Package ‘kriens’

December 2, 2015

Type Package
Title Continuation Passing Style Development
Version 0.1
Date 2015-11-30
Author Matteo Provenzano
Maintainer Matteo Provenzano <matteo.provenzano@alephdue.com>
Description Provides basic functions for Continuation-Passing Style development.
License BSD_3_clause + file LICENSE
URL http://www.alephdue.com
LazyData TRUE
Suggests testthat
NeedsCompilation no
Repository CRAN
Date/Publication 2015-12-02 15:10:26

R topics documented:

compose ......................................................... 2
do .............................................................. 3
forget .......................................................... 4
identity2 ....................................................... 5
monoid ......................................................... 6
path ............................................................ 7

Index 8

1
**Description**

It allows to compose two functions of the form $f(x, \ ret)$ and $g(x, \ ret)$ returning a function $h(x, \ ret)$ which is the composition $f \circ g$. It implements the composition operator of the Continuation category.

The composition has the following properties:

1. **Associativity**: $h \circ (f \circ g) = (h \circ g) \circ f$
2. **Unity**: $f \circ \text{identity}2 = f = \text{identity}2 \circ f$

In order for these relations to hold, the function $f$ and $g$ must not deal with global mutable states.

**Usage**

```plaintext
compose(f, g)
```

**Arguments**

- $f$ : The first function that must be composed
- $g$ : The first function that must be composed

**Value**

Returns the composite function of $f$ and $g$

**Note**

The composition is performed from left to right i.e. such that the first function executed is $f$.

**Author(s)**

Matteo Provenzano

[http://www.alephdue.com](http://www.alephdue.com)

**See Also**

`forget`
do

Examples

# Example 1

# define an arrow in the Continuation category.
# this function applies the continuation to the
# increment of its argument and then decrements it.
one <- function(x, ret) {
  return(ret(x+1) - 1)
}

# define another arrow in the Continuation category.
two <- function(x, ret) {
  return(ret(2*x))
}

# create the composition
# this is exactly the same as one %>% two
composite <- compose(one, two)

# build the function (forget the continuation)
execute1 <- forget(composite)
execute1(1)
# returns 3

# Example 2
# compose the function further to loop over an array of elements
# lapply and sapply are already arrow in the Continuation category
loop <- compose(lapply, composite)

# build the function
execute2 <- forget(loop)
execute2(1:10)

---

**do**

Compose and Forget in one go.

Description

do allows to specify the list of function directly as its arguments. It return a function which is the composition of every argument with the continuation already forgotten.

Usage

do(...)

Arguments

... The functions that must be composed together.
Value

A function of the type $g(x)$ which can be directly used on the input.

Author(s)

Matteo Provenzano

http://www.alephdue.com

See Also

path, forget

Examples

```c
# define a function that doubles its argument
times.two <- function(x, ret) {
  ret(x*2)
}

# define a function that loops over a list of list and double every element
loop <- do(lapply, lapply, times.two)

#returns list(list(2, 4, 6), list(8,10,12))
loop(list(list(1,2,3),list(4,5,6)))
```

---

**Description**

This function takes a function of the form $f(x, \text{ret})$ and forgets the $\text{ret}$ part returning a function of the form $g(x)$.

Usage

`forget(f)`

Arguments

$f$ a function of the form $f(x, \text{ret})$.

Value

a function of the form $f(x)$.

Author(s)

Matteo Provenzano

http://www.alephdue.com
**identity2**

### See Also

compose

### Examples

```r
# forget the FUN part in lapply
to.list <- forget(lapply)

# returns the list of the natural numbers from 1 to 10
to.list(1:10)
```

---

**identity2**  
*The Identity Arrow*

---

### Description

The identity arrow for the Continuation category for which holds:  
\[ f \circ \text{identity2} = f = \text{identity2} \circ f \]

### Usage

```r
identity2(x, ret)
```

### Arguments

- **x**: The value on which the function operates
- **ret**: The following computation

### Value

This function always returns the original arrow.

### Author(s)

Matteo Provenzano

[http://www.alephdue.com](http://www.alephdue.com)
monoid  

*Description*

Creates the monoid binary operator for a monoid in the Continuation category.

*Usage*

```javascript
monoid(op)
```

*Arguments*

- `op`  
The binary operator to be inserted in the monoid (multiplication).

*Value*

It returns a function of the type `h(f, g)` where `f` and `g` must be elements of the monoid and objects in the Continuation category. The function `h` will return a function of the type `t(x, ret)` which can be used in the Continuation category.

*Note*

The developer must make sure that the function `f` and `g` are elements of a monoid and of the Continuation category. The developer must also ensure that the operator `op` is the monoid’s binary operator.

*Author(s)*

Matteo Provenzano  
http://www.alephdue.com

*References*

https://en.wikipedia.org/wiki/Monoid_(category_theory)

*See Also*

- `do`

*Examples*

```javascript
#   A list is a monoid
replicate.10 <- function(x, ret) {
    ret(rep(x, 10))
}

# concatenation is the binary operator for the list monoid
# the empty list is the unit
```
path

`
`%et` <- monoid(c)

replicate.20 <- do(replicate.10 `%et` replicate.10)

# returns a list of 20 "a"s
replicate.20("a")

---

**Description**

It applies the compose operator recursively on all the elements of the list provided as argument.

**Usage**

`path(fs)`

**Arguments**

- **fs**  
  The list of the functions that must be composed together (e.g: list(f1, f2, f3, ...)).

**Value**

A function of the type `g(x, ret)` result of the pairwise composition of each element in the list.

**Author(s)**

Matteo Provenzano  
http://www.alephdue.com

**Examples**

# define a function that doubles its argument
```r
times.two <- function(x, ret) {
  ret(x*2)
}
```

# define a function that loops over a list of list and double every element
```r
loop <- forget(path(list(lapply, lapply, times.two)))
```

# returns list(list(2, 4, 6), list(8,10,12))
```r
loop(list(list(1,2,3),list(4,5,6)))
```
Index

%. (compose), 2
compose, 2, 5
do, 3, 6
forget, 2, 4, 4
identity2, 5
monoid, 6
path, 4, 7