Package: rco (via r-universe)

November 11, 2024

Type Package

Title The R Code Optimizer

Version 1.0.2

Maintainer Juan Cruz Rodriguez <jcrodriguez@unc.edu.ar>

Description Automatically apply different strategies to optimize R code. 'rco' functions take R code as input, and returns R code as output.

Depends R (>= 3.6.0)

License GPL-3

URL https://jcrodriguez1989.github.io/rco/

BugReports https://github.com/jcrodriguez1989/rco/issues

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.1.1

Suggests covr, diffr, ggplot2, httr, knitr, markdown, microbenchmark, rmarkdown, rstudioapi, rvest, shiny, shinythemes, testthat, xml2

Repository https://jcrodriguez1989.r-universe.dev

RemoteUrl https://github.com/jcrodriguez1989/rco

RemoteRef HEAD

RemoteSha 1df14992b7351bf68f28ae316abd8715ed7ff633

Contents

`all_optimizers` *All optimizers list.*

Description

List of all the optimizer functions:

- Conditional Threading `opt_cond_thread`
- Constant Folding `opt_constant_folding`
- Constant Propagation `opt_constant_propagation`
- Dead Code Elimination `opt_dead_code`
- Dead Store Elimination `opt_dead_store`
- Dead Expression Elimination `opt_dead_expr`
- Common Subexpression Elimination `opt_common_subexpr`
- Loop-invariant Code Motion `opt_loop_invariant`
- Memory Allocation `opt_memory_alloc`

Usage

```
all_optimizers
```

Format

An object of class `list` of length 9.

`generate_files_opt_report`
Report possible optimizations in ‘.R‘ files.

Description

Report possible optimizations in ‘.R‘ files.

Usage

```
generate_files_opt_report(files, optimizers = rco:::all_optimizers)
```

Arguments

`files`	A character vector with paths to files to optimize.
`optimizers`	A named list of optimizer functions.

`generate_folder_opt_report`
Report possible optimizations in folder containing ‘.R‘ files.

Description

Report possible optimizations in folder containing ‘.R‘ files.

Usage

```
generate_folder_opt_report(
  folder,
  optimizers = all_optimizers,
  pattern = "\\.R$",
  recursive = TRUE
)
```

Arguments

`folder`	Path to a directory with files to optimize.
`optimizers`	A named list of optimizer functions.
`pattern`	An optional regular expression. Only file names which match the regular expression will be optimized.
`recursive`	A logical value indicating whether or not files in subdirectories of ‘folder‘ should be optimized as well.

generate_text_opt_report *Report possible optimizations in '.R' code snippet.*

Description

Report possible optimizations in '.R' code snippet.

Usage

```
generate_text_opt_report(text, optimizers = all_optimizers)
```

Arguments

`text`	A character vector with the code to optimize.
`optimizers`	A named list of optimizer functions.

max_optimizers *Max optimizers list.*

Description

List of all the optimizer functions, with maximum optimization techniques enabled. Note that using this optimizers could change the semantics of the program!

- Conditional Threading `opt_cond_thread`
- Constant Folding `opt_constant_folding`
- Constant Propagation `opt_constant_propagation`
- Dead Code Elimination `opt_dead_code`
- Dead Store Elimination `opt_dead_store`
- Dead Expression Elimination `opt_dead_expr`
- Common Subexpression Elimination `opt_common_subexpr`
- Loop-invariant Code Motion `opt_loop_invariant`
- Memory Allocation `opt_memory_alloc`

Usage

```
max_optimizers
```

Format

An object of class `list` of length 9.

`optimize_files` *Optimize '.R' files.*

Description

Performs the desired optimization strategies in the files specified. Carefully examine the results after running this function! If several files interact between them (functions from one file use functions from the other), then optimizing all of them together gives more information to rco.

Usage

```
optimize_files(
  files,
  optimizers = all_optimizers,
  overwrite = FALSE,
  iterations = Inf
)
```

Arguments

`files`	A character vector with paths to files to optimize.
`optimizers`	A named list of optimizer functions.
`overwrite`	A logical indicating if files should be overwritten, or saved into new files with "optimized_" prefix.
`iterations`	Numeric indicating the number of times optimizers should pass. If there was no change after current pass, it will stop.

`optimize_folder` *Optimize a folder with '.R' files.*

Description

Performs the desired optimization strategies in all the '.R' in a directory. Carefully examine the results after running this function! If several files interact between them (functions from one file use functions from the other), then optimizing all of them together gives more information to rco.

Usage

```
optimize_folder(
  folder,
  optimizers = all_optimizers,
  overwrite = FALSE,
  iterations = Inf,
  pattern = "\\.R$",
  recursive = TRUE
)
```

Arguments

`folder`	Path to a directory with files to optimize.
`optimizers`	A named list of optimizer functions.
`overwrite`	A logical indicating if files should be overwritten, or saved into new files with "optimized_" prefix.
`iterations`	Numeric indicating the number of times optimizers should pass. If there was no change after current pass, it will stop.
`pattern`	An optional regular expression. Only file names which match the regular expression will be optimized.
`recursive`	A logical value indicating whether or not files in subdirectories of ‘folder‘ should be optimized as well.

`optimize_text` *Optimize text containing code.*

Description

Performs the desired optimization strategies in the text. Carefully examine the results after running this function!

Usage

```
optimize_text(text, optimizers = all_optimizers, iterations = Inf)
```

Arguments

`text`	A character vector with the code to optimize.
`optimizers`	A named list of optimizer functions.
`iterations`	Numeric indicating the number of times optimizers should pass. If there was no change after current pass, it will stop.

`opt_common_subexpr` *Optimizer: Common Subexpression Elimination.*

Description

Performs one common subexpression elimination pass. Carefully examine the results after running this function!

Usage

```
opt_common_subexpr(texts, n_values = 2, in_fun_call = FALSE)
```

Arguments

`texts`	A list of character vectors with the code to optimize.
`n_values`	A numeric indicating the minimum number of values to consider a subexpression.
`in_fun_call`	A logical indicating whether it should propagate in function calls. Note: this could change the semantics of the program.

Examples

```
code <- paste(
  "a <- b * c + g",
  "d = b * c * e",
  sep = "\n"
)
cat(opt_common_subexpr(list(code))$codes[[1]])

heron_formula <- paste(
  "area <- (a/2 + b/2 +c/2) * (a/2 + b/2 + c/2 - a) * (a/2 + b/2 + c/2 - b) *",
  "  (a/2 + b/2 + c/2 - c)",
  "area <- sqrt(area)",
  sep = '\n'
)
cat(opt_common_subexpr(list(heron_formula))$codes[[1]])
```

`opt_cond_thread` *Optimizer: Conditional Threading.*

Description

Performs one conditional threading pass. Carefully examine the results after running this function!

Usage

```
opt_cond_thread(code)
```

Arguments

`code`	A list of character vectors with the code to optimize.

Examples

```
code <- paste(
  "num <- sample(100, 1)",
  "even_sum <- 0",
  "odd_sum_a <- 0",
  "odd_sum_b <- 0",
  "if (num %% 2 == 1) {",
  "  odd_sum_a <- odd_sum_a + num",
  "}",
  "if (num %% 2 == 1) {",
  "  odd_num_b <- odd_num_b + num",
  "}",
  "if (!(num %% 2 == 1)) {",
  "  even_sum <- even_sum + num",
  "}",
  sep = "\n"
)
cat(opt_cond_thread(list(code))$codes[[1]])
```

opt_constant_folding *Optimizer: Constant Folding.*

Description

Performs one constant folding pass. Carefully examine the results after running this function!

Usage

```
opt_constant_folding(texts, fold_floats = FALSE, in_fun_call = FALSE)
```

Arguments

`texts`	A list of character vectors with the code to optimize.
`fold_floats`	A logical indicating if floating-point results should be folded (will reduce precision).
`in_fun_call`	A logical indicating whether it should propagate in function calls. Note: this could change the semantics of the program.

Examples

```
code <- paste(
  "i <- 320 * 200 * 32",
  "x <- i * 20 + 100",
  sep = "\n"
)
cat(opt_constant_folding(list(code))$codes[[1]])
```

`opt_constant_propagation` *Optimizer: Constant Propagation.*

Description

Performs one constant propagation pass. Carefully examine the results after running this function!

Usage

```
opt_constant_propagation(texts, in_fun_call = FALSE)
```

Arguments

`texts`	A list of character vectors with the code to optimize.
`in_fun_call`	A logical indicating whether it should propagate in function calls. Note: this could change the semantics of the program.

Examples

```
code <- paste(
  "i <- 170",
  "x <- -170",
  "y <- x + 124",
  "z <- i - 124",
  sep = "\n"
)
cat(opt_constant_propagation(list(code))$codes[[1]])

hemisphere_vol <- paste(
  "pi <- 3.141593 ",
  "radius <- 25 ",
  "hemis_vol <- 2/3 * pi * radius ^ 3 ",
  sep = "\n"
)
cat(opt_constant_propagation(list(hemisphere_vol))$codes[[1]])
```

`opt_dead_code` *Optimizer: Dead Code Elimination.*

Description

Performs one dead code elimination pass. Carefully examine the results after running this function!

Usage

```
opt_dead_code(texts)
```

Arguments

`texts`	A list of character vectors with the code to optimize.

Examples

```
code <- paste(
  "while (TRUE) {",
  "  break",
  "  dead_code()",
  "}",
  sep = "\n"
)
cat(opt_dead_code(list(code))$codes[[1]])
```

`opt_dead_expr` *Optimizer: Dead Expression Elimination.*

Description

Performs one dead expression elimination pass. Carefully examine the results after running this function!

Usage

```
opt_dead_expr(texts)
```

Arguments

`texts`	A list of character vectors with the code to optimize.

Examples

```
code <- paste(
  "foo <- function(x) {",
  "  x ^ 3",
  "  return(x ^ 3)",
  "}",
  sep = "\n"
)
cat(opt_dead_expr(list(code))$codes[[1]])
```

`opt_dead_store` *Optimizer: Dead Store Elimination.*

Description

Performs one dead store elimination pass. Carefully examine the results after running this function!

Usage

```
opt_dead_store(texts)
```

Arguments

`texts`	A list of character vectors with the code to optimize.

Examples

```
code <- paste(
  "foo <- function() {",
  "  x <- 128 ^ 2",
  "  return(TRUE)",
  "}",
  sep = "\n"
)
cat(opt_dead_store(list(code))$codes[[1]])

code <- paste(
  "sinpi <- function() {",
  "  pi <- 3.1415",
  "  e <- 2.718",
  "  phi <- 1.618",
  "  sin(pi)",
  "}",
  sep = "\n"
)
cat(opt_dead_store(list(code))$codes[[1]])
```

`opt_loop_invariant` *Optimizer: Loop-invariant Code Motion.*

Description

Performs one loop-invariant code motion pass. Carefully examine the results after running this function!

Usage

```
opt_loop_invariant(texts)
```

Arguments

`texts`	A list of character vectors with the code to optimize.

Examples

```
code <- paste(
  "i <- 0",
  "while (i < n) {",
  "  x <- y + z",
  "  a[i] <- 6 * i + x * x",
  "  i <- i + 1",
  "}",
  sep = "\n"
)
cat(opt_loop_invariant(list(code))$codes[[1]])
```

`opt_memory_alloc` *Optimizer: Memory Allocation.*

Description

Performs one memory allocation pass. Carefully examine the results after running this function!

Usage

```
opt_memory_alloc(code)
```

Arguments

`code`	A list of character vectors with the code to optimize.

Examples

```
code <- paste(
  "v <- NULL",
  "for (i in 1:5) {",
  "  v[i] <- i^2",
  "}",
  sep = "\n"
)
cat(opt_memory_alloc(list(code))$codes[[1]])
```

`rco_gui` *rco GUI selector.*

Description

Starts the selected rco Graphical User Interface (GUI).

Usage

```
rco_gui(option)
```

Arguments

`option`	A character indicating which GUI to open. One from: • "code_optimizer" for single code optimizing. • "pkg_optimizer" for package optimizing.

Examples

```
## Start the GUI
## Not run:
rco_gui("code_optimizer")

## End(Not run)
```

Index