Package 're2r'

September 4, 2017

Type Package

Title RE2 Regular Expression

Version 0.2.0

Maintainer Qin Wenfeng <mail@qinwenfeng.com>

Description RE2 <https://github.com/google/re2> is a primarily deterministic finite automaton based regular expression engine from Google that is very fast at matching large amounts of text.

License BSD_3_clause + file LICENSE

LazyData TRUE

Depends R (>= 3.3)

Imports Rcpp (>= 0.12.2), stringi, RcppParallel, htmlwidgets (>= 0.6),

LinkingTo Rcpp, RcppParallel

Suggests knitr (>= 1.12.3),testthat, rmarkdown (>= 0.9.5),microbenchmark, ggplot2, directlabels

URL https://github.com/qinwf/re2r/

BugReports https://github.com/qinwf/re2r/issues

VignetteBuilder knitr

NeedsCompilation yes

RoxygenNote 6.0.1

SystemRequirements GNU make

Author Qin Wenfeng [aut, cre],
Toby Dylan Hocking [ctb] (benchmarks),
Marek Gagolewski [ctb] (stringi benchmarks and test cases),
RE2 developers [ctb] (RE2 library),
Google Inc. [ctb, cph] (RE2 library),
Lucent Technologies [ctb, cph] (RE2 library),
Rob Pike [ctb] (RE2 library),
Ken Thompson [ctb] (RE2 library),
Julian Seward [ctb, cph] (valgrind.h),
Andrzej Krzemienski [ctb] (optional.hpp),
Jeffrey Avallone [ctb] (regexper library)

Repository CRAN

Date/Publication 2017-09-04 17:32:09 UTC

R topics documented:

`get_expression_size` *Get pre-compiled regular expression program size*

Description

Returns the program size, a very approximate measure of a regexp's "cost". Larger numbers are more expensive than smaller numbers.

Usage

```
get_expression_size(pattern, ...)
```

Arguments

pattern	a pre-compiled regular expression or a string
...	further arguments passed to re2

Value

a integer

Examples

```
get_expression_size(re2("1"))
get_expression_size(re2("(1)"))
get_expression_size(re2("(?:(?:(?:(?:(?:.)?){100})*)+)"))
```

---

get_named_groups *Return capturing names for a pre-compiled regular expression.*

Description

Return capturing names.

Usage

```
get_named_groups(pattern, ...)
```

Arguments

pattern	a pre-compiled regular expression or a string
...	further arguments passed to re2

Value

capturing names

Examples

```
get_named_groups(re2("(a)(?P<name>b)"))

regexp = re2("(?P<A>exprA(?P<B>exprB)(?P<C>exprC))((expr5)(?P<D>exprD))")

print(regexp)
(res = get_named_groups(regexp))
re2_match("exprAexprBexprCexpr5exprD", regexp)
```

get_number_of_groups *Return the number of capturing subpatterns*

Description

Return the number of capturing subpatterns, or -1 if the regexp wasn't valid on construction. The overall match ($0) does not count: if the regexp is "(a)(b)", returns 2.

Usage

```
get_number_of_groups(pattern, ...)
```

Arguments

pattern	a pre-compiled regular expression or a string
...	further arguments passed to re2

Value

a integer

Examples

```
regexp = re2("1")
get_number_of_groups(regexp)

get_number_of_groups(re2("((?P<a>123)(12))"))

# uncaptured groups
get_number_of_groups(re2("(?:(?:(?:(?:(?:.)?){100})*)+)"))
```

get_options *Get options of a pre-compiled regular expression*

Description

Returns options of a pre-compiled regular expression

Usage

```
get_options(pattern, ...)
```

Arguments

pattern	a pre-compiled regular expression or a string
...	further arguments passed to re2

Value

an list

Examples

```
get_options(re2("test"))
```

---

get_pattern *The string specification for this RE2.*

Description

The string specification for this RE2.

Usage

```
get_pattern(pattern, ...)
```

Arguments

pattern	a pre-compiled regular expression or a string
...	further arguments passed to re2

Value

a string

Examples

```
regexp = re2("1")
get_pattern(regexp)

get_pattern(re2("^(?P<abc>abc)a"))
```

---

get_program_fanout *Get program fanout*

Description

Return the program fanout as a histogram bucketed by powers of 2.

Usage

```
get_program_fanout(pattern, ...)
```

Arguments

pattern	a pre-compiled regular expression or a string
...	further arguments passed to re2

Examples

```
re1 = re2("(?:(?:(?:(?:(?:.)?){1})*)+)")
re10 = re2("(?:(?:(?:(?:(?:.)?){10})*)+)")
re100 = re2("(?:(?:(?:(?:(?:.)?){100})*)+)")
re1000 = re2("(?:(?:(?:(?:(?:.)?){1000})*)+)")

get_program_fanout(re1)
get_program_fanout(re10)
get_program_fanout(re100)
get_program_fanout(re1000)
```

---

get_simplify *Simplify pattern strings.*

Description

Simplify pattern strings.

Usage

```
get_simplify(pattern, ...)
```

Arguments

pattern	a pre-compiled regular expression or a string
...	further arguments passed to re2

Examples

```
get_simplify("a{1}")
get_simplify("a{3}b+(:?abc(a))")
get_simplify("a{2,3}a{2}")
get_simplify(re2("1+2",literal = TRUE))
get_pattern(re2("1+2",literal = TRUE))
```

is_re2c_na *Check NA pattern*

Description

Returns whether a pre-compiled regular expression is NA.

Usage

```
is_re2c_na(pattern, ...)
```

Arguments

pattern	a pre-compiled regular expression or a string
...	further arguments passed to re2

Value

a boolean

Examples

```
is_re2c_na(re2(NA))
```

print.re2c *Print information about a pre-compiled regular expression*

Description

Print information about a pre-compiled regular expression

Usage

```
## S3 method for class 're2c'
print(x, options = FALSE, ...)
```

Arguments

x	a pre-compiled regular expression
options	print options
...	further arguments passed to or from other methods.

Examples

```
re2("(.*)@([^.]*)")
re2("(?P<name>sd)")
print(re2("sd"), options = TRUE)
```

print.re2_matrix *Print information about a re2_matrix*

Description

Print information about a re2_matrix

Usage

```
## S3 method for class 're2_matrix'
print(x, ...)
```

Arguments

x	an re2_matrix
...	further arguments passed to or from other methods.

Examples

```
print(re2_match(".*", "abc"))
```

quote_meta *Escapes all potentially meaningful regexp characters in 'unquoted'.*

Description

The returned string, used as a regular expression, will exactly match the original string.

Usage

```
quote_meta(unquoted, parallel = FALSE, grain_size = 1e+05)
```

Arguments

unquoted	unquoted string
parallel	multithreading support
grain_size	a minimum chunk size for tuning the behavior of parallel algorithms.

Value

quoted string

Examples

```
quote_meta(c("1.2","abc"))
re2_detect("1+2", "1+2")
re2_detect("1+2", quote_meta("1+2"))
re2_detect("1+2", re2("1+2",literal = TRUE))
```

re2 *Create a pre-compiled regular expression*

Description

Create a pre-compiled regular expression from a string.

Usage

```
re2(pattern, utf_8 = TRUE, case_sensitive = TRUE, posix_syntax = FALSE,
  dot_nl = FALSE, literal = FALSE, longest_match = FALSE,
  never_nl = FALSE, never_capture = FALSE, one_line = FALSE,
  perl_classes = FALSE, word_boundary = FALSE, log_error = FALSE,
  max_mem = 8388608, simplify = TRUE)
```

Arguments

pattern	regular expression pattern
utf_8	(true) text and pattern are UTF-8; otherwise Latin-1
case_sensitive	(true) match is case-sensitive (regexp can override with (?i) unless in posix_syntax mode)
posix_syntax	(false) restrict regexps to POSIX egrep syntax
dot_nl	(false) dot matches everything including new line
literal	(false) interpret string as literal, not regexp
longest_match	(false) search for longest match, not first match
never_nl	(false) never match \n, even if it is in regexp
never_capture	(false) parse all parens as non-capturing
one_line	(false) ^ and $ only match beginning and end of text, when posix_syntax == false this features are always enabled
perl_classes	(false) allow Perl's \d \s \w \D \S \W, when posix_syntax == false this features are always enabled
word_boundary	(false) allow Perl's \b \B (word boundary and not), when posix_syntax == false this features are always enabled
log_error	(false) log syntax and execution errors
max_mem	(see details) approx. max memory footprint of RE2
simplify	(true) return a object instead of a list when pattern length is 1.

Details

The max_mem option controls how much memory can be used to hold the compiled form of the regexp (the Prog) and its cached DFA graphs.

Once a DFA fills its budget, it flushes its cache and starts over. If this happens too often, RE2 falls back on the NFA implementation.

For now, make the default budget something close to Code Search.

Default maxmem = 8«20 = 8388608;

Value

a pre-compiled regular expression

Examples

```
regexp = re2("test")
regexp

re2_match("abc\ndef","(?s)(.*)")
re2_match("abc\ndef", re2("(?s)(.*)", never_nl = TRUE))

re2_detect("\n", re2(".", dot_nl = TRUE))
re2_detect("\n", ".")

get_number_of_groups(re2("(A)(v)",never_capture = TRUE))

re2_match("aaabaaaa",re2("(a|aaa)",longest_match = TRUE))
re2_match("aaabaaaa",re2("(a|aaa)",longest_match = FALSE))

re2_match("a+b", re2("a+b", literal = TRUE))

re2_detect("abc" , re2("abc", posix_syntax = TRUE))
re2("(?P<name>re)")

## Not run:

expect_error(re2("(?P<name>re)", posix_syntax = TRUE))

## End(Not run)
```

---

re2_count *Count the number of matches in a string.*

Description

Count the number of matches in a string. Vectorised over strings and patterns.

Usage

```
re2_count(string, pattern, anchor = UNANCHORED, parallel = FALSE,
  grain_size = 1e+05, ...)
```

Arguments

string	a character vector
pattern	a character vector or pre-compiled regular expressions
anchor	see UNANCHORED

parallel	use multithread
grain_size	a minimum chunk size for tuning the behavior of parallel algorithms
...	further arguments passed to re2

Value

An integer vector.

Examples

```
re2_count("one", "(o.e)")
re2_count("123-234-2222", "\\d\\d\\d-\\d\\d\\d-\\d\\d\\d\\d")

words = c("sunny","beach","happy","really")
re2_count(words, "y")
re2_count(words, "^b")
re2_count(words, "[abc]")

# vectorize
re2_count("This", letters)
```

---

re2_detect *Test a pattern in strings, and return boolean.*

Description

Test a pattern in strings, and return boolean. Vectorised over strings and patterns.

Usage

```
re2_detect(string, pattern, anchor = UNANCHORED, parallel = FALSE,
  grain_size = 1e+05, ...)
```

Arguments

string	a character vector
pattern	a character vector or pre-compiled regular expressions
anchor	see UNANCHORED
parallel	use multithread
grain_size	a minimum chunk size for tuning the behavior of parallel algorithms
...	further arguments passed to re2

Value

A logical vector.

Examples

```
re2_detect("one", "(o.e)")
re2_detect("123-234-2222", "\\d\\d\\d-\\d\\d\\d-\\d\\d\\d\\d")

words = c("sunny","beach","happy","really")
re2_detect(words, "y")
re2_detect(words, "^b")
re2_detect(words, "[abc]")

# vectorize
(res = re2_detect("This", letters))
letters[res]

letters[re2_detect("This", letters, case_sensitive = FALSE)]

letters[re2_detect("This", re2(letters, case_sensitive = FALSE))]

# In stringi, "" empty search patterns return NA.
# In re2r, empty search patterns will match
# empty string.

re2_detect("abc", "")
stringi::stri_detect("abc", regex = "")
```

---

re2_extract *Extract matching patterns from a string.*

Description

Extract matching patterns from a string. Vectorised over string and pattern.

Usage

```
re2_extract(string, pattern, anchor = UNANCHORED, parallel = FALSE,
  grain_size = 1e+05, ...)

re2_extract_all(string, pattern, anchor = UNANCHORED, parallel = FALSE,
  grain_size = 1e+05, ...)
```

Arguments

string	a character vector
pattern	a character vector or pre-compiled regular expressions
anchor	see UNANCHORED
parallel	use multithread
grain_size	a minimum chunk size for tuning the behavior of parallel algorithms
...	further arguments passed to re2

Value

A character vector for re2_extract, and a list for re2_extract_all.

See Also

re2_match to extract matched groups.

Examples

```
re2_extract("yabba dabba doo", "(.)")
re2_extract_all("yabba dabba doo", "(.)")

str <- c("Aster", "Azalea x2", "Baby's Breath", "Bellflower")
re2_extract(str, "\\d")
re2_extract(str, "[a-z]+")
re2_extract(str, "\\b\\w{1,3}\\b")

# Extract all matches
re2_extract_all(str, "[A-Za-z]+")
re2_extract_all(str, "\\b\\w{1,3}\\b")
re2_extract_all(str, "\\d")
```

---

re2_locate *Locate the position of patterns in a string.*

Description

Locate the position of patterns in a string.If the match is of length 0, (e.g. from a special match like $) end will be one character less than start. Vectorised over string and pattern.

Usage

```
re2_locate(string, pattern, parallel = FALSE, grain_size = 1e+05, ...)

re2_locate_all(string, pattern, parallel = FALSE, grain_size = 1e+05, ...)
```

Arguments

string	a character vector
pattern	a character vector or pre-compiled regular expressions
parallel	use multithread
grain_size	a minimum chunk size for tuning the behavior of parallel algorithms
...	further arguments passed to re2

Value

For re2_locate, an integer matrix. First column gives start postion of match, and second column gives end position. For re2_locate_all a list of integer matrices.

Examples

```
re2_locate("yabba dabba doo", "d")

fruit <- c("apple", "banana", "pear", "pineapple")
re2_locate(fruit, "$")
re2_locate(fruit, "a")
re2_locate(fruit, "e")
re2_locate(fruit, c("a", "b", "p", "p"))

re2_locate_all(fruit, "a")
re2_locate_all(fruit, "e")
re2_locate_all(fruit, c("a", "b", "p", "p"))

# Find location of every character
re2_locate_all(fruit, "\\P{M}")
```

---

re2_match *Find matched groups from strings.*

Description

Find matched groups from strings.

Usage

```
re2_match(string, pattern, anchor = UNANCHORED, parallel = FALSE,
  grain_size = 1e+05, ...)

re2_match_all(string, pattern, anchor = UNANCHORED, parallel = FALSE,
  grain_size = 1e+05, ...)
```

Arguments

string	a character vector
pattern	a character vector or pre-compiled regular expressions
anchor	see UNANCHORED
parallel	use multithread
grain_size	a minimum chunk size for tuning the behavior of parallel algorithms
...	further arguments passed to re2

Value

For re2_match, a character matrix. First column is the complete match, followed by one column for each capture group with names.

For re2_match_all, a list of character matrices.

Examples

```
strings <- c("Gym: 627-112-1433", "Apple x2",
             "888 888 8888", "This is a test.",
             "627-112-1433 223-343-2232")
phone <- "([2-9][0-9]{2})[- .](?P<second>[0-9]{3})[- .]([0-9]{4})"
re2_extract(strings, phone)
re2_match(strings, phone)

re2_extract_all(strings, phone)
re2_match_all(strings, phone)

regexp = re2("test",case_sensitive = FALSE)
re2_match("TEST", regexp)

# differences from stringi

# This kind of repeating capturing group works differently.
re2_match("aasd", "(a*)+")
stringi::stri_match("aasd", regex = "(a*)+")

# In stringi, "" empty search patterns return NA.
# In re2r, empty search patterns will match
# empty string.

re2_match("abc", "")
stringi::stri_match("abc", regex = "")

dates <- c("2008-08-08", "2020", "a string",
           "12-12-72", "1989-06-30", "2115-11-21 09:21")
pattern <- "([0-9]{4})-([0-1][0-9])-([0-3][0-9])"
re2_match(dates, pattern)

pattern <- "(?P<y>[0-9]{4})-(?P<m>[0-1][0-9])-(?P<d>[0-3][0-9])"
(res = re2_match(dates, pattern))
res$y
res$m
res$d

pattern <- paste0(
"(?P<first>[A-Z][a-z]+) ",
"(?P<last>[A-Z][a-z]+)"
)
texts <- c(
    "  Taylor Swift and Lady Gaga",
    "One Direction hit the road agains"
```

```
)
re2_match_all(texts, pattern)

texts = c("pi is 3.14529..",
          "-15.34 F",
          "128 days",
          "1.9e10",
          "123,340.00$",
          "only texts")
(number_pattern = re2(".*?(?P<number>-?\\d+(,\\d+)*(\\d+(e\\d+)?)?).*?"))

(res = re2_match(texts, number_pattern))
res$number

# show_regex(number_pattern)
```

`re2_replace` *Replace matched patterns in a string.*

Description

Replace the the first match or all matches of `pattern` in `string` with `replacement`.

Usage

```
re2_replace(string, pattern, replacement, parallel = FALSE,
  grain_size = 1e+05, ...)

re2_replace_all(string, pattern, replacement, parallel = FALSE,
  grain_size = 1e+05, ...)
```

Arguments

`string`	a character vector
`pattern`	a pre-compiled regular expression or a string
`replacement`	replace the first match or all of the match of `pattern` in `string` with "rewrite"
`parallel`	use multithread
`grain_size`	a minimum chunk size for tuning the behavior of parallel algorithms
`...`	further arguments passed to `re2`

Details

Within `replacement`, backslash-escaped digits (\1 to \9) can be used to insert text matching corresponding parenthesized group from the pattern. \0 in `replacement` refers to the entire matching text.

Vectorised over strings, patterns and replacements.

Value

For re2_replace, a character vector. For re2_replace_all, a character vector with the number of replacements.

Examples

```
# replace one or more b, prefer more
regexp = re2("b+")
re2_replace_all("yabba dabba doo", regexp,"d")
re2_replace("yabba dabba doo", "b+","d")

# trim string
pattern = "^\\s+|\\s+$"
re2_replace_all(c("  abc  "," this is "), pattern, "")

# mask the middle three digits of a US phone number
texts = c("415-555-1234",
          "650-555-2345",
          "(416)555-3456",
          "202 555 4567",
          "4035555678",
          "1 416 555 9292")

us_phone_pattern = re2("(1?[\\s-]?\\(?\\d{3}\\)?[\\s-]?)(\\d{3})([\\s-]?\\d{4})")

re2_replace(texts, us_phone_pattern, "\\1***\\3")

# show_regex(us_phone_pattern)
```

---

re2_split *Split a string into pieces.*

Description

Split a string into pieces. Vectorised over string and pattern.

Usage

```
re2_split(string, pattern, n = Inf, parallel = FALSE, grain_size = 10000,
  ...)

re2_split_fixed(string, pattern, n, parallel = FALSE, grain_size = 10000,
  ...)
```

Arguments

string	a character vector
pattern	a character vector or pre-compiled regular expressions
n	number of pieces to return. Default (Inf) for re2_split uses all possible split positions. For re2_split_fixed, if n is greater than the number of pieces, the result will be padded with empty strings.
parallel	use multithread
grain_size	a minimum chunk size for tuning the behavior of parallel algorithms
...	further arguments passed to re2

Value

For re2_split_fixed, a character matrix with n columns.

For re2_split, a list of character vectors.

Examples

```
re2_split("yabba dabba doo", " ")
re2_split_fixed(c("yabba dabba doo", "a bc"), " ", 2)
```

---

re2_subset *Keep strings matching a pattern.*

Description

This is a convenient wrapper around x[re2_detect(x, pattern)]. Vectorised over strings and patterns.

Usage

```
re2_subset(string, pattern, anchor = 0, omit_na = TRUE, parallel = FALSE,
  grain_size = 1e+05, ...)
```

Arguments

string	a character vector
pattern	a character vector or pre-compiled regular expressions
anchor	see UNANCHORED
omit_na	omit na result
parallel	use multithread
grain_size	a minimum chunk size for tuning the behavior of parallel algorithms
...	further arguments passed to re2

Value

A character vector.

Examples

```
fruit <- c("apple", "banana", "pear", "pinapple")
re2_subset(fruit, "a")
re2_subset(fruit, "^a")
re2_subset(fruit, "a$")
re2_subset(fruit, "b")
re2_subset(fruit, "[aeiou]")

re2_subset(c("a", NA, "b"), ".")
```

---

show_regex *Show regex pattern in a htmlwidget*

Description

Show JS-style regex pattern in an htmlwidget.

Usage

```
show_regex(pattern, width = NULL, height = NULL)
```

Arguments

pattern	a pattern string
width	the widget width
height	the widget height

Details

Most parts of RE2 regex syntax are supported, except for some special Unicode character classes.

See Also

https://regexper.com/documentation.html

Examples

```
# Skip on CRAN

## Not run:

# US ZIP code

show_regex("[0-9]{5}(?:-[0-9]{4})?")
```

```
# Email

show_regex("\\b[a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4}\\b")

# Hex value

show_regex("#?([a-f0-9]{6}|[a-f0-9]{3})")


## End(Not run)
```

`sub_string` *Extract and replace substrings from a character vector.*

Description

`sub_string` extracts substrings under code point-based index ranges provided. `sub_string<-` allows to substitute parts of a string with given strings.

Usage

```
sub_string(string, from = 1L, to = -1L)

sub_string(string, from = 1L, to = -1L) <- value
```

Arguments

`string`	input character vector.
`from`	an integer vector or a two-column matrix. `from` gives the position of the first character (defaults to first). Negative values count backwards from the last character.
`to`	an integer vector. `to` gives the position of the last (defaults to last character).
`value`	replacement string

Value

A character vector of substring from `start` to `end` (inclusive). Will be length of longest input argument.

See Also

The underlying implementation in `stri_sub`

Examples

```
sub_string("test", 1, 2)

x <- "ABC"

(sub_string(x, 1, 1) <- "A")
x
(sub_string(x, -2, -2) <- "GHIJ")
x
(sub_string(x, 2, -2) <- "")
x
```

UNANCHORED *Anchors for pattern*

Description

Anchors for regular expression pattern string.

Usage

```
UNANCHORED

ANCHOR_START

ANCHOR_BOTH
```

Format

An object of class `numeric` of length 1.

Details

UNANCHORED - No anchoring

ANCHOR_START - Anchor at start only

ANCHOR_BOTH - Anchor at start and end

Examples

```
re2_match("This is an apple.", "(is)",anchor = ANCHOR_BOTH)
re2_match("This is an apple.", "(is)",anchor = UNANCHORED)
re2_match("This is an apple.", "(is)",anchor = ANCHOR_START)
re2_match("This is an apple.", "(This)",anchor = ANCHOR_START)
re2_match("This is an apple.", "(This)",anchor = ANCHOR_BOTH)
```

`$.re2_matrix` *Get a match group*

Description

Get a match group

Usage

```
## S3 method for class 're2_matrix'
self$group
```

Arguments

`self`	re2_matrix
`group`	group name

Examples

```
text = c("this is test",
    "this is test, and this is not test",
    "they are tests")
res = re2_match(
    string = text,
    pattern = "(?P<testname>this)( is)"
)
class(res)
is.matrix(res)
is.character(res)
print(res)
res$testname
res$.match
res$`.2`
res[, ".2"]
res[, ".match"]
res[, "testname"]
```

Index