Package 'deduped'

June 15, 2026

Type Package

Title Making ``Deduplicated'' Functions

Version 0.5.0

Description Contains one main function deduped() which speeds up slow, vectorized functions by only performing computations on the unique values of the input and expanding the results at the end.

License MIT + file LICENSE

Encoding UTF-8

Imports collapse,

Suggests testthat (>= 3.0.0), withr, fs

Config/testthat/edition 3

URL https://github.com/orgadish/deduped

BugReports https://github.com/orgadish/deduped/issues

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Or Gadish [aut, cre, cph]

Maintainer Or Gadish <orgadish@gmail.com>

Repository CRAN

Date/Publication 2026-06-15 05:40:02 UTC

Contents

deduped *Deduplicate a vectorized function to act on* unique *elements*

Description

Converts a vectorized function into one that only performs the computations on unique values in the first argument. The result is then expanded so that it is the same as if the computation was performed on all elements.

Note: This only works with functions that preserve length and order.

Usage

```
deduped(f, verbose = NULL)
```

Arguments

f	A length-preserving, order-preserving function that accepts a vector or list as its first input.
verbose	If TRUE, prints the number of unique values and reduction percentage on each call. If NULL (default), reads getOption("deduped.verbose", FALSE) at call time, so setting options(deduped.verbose = TRUE) enables it for the entire session.

Details

We make a best effort to preserve the two main cases for named inputs:

1. If f() drops names, names are dropped in the output.
2. Otherwise, we preserve the names from the input x. We cannot reliably re-expand the names from the deduped output, since duplicate values would always map back to their first occurrence's name.

Value

Deduplicated version of f.

Examples

```
x <- sample(LETTERS, 10)
x

large_x <- sample(rep(x, 10))
length(large_x)

slow_func <- function(x) {
  for (i in x) {
    Sys.sleep(0.001)
  }
```

```
  tolower(x)
}

system.time({
  y1 <- slow_func(large_x)
})

system.time({
  y2 <- deduped(slow_func)(large_x)
})

all(y1 == y2)
```

`with_deduped` *Deduplicate the first argument in an expression*

Description

This is a convenience wrapper for `deduped()` to allow it to be piped into an expression. It will recursively parse the first arguments of the expression call tree to find the bottom – when the first argument is not itself a function call.

- Without nesting: `f(x, ...) |> with_deduped()` is equivalent to `deduped(\(.z) f(.z, ...))(x)`.
- With nesting: `f(g(x, g2), f2) |> with_deduped()` is equivalent to `deduped(\(.z) f(g(.z, g2), f2))(x)`.

Usage

```
with_deduped(expr, env = parent.frame(), verbose = NULL)
```

Arguments

`expr`	The expression to evaluate.
`env`	The environment within which to evaluate the expression. Can be modified when calling inside other functions.
`verbose`	If `TRUE`, prints the number of unique values and reduction percentage on each call. If `NULL` (default), reads `getOption("deduped.verbose", FALSE)` at call time, so setting `options(deduped.verbose = TRUE)` enables it for the entire session.

Details

`with_deduped()` reconstructs the wrapper function on every call, so it is best suited for one-off or interactive use. For repeated calls such as inside a loop, build the wrapper once with `deduped()` instead:

```
# Preferred for loops:
deduped_f <- deduped(slow_func)
for (...) deduped_f(x)

# Rather than:
for (...) slow_func(x) |> with_deduped()
```

Value

The result of evaluating the expression.

Examples

```
x <- sample(LETTERS, 10)
x

large_x <- sample(rep(x, 10))
length(large_x)

slow_func <- function(x) {
  for (i in x) {
    Sys.sleep(0.001)
  }
  tolower(x)
}

system.time({
  y1 <- slow_func(large_x)
})


system.time({
  y2 <- with_deduped(slow_func(large_x))

  # Can also use the R pipe (R >= 4.1.0) or magrittr pipe, for convenience.
  # slow_func(large_x) |> with_deduped()
})

all(y1 == y2)
```

Index