Package 'evalR'

May 8, 2026

Type Package

Title Evaluation of Unverified Code

Version 0.0.1

Description The purpose of this package is to generate trees and validate unverified code. Trees are made by parsing a statement into a verification tree data structure. This will make it easy to port the statement into another language. Safe statement evaluations are done by executing the verification trees.

Depends R (>= 3.5.0)

License MIT + file LICENSE

Imports Rcpp (>= 1.0.1)

LinkingTo Rcpp

Suggests testthat, rmarkdown, knitr, tufte, microbenchmark

RoxygenNote 7.1.2

VignetteBuilder knitr

NeedsCompilation yes

Author Numerious Inc. [cph, fnd],
Trevor Olsen [aut, cre]

Maintainer Trevor Olsen <trevor@numerious.com>

Repository CRAN

Date/Publication 2022-08-25 09:32:34 UTC

Contents

`create_tree` *Convert a statement into an evaluation tree*

Description

function will break `text` into a list of lists.

Usage

```
create_tree(
  text,
  singular_operators = NULL,
  binary_operators = NULL,
  valid_functions = NULL
)
```

Arguments

`text`	the string/code/statement you want to parse.
`singular_operators`	tokens of length 1 that operate on a right hand value. For example, the '-' token is an operator to negate a vector. `NULL` value will be replaced with `c("-", "!")`.
`binary_operators`	tokens of any length that operate on a left and right hand values. For example, the '+' token is an operator that adds a left vector to a right vector. `NULL` value will be replaced with `c(",", "|", "&", "<=", "<", ">=", ">", "==", "!=", "+", "-", "*", "%/%", "/", "%%", "%in%", ":", "^")`. The order determines the precedence of the operators.
`valid_functions`	tokens of any length that are prefixed on a parenthesis block and specify a function to run on the provided parameters within the block. For example, the 'log' token will evaluate the logarithm value of the first parameter. Note named parameters are not support. `NULL` value will be replaced with `c("log", "c", "any", "all", "abs", "ifelse")`.

Details

See `vignette("Overview", package = "evalR")`

Value

a list of lists. In other words, a tree data structure made from lists.

Examples

```
x <- create_tree("2 * (3 + 5)")
str(x)
```

`eval_text` *safely evaluate text*

Description

Safe alternative to using eval + parse

Usage

```
eval_text(
  text,
  singular_operators = NULL,
  binary_operators = NULL,
  valid_functions = NULL,
  map = NULL,
  mapping_names = NULL
)
```

Arguments

`text`	the string/code/statement you want to parse.
`singular_operators`	tokens of length 1 that operate on a right hand value. For example, the '-' token is an operator to negate a vector. `NULL` value will be replaced with `c("-", "!")`.
`binary_operators`	tokens of any length that operate on a left and right hand values. For example, the '+' token is an operator that adds a left vector to a right vector. `NULL` value will be replaced with `c(",", "|", "&", "<=", "<", ">=", ">", "==", "!=", "+", "-", "*", "%/%", "/", "%%", "%in%", ":", "^")`. The order determines the precedence of the operators.
`valid_functions`	tokens of any length that are prefixed on a parenthesis block and specify a function to run on the provided parameters within the block. For example, the 'log' token will evaluate the logarithm value of the first parameter. Note named parameters are not support. `NULL` value will be replaced with `c("log", "c", "any", "all", "abs", "ifelse")`.
`map`	a named list of data.frames/lists/matrices. Where names are keys for referencing the values in the `text` parameters.
`mapping_names`	optional argument to make the function faster or limit which map elements can be referenced.

Details

See `vignette("Overview", package = "evalR")`

Value

numeric or logical vector

Examples

```
eval_text("1 + 2")

# using the map parameter
map_obj <- list("#" = data.frame(x = 1:5, y = 5:1),"$" = list(z = -(1:5)))
y <- evalR::eval_text("#x# + $z$", map=map_obj)
```

`eval_tree` *safely evaluate tree*

Description

Safe alternative to using eval + parse on some string that has already been converted into a tree.

Usage

```
eval_tree(
  tree,
  singular_operators = NULL,
  binary_operators = NULL,
  valid_functions = NULL,
  map = NULL,
  mapping_names = NULL
)
```

Arguments

`tree`	the output object from create_tree
`singular_operators`	tokens of length 1 that operate on a right hand value. For example, the '-' token is an operator to negate a vector. `NULL` value will be replaced with `c("-", "!")`.
`binary_operators`	tokens of any length that operate on a left and right hand values. For example, the '+' token is an operator that adds a left vector to a right vector. `NULL` value will be replaced with `c(",", "|", "&", "<=", "<", ">=", ">", "==", "!=", "+", "-", "*", "%/%", "/", "%%", "%in%", ":", "^")`. The order determines the precedence of the operators.
`valid_functions`	tokens of any length that are prefixed on a parenthesis block and specify a function to run on the provided parameters within the block. For example, the 'log' token will evaluate the logarithm value of the first parameter. Note named parameters are not support. `NULL` value will be replaced with `c("log", "c", "any", "all", "abs", "ifelse")`.

map	a named list of data.frames/lists/matrices. Where names are keys for referencing the values in the text parameters.
mapping_names	optional argument to make the function faster or limit which map elements can be referenced.

Details

See vignette("Overview", package = "evalR")

Value

numeric or logical vector

Examples

```
tree <- create_tree("1 + 2")
eval_tree(tree)
```

---

find_parenthesis *Helper to find first block of parenthesis*

Description

This function will search for the first block of parenthesis and return it if found. Otherwise, it will return "".

Usage

```
find_parenthesis(text)
```

Arguments

text	the string/code/statement you want to parse.

Value

a substring. Either "" or the first parenthesis block.

Examples

```
# returns ""
find_parenthesis("3 + 5")
# returns "(3 + 5)"
find_parenthesis("2 * (3 + 5)")
```

Index