method for signature 'formula'
lhs.vars(x, ..., data = NULL)

## S4 method for signature 'call'
lhs.vars(x, ..., data = NULL)

## S4 method for signature 'expression'
lhs.vars(x, ...)

rhs.vars(x, ...)

.rhs.vars(x, ..., data = NULL)

## S4 method for signature 'formula'
rhs.vars(x, ..., data = NULL)

## S4 method for signature 'call'
rhs.vars(x, ..., data = NULL)

## S4 method for signature 'expression'
rhs.vars(x, ...)
```

Arguments

x	object to extract vars from.
data	data set/list or environment on which the names are defined
...	arguments passed to subsequent functions
	get.vars and variant get the variables from objects optionally interpreting on . on the data. This is useful, for example, when you wish to know what data is used based on a given formula.
	Methods/functions beginning with . are not exported

Value

character vector of variables names in order that they appear in x.

See Also

all.vars

Examples

```
get.vars( Species ~ ., iris )
get.vars( quote( Sepal.Length * Sepal.Width ), iris )
```

---

invert *invert*

Description

Invert the operators in an object, usually a formula or expression

Usage

```
invert(x, ...)

## S4 method for signature 'call'
invert(x)

## S4 method for signature 'expression'
invert(x)
```

Arguments

x	function for invert
...	additional arguments passed other functions
	invert is a S4 generic method for inverting relational operators, i.e.
	functions prefixed with a . are not exported and should probably not be called directly

Value

The operand is returned with the relational operators inverted.

Author(s)

Christopher Brown

See Also

op, op.type

Examples

```
invert( quote( A >  5 ) )
invert( quote( A >= 5 ) )
invert( quote( A <  5 ) )
invert( quote( A <= 5 ) )
invert( quote( A == 5 ) )
invert( quote( A != 5 ) )
invert( quote( A %in% lettters[1:5] ) )
invert( quote( A %!in% letters[1:5] ) )
```

---

is.cat *Work with variables as categorical or continuous*

Description

These functions are used to identify which/if a variable or variables are categorical or continuos. `is.cat` and `is.cont` take single variable arguments. `which.cat` and `which.cont` take a list or data.frame or any structures whose elements have a class method.

Usage

```
is.cat(x, ...)

## S4 method for signature 'character'
is.cat(x)

## S4 method for signature 'factor'
is.cat(x)

## S4 method for signature 'logical'
is.cat(x)

## S4 method for signature 'ANY'
is.cat(x)

is.cont(x, ...)

## S4 method for signature 'numeric'
is.cont(x)

## S4 method for signature 'integer'
is.cont(x)

## S4 method for signature 'complex'
is.cont(x)
```

```
## S4 method for signature 'Date'
is.cont(x)

## S4 method for signature 'POSIXct'
is.cont(x)

## S4 method for signature 'factor'
is.cont(x)

## S4 method for signature 'ANY'
is.cont(x)

which.cat(x, ..., names = FALSE)

which.cont(x, ..., names = FALSE)
```

Arguments

x	object
...	arguments passed to other functions
names	logical; whether to return the names of the variables instead of their index

Details

These functions facilitate working with variables as categorical or continous rather than logical, integer, numeric, factor, character, ..

Value

`is.cat` returns `TRUE` if `x` is `character`, `factor` or `logical`. It is `FALSE` otherwise.

`is.cont` returns `TRUE` if `x` is `numeric`, `integer`, `complex`, `Date`, `POSIXct`

`which.cat` and `which.cont` report which variables in an object are categorical and which are continuous. By default, interger indices are return. If `names=TRUE`, the names of the variables are returned instead.

See Also

`which`, `is`

Examples

```
## Not run:
 is.cat(letters)          # TRUE
 is.cat(factor(letters))  # TRUE
 is.cat(TRUE)             # TRUE
 is.cat(FALSE)            # TRUE
 is.cat(1:10)             # FALSE
 is.cat(rnorm(10))        # FALSE
 is.cat( now() )          # FALSE
```

```
  is.cont(letters)         # FALSE
  is.cont(factor(letters)) # FALSE
  is.cont(TRUE)            # FALSE
  is.cont(FALSE)           # FALSE
  is.cont(1:10)            # TRUE
  is.cont(rnorm(10))      # TRUE

  which.cat(iris)
  which.cat( iris, names=TRUE )

  which.cont( iris )
  which.cont( iris, names=TRUE )

## End(Not run)
```

`is.one.sided` *Determine if an object is one- or two-sided. Test whether a object (typically formula, call or expression) is one- (e.g.* `~x`*) or two-sided (e.g.* `x~y`*).*

Description

Determine if an object is one- or two-sided.

Test whether a object (typically formula, call or expression) is one- (e.g. `~x`) or two-sided (e.g. `x~y`).

Usage

```
is.one.sided(x, ...)

## S4 method for signature 'formula'
is.one.sided(x, ...)

## S4 method for signature 'call'
is.one.sided(x, ...)



## S4 method for signature 'expression'
is.one.sided(x, ...)

## S4 method for signature 'list'
is.one.sided(x, ...)

## S4 method for signature 'ANY'
is.one.sided(x, ...)
```

```
is.two.sided(x, ...)

## S4 method for signature 'formula'
is.two.sided(x, ...)

## S4 method for signature 'call'
is.two.sided(x, ...)



## S4 method for signature 'expression'
is.two.sided(x, ...)

## S4 method for signature 'list'
is.two.sided(x, ...)

## S4 method for signature 'ANY'
is.two.sided(x, ...)
```

Arguments

x	object to test for one-sidedness.
...	arguments passed to called functions

Details

These functions detect whether the formula is single- (unary) or double- sided. They work on formulas, expression, calls, assignments, etc.

`is.single.sided` and `is.unary` are alias for `is.single.sided`. `is.double.sided` and `is.binary` are aliases for `is.two.sided`.

Value

logical; whether `x` is an object is one-sided or two-sided formula.

Note

Methods for the "<-" class exist and are not included in the usage documentation because CRAN does not support S4 documentation for this class.

Examples

```
form <- y ~ x

is.one.sided(form)
# is.single.sided(form)
# is.unary(form)

is.two.sided(form)
```

```
# is.double.sided(form)
# is.binary(form)
```

---

op.type *Get the operator type used in an call, formula, expression, etc.*

Description

Get the operator type used in an call, formula, expression, etc.

Usage

```
op.type(x)

## S4 method for signature 'call'
op.type(x)

## S4 method for signature 'formula'
op.type(x)



## S4 method for signature 'ANY'
op.type(x)

## S4 method for signature 'expression'
op.type(x)

## S4 method for signature 'list'
op.type(x)
```

Arguments

x	object from which to extract the operator type

Value

a character vector of the operator type(s)

See Also

op, operator.type

terms.call *terms*

Description

terms method for call and expression objects

Usage

```
## S3 method for class 'call'
terms(x, ...)

## S3 method for class 'expression'
terms(x, ...)
```

Arguments

x	A call object
...	Arguments passed to terms.formula This S3 method returns a terms object for a call methods using a dispatch to terms.formula . The terms are generated by making a rhs only call to terms.formula . data is only needed and must be explicitly specified, i.e. data = if there are special elements such as '.'. Otherwise the data argument is unused. Some edge cases may not be supported.

Value

A terms object. See terms.object for details.

Author(s)

Christopher Brown

See Also

terms.object and terms.formula

Examples

```
terms( quote( A + B ) )

data(iris)
x <- terms( quote( . - Species ) , data=iris )
```

Index