Package ‘wyz.code.metaTesting’

February 12, 2020

Type Package
Title Wizardry Code Meta Testing
Version 1.1.11
Author Fabien Gelineau <neonira@gmail.com>
Maintainer Fabien Gelineau <neonira@gmail.com>
Description Test R any R function without having to provide parameter values. Values will be generated, based on semantic naming of parameters as introduced by package ‘wyz.code.offensiveProgramming’. Generated tests can be saved and reused. Value generation logic can be completed with your own specific data types and generation schemes, to meet your requirements. Main benefits of ‘wyz.code.metaTesting’ is higher developer productivity, reduced time to production, and industrial inference testing. Refer to chapter 10 of Offensive Programming Book, Fabien GELINEAU (2019, ISBN:979-10-699-4075-8), to learn about details and get value from this package.

Encoding UTF-8
LazyData true
License GPL-3
Depends R (>= 3.6)
Imports methods, data.table (>= 1.11.8), tidyr, lubridate (>= 1.7.4), wyz.code.offensiveProgramming (>= 1.1.17), crayon, utils, stats
Suggests testthat, knitr, rmarkdown
RoxygenNote 6.1.1
VignetteBuilder knitr
NeedsCompilation no
Repository CRAN
Date/Publication 2020-02-12 08:00:02 UTC
R topics documented:

- buildSemanticArgumentName
- computeArgumentsCombination
- exploreSignatures
- generateData
- opMetaTestingInformation
- opwf
- qualifyFunctionArguments
- retrieveDataFactory
- setDefaultArgumentsGenerationContext
- setGenerationContext
- testFunction
- usesSemanticArgumentNames

buildSemanticArgumentName

Build semantic argument name

Description

Build a semantic argument name from the suffix you provide.

Usage

buildSemanticArgumentName(suffix_s_1, prefix_s_1 = "x_")

Arguments

- **suffix_s_1**: one legal suffix. Use retrieveDataFactory()$getKnownSuffixes() to get a vector of known suffixes. Legal suffixes are known suffixes, augmented by suffixes you registered.
- **prefix_s_1**: the variable prefix name you want to use.

Details

Know that no checks are done on suffix_s_1. Value you provide will be trusted, regular or irregular one.

Value

A single string.

Author(s)

Fabien Gelineau <neonira@gmail.com>
Maintainer: Fabien Gelineau <neonira@gmail.com>
computeArgumentsCombination

See Also

Refer to testFunction

Examples

# typical example
buildSemanticArgumentName('i') # x_i

buildSemanticArgumentName('ui_1', 'numberOfItems') # numberOfItems_ui_1

computeArgumentsCombination

Compute Function Arguments Combination

Description

Computes a priori legal combinations of function arguments, according to the function definition (see formals).

Allows to foresee test complexity for a function, as this is in narrow relationship, with the number of various call signatures that could be used. The number of signatures is in itself a good indicator of complexity.

Usage

computeArgumentsCombination(fun_f_1)

Arguments

fun_f_1 an R function

Details

Computes an a priori legal list of argument signatures for the provided function.

Value

A list holding names, number and signatures, each of them are list.

The names provides names of mandatory arguments, ellipsis(...) arguments and of default arguments.

The number provides the number of replacements per argument.

The signatures are the resulting textual argument combinations.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>
exploreSignatures

See Also

Refer to testFunction

Examples

# typical example
computeArgumentsCombination(append)

computeArgumentsCombination(kronecker)

desc
exploreSignatures Explore Signatures

desc
exploreSignatures

Description

Test an offensive programming wrapper function, applying various argument signatures.

Usage

exploreSignatures(fun_f_1,
    argumentsTypeRestrictions_l = list(),
    signaturesRestrictions_l = list())

Arguments

fun_f_1 a single R function. Must be an offensive programming wrapper function. See opwf.
argumentsTypeRestrictions_l a named list. Each name must match a function argument name. Each content must be a vector of strings, each of them matching a retrieveDataFactory()$getKnownSuffixes() known suffix.
signaturesRestrictions_l an unnamed list of single strings, each of them matching one of computeArgumentsCombination(fun_f_1)

Details

This function offers a really convenient way to test your own functions, without the burden of building the execution context, that is much trickier than one can imagine at first glance.

Moreover it provides argument signature analysis, which is not provided by testFunction.

Arguments restriction parameter argumentsTypeRestrictions_l allows to restrict on demand, value types exploration. It is very useful and convenient to reduce the exploration tree, and to shorten execution time.

By default, a total of 768 tests will run for a single function, when no signaturesRestrictions_l is set. This may requires some time to achieve.

When working interactively, a good practice is to use computeArgumentsCombination prior to use function computeArgumentsCombination, as it will provide complexity information about the

function you wish to test. The number of signature is a good metric of function call complexity. Know that each of them will be tested, and data generation has to be achieved for each parameter according to global or restricted scheme, depending on your argumentsTypeRestrictions.1 inputs.

Value

A list with names info, success, failure, each of them being a list.
The info sub-list holds execution results. It holds following entries

- raw is a list, providing capture of execution context, data and results.
- good is a list, providing same information as raw, filtered to retain only tests that do not generate any error.
- bad is a list, providing same information as raw, filtered to retain only tests that do generate error.

The success sub-list holds analysis results for tests which do not generate errors. It holds following entries

- code is a data.table, providing used call code and results.
- table is a data.table, providing used argument signatures and execution context information.
- synthesis is a list, providing synthesis information. Much easier to read, than table entry.

The failure subsist holds analysis results for tests which do generate errors. It holds following entries

- table is a data.table, providing encountered error messages and execution context information
- synthesis is a list, providing synthesis information. Much easier to read, than table entry.

Author(s)

Fabien Gelineau <neonira@gmail.com>
Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to testFunction and to generateData.

Examples

# typical use case
op_sum <- opwf(sum, c('...','removeNA_b'))

rv_sum <- exploreSignatures(op_sum, list(... = c('im', 'r', 'cm')))

# which are the errors of exploration and in what context do they occur?
print(rv_sum$failure$synthesis)
# which are the good behaviors of exploration and in what context do they occur?
print(rv_sum$south$synthesis)

# Restrict signatures to use for exploration testing on op_sum
# Consider only two cases: no argument and ellipsis1_, ellipsis2_
cac_sum <- computeArgumentsCombination(op_sum)
rv_sum_f <- exploreSignatures(op_sum, list(... = c('im', 'r', 'cm'))),
cac_sum$signatures[c(1, 5)])

generateData

Generate Data

Description
Function to generate data.

Usage
generateData(function_f_1,
    argumentsTypeRestrictions_l = list(),
    replacementContext_l = setGenerationContext(),
    ellipsisReplacementContext_l = setGenerationContext(),
    defaultArgumentsContext_l = setDefaultArgumentsGenerationContext(),
    functionName_s_1 = deparse(substitute(function_f_1))
)

Arguments

function_f_1 a single R function, offensive programming ready, with using semantic argument names
argumentsTypeRestrictions_l
    a named list. Each name must match a function argument name. Each content must be a vector of strings, each of them matching a retrieveDataFactory()$getKnownSuffixes() known suffix.
replacementContext_l
    a generation context, as defined by setGenerationContext function, applicable to standard arguments of the function, if any.
ellipsisReplacementContext_l
    a generation context, as defined by setGenerationContext function, applicable to ...arguments of the function.
defaultArgumentsContext_l
    a generation context, as defined by setDefaultArgumentsGenerationContext function, applicable to default arguments of the function.
functionName_s_1
    A character vector of length 1, holding the function name. This is particularly useful, in scripts.
Details

Generate a driven aleatory set of data to be used as argument in a call to function `fun_f_1`. Generation is driven by the `argumentsTypeRestrictions_l` argument.

Value

A list with names `generation`, `data`, `context`, and `n`, representing respectively, argument name generation, data, data type generation context, and the number of data generated at the first level. Generated data are ready for use and accessible using the `data` name of the list.

See Also

Refer to `retrievedatafactory` and to `testFunction`.

Examples

```r
# typical example
op_sum <- opwf(sum, c('...','removeNA_b_1'))
op_sum_atr <- list('...'=c('i','d','c'))
ec <- setGenerationContext(0, TRUE, FALSE)
gd <- generateData(op_sum, op_sum_atr, ec, erc$hetero_vector[[1]], dac$none)
```

---

### opMetaTestingInformation

**Package functions information**

**Description**

A reminder of available functions from this package, and, most common usage intent. A poor man CLI cheat sheet.

**Usage**

```r
opMetaTestingInformation()
```

**Value**

A data.table with following column names

- `function` a function name provided by this package
- `elaboration` a boolean stating if function could/should be use for elaboration purpose
- `verification` a boolean stating if function could/should be use for verification purpose
- `exploitation` a boolean stating if function could/should be use for exploitation purpose
- `information` a boolean stating if function could/should be use for informative purpose
- `kind` the classification kind of the chosen function
- `user` a typical user role/profile that may use the function
- `domain` internal domain function belongs to
**Description**

Create an offensive programming function, wrapping a standard \( \text{R} \) function.

**Usage**

\[
\text{opwf}(\text{fun}_f_1, \text{parameterNames}_s, \text{functionName}_s_1 = \text{NA_character}_s)
\]

**Arguments**

- \( \text{fun}_f_1 \): a single \( \text{R} \) function
- \( \text{parameterNames}_s \): the new names of the parameter function, must be semantic argument names. Must be a bijection to real \( \text{fun}_f_1 \) argument names.
- \( \text{functionName}_s_1 \): A character vector of length 1, holding the function name. Default value, implies evaluation using `deparse(substitute(\text{fun}_f_1))`. This is particularly useful, in scripts.

**Details**

Arguments default values, if any, are managed, and should be correctly and automatically substituted.

**Value**

A \( \text{R} \) function which takes given \( \text{parameterNames}_s \) as arguments.

**Author(s)**

Fabien Gelineau <neonira@gmail.com>
Maintainer: Fabien Gelineau <neonira@gmail.com>
**qualifyFunctionArguments**

**See Also**
Refer to `testFunction`

**Examples**

```r
# typical example
op_sum <- opwf(sum, c('...','removeNA_b_1'))

# example with substituted argument in existing default valued arguments
op_append <- opwf(append, c('originalValues_','valuesToInsert_','afterIndex_ui_1'))
```

---

**Description**
Use method `qualifyFunctionArguments` to retrieve information about function arguments.

**Usage**

```r
qualifyFunctionArguments(fun_f_1)
```

**Arguments**

- `fun_f_1` A constrained vector of function values. Vector length must be 1.

**Value**
A list with following names:

- `argument_names` a character vector of all the function argument names
- `owns_ellipsis` a boolean. Is TRUE when ... belongs to argument names
- `symbol_names` a character vector of argument names that are symbols
- `symbol_indexes` the integer indexes of symbol names in the argument names
- `stripped_symbol_names` a character vector of argument names that are symbols, not considering ...
- `stripped_symbol_indexes` the integer indexes of stripped symbol names in the argument names
- `default_names` a character vector of argument names that owns default values
- `default_indexes` the integer indexes of default valued arguments names in the argument names
- `arguments` a pairList of argument names and values. See `formals`
Author(s)

Fabien Gelineau <neonira@gmail.com>
Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

# typical examples
qualifyFunctionArguments(Sys.Date)
qualifyFunctionArguments(cos)
qualifyFunctionArguments(sum)

retrieveDataFactory  Retrieve Data Factory

Description

As the data factory may be modified, this function allows you to make changes and to record them in your own specialized data generation factory, to match various needs and ease reuse.

Usage

retrieveDataFactory()

Details

Provides a data factory.

Retrieves a DataFactory from options variable op_mt_data_factory to consider your customized data factory.

Value

An R object that is a DataFactory.

Author(s)

Fabien Gelineau <neonira@gmail.com>
Maintainer: Fabien Gelineau <neonira@gmail.com>
### Examples

```r
#---- typical case ----

draw_integer_array_dim2 <- function(n, replace_b_1 = TRUE) {
  m <- n + sample(0:3, 1)
  matrix(seq(1, n * m), byrow = TRUE, nrow = n,
         dimnames = list(paste('row_', 1:n), paste('col_', 1:m)))
}

df <- retrieveDataFactory()
df$addSuffix('a', "array", draw_integer_array_dim2)

options(op_mt_data_factory = df)
fg <- retrieveDataFactory() # retrieves the user defined data factory
fg$getRecordedTypes()[suffix == 'a'] # right behavior !

# wrong behavior as retrieveDataFactory will provide the default factory and not yours!
options(op_mt_data_factory = NULL)
fh <- retrieveDataFactory() # retrieves the default factory
fh$getRecordedTypes()[suffix == 'a']
```

---

**setDefaultArgumentsGenerationContext**

*Set default arguments generation context.*

**Description**

Use method `setDefaultArgumentsGenerationContext` to set default arguments generation context.

**Usage**

```r
setDefaultArgumentsGenerationContext(useDefaultArguments_b_1 = TRUE,
                                      useAllDefaultArguments_b_1 = FALSE)
```

**Arguments**

- **useDefaultArguments_b_1**
  - A single boolean value to specify the usage of default arguments in generated function call

- **useAllDefaultArguments_b_1**
  - A single boolean value to specify usage of all default valued arguments in generated function call. Second argument is considered only when first argument is `TRUE`. 

Value

A list holding the provided values, allowing easy reuse either interactively or programmatically, accessible through names use, and use_all.

Predefined variables named default_arguments_context and dac hold most common definition cases, and therefore greatly eases use and simplify writing.

Author(s)

Fabien Gelineau <neonira@gmail.com>
Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```r
# a typical instanciation
mydgc <- list(
  setDefaultArgumentsGenerationContext(FALSE, FALSE),
  setDefaultArgumentsGenerationContext(TRUE, FALSE),
  setDefaultArgumentsGenerationContext(TRUE, TRUE)
)

# uses predefined variable
print(dac$partial)
```

---

**setGenerationContext**   *Set generation context.*

Description

Use this function to set generation context.

Usage

```r
setGenerationContext(replacementNumber_ui_1 = sample(0:3L, 1),
  homogeneousTypeReplacement_b_1 = FALSE,
  allowList_b_1 = TRUE,
  forceList_b_1 = FALSE)
```

Arguments

- **replacementNumber_ui_1**
  
  A single unsigned integer expressing the number of arguments to generate.

- **homogeneousTypeReplacement_b_1**
  
  A single boolean expressing willingness to replace chosen argument with same type arguments, or not. Useful when dealing with ....

- **allowList_b_1**
  
  A single boolean, expressing the desired result. When TRUE result is a list, a vector otherwise.

- **forceList_b_1**
  
  A single boolean, expressing the desire to get the result as a list.
Value

A list containing all the provided arguments, accessible through names homogeneous_type, number_replacements, and allow_list.

Predefined variables named established_replacement_context and erc hold most common definition cases, and therefore greatly eases use and simplify writing.

Author(s)

Fabien Gelineau <neonira@gmail.com>
Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

# a typical instanciation
egc <- list(
  setGenerationContext(homogeneous = TRUE),
  setGenerationContext(allowList = FALSE)
)

# uses predefined variable
print(erc$homo_vector[[2]])

testFunction

Test function

Description

Apply data to function signature and record results.

Usage

testFunction(function_f_1,
generatedData_l,
functionName_s_1 = deparse(substitute(function_f_1)))

Arguments

function_f_1 a single R function, offensive programming ready, with using semantic argument names

generatedData_l data to apply to the function. Could be generated by generateData function is desired.

functionName_s_1 A character vector of length 1, holding the function name. This is particularly useful, in scripts.
Details

Executes code and captures execution context and result, for posterior analysis.

Value

A list with names generation, data, context, and n, representing respectively, argument name generation, data, data type generation context, and the number of data generated at the first level.

Generated data are ready for use and accessible using the data name of the list.

See Also

Refer to opwf.

Examples

```r
# typical example
op_sum <- opwf(sum, c('...', 'removeNA_b_1'))
op_sum_atr <- list('...' = c('i', 'd', 'c'))
ec <- setGenerationContext(0, TRUE, FALSE)
gd <- generateData(op_sum, op_sum_atr, ec, erc$hetero_vector[[1]], dac$none)
tf <- testFunction(op_sum, gd$data)
```

usesSemanticArgumentNames

Uses semantic argument names.

Description

Determine if the given function uses semantic argument names.

Usage

```r
usesSemanticArgumentNames(fun_f_1)
```

Arguments

- `fun_f_1`: A single function

Value

A boolean value

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>
Examples

```r
f <- function(x_) x_

usesSemanticArgumentNames(f)
# TRUE

usesSemanticArgumentNames(sum)
# FALSE
```
Index

*Topic programation
  exploreSignatures, 4
  generateData, 6
  qualifyFunctionArguments, 9
  setDefaultArgumentsGenerationContext, 11
  setGenerationContext, 12
  usesSemanticArgumentNames, 14

*Topic programming
  buildSemanticArgumentName, 2
  computeArgumentsCombination, 3
  opMetaTestingInformation, 7
  opwf, 8
  retrieveDataFactory, 10
  testFunction, 13

*Topic utilities
  buildSemanticArgumentName, 2
  computeArgumentsCombination, 3
  exploreSignatures, 4
  generateData, 6
  opMetaTestingInformation, 7
  opwf, 8
  qualifyFunctionArguments, 9
  retrieveDataFactory, 10
  setDefaultArgumentsGenerationContext, 11
  setGenerationContext, 12
  testFunction, 13
  usesSemanticArgumentNames, 14

buildSemanticArgumentName, 2

computeArgumentsCombination, 3, 4

dac
  (setDefaultArgumentsGenerationContext), 11

default_arguments_context
  (setDefaultArgumentsGenerationContext), 11

erc (setGenerationContext), 12
established_replacement_context
  (setGenerationContext), 12
exploreSignatures, 4
formals, 3, 9
generateData, 5, 6, 13
offensiveProgrammingWrapFunction
  (opwf), 8
opMetaTestingInformation, 7
opwf, 4, 8, 14
qualifyFunctionArguments, 9
retrieveDataFactory, 7, 10
setDefaultArgumentsGenerationContext, 6, 11
setGenerationContext, 6, 12
testFunction, 3–5, 7, 9, 13
usesSemanticArgumentNames, 14