

Package ‘broadcast’

January 17, 2025

Title Simple Broadcasted Binary Operations for Atomic and Recursive Arrays with Minimal Dependencies

Version 0.0.0.9000

Description Implements simple broadcasted binary operations, for atomic and recursive arrays.

All operations are element-wise binary operations (i.e. involving only 2 arrays at a time).

Besides linking to 'Rcpp',

'broadcast' does not depend on, vendor, link to, or otherwise use any external libraries;

'broadcast' was essentially made from scratch and can be installed out-of-the-box.

The broadcasted implementations include, but are not limited to, the following.

- 1) Relational operators (like ``==``, ``!=``, ``<``, ``>``, ``<=``, ``>=``; also supports precision-based relational operators);
- 2) Boolean combiner operators (like ``&``, ``|``, ``xor()``, ``not-and``);
- 3) Arithmetic operators (like ``+``, ``-``, ``*``, ``/``, ``^``, ``pmin()``, ``pmax()``);
- 4) String distance and concatenation operators;
- 5) Broadcasted S4 implementations of ``ifelse()`` and ``vapply()``.

The broadcasted implementations strive to minimize computation time and memory usage (which is not just good for computer efficiency, but also for the environment).

License MPL-2.0 | file LICENSE

Encoding UTF-8

LinkingTo Rcpp

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Depends R (>= 4.2.0)

Imports Rcpp (>= 1.0.11)

Suggests tinytest, roxygen2

Contents

aaa00_broadcast_help	2
array_recycle	3
atomic_typecast	3
bc.b	5
bc.cplx	6
bc.list	7

bc.num	8
bc.str	9
bcapply	10
bc_dim	11
bc_ifelse	11
Index	13

aaa00_broadcast_help	<i>broadcast: Subset Methods as Alternatives to the Square Brackets Operators for Programming</i>
----------------------	---------------------------------------------------------------------------------------------------

Description

broadcast:
Simple broadcasted binary operations for atomic and recursive arrays in 'R'.

Details

The 'broadcast' package provides a set of type-specific functions for broadcasted operations. The functions use an API similar to the [outer](#) and [sweep](#) functions.

The following functions are available:

- [bc.num](#): Broadcasted operations for numeric (types int and dbl) arrays.
- [bc.bool](#): Broadcasted operations for logical/Boolean arrays.
- [bc.cplx](#): Broadcasted operations for complex arrays.
- [bc.str](#): Broadcasted operations for character arrays.
- [bc.list](#): Broadcasted operations for recursive arrays.

Each of these functions support 2 types of operations:

- regular operations, which return an array of the same type.
For example: +, -, *, /, etc.
- relational operations, which return a logical array.
For example: ==, !=, etc.

The 'broadcast' package supports relational operators (==, !=, <, >, <=, >=), logical combiners (&, |, xor, nand), arithmetic (+, -, *, /, ^)

Author(s)

Author, Maintainer: Tony Wilkes <tony_a_wilkes@outlook.com> ([ORCID](#))

References

The badges shown in the documentation of this R-package were made using the services of: <https://shields.io/>

array_recycle	<i>Recycle Array Dimensions</i>
---------------	---------------------------------

Description

The `array_recycle()` function recycles array dimensions until the specified dimension sizes are reached, and returns the array.

The various broadcasting functions "recycle" an array virtually, meaning little to no additional memory is needed.

The `array_recycle()` function, however, physically recycles an array (and thus actually occupies memory space).

Usage

```
array_recycle(x, tdim)
```

Arguments

<code>x</code>	an atomic or recursive array or matrix.
<code>tdim</code>	an integer vector, giving the target dimension to reach.

Value

Returns the recycled array.

Examples

```
x <- matrix(1:9, 3,3)
colnames(x) <- LETTERS[1:3]
rownames(x) <- letters[1:3]
names(x) <- month.abb[1:9]
print(x)

array_recycle(x, c(3,3,2)) # recycle to larger size
```

atomic_typecast	<i>Atomic Type Casting With Names and Dimensions Preserved</i>
-----------------	----------------------------------------------------------------

Description

Atomic type casting in R is generally performed using the functions [as.logical](#), [as.integer](#), [as.double](#), [as.character](#), [as.complex](#), and [as.raw](#).

Converting an object between atomic types using these functions strips the object of its attributes, including (dim)names and dimensions.

The functions provided here by the 'tinycodet' package preserve the dimensions, dimnames, and

names.

The functions are as follows:

- `as_bool()`: converts object to atomic type logical (TRUE, FALSE, NA).
- `as_int()`: converts object to atomic type integer.
- `as_dbl()`: converts object to atomic type double (AKA decimal numbers).
- `as_chr()`: converts object to atomic type character.
- `as_cplx()`: converts object to atomic type complex.
- `as_raw()`: converts object to atomic type raw.

Usage

```
as_bool(x, ...)
```

```
as_int(x, ...)
```

```
as_dbl(x, ...)
```

```
as_chr(x, ...)
```

```
as_cplx(x, ...)
```

```
as_raw(x, ...)
```

Arguments

<code>x</code>	vector, matrix, array (or a similar object where all elements share the same type).
<code>...</code>	further arguments passed to or from other methods.

Value

The converted object.

Examples

```
# matrix example ====
x <- matrix(sample(-1:28), ncol = 5)
colnames(x) <- month.name[1:5]
rownames(x) <- month.abb[1:6]
names(x) <- c(letters[1:20], LETTERS[1:10])
print(x)

as_bool(x)
as_int(x)
as_dbl(x)
as_chr(x)
as_cplx(x)
as_raw(x)
```

```
#####

# factor example ====
x <- factor(month.abb, levels = month.abb)
names(x) <- month.name
print(x)

as_bool(as_int(x) > 6)
as_int(x)
as_dbl(x)
as_chr(x)
as_cplx(x)
as_raw(x)
```

bc.b

*Broadcasted Operations for Logical Arrays***Description**

The `bc.b()` function performs broadcasted operations on 2 logical arrays.

Usage

```
bc.b(x, y, op)
```

Arguments

<code>x, y</code>	conformable atomic arrays of types logical, integer, or double.
<code>op</code>	a single string, giving the operator. Supported Boolean combiner operators: <code>&</code> , <code> </code> , <code>xor</code> , <code>nand</code> . Supported relational operators: <code>==</code> , <code>!=</code> , <code><</code> , <code>></code> , <code><=</code> , <code>>=</code> .

Value

For the boolean combiner operators:

A logical array as a result of the broadcasted arithmetic operation.

For relational operators:

A logical array as a result of the broadcasted relational comparison.

Examples

```
x.dim <- c(10:8)
x.len <- prod(x.dim)
x.data <- sample(c(TRUE, FALSE, NA), x.len, TRUE)
x <- array(x.data, x.dim)
y <- array(1:50, c(10,1,1))
```

```
bc.b(x, y, "&")
bc.b(x, y, "|")
bc.b(x, y, "xor")
bc.b(x, y, "nand")
```

```
bc.b(x, y, "==")
bc.b(x, y, "!=")
bc.b(x, y, "<")
bc.b(x, y, ">")
bc.b(x, y, "<=")
bc.b(x, y, ">=")
```

bc.cplx

Broadcasted Operations for Character/String Arrays

Description

The `bc.cplx()` function performs broadcasted operations on 2 complex arrays.

Note that `bc.cplx()` uses more strict NA checks than base 'R':

If for an element of either `x` or `y`, either the real or imaginary part is NA or NaN, then the result of the operation for that element is necessarily NA.

Usage

```
bc.cplx(x, y, op)
```

Arguments

<code>x, y</code>	conformable atomic arrays of type <code>complex</code> .
<code>op</code>	a single string, giving the operator. Supported arithmetic operators: <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> . Supported relational operators: <code>==</code> , <code>!=</code> .

Value

For arithmetic operators:

A complex array as a result of the broadcasted arithmetic operation.

For relational operators:

A logical array as a result of the broadcasted relational comparison.

Examples

```
x.dim <- c(10:8)
x.len <- prod(x.dim)
gen <- function() sample(c(rnorm(10), NA, NA, NaN, NaN, Inf, Inf, -Inf, -Inf))
x <- array(gen() + gen() * -1i, x.dim)
y <- array(gen() + gen() * -1i, c(10,1,1))

bc.cplx(x, y, "==")
bc.cplx(x, y, "!=")

bc.cplx(x, y, "+")

bc.cplx(array(gen() + gen() * -1i), array(gen() + gen() * -1i), "==")
bc.cplx(array(gen() + gen() * -1i), array(gen() + gen() * -1i), "!=")

x <- gen() + gen() * -1i
y <- gen() + gen() * -1i
out <- bc.cplx(array(x), array(y), "*")
cbind(x, y, x*y, out)
```

bc.list

Broadcasted Operations for Recursive Arrays

Description

The `bc.list()` function performs broadcasted operations on 2 Recursive arrays.

Usage

```
bc.list(x, y, f)
```

Arguments

<code>x, y</code>	conformable Recursive arrays (i.e. arrays of type <code>list</code>).
<code>f</code>	a function that takes in exactly 2 arguments, and returns a result that can be stored in a single element of a list.

Value

A recursive array.

Examples

```
x.dim <- c(c(10, 2,2))
x.len <- prod(x.dim)

gen <- function(n) sample(list(letters, month.abb, 1:10), n, TRUE)

x <- array(gen(10), x.dim)
y <- array(gen(10), c(10,1,1))
```

```
bc.list(
  x, y,
  \(x, y)c(length(x) == length(y), typeof(x) == typeof(y))
)
```

bc.num

*Broadcasted Operations for Numeric Arrays***Description**

The `bc.num()` function performs broadcasted operations on 2 numeric arrays.

Usage

```
bc.num(x, y, op, prec = sqrt(.Machine$double.eps))
```

Arguments

<code>x, y</code>	conformable atomic arrays of types <code>logical</code> , <code>integer</code> , or <code>double</code> .
<code>op</code>	a single string, giving the operator. Supported arithmetic operators: <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>^</code> , <code>intmod</code> , <code>pmin</code> , <code>pmax</code> . Supported relational operators: <code>==</code> , <code>!=</code> , <code><</code> , <code>></code> , <code><=</code> , <code>>=</code> , <code>d==</code> , <code>d!=</code> , <code>d<</code> , <code>d></code> , <code>d<=</code> , <code>d>=</code> .
<code>prec</code>	a single number between 0 and 0.1, giving the machine precision to use. Only relevant for the following operators: <code>d==</code> , <code>d!=</code> , <code>d<</code> , <code>d></code> , <code>d<=</code> , <code>d>=</code> See the <code>d==</code> , <code>d!=</code> , <code>d<</code> , <code>d></code> , <code>d<=</code> , <code>d>=</code> operators from the 'tinycodet' package for details.

Value

For arithmetic operators:

A numeric array as a result of the broadcasted arithmetic operation.

For relational operators:

A logical array as a result of the broadcasted relational comparison.

Examples

```
x.dim <- c(10:8)
x.len <- prod(x.dim)
x.data <- sample(c(NA, 1.1:1000.1), x.len, TRUE)
x <- array(x.data, x.dim)
y <- array(1:50, c(10,1,1))

bc.num(x, y, "+")
bc.num(x, y, "-")
```



```

bc.num(x, y, "*")
bc.num(x, y, "/" )
bc.num(x, y, "^")

bc.num(x, y, "==")
bc.num(x, y, "!=")
bc.num(x, y, "<")
bc.num(x, y, ">")
bc.num(x, y, "<=")
bc.num(x, y, ">=")

```

bc.str

Broadcasted Operations for Character/String Arrays

Description

The `bc.str()` function performs broadcasted operations on 2 character/string arrays.
`bc.chr()` is an alias for `bc.str()`.

Usage

```
bc.str(x, y, op)
```

```
bc.chr(x, y, op)
```

Arguments

<code>x, y</code>	conformable atomic arrays of type <code>character</code> .
<code>op</code>	a single string, giving the operator. Supported concatenation operators: <code>+</code> . Supported relational operators: <code>==</code> , <code>!=</code> . Supported distance operators: <code>levenshtein</code> .

Value

For concatenation operation:
A character array as a result of the broadcasted concatenation operation.

For relational operation:
A logical array as a result of the broadcasted relational comparison.

For distance operation:
An integer array as a result of the broadcasted relational comparison.

Examples

```

x.dim <- c(10:8)
x.len <- prod(x.dim)
x <- array(letters, x.dim)
y <- array(letters, c(10,1,1))

bc.str(x, y, "==")
bc.str(x, y, "!=")

bc.str(x, y, "+")

bc.str(array(letters), array(letters), "==")
bc.str(array(letters), array(letters), "!=")

```

bcapply*Apply a Function to 2 Broadcasted Arrays*

Description

The `bcapply()` function applies a function to 2 arrays with broadcasting.

Usage

```
bcapply(x, y, f)
```

Arguments

<code>x, y</code>	conformable atomic or recursive arrays.
<code>f</code>	a function that takes in exactly 2 arguments, and returns a result that can be stored in a single element of a recursive or atomic array.

Value

An atomic or recursive array with dimensions `bc_dim(x, y)`.

Examples

```

x.dim <- c(c(10, 2,2))
x.len <- prod(x.dim)

gen <- function(n) sample(list(letters, month.abb, 1:10), n, TRUE)

x <- array(gen(10), x.dim)
y <- array(gen(10), c(10,1,1))

bc.list(
  x, y,
  \(x, y) c(length(x) == length(y), typeof(x) == typeof(y))
)

```

bc_dim	<i>Predict Broadcasted dimensions</i>
--------	---------------------------------------

Description

bc_dim(x, y) gives the dimensions an array would have, as the result of an broadcasted binary element-wise operation between 2 arrays x and y.

Usage

```
bc_dim(x, y)
```

Arguments

x, y an atomic array or matrix.

Value

Returns the recycled array.

Examples

```
x.dim <- c(10:8)
x.len <- prod(x.dim)
x.data <- sample(c(TRUE, FALSE, NA), x.len, TRUE)
x <- array(x.data, x.dim)
y <- array(1:50, c(10,1,1))

dim(bc.b(x, y, "&")) == bc_dim(x, y)
dim(bc.b(x, y, "|")) == bc_dim(x, y)
```

bc_ifelse	<i>Broadcasted Ifelse</i>
-----------	---------------------------

Description

The bc_ifelse() S4 generic method performs a broadcasted form of [ifelse](#).

Usage

```
bc_ifelse(cond, yes, no)
```

Arguments

cond logical vector or array with the length equal to prod(bc_dim(yes, no)).

yes, no conformable arrays of the same type.

All atomic types (see [atomic](#)) are supported.

Recursive arrays of type list are also supported.

since bc_ifelse() is an S4 generic, it can be extended to support special array classes.

Value

The output, here referred to as `out`, will be an array of the same type as `yes` and `no`.

After broadcasting `yes` against `no`, given any element index `i`, the following will hold for the output:

- when `cond[i] == TRUE`, `out[i]` is `yes[i]`;
- when `cond[i] == FALSE`, `out[i]` is `no[i]`;
- when `cond[i]` is `NA`, `out[i]` is `NA` when `yes` and `no` are atomic, and `out[i]` is `list(NULL)` when `yes` and `no` are recursive.

Examples

```
x.dim <- c(c(10, 2,2))
x.len <- prod(x.dim)

gen <- function(n) sample(list(letters, month.abb, 1:10), n, TRUE)

x <- array(gen(10), x.dim)
y <- array(gen(10), c(10,1,1))

cond <- bc.list(
  x, y,
  \(x, y) c(length(x) == length(y) && typeof(x) == typeof(y))
) |> as_bool()

bc_ifelse(cond, yes = x, no = y)
```

Index

`aaa00_broadcast_help`, 2
`array_recycle`, 3
`as.character`, 3
`as.complex`, 3
`as.double`, 3
`as.integer`, 3
`as.logical`, 3
`as.raw`, 3
`as_bool (atomic_typecast)`, 3
`as_chr (atomic_typecast)`, 3
`as_cplx (atomic_typecast)`, 3
`as_dbl (atomic_typecast)`, 3
`as_int (atomic_typecast)`, 3
`as_raw (atomic_typecast)`, 3
`atomic`, 11
`atomic_typecast`, 3

`bc.b`, 5
`bc.chr (bc.str)`, 9
`bc.cplx`, 6
`bc.list`, 7
`bc.num`, 2, 8
`bc.str`, 9
`bc_dim`, 11
`bc_ifelse`, 11
`bcapply`, 10
`broadcast (aaa00_broadcast_help)`, 2
`broadcast-package`
 (`aaa00_broadcast_help`), 2
`broadcast_help (aaa00_broadcast_help)`, 2

`ifelse`, 11

`outer`, 2

`sweep`, 2