

# Package ‘formula.tools’

March 1, 2018

**Type** Package

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

**Version** 1.7.1

**Date** 2018-02-22

**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' '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' 'split.terms.R' 'terms.R' 'toggle.sign.R' 'zzz.R'

**RoxygenNote** 6.0.1.9000

**Repository** CRAN

**Encoding** UTF-8

**NeedsCompilation** no

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

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

**Date/Publication** 2018-03-01 12:56:01 UTC

## R topics documented:

<code>.invert.single</code> . . . . .	2
<code>as.character.formula</code> . . . . .	3
<code>env</code> . . . . .	4
<code>formula.parts</code> . . . . .	4
<code>get.vars</code> . . . . .	9
<code>invert</code> . . . . .	10
<code>is.one.sided</code> . . . . .	12
<code>op.type</code> . . . . .	13
<code>split_terms</code> . . . . .	14
<code>terms.call</code> . . . . .	16
<code>toggle.sign</code> . . . . .	17
<b>Index</b>	<b>18</b>

---

<code>.invert.single</code>	<i>Invert multiple elements of a multiple element object</i>
-----------------------------	--

---

### 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

Converts a formula to character representation

## 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	<i>Get the environment</i>
-----	----------------------------

---

**Description**

Get the environment

**Usage**

```
env(x)
```

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

**Arguments**

x                    object to get environment 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	<i>Manipulate the component parts of formulas, expressions, calls, name/symbols and list and vectors of such objects.</i>
---------------	---

---

**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 manipulation 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 inferred.

### 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

<code>x</code>	object to extract vars from.
<code>data</code>	data set/list or environment on which the names are defined
<code>...</code>	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.one.sided	<i>Determine if an object is one- or two-sided. Test whether a object (typically formula, call or expression) is one- (e.g. <math>\sim x</math>) or two-sided (e.g. <math>x \sim y</math>).</i>
--------------	---

---

### Description

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

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

### 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](#)

---

split\_terms

*Split object into terms*

---

**Description**

Split formulas, call and expressions into terms.

**Usage**

```
split_terms(x, recursive = FALSE)
```

**Arguments**

x                    object to split terms from  
recursive            logical; whether to split terms recursively in parenthetical enclosed terms (depth-first). (Default: FALSE)

**Details**

Unlike the [terms()] function, 'split\_terms' does not use [stats::terms()] and instead just splits 'x' into an **expression vector** of terms. (Terms are the mathematical notion of terms). The signs of the terms are preserved.

If 'recursive' is 'TRUE', splitting occurs recursively, i.e. parsing of the input descends into parenthetical expressions '(...)'.  
 See **examples**.

**Value**

expression vector of terms

**References**

\* [SO: How to split a formula](<https://stackoverflow.com/questions/39155701/how-to-split-a-formula-in-r>)

**See Also**

\* [terms()]

**Examples**

```
split_terms(1)           # 1
split_terms( quote(a) )  # a
split_terms( quote(-a) ) # -a
split_terms( quote(a+1) ) # a, 1
split_terms( quote(1+a) ) # 1, a
split_terms( quote(-1+a) ) # -1, a
split_terms( quote(-1-a) )

split_terms( quote(a+b+c) ) # a,b,c
split_terms( quote((a+b)+1) ) # (a+b),1
split_terms( quote((a+b)+1), recursive=TRUE ) # a,b,1
split_terms( quote((a-b)+1), recursive=TRUE ) # a,-b,1
split_terms( quote(-a) ) # -a

split_terms( quote(a-1) ) # a, -1
split_terms( quote(-a-1)) # -a, -1
split_terms( quote( -(a+1) ) ) # -(a+1)
split_terms( quote( -(a+1) ), recursive=TRUE ) # -a,-1

split_terms( quote( ---a ) )
split_terms( quote( -(a+(b-(c+d))) ), recursive=TRUE )
```

---

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 <a href="#">terms.formula</a>

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

- [stats::terms.object\(\)](#)
- [stats::terms.formula\(\)](#)

## Examples

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



---

toggle.sign	<i>toggle the sign of an expression</i>
-------------	---

---

**Description**

toggle the sign of an expression

**Usage**

```
toggle.sign(x)
```

**Arguments**

x                    expression

**Details**

'toggle.sign' changes the sign of an expression for '+' to '-' and visa-versa.

**Examples**

```
toggle.sign(1:3)
toggle.sign( quote(a) )
toggle.sign( quote(-a) )

exp <- expression( a, -b, -(a-b) )
toggle.sign(exp)
```

# Index

- \*Topic **manip**
  - as.character.formula, 3
  - terms.call, 16
- \*Topic **symbolmath**
  - terms.call, 16
- \*Topic **utilities**
  - as.character.formula, 3
  - terms.call, 16
- .invert.plural (.invert.single), 2
- .invert.single, 2
- .lhs.vars (get.vars), 9
- .replace.lhs.plural (formula.parts), 4
- .replace.op.plural (formula.parts), 4
- .replace.rhs.plural (formula.parts), 4
- .replace.rhs.singular (formula.parts), 4
- .replace.ths.plural (formula.parts), 4
- .rhs.singular (formula.parts), 4
- .rhs.vars (get.vars), 9
- 'lhs<-', <--method (formula.parts), 4
  
- all.vars, 7, 9, 10
- as.character (as.character.formula), 3
- as.character.formula, 3
  
- deparse, 3
  
- env, 4
  
- formula.parts, 4
  
- get.vars, 9
- get.vars, ANY, ANY-method (get.vars), 9
- get.vars, ANY, ANY-methods (get.vars), 9
- get.vars, call, ANY-method (get.vars), 9
- get.vars, expression, missing-method (get.vars), 9
- get.vars, formula, ANY-method (get.vars), 9
- get.vars, name, ANY-method (get.vars), 9
- get.vars, NULL, ANY-method (get.vars), 9
- get.vars, NULL, ANY-methods (get.vars), 9
  
- invert, 10
- invert, call-method (invert), 10
- invert, expression-method (invert), 10
- is.one.sided, 12
- is.one.sided, <--method (is.one.sided), 12
- is.one.sided, ANY-method (is.one.sided), 12
- is.one.sided, call-method (is.one.sided), 12
- is.one.sided, expression-method (is.one.sided), 12
- is.one.sided, formula-method (is.one.sided), 12
- is.one.sided, list-method (is.one.sided), 12
- is.two.sided (is.one.sided), 12
- is.two.sided, <--method (is.one.sided), 12
- is.two.sided, ANY-method (is.one.sided), 12
- is.two.sided, call-method (is.one.sided), 12
- is.two.sided, expression-method (is.one.sided), 12
- is.two.sided, formula-method (is.one.sided), 12
- is.two.sided, list-method (is.one.sided), 12
  
- lhs (formula.parts), 4
- lhs, <--method (formula.parts), 4
- lhs, call-method (formula.parts), 4
- lhs, expression-method (formula.parts), 4
- lhs, formula-method (formula.parts), 4
- lhs, list-method (formula.parts), 4
- lhs.vars (get.vars), 9
- lhs.vars, call-method (get.vars), 9
- lhs.vars, expression-method (get.vars), 9
- lhs.vars, formula-method (get.vars), 9

- lhs<- (formula.parts), 4
- lhs<- ,<--method (formula.parts), 4
- lhs<- ,call-method (formula.parts), 4
- lhs<- ,expression-method (formula.parts), 4
- lhs<- ,formula-method (formula.parts), 4
- lhs<- ,list-method (formula.parts), 4
  
- op, 11, 14
- op (formula.parts), 4
- op,<--method (formula.parts), 4
- op,call-method (formula.parts), 4
- op,expression-method (formula.parts), 4
- op,formula-method (formula.parts), 4
- op,list-method (formula.parts), 4
- op,name-method (formula.parts), 4
- op.type, 11, 13
- op.type,<--method (op.type), 13
- op.type,ANY-method (op.type), 13
- op.type,call-method (op.type), 13
- op.type,expression-method (op.type), 13
- op.type,formula-method (op.type), 13
- op.type,list-method (op.type), 13
- op.type-methods (op.type), 13
- op<- (formula.parts), 4
- op<- ,<--method (formula.parts), 4
- op<- ,call-method (formula.parts), 4
- op<- ,expression-method (formula.parts), 4
- op<- ,formula-method (formula.parts), 4
- op<- ,list-method (formula.parts), 4
- operator.type, 14
- operators, 7, 8
  
- rhs (formula.parts), 4
- rhs,<--method (formula.parts), 4
- rhs,call-method (formula.parts), 4
- rhs,expression-method (formula.parts), 4
- rhs,formula-method (formula.parts), 4
- rhs,list-method (formula.parts), 4
- rhs.vars (get.vars), 9
- rhs.vars,call-method (get.vars), 9
- rhs.vars,expression-method (get.vars), 9
- rhs.vars,formula-method (get.vars), 9
- rhs<- (formula.parts), 4
- rhs<- ,<--method (formula.parts), 4
- rhs<- ,call-method (formula.parts), 4
- rhs<- ,expression-method (formula.parts), 4
- rhs<- ,formula-method (formula.parts), 4
- rhs<- ,list-method (formula.parts), 4
- setOperator, 7
- split\_terms, 14
- stats::terms.formula(), 16
- stats::terms.object(), 16
  
- terms (terms.call), 16
- terms.call, 16
- terms.formula, 16
- terms.object, 16
- toggle.sign, 17