Package 'minex'

July 16, 2026

Title Automatically Reduce Failing R Scripts to a Minimal Reproducible Example

Version 0.1.0

Description Shrinks a failing R script to the smallest subset of statements that still triggers the same error, using the delta debugging algorithm of Zeller and Hildebrandt (2002) <doi:10.1109/32.988498>. Each candidate reduction is evaluated in a separate R process, so dependencies between statements and their side effects are respected. The result is a one-minimal example, in which removing any remaining statement makes the error disappear; this is the form most useful for bug reports and for questions on community forums. A general delta debugging routine and a helper for reducing data frames to the rows that reproduce a failure are also provided.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports callr

Suggests knitr, rmarkdown, spelling, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Language en-US

URL https://github.com/DIGlabUAB/minex

BugReports https://github.com/DIGlabUAB/minex/issues

NeedsCompilation no

Author Sandeep Bodduluri [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-8201-5262>)

Maintainer Sandeep Bodduluri <sbodduluri@uabmc.edu>

Repository CRAN

Date/Publication 2026-07-16 14:00:31 UTC

Contents

`ddmin` *Delta debugging*

Description

General implementation of the `ddmin` minimization algorithm of Zeller and Hildebrandt (2002). Given a collection of elements and a predicate that reports whether a subset still exhibits some behavior of interest, `ddmin()` returns a subset that is *one-minimal*: the predicate holds for it, but fails for every subset obtained by removing a single element.

Usage

```
ddmin(items, interesting, verbose = FALSE)
```

Arguments

`items`	A list or atomic vector of elements to minimize.
`interesting`	A predicate applied to a subset of `items`, in the same form as `items`, returning a single logical. It should return `TRUE` when the subset still reproduces the behavior of interest.
`verbose`	Logical. If `TRUE`, report each reduction step via `message()`.

Details

The algorithm partitions the current candidate into `n` blocks (starting with `n = 2`). It first tests whether any single block reproduces the behavior; if so it continues with that block. Otherwise it tests each complement (the candidate with one block removed) and continues with the first that reproduces. If neither succeeds the granularity is doubled, up to the point where each element sits in its own block, which guarantees one-minimality.

Results of the predicate are cached on the set of element indices, so an identical configuration is never evaluated twice.

Value

The one-minimal subset of `items`, in the original order.

References

Zeller A, Hildebrandt R (2002). "Simplifying and Isolating Failure-Inducing Input." *IEEE Transactions on Software Engineering*, 28(2), 183-200. doi:10.1109/32.988498

See Also

`minex()` for the script-reduction front end and `reduce_rows()` for reducing data frames.

Examples

```
# Reduce a sentence to the single word a predicate depends on.
words <- strsplit("the quick brown fox", " ")[[1]]
ddmin(words, function(s) "fox" %in% s)

# When several elements are jointly required, all of them are kept.
nums <- 1:6
ddmin(nums, function(s) sum(s) >= 11 && 6 %in% s)
```

`minex` *Minimize a failing R script to a reproducible example*

Description

Reduces a failing piece of R code to the smallest subset of its top-level statements that still triggers the same failure. The result is a *one-minimal* example: removing any remaining statement makes the failure disappear. This is the form requested when reporting bugs or asking for help, and the part of preparing such an example that is usually done by hand.

Usage

```
minex(
  file = NULL,
  code = NULL,
  oracle = NULL,
  match = c("message", "class", "both"),
  backend = c("callr", "inprocess"),
  timeout = 60,
  verbose = FALSE
)
```

Arguments

`file`	Path to a file containing the R code to minimize. Ignored if `code` is supplied.
`code`	A character vector of R source lines, or a single string. Takes precedence over `file`.
`oracle`	Optional predicate taking a character vector of statements and returning a single logical. When supplied, the target failure is not recorded automatically and `match` is ignored; you are fully responsible for defining what counts as reproducing the failure.

match	How a candidate's failure must match the recorded one when no oracle is given: "message" (identical message, the default), "class" (shares a condition class) or "both". Matching on the message is usually right, because an over-reduced fragment tends to fail with a different message (for example "object not found").
backend	Either "callr" (evaluate each candidate in a fresh R process, the default and the only choice that fully isolates state) or "inprocess" (evaluate in the current session, faster but without isolation).
timeout	Maximum seconds allowed for a single callr evaluation.
verbose	Logical. If TRUE, report progress.

Details

By default minex() first runs the whole input to record the failure it produces (its condition message and class), then uses ddmin() to search for a minimal subset that reproduces it. Each candidate is evaluated in a separate R process so that dependencies between statements and their side effects are respected; removing a statement that a later one needs typically changes the error, and such a removal is therefore rejected.

Supply a custom oracle to minimize against any condition you can express as a predicate, rather than against the recorded failure. The oracle receives a character vector of statements and must return a single logical.

Value

An object of class "minex_result": a list with the minimized code (a character vector of statements), the original statements, the statement counts n_original and n_minimal, the number of oracle_calls, the recorded target failure (or NULL for a custom oracle), and the match and backend settings.

See Also

ddmin() for the underlying algorithm and reduce_rows() for reducing data frames.

Examples

```
# A failing script padded with irrelevant setup.
script <- c(
  "a <- 10",
  "b <- 20",
  "log('not a number')"
)
res <- minex(code = script, backend = "inprocess")
res
cat(as.character(res), "\n")

# A failure that genuinely depends on an earlier statement: both are kept.
script2 <- c(
  "x <- c(1, 2, NA)",
  "m <- mean(x)",
  "if (is.na(m)) stop('mean is NA')"
```

```
)
minex(code = script2, backend = "inprocess")


# The default backend runs candidates in fresh R processes.
minex(code = script)
```

---

reduce_rows *Reduce a data frame to the rows that reproduce a failure*

Description

Often a bug only shows up with a large data frame, even though a handful of rows is enough to trigger it. `reduce_rows()` applies `ddmin()` over the rows of `data` and returns the smallest subset for which `predicate` still holds, preserving the original row order. The result is typically small enough to paste into a bug report with `dput()`.

Usage

```
reduce_rows(data, predicate, verbose = FALSE)
```

Arguments

`data`	A data frame.
`predicate`	A function taking a data frame (a subset of `data`'s rows) and returning a single logical: `TRUE` when the subset still reproduces the failure of interest.
`verbose`	Logical. If `TRUE`, report progress.

Value

A data frame containing the one-minimal subset of rows.

See Also

`ddmin()`, `minex()`.

Examples

```
df <- data.frame(id = 1:6, value = c(3, 8, 999, 2, 5, 7))
# The failure: any value greater than 100.
reduce_rows(df, function(d) any(d$value > 100))
```

Index