Package ‘sets’

December 12, 2017

Version 1.0-18

Title Sets, Generalized Sets, Customizable Sets and Intervals

Description Data structures and basic operations for ordinary sets,
generalizations such as fuzzy sets, multisets, and
fuzzy multisets, customizable sets, and intervals.

Depends R (>= 2.7.0)

Suggests proxy

Imports graphics,grDevices,stats,utils

License GPL-2

NeedsCompilation yes

Author David Meyer [aut, cre],
Kurt Hornik [aut],
Christian Buchta [ctb]

Maintainer David Meyer <David.Meyer@R-project.org>

Repository CRAN

Date/Publication 2017-12-12 06:47:44 UTC

R topics documented:

canonicalize set and mapping 2
closure 3
cset 4
fuzzy 8
fuzzydocs 11
fuzzyfuns 12
fuzzyinference 15
gset 17
interval 22
labels 26
options 27
outer 28
plot 29

set 31
similarity 34
tuple 36

Index **38**

`canonicalize set and mapping` *Canonicalize set and mapping*

Description

Helper function that canonicalizes set elements, and possibly reorders a given mapping accordingly.

Usage

```
canonicalize_set_and_mapping(x, mapping = NULL, margin = NULL)
```

Arguments

`x`	An object to be transformed into a set.
`mapping`	A list, array or data frame representing a mapping of the set.
`margin`	Margins to be reordered (ignored if `mapping` is a list). If `NULL`, all margins with the same length of `x` will be used.

Details

This helper function can be used when a set is to be created from some object `x`, and another object contains some meta-information on the set elements in the same order than the elements of `x`. The set creation can cause the input elements to be permuted. By the use of this function, the meta information can be kept in sync with the result of iterating over the associated set.

Value

A list with three named components:

`set`	The set created from `x`.
`mapping`	`mapping`, possibly reordered to match the order of `set`.
`order`	The order used for rearranging the mapping.

See Also

`set`.

Examples

```
L <- list(c, "a", 3)
M1 <- list("a","b","c")
M2 <- matrix(1:9, ncol = 3)
canonicalize_set_and_mapping(L, M1)
canonicalize_set_and_mapping(L, M2)
canonicalize_set_and_mapping(L, M2, 1)
```

---

closure *Closure and reduction*

Description

Closure and reduction of (g)sets.

Usage

```
## S3 method for class 'set'
closure(x, operation = c("union", "intersection"), ...)
binary_closure(x, operation = c("union", "intersection"))
## S3 method for class 'set'
reduction(x, operation = c("union", "intersection"), ...)
binary_reduction(x, operation = c("union", "intersection"))
```

Arguments

x	For `binary_closure` and `binary_reduction`: a binary matrix. A set of (g)sets otherwise.
operation	The set operation under which the closure or reduction shall be computed.
...	Currently not used.

Details

The closure of a set S under some operation OP contains all elements of S, and the results of OP applied to all element pairs of S.

The reduction of a set S under some operation OP is the minimal subset of S having the same closure than S under OP.

Note that the closure and reduction methods for sets are currently only implemented for sets of *(g)sets* (families) and will give an error for other cases.

`binary_closure` and `binary_reduction` interface efficient C code for computing closures and reductions of binary patterns. They are used by the high-level methods if `x` contains only objects of class `sets`.

Value

An object of same type than `x`.

Author(s)

The C code for binary closures is provided by Christian Buchta.

See Also

`set`, `gset`.

Examples

```
## ordinary set
s <- set(set(1),set(2),set(3))
(cl <- closure(s))
(re <- reduction(cl))
stopifnot(s == re)

(cl <- closure(s, "intersection"))
(re <- reduction(cl, "intersection"))
stopifnot(s == re)

## multi set
s <- set(gset(1,1),gset(2,2),gset(3,3))
(cl <- closure(s))
(re <- reduction(cl))
stopifnot(s == re)

## fuzzy set
s <- set(gset(1,1/3),gset(2,2/3),gset(3,3/3))
(cl <- closure(s))
(re <- reduction(cl))
stopifnot(s == re)

## fuzzy multiset
s <- set(gset(1,list(set(1,0.8))), gset(2, list(gset(1,3))), gset(3,0.3))
(cl <- closure(s))
(re <- reduction(cl))
stopifnot(s == re)
```

`cset` *Customizable sets*

Description

Creation and manipulation of customizable sets.

Usage

```
cset(gset,
     orderfun = sets_options("orderfun"),
     matchfun = sets_options("matchfun"))
```

```
cset_support(x)
cset_core(x, na.rm = FALSE)
cset_peak(x, na.rm = FALSE)
cset_height(x, na.rm = FALSE)
cset_memberships(x, filter = NULL)
cset_universe(x)
cset_bound(x)

cset_transform_memberships(x, FUN, ...)
cset_concentrate(x)
cset_dilate(x)
cset_normalize(x, height = 1)
cset_defuzzify(x,
               method = c("meanofmax", "smallestofmax",
                          "largestofmax", "centroid"))

matchfun(FUN)

cset_orderfun(x)
cset_matchfun(x)
cset_orderfun(x) <- value
cset_matchfun(x) <- value

as.cset(x)
is.cset(x)

cset_is_empty(x, na.rm = FALSE)
cset_is_subset(x, y, na.rm = FALSE)
cset_is_proper_subset(x, y, na.rm = FALSE)
cset_is_equal(x, y, na.rm = FALSE)
cset_contains_element(x, e)

cset_is_set(x, na.rm = FALSE)
cset_is_multiset(x, na.rm = FALSE)
cset_is_fuzzy_set(x, na.rm = FALSE)
cset_is_set_or_multiset(x, na.rm = FALSE)
cset_is_set_or_fuzzy_set(x, na.rm = FALSE)
cset_is_fuzzy_multiset(x)
cset_is_crisp(x, na.rm = FALSE)
cset_has_missings(x)

cset_cardinality(x, type = c("absolute", "relative"), na.rm = FALSE)
cset_union(...)
cset_mean(x, y, type = c("arithmetic", "geometric", "harmonic"))
cset_product(...)
cset_difference(...)
cset_intersection(...)
cset_symdiff(...)
```

```
cset_complement(x, y)
cset_power(x)
cset_cartesian(...)
cset_combn(x, m)

## S3 method for class 'cset'
cut(x, level = 1, type = c("alpha", "nu"), strict = FALSE, ...)
## S3 method for class 'cset'
mean(x, ..., na.rm = FALSE)

## S3 method for class 'cset'
## median(x, na.rm = FALSE, ...)     [R >= 3.4.0]
## median(x, na.rm)                  [R < 3.4.0]
## S3 method for class 'cset'
length(x)
## S3 method for class 'cset'
lengths(x, use.names = TRUE)
```

Arguments

`x`	For `as.cset()` and `is.cset()`: an R object. A (c)set object otherwise.
`y`	A (c)set object.
`gset`	A generalized set (or some other R object coercible to it).
`matchfun`	A function for matching similar elements, comparable to `match`, taking two arguments: `x` (vector of elements to be matched) and `table` (vector of elements to be matched against). The return value is an integer vector of the matching positions (or `NA` if there is no match). Note that the default behavior is to test for *identity*.
`FUN`	A predicate testing for equality of two objects.
`orderfun`	A function taking a list and returning an integer vector, specifying the order in which an iterator processes the set elements. Alternatively, the index vector can be specified directly.
`value`	A new match function (order function).
`type`	For `gset_cardinality()`: cardinality type (either `"absolute"` or `"relative"`). For `gset_mean()`: mean type (`"arithmetic"`, `"geometric"`, or `"harmonic"`). For `"cut"`: either `"alpha"` or `"nu"`.
`strict`	Logical indicating whether the cut level must be exceeded strictly ("greater than") or not ("greater than or equal").
`height`	Double from the unit interval for scaling memberships.
`e`	An object of class `element`.
`filter`	Optional vector of elements to be filtered.
`m`	Number of elements to choose.
`method`	Currently, only `"Jaccard"` is implemented.
`level`	The minimum membership level.

use.names	logical; should the names of x be used in the result?
na.rm	logical indicating whether NA values should be removed.
...	For cset_*foo*(): (c)set objects. For the mean and sort methods: additional parameters internally passed to mean and order, respectively. For gset_transform_memberships: further arguments passed to FUN. For cut: currently not used.

Details

Customizable sets extend generalized sets in two ways: First, users can control the way elements are matched, i.e., define equivalence classes of elements. Second, an order function (or permutation index) can be specified for each set for changing the order in which iterators such as as.list process the elements. The latter in particular influences the labeling and print methods for customizable sets.

The match function needs to be vectorized in a similar way than match. matchfun can be used to create such a function from a "simple" predicate testing for equality (such as, e.g., identical). Make sure, however, to create the same function only once.

Note that operations on customizable sets require the same match function for all sets involved. The order function can differ, but will then be stripped from the result.

sets_options can be used to conveniently switch the default match and/or order function if a number of cset objects need to be created.

References

D. Meyer and K. Hornik (2009), Generalized and customizable sets in R, *Journal of Statistical Software* **31**(2), 1–27. http://www.jstatsoft.org/v31/i02/

See Also

set for ("ordinary") sets, gset for generalized sets, cset_outer, and tuple for tuples ("vectors").

Examples

```
## default behavior of sets: matching of elements is very strict
## Note that on most systems, 3.3 - 2.2 != 1.1
x <- set("1", 1L, 1, 3.3 - 2.2, 1.1)
print(x)

y <- set(1, 1.1, 2L, "2")
print(y)
1L %e% y

set_union(x, y)
set_intersection(x, y)
set_complement(x, y)

## Now use the more sloppy match()-function.
## Note that 1 == "1" == 1L ...
X <- cset(x, matchfun = match)
print(X)
```

```
Y <- cset(y, matchfun = match)
print(Y)
1L %e% Y

cset_union(X, Y)
cset_intersection(X, Y)
cset_complement(X, Y)

## Same using all.equal().
## This is a non-vectorized predicate, so use matchfun
## to generate a vectorized version:
FUN <- matchfun(function(x, y) isTRUE(all.equal(x, y)))
X <- cset(x, matchfun = FUN)
print(X)
Y <- cset(y, matchfun = FUN)
print(Y)
1L %e% Y

cset_union(X, Y)
cset_intersection(X, Y)
cset_complement(X, Y)

### change default functions via set_option
sets_options("matchfun", match)
cset(x)
cset(y)

cset(1:3) <= cset(c(1,2,3))

### restore package defaults
sets_options("matchfun", NULL)

### customized order function
FUN <- function(x) order(as.character(x), decreasing = TRUE)
Z <- cset(letters[1:5], orderfun = FUN)
print(Z)
as.character(Z)

## converter for ordered factors keeps order
o <- ordered(c("a", "b", "a"), levels = c("b", "a"))
as.set(o)
as.cset(o)

## converter for other data types keep order if the elements are unique:
as.cset(c("A", "quick", "brown", "fox"))
as.cset(c("A", "quick", "brown", "fox", "quick"))
```

---

fuzzy *Fuzzy logic*

Description

Fuzzy Logic

Usage

```
fuzzy_logic(new, ...)
.N.(x)
.T.(x, y)
.S.(x, y)
.I.(x, y)
```

Arguments

`x, y`	Numeric vectors.
`new`	A character string specifying one of the available fuzzy logic “families” (see details).
`...`	optional parameters for the selected family.

Details

A call to `fuzzy_logic()` without arguments returns the currently set fuzzy logic, i.e., a named list with four components `N`, `T`, `S`, and `I` containing the corresponding functions for negation, conjunction (“t-norm”), disjunction (“t-conorm”), and residual implication (which may not be available).

The package provides several fuzzy logic *families*. A concrete fuzzy logic is selected by calling `fuzzy_logic` with a character string specifying the family name, and optional parameters. Let us refer to $N(x) = 1-x$ as the *standard* negation, and, for a t-norm T, let $S(x,y) = 1-T(1-x, 1-y)$ be the *dual* (or complementary) t-conorm. Available specifications and corresponding families are as follows, with the standard negation used unless stated otherwise.

`"Zadeh"` Zadeh’s logic with $T = \min$ and $S = \max$. Note that the minimum t-norm, also known as the Gödel t-norm, is the pointwise largest t-norm, and that the maximum t-conorm is the smallest t-conorm.

`"drastic"` the drastic logic with t-norm $T(x,y) = y$ if $x = 1$, x if $y = 1$, and 0 otherwise, and complementary t-conorm $S(x,y) = y$ if $x = 0$, x if $y = 0$, and 1 otherwise. Note that the drastic t-norm and t-conorm are the smallest t-norm and largest t-conorm, respectively.

`"product"` the family with the product t-norm $T(x,y) = xy$ and dual t-conorm $S(x,y) = x+y-xy$.

`"Lukasiewicz"` the Lukasiewicz logic with t-norm $T(x,y) = \max(0, x+y-1)$ and dual t-conorm $S(x,y) = \min(x+y, 1)$.

`"Fodor"` the family with Fodor’s *nilpotent minimum* t-norm given by $T(x,y) = \min(x,y)$ if $x+y > 1$, and 0 otherwise, and the dual t-conorm given by $S(x,y) = \max(x,y)$ if $x+y < 1$, and 1 otherwise.

`"Frank"` the family of Frank t-norms T_p, $p \geq 0$, which gives the Zadeh, product and Lukasiewicz t-norms for $p = 0$, 1, and ∞, respectively, and otherwise is given by $T(x,y) = \log_p(1 + (p^x - 1)(p^y - 1)/(p-1))$.

`"Hamacher"` the three-parameter family of Hamacher, with negation $N_\gamma(x) = (1-x)/(1+\gamma x)$, t-norm $T_\alpha(x,y) = xy/(\alpha + (1-\alpha)(x+y-xy))$, and t-conorm $S_\beta(x,y) = (x+y+(\beta-1)xy)/(1+\beta xy)$, where $\alpha \geq 0$ and $\beta, \gamma \geq -1$. This gives a deMorgan triple (for which $N(S(x,y)) = T(N(x), N(y))$ iff $\alpha = (1+\beta)/(1+\gamma)$. The parameters can be specified as `alpha`, `beta` and `gamma`, respectively. If α is not given, it is taken as $\alpha = (1+\beta)/(1+\gamma)$. The default values for β and γ are 0, so that by default, the product family is obtained.

The following parametric families are obtained by combining the corresponding families of t-norms with the standard negation.

`"Schweizer-Sklar"` the Schweizer-Sklar family T_p, $-\infty \leq p \leq \infty$, which gives the Zadeh (minimum), product and drastic t-norms for $p = -\infty$, 0, and ∞, respectively, and otherwise is given by $T_p(x,y) = \max(0, (x^p + y^p - 1)^{1/p})$.

`"Yager"` the Yager family T_p, $p \geq 0$, which gives the drastic and minimum t-norms for $p = 0$ and ∞, respectively, and otherwise is given by $T_p(x,y) = \max(0, 1 - ((1-x)^p + (1-y)^p)^{1/p})$.

`"Dombi"` the Dombi family T_p, $p \geq 0$, which gives the drastic and minimum t-norms for $p = 0$ and ∞, respectively, and otherwise is given by $T_p(x,y) = 0$ if $x = 0$ or $y = 0$, and $T_p(x,y) = 1/(1 + ((1/x - 1)^p + (1/y - 1)^p)^{1/p})$ if both $x > 0$ and $y > 0$.

`"Aczel-Alsina"` the family of t-norms T_p, $p \geq 0$, introduced by Aczél and Alsina, which gives the drastic and minimum t-norms for $p = 0$ and ∞, respectively, and otherwise is given by $T_p(x,y) = \exp(-(|\log(x)|^p + |\log(y)|^p)^{1/p})$.

`"Sugeno-Weber"` the family of t-norms T_p, $-1 \leq p \leq \infty$, introduced by Weber with dual t-conorms introduced by Sugeno, which gives the drastic and product t-norms for $p = -1$ and ∞, respectively, and otherwise is given by $T_p(x,y) = \max(0, (x+y-1+pxy)/(1+p))$.

`"Dubois-Prade"` the family of t-norms T_p, $0 \leq p \leq 1$, introduced by Dubois and Prade, which gives the minimum and product t-norms for $p = 0$ and 1, respectively, and otherwise is given by $T_p(x,y) = xy/\max(x,y,p)$.

`"Yu"` the family of t-norms T_p, $p \geq -1$, introduced by Yu, which gives the product and drastic t-norms for $p = -1$ and ∞, respectively, and otherwise is given by $T(x,y) = \max(0, (1+p)(x+y-1) - pxy)$.

By default, the Zadeh logic is used.

`.N.`, `.T.`, `.S.`, and `.I.` are dynamic functions, i.e., wrappers that call the corresponding function of the current fuzzy logic. Thus, the behavior of code using these functions will change according to the chosen logic.

References

C. Alsina, M. J. Frank and B. Schweizer (2006), *Associative Functions: Triangular Norms and Copulas*. World Scientific. ISBN 981-256-671-6.

J. Dombi (1982), A general class of fuzzy operators, the De Morgan class of fuzzy operators and fuzziness measures induced by fuzzy operators, *Fuzzy Sets and Systems* **8**, 149–163.

J. Fodor and M. Roubens (1994), *Fuzzy Preference Modelling and Multicriteria Decision Support*. Kluwer Academic Publishers, Dordrecht.

D. Meyer and K. Hornik (2009), Generalized and customizable sets in R, *Journal of Statistical Software* **31**(2), 1–27. http://www.jstatsoft.org/v31/i02/

B. Schweizer and A. Sklar (1983), *Probabilistic Metric Spaces*. North-Holland, New York. ISBN 0-444-00666-4.

Examples

```
x <- c(0.7, 0.8)
y <- c(0.2, 0.3)

## Use default family ("Zadeh")
.N.(x)
.T.(x, y)
.S.(x, y)
.I.(x, y)

## Switch family and try again
fuzzy_logic("Fodor")
.N.(x)
.T.(x, y)
.S.(x, y)
.I.(x, y)
```

`fuzzydocs` *Documents on Fuzzy Theory*

Description

Occurence of three terms (neural networks, fuzzy, and image) in 30 documents retrieved from a Japanese article data base on fuzzy theory and systems.

Usage

```
data("fuzzy_docs")
```

Format

`fuzzy_docs` is a list of 30 fuzzy multisets, representing the occurrence of the terms "neural networks", "fuzzy", and "image" in each document. Each term appears with up to three membership values representing weights, depending on whether the term occurred in the abstract (0.2), the keywords section (0.6), and/or the title (1). The first 12 documents concern neural networks, the remaining 18 image processing. In the reference, various clustering methods have been employed to recover the two groups in the data set.

Source

K. Mizutani, R. Inokuchi, and S. Miyamoto (2008), Algorithms of Nonlinear Document Clustering Based on Fuzzy Multiset Model, *International Journal of Intelligent Systems*, **23**, 176–198.

Examples

```
data(fuzzy_docs)

## compute distance matrix using Jaccard dissimilarity
d <- as.dist(set_outer(fuzzy_docs, gset_dissimilarity))

## apply hierarchical clustering (Ward method)
cl <- hclust(d, "ward")

## retrieve two clusters
cutree(cl, 2)

## -> clearly, the clusters are formed by docs 1--12 and 13--30,
## respectively.
```

---

fuzzyfuns *Fuzzy membership functions*

Description

Fuzzy membership and set creator functions.

Usage

```
charfun_generator(FUN, height = 1)
fuzzy_tuple(FUN = fuzzy_normal, n = 5, ...,
                 universe = NULL, names = NULL)
is.charfun_generator(x)

fuzzy_normal(mean = NULL, sd = 1, log = FALSE, height = 1, chop = 0)
fuzzy_two_normals(mean = NULL, sd = c(1,1), log = c(FALSE, FALSE),
                  height = 1, chop = 0)
fuzzy_bell(center = NULL, cross = NULL, slope = 4, height = 1, chop = 0)
fuzzy_sigmoid(cross = NULL, slope = 0.5, height = 1, chop = 0)
fuzzy_trapezoid(corners = NULL, height = c(1,1),
                return_base_corners = TRUE)
fuzzy_triangular(corners = NULL, height = 1,
                 return_base_corners = TRUE)
fuzzy_cone(center = NULL, radius = 2, height = 1,
           return_base_corners = TRUE)
fuzzy_pi3(mid = NULL, min = NULL, max = NULL, height = 1,
          return_base_corners = TRUE)
fuzzy_pi4(knots, height = 1, return_base_corners = TRUE)

fuzzy_normal_gset(mean = NULL, sd = 1, log = FALSE, height = 1,
                  chop = 0, universe = NULL)
fuzzy_two_normals_gset(mean = NULL, sd = c(1,1), log = c(FALSE, FALSE),
```

```
                      height = 1, chop = 0, universe = NULL)
fuzzy_bell_gset(center = NULL, cross = NULL, slope = 4, height = 1,
                chop = 0, universe = NULL)
fuzzy_sigmoid_gset(cross = NULL, slope = 0.5, height = 1,
                   chop = 0, universe = NULL)
fuzzy_trapezoid_gset(corners = NULL, height = c(1,1), universe = NULL,
                     return_base_corners = TRUE)
fuzzy_triangular_gset(corners = NULL, height = 1, universe = NULL,
                      return_base_corners = TRUE)
fuzzy_cone_gset(center = NULL, radius = 2, height = 1, universe = NULL,
                return_base_corners = TRUE)
fuzzy_pi3_gset(mid = NULL, min = NULL, max = NULL, height = 1,
              universe = NULL, return_base_corners = TRUE)
fuzzy_pi4_gset(knots, height = 1,
               universe = NULL, return_base_corners = TRUE)
```

Arguments

`x`	An R object.
`n`	Positive integer indicating the number of sets to be created, or integer vector of location parameters used to create the sets.
`FUN`	Function to be used for creating a membership function. Needs to be vectorized, i.e., is expected to take a vector of set elements and to return a vector of numeric values.
`height`	Numeric value in the unit interval specifying the height of the set resulting from applying the membership function to the universe, i.e., the maximum value to which the values will be scaled to.
`chop`	Threshold value below which the membership function is truncated, i.e., has a value of 0.
`center, mean`	Numeric mean value(s) used for the resulting membership function.
`sd`	Numeric scale factor(s) (standard deviation(s)) used for the resulting membership function.
`radius`	Double added/subtracted from `center` to get the base line corners of the cone.
`log`	Logical (vector), indicating whether normal or log-normal distributions should be used.
`cross`	Double indicating the cross-over point for the sigmoidal distribution.
`slope`	Double indicating the slope at the cross-over point.
`corners`	Double values (vector of length four) indicating the abscissas of the four corners of the resulting trapezoid.
`min, mid, max`	Doubles indicating the abscissas of the three spline knots (`min`, 0), (`mid`, `height`) and (`max`, 0) the curve passes through.
`knots`	Vector of doubles of length four indicating the abscissas of the spline knots the curve passes through.

`return_base_corners`
: Logical indicating whether membership grades for the base line corner elements should be returned as small values (`.Machine$double.eps`) instead of 0. Otherwise, if a set is created from the memberships, the corner elements would be removed from the set.

`universe`
: Universal set used for computing the memberships grades.

`names`
: optional character vector of component labels for the return value.

`...`
: Further arguments passed to `FUN`.

Details

These functions can be used to create sets with certain membership patterns.

The core functions are function *generators*, taking parameters and returning a corresponding fuzzy function (i.e., with values in the unit interval). All of them are normalized, i.e., scaled to have a maximum value of `height` (default: 1):

`fuzzy_normal` is simply based on `dnorm`.

`fuzzy_two_normals` returns a function composed of the left and right parts of two normal distributions (each normalized), with possibly different means and standard deviations.

`fuzzy_bell` returns a function defined as: $\frac{1}{\left(1+\left|\frac{x-c}{w}\right|\right)^{2s}}$ with center c, crossover points $c \pm w$, and slope at the crossover points of $\frac{s}{2w}$.

`fuzzy_sigmoid` yields a function whose values are computed as $\frac{1}{1+e^{s(c-x)}}$ with slope s at crossover point c.

`fuzzy_trapezoid` creates a function with trapezoidal shape, defined by four `corners` elements and two `height` values for the second and third corner (the heights of the first and fourth corner being fixed at 0).

`fuzzy_triangular` similar to the above with only three corners.

`fuzzy_cone` is a special case of `fuzzy_triangular`, defining an isosceles triangle with corners (element, membership degree): $(\texttt{center} - \texttt{radius}, 0)$, $(\texttt{center}, \texttt{height})$, and $(\texttt{center} + \texttt{radius}, 0)$.

`fuzzi_pi3` constructs a spline based on two quadratic functions, passing through the knot points $(\texttt{min}, 0)$, $(\texttt{mid}, \texttt{height})$ and $(\texttt{max}, 0)$.

`fuzzi_pi4` constructs a spline based on an S-shaped and a Z-shaped curve forming a π-shaped one, passing through the four knots $(\texttt{knots[1]}, 0)$, $(\texttt{knots[2]}, \texttt{height})$, $(\texttt{knots[3]}, \texttt{height})$, and $(\texttt{knots[4]}, 0)$.

`charfun_generator` takes a vectorized function as argument, returning a function normalized by `height`.

The `fuzzy_`*foo*`_gset` functions directly generate generalized sets from `fuzzy_`*foo*, using the values defined by `universe`, `sets_options("universe")`, or `seq(0, 20, by = 0.1)` (in that order, whichever is not `NULL`).

`fuzzy_tuple` generates a sequence of `n` sets based on any of the generating functions (except `fuzzy_trapezoid` and `fuzzy_triangular`). The chosen generating function `FUN` is called with `n` different values chosen along the `universe` passed to the first argument, thus varying the position or the resulting graph.

Value

For charfun_generator, a *generating function* taking an argument list of parameters, and returning a membership function, mapping elements to membership values (from of the unit interval).

For fuzzy_tuple, a tuple of n fuzzy sets.

For is.charfun_generator, a logical.

For fuzzy_*foo*_gset, a fuzzy set.

For the other functions, a membership function.

See Also

set, gset, and tuple for the set types, and plot.gset for the available plot functions.

Examples

```
## creating a fuzzy normal function
N <- fuzzy_normal(mean = 0, sd = 1)
N(-3:3)

## create a fuzzy set with it
gset(charfun = N, universe = -3:3)

## same using wrapper
fuzzy_normal_gset(universe = -3:3)

## creating a user-defined fuzzy function
fuzzy_poisson <- charfun_generator(dpois)
gset(charfun = fuzzy_poisson(10), universe = seq(0, 20, 2))

## creating a series of fuzzy normal sets
fuzzy_tuple(fuzzy_normal, 5)

## creating a series of fuzzy cones with specific locations
fuzzy_tuple(fuzzy_cone, c(2,3,7))
```

---

fuzzyinference *Fuzzy inference*

Description

Basic infrastructure for building and using fuzzy inference systems.

Usage

```
fuzzy_inference(system, values, implication = c("minimum", "product"))
fuzzy_rule(antecedent, consequent)
fuzzy_system(variables, rules)
fuzzy_partition(varnames, FUN = fuzzy_normal, universe = NULL, ...)
fuzzy_variable(...)
x %is% y
```

Arguments

`...`	For `fuzzy_variable`: named list of fuzzy sets (or membership functions from which the fuzzy sets will be created using the default universe). For `fuzy_partition`: further arguments passed to `FUN`.
`antecedent, consequent`	parts of an inference rule (see details).
`variables`	Set or tuple of fuzzy variables (note that tuples must be used if two variables have the same definition).
`rules`	Set of rules.
`system`	A fuzzy system.
`values`	Named list of input values to the system. The names must match the labels of the variable set.
`implication`	A vectorized function taking two arguments, or a character string indicating the parallel minimum or the product function.
`varnames`	Either a character vector of fuzzy category labels, to be used with the default locations, or a named numeric vector of locations.
`FUN`	Function generator for membership functions to be used for the fuzzy partition.
`universe`	Universal set used for computing the memberships grades.
`x`	The name of a fuzzy variable.
`y`	The name of a category, belonging to a fuzzy variable.

Details

These functions can be used to create simple fuzzy inference machines based on fuzzy ("linguistic") variables and fuzzy rules. This involves five steps:

1. Fuzzification of the input variables.
2. Application of fuzzy operators (AND, OR, NOT) in the antecedents of some given rules.
3. Implication from the antecedent to the consequent.
4. Aggregation of the consequents across the rules.
5. Defuzzification of the resulting fuzzy set.

Implication is based on either the minimum or the product. The evaluation of the logical expressions in the antecedents, as well as the aggregation of the evaluation result for each single rule, depends on the fuzzy logic currently set.

Value

For `fuzzy_inference`: a generalized set. For `fuzzy_rule` and `fuzzy_system`: an object of class `fuzzy_rule` and `fuzzy_system`, respectively. For `fuzzy_variable` and `fuzzy_partition`: an object of class `fuzzy_variable`, inheriting from `tuple`.

See Also

`set` and `gset` for the set types, `fuzzy_tuple` for available fuzzy functions, and `fuzzy_logic` on the behavior of the implemented fuzzy operators.

Examples

```
## set universe
sets_options("universe", seq(from = 0, to = 25, by = 0.1))

## set up fuzzy variables
variables <-
set(service =
    fuzzy_partition(varnames =
                    c(poor = 0, good = 5, excellent = 10),
                    sd = 1.5),
    food =
    fuzzy_variable(rancid =
                   fuzzy_trapezoid(corners = c(-2, 0, 2, 4)),
                   delicious =
                   fuzzy_trapezoid(corners = c(7, 9, 11, 13))),
    tip =
    fuzzy_partition(varnames =
                    c(cheap = 5, average = 12.5, generous = 20),
                    FUN = fuzzy_cone, radius = 5)
    )

## set up rules
rules <-
set(
    fuzzy_rule(service %is% poor || food %is% rancid,
               tip %is% cheap),
    fuzzy_rule(service %is% good,
               tip %is% average),
    fuzzy_rule(service %is% excellent || food %is% delicious,
               tip %is% generous)
    )

## combine to a system
system <- fuzzy_system(variables, rules)
print(system)
plot(system) ## plots variables

## do inference
fi <- fuzzy_inference(system, list(service = 3, food = 8))

## plot resulting fuzzy set
plot(fi)

## defuzzify
gset_defuzzify(fi, "centroid")

## reset universe
sets_options("universe", NULL)
```

---

gset *Generalized sets*

Description

Creation and manipulation of generalized sets.

Usage

```
gset(support, memberships, charfun, elements, universe, bound)
as.gset(x)
is.gset(x)
gset_support(x)
gset_core(x, na.rm = FALSE)
gset_peak(x, na.rm = FALSE)
gset_height(x, na.rm = FALSE)
gset_universe(x)
gset_bound(x)

gset_memberships(x, filter = NULL)
gset_transform_memberships(x, FUN, ...)
gset_concentrate(x)
gset_dilate(x)
gset_normalize(x, height = 1)
gset_defuzzify(x,
               method = c("meanofmax", "smallestofmax",
                          "largestofmax", "centroid"))

gset_is_empty(x, na.rm = FALSE)
gset_is_subset(x, y, na.rm = FALSE)
gset_is_proper_subset(x, y, na.rm = FALSE)
gset_is_equal(x, y, na.rm = FALSE)
gset_contains_element(x, e)

gset_is_set(x, na.rm = FALSE)
gset_is_multiset(x, na.rm = FALSE)
gset_is_fuzzy_set(x, na.rm = FALSE)
gset_is_set_or_multiset(x, na.rm = FALSE)
gset_is_set_or_fuzzy_set(x, na.rm = FALSE)
gset_is_fuzzy_multiset(x)
gset_is_crisp(x, na.rm = FALSE)
gset_has_missings(x)

gset_cardinality(x, type = c("absolute", "relative"), na.rm = FALSE)
gset_union(...)
gset_sum(...)
gset_difference(...)
gset_product(...)
gset_mean(x, y, type = c("arithmetic", "geometric", "harmonic"))
gset_intersection(...)
gset_symdiff(...)
gset_complement(x, y)
```

```
gset_power(x)
gset_cartesian(...)
gset_combn(x, m)

e(x, memberships = 1L)
is_element(e)

## S3 method for class 'gset'
cut(x, level = 1, type = c("alpha", "nu"), strict = FALSE, ...)
## S3 method for class 'gset'
mean(x, ..., na.rm = FALSE)

## S3 method for class 'gset'
## median(x, na.rm = FALSE, ...)     [R >= 3.4.0]
## median(x, na.rm)                  [R < 3.4.0]
## S3 method for class 'gset'
length(x)
## S3 method for class 'gset'
lengths(x, use.names = TRUE)
```

Arguments

x	For `e()`, `as.gset()` and `is.gset()`: an R object. A (g)set object otherwise. `gset_memberships()` also accepts tuple objects.
y	A (g)set object.
e	An object of class `element`.
filter	Optional vector of elements to be filtered.
m	Number of elements to choose.
support	A set of elements giving the support of the gset (elements with non-zero memberships). Must be a subset of the universe, if specified.
memberships	For an ("ordinary") set: 1L (or simply missing). For a fuzzy set: a value between 0 and 1. For a multiset: a positive integer. For a fuzzy multiset: a list of multisets with elements from the unit interval (or a list of vectors interpreted as such). Otherwise, the argument will be transformed using `as.gset`.
elements	A set (or list) of `e` objects which are object/memberships-pairs.
charfun	A function taking an object and returning the membership.
bound	Integer used to compute the absolute complement for (fuzzy) multisets. If `NULL`, defaults to the value of `sets_options("bound")`. If the latter is also `NULL`, the maximum multiplicity will be used in computations.
FUN	A function, to be applied to a membership vector.
type	For `gset_cardinality()`: cardinality type (either `"absolute"` or `"relative"`). For `gset_mean()`: mean type (`"arithmetic"`, `"geometric"`, or `"harmonic"`). For `"cut"`: either `"alpha"` or `"nu"`.
strict	Logical indicating whether the cut level must be exceeded strictly ("greater than") or not ("greater than or equal").

height	Double from the unit interval for scaling memberships.
universe	An optional set of elements. If NULL, defaults to the value of sets_options("universe"). If the latter is also NULL, the support will be used in computations.
method	"centroid" computes the arithmetic mean of the set elements, using the membership values as weights. "smallestofmax" / "meanofmax" / "largestofmax" returns the minimum/mean/maximum of all set elements with maximal membership degree.
level	The minimum membership level.
na.rm	logical indicating whether NA values should be removed.
use.names	logical; should the names of x be used in the result?
...	For gset_*foo*(): (g)set objects. For the mean and sort methods: additional parameters internally passed to mean and order, respectively. For gset_transform_memberships: further arguments passed to FUN. For cut: currently not used.

Details

These functions represent basic infrastructure for handling *generalized* sets of general (R) objects.

A generalized set (or gset) is set of pairs (e, f), where e is some set element and f is the characteristic (or membership) function. For ("ordinary") sets f maps to $\{0, 1\}$, for fuzzy sets into the unit interval, for multisets into the natural numbers, and for fuzzy multisets f maps to the set of multisets over the unit interval.

The gset_is_*foo*() predicates are vectorized. In addition to the methods defined, one can use the following operators: | for the union, & for the intersection, + for the sum, - for the difference, %D% for the symmetric difference, * and ^*n* for the (n-fold) cartesian product, 2^ for the power set, %e% for the element-of predicate, < and <= for the (proper) subset predicate, > and >= for the (proper) superset predicate, and == and != for (in)equality. The Summary methods do also work if defined for the set elements. The mean and median methods try to convert the object to a numeric vector before calling the default methods. set_combn returns the gset of all subsets of specified length.

gset_support, gset_core, and gset_peak return the set of elements with memberships greater than zero, equal to one, and equal to the maximum membership, respectively. gset_memberships returns the membership vector(s) of a given (tuple of) gset(s), optionally restricted to the elements specified by filter. gset_height returns only the largest membership degree. gset_cardinality computes either the absolute or the relative cardinality, i.e. the memberships sum, or the absolute cardinality divided by the number of elements, respectively. The length method for gsets gives the (absolute) cardinality. The lengths method coerces the set to a list before applying the length method on its elements. gset_transform_memberships applies function FOO to the membership vector of the supplied gset and returns the transformed gset. The transformed memberships are guaranteed to be in the unit interval. gset_concentrate and gset_dilate are convenience functions, using the square and the square root, respectively. gset_normalize divides the memberships by their maximum and scales with height. gset_product (gset_mean) of some gsets compute the gset with the corresponding memberships multiplied (averaged).

The cut method provides both α- and ν-cuts. α-cuts "filter" all elements with memberships greater than (or equal to) level—the result, thus, is a crisp (multi)set. ν-cuts select those elements with a *multiplicity* exceeding level (only sensible for (fuzzy) multisets).

Because set elements are unordered, it is not allowed to use *positional* indexing. However, it is possible to do indexing using element labels or simply the elements themselves (useful, e.g., for subassignment). In addition, it is possible to iterate over *all* elements using `for` and `lapply`/`sapply`.

`gset_contains_element` is vectorized in `e`, that is, if `e` is an atomic vector or list, the is-element operation is performed element-wise, and a logical vector returned. Note that, however, objects of class `tuple` are taken as atomic objects to correctly handle sets of tuples.

References

D. Meyer and K. Hornik (2009), Generalized and customizable sets in R, *Journal of Statistical Software* **31**(2), 1–27. http://www.jstatsoft.org/v31/i02/

See Also

`set` for "ordinary" sets, `gset_outer`, and `tuple` for tuples ("vectors").

Examples

```
## multisets
(A <- gset(letters[1:5], memberships = c(3, 2, 1, 1, 1)))
(B <- gset(c("a", "c", "e", "f"), memberships = c(2, 2, 1, 2)))
rep(B, 2)
gset_memberships(tuple(A, B), c("a","c"))

gset_union(A, B)
gset_intersection(A, B)
gset_complement(A, B)

gset_is_multiset(A)
gset_sum(A, B)
gset_difference(A, B)

## fuzzy sets
(A <- gset(letters[1:5], memberships = c(1, 0.3, 0.8, 0.6, 0.2)))
(B <- gset(c("a", "c", "e", "f"), memberships = c(0.7, 1, 0.4, 0.9)))
cut(B, 0.5)
A * B
A <- gset(3L, memberships = 0.5, universe = 1:5)
!A

## fuzzy multisets
(A <- gset(c("a", "b", "d"),
         memberships = list(c(0.3, 1, 0.5), c(0.9, 0.1),
                            gset(c(0.4, 0.7), c(1, 2)))))
(B <- gset(c("a", "c", "d", "e"),
         memberships = list(c(0.6, 0.7), c(1, 0.3), c(0.4, 0.5), 0.9)))
gset_union(A, B)
gset_intersection(A, B)
gset_complement(A, B)

## other operations
mean(gset(1:3, c(0.1,0.5,0.9)))
```

```
median(gset(1:3, c(0.1,0.5,0.9)))
```

---

interval *Intervals*

Description

Interval class for countable and uncountable numeric sets.

Usage

```
interval(l=NULL, r=l,
         bounds=c("[]", "[)", "(]", "()", "[[", "]]", "][",
                  "open", "closed", "left-open", "right-open",
                  "left-closed", "right-closed"),
         domain=NULL)

reals(l=NULL, r=NULL,
      bounds=c("[]", "[)", "(]", "()", "[[", "]]", "][",
               "open", "closed", "left-open", "right-open",
               "left-closed", "right-closed"))
integers(l=NULL, r=NULL)
naturals(l=NULL, r=NULL)
naturals0(l=NULL, r=NULL)
l %..% r

interval_domain(x)

as.interval(x)
integers2reals(x, min=-Inf, max=Inf)
reals2integers(x)

interval_complement(x, y=NULL)
interval_intersection(...)
interval_symdiff(...)
interval_union(...)

interval_difference(...)
interval_division(...)
interval_product(...)
interval_sum(...)

is.interval(x)
interval_contains_element(x, y)
interval_is_bounded(x)
interval_is_closed(x)
interval_is_countable(...)
```

```
interval_is_degenerate(x)
interval_is_empty(x)
interval_is_equal(x, y)
interval_is_less_than_or_equal(x, y)
interval_is_less_than(x, y)
interval_is_greater_than_or_equal(x, y)
interval_is_greater_than(x, y)
interval_is_finite(x)
interval_is_half_bounded(x)
interval_is_left_bounded(x)
interval_is_left_closed(x)
interval_is_left_open(...)
interval_is_left_unbounded(x)
interval_measure(x)
interval_is_proper(...)
interval_is_proper_subinterval(x, y)
interval_is_right_bounded(x)
interval_is_right_closed(x)
interval_is_right_open(...)
interval_is_right_unbounded(x)
interval_is_subinterval(x, y)
interval_is_unbounded(x)
interval_is_uncountable(x)
interval_power(x, n)
x %<% y
x %>% y
x %<=% y
x %>=% y
```

Arguments

`x`	For `as.interval()` and `is.interval()`: an R object. For all other functions: an interval object (or any other R object coercible to one).
`y`	An interval object (or any other R object coercible to one).
`min, max`	Integers defining the range to be coerced.
`l, r`	Numeric values defining the bounds of the interval. For integer domains, these will be rounded.
`bounds`	Character string specifying whether the interval is open, closed, or left/right-open/closed. Symbolic shortcuts such as `"()"` or `"]["` for an open interval, etc., are also accepted.
`domain`	Character string specifying the domain of the interval: `"R"`, `"Z"`, `"N"`, and `"N0"` for the reals, integers, positive integers and non-negative integers, respectively. If unspecified, the domain will be guessed from the mode of the numeric values specifying the bounds.
`n`	Integer exponent.
`...`	Interval objects (or other R objects coercible to interval objects).

Details

An interval object represents a multi-interval, i.e., a union of disjoint, possibly unbounded (i.e., infinite) ranges of numbers—either the extended reals, or sequences of integers. The usual set operations (union, complement, intersection) and predicates (equality, (proper) inclusion) are implemented. If (numeric) sets and interval objects are mixed, the result will be an interval object. Some basic interval arithmetic operations (addition, subtraction, multiplication, division, power) as well mathematical functions (`log`, `log2`, `log10`, `exp`, `abs`, `sqrt`, `trunc`, `round`, `floor`, `ceiling`, `signif`, and the trigonometric functions) are defined. Note that the rounding functions will discretize the interval.

Coercion methods for the `as.numeric`, `as.list`, and `as.set` generics are implemented. `reals2integers()` discretizes a real multi-interval. `integers2reals()` returns a multi-interval of corresponding (degenerate) real intervals.

The summary functions `min`, `max`, `range`, `sum`, `mean` and `prod` are implemented and work on the interval bounds.

`sets_options()` allows to change the style of open bounds according to the ISO 31-11 standard using reversed brackets instead of round parentheses (see examples).

Value

For the predicates: a logical value. For all other functions: an interval object.

See Also

`set` and `gset` for *finite* (generalized) sets.

Examples

```
#### * general interval constructor

interval(1,5)
interval(1,5, "[)")
interval(1,5, "()")

## ambiguous notation -> use alternative style
sets_options("openbounds", "][")
interval(1,5, "()")
sets_options("openbounds", "()")

interval(1,5, domain = "Z")
interval(1L, 5L)

## degenerate interval
interval(3)

## empty interval
interval()

#### * reals
reals()
reals(1,5)
```

```
reals(1,5,"()")
reals(1) ## half-unbounded

## (auto-)complement
!reals(1,5)
interval_complement(reals(1,5), reals(2, Inf))

## combine/c(reals(2,4), reals(3,5))
reals(2,4) | reals(3,5)

## intersection
reals(2,4) & reals(3,5)

## overlapping intervals
reals(2,4) & reals(3,5)
reals(2,4) & reals(4,5,"(]")

## non-overlapping
reals(2,4) & reals(7,8)
reals(2,4) | reals(7,8)
reals(2,4,"[)") | reals(4,5,"(]")

## degenerated cases
reals(2,4) | interval()
c(reals(2,4), set())

reals(2,4) | interval(6)
c(reals(2,4), set(6), 9)

## predicates
interval_is_empty(interval())
interval_is_degenerate(interval(4))
interval_is_bounded(reals(1,2))
interval_is_bounded(reals(1,Inf)) ## !! FALSE, because extended reals
interval_is_half_bounded(reals(1,Inf))
interval_is_left_bounded(reals(1,Inf))
interval_is_right_unbounded(reals(1,Inf))
interval_is_left_closed(reals(1,Inf))
interval_is_right_closed(reals(1,Inf)) ## !! TRUE

reals(1,2) <= reals(1,5)
reals(1,2) < reals(1,2)
reals(1,2) <= reals(1,2,"[)")
reals(1,2,"[)") < reals(1,2)

#### * integers
integers()
naturals()
naturals0()

3 %..% 5
integers(3, 5)
integers(3, 5) | integers(6,9)
```

```
integers(3, 5) | integers(7,9)

interval_complement(naturals(), integers())

naturals() <= naturals0()
naturals0() <= integers()

## mix reals and integers
c(reals(2,5), integers(7,9))
interval_complement(reals(2,5), integers())
interval_complement(integers(2,5), reals())

try(interval_complement(integers(), reals()), silent = TRUE)
## infeasible --> error

integers() <= reals()
reals() <= integers()

### interval arithmetic
x <- interval(2,4)
y <- interval(3,6)
x + y
x - y
x * y
x / y

## summary functions
min(x, y)
max(y)
range(y)
mean(y)
```

---

labels *Labels from objects*

Description

Creates “nice” labels from objects.

Usage

```
LABELS(x, max_width = NULL, dots = "...", unique = FALSE,
       limit = NULL, ...)
LABEL(x, limit = NULL, ...)
## S3 method for class 'character'
LABEL(x, limit = NULL, quote = sets_options("quote"), ...)
```

Arguments

`x`	For `LABELS`, a vector of R objects (if the object is not a vector, it is converted using `as.list`). For `LABEL`, an R object.
`max_width`	Integer vector (recycled as needed) specifying the maximum label width for each component of `x`. If `NULL`, there is no limit, otherwise, the label will be truncated to `max_width`.
`dots`	A character string appended to a truncated label. If `NULL`, nothing is appended.
`unique`	Logical indicating whether `make.unique` should be called on the final result.
`limit`	Maximum length of vectors or sets to be represented as is. Longer elements will be replaced by a label.
`quote`	Should character strings be quoted, or not? (default: `TRUE`)
`...`	Optional arguments passed to the `LABEL` methods.

Value

A character vector of labels generated from the supplied object(s). `LABELS` first checks whether the object has names and uses these if any; otherwise, `LABEL` is called for each element to generate a "short" representation.

`LABEL` is generic to allow user extensions. The current methods return the result of `format` if the argument is of length 1 (for objects of classes `set` and `tuple`: by default of length 5), and create a simple class information otherwise.

Examples

```
LABELS(list(1, "test", X = "1", 1:5))
LABELS(set(X = as.tuple(1:20), "test", list(list(list(1,2)))))
LABELS(set(pair(1,2), set("a", 2), as.tuple(1:10)))
LABELS(set(pair(1,2), set("a", 2), as.tuple(1:10)), limit = 11)
```

`options` *Options for the 'sets' package*

Description

Function for getting and setting options for the **sets** package.

Usage

```
sets_options(option, value)
```

Arguments

`option`	character string indicating the option to get or set (see details). If missing, all options are returned as a list.
`value`	Value to be set. If omitted, the current value is returned.

Details

Currently, the following options are available:

"quote": logical specifying whether labels for character elements are quoted or not (default: TRUE).

"hash": logical specifying whether set elements are hashed or not (default: TRUE).

"matchfun": the default matching function for cset (default: NULL).

"orderfun": the default ordering function for cset (default: NULL).

"universe": the default universe for generalized sets (default: NULL).

See Also

cset

Examples

```
sets_options()
sets_options("quote", TRUE)
print(set("a"))
sets_options("quote", FALSE)
print(set("a"))
```

---

outer — *Outer Product of Sets (Tuples)*

Description

Outer "product" of (g)sets (tuples).

Usage

```
set_outer(X, Y, FUN = "*", ..., SIMPLIFY = TRUE, quote = FALSE)
gset_outer(X, Y, FUN = "*", ..., SIMPLIFY = TRUE, quote = FALSE)
cset_outer(X, Y, FUN = "*", ..., SIMPLIFY = TRUE, quote = FALSE)
tuple_outer(X, Y, FUN = "*", ..., SIMPLIFY = TRUE, quote = FALSE)
```

Arguments

X, Y	Set (tuple) objects or vectors. If Y is omitted, X will be used instead. In this case, FUN can also be specified as Y for convenience.
FUN	A function or function name (character string).
SIMPLIFY	Logical. If TRUE and all return values of FUN are atomic and of length 1, the result will be an atomic matrix; otherwise, a recursive one (a list with dim attribute).
quote	logical indicating whether the character strings used for the row and column names of the returned matrix should be quoted.
...	Additional arguments passed to the FUN.

Details

This function applies FUN to all pairs of elements specified in X and Y. Basically intended as a replacement for outer for sets (tuples), it will also accept any vector for X and Y. The return value will be a matrix of dimension length(X) times length(Y), atomic or recursive depending on the complexity of FUN's return type and the SIMPLIFY argument.

See Also

set, tuple, outer.

Examples

```
set_outer(set(1,2), set(1,2,3), "/")
X <- set_outer(set(1,2), set(1,2,3), pair)
X[[1,1]]
Y <- set_outer(set(1,2), set(1,2,3), set)
Y[[1,1]]
set_outer(2 ^ set(1,2,3), set_is_subset)

tuple_outer(pair(1,2), triple(1,2,3))
tuple_outer(1:5, 1:4, "^")
```

---

plot *Plot functions for generalized sets*

Description

Plot and lines functions for (tuples of) generalized sets and function generators of characteristic functions.

Usage

```
## S3 method for class 'gset'
plot(x, type = NULL, ylim = NULL,
         xlab = "Universe", ylab = "Membership Grade", ...)
## S3 method for class 'cset'
plot(x, ...)
## S3 method for class 'set'
plot(x, ...)
## S3 method for class 'tuple'
plot(x, type = "l", ylim = NULL,
         xlab = "Universe", ylab = "Membership Grade", col = 1,
         continuous = TRUE, ...)
## S3 method for class 'charfun_generator'
plot(x, universe = NULL, ...)

## S3 method for class 'gset'
lines(x, type = "l", col = 1, continuous = TRUE,
```

```
         universe = NULL, ...)
## S3 method for class 'cset'
lines(x, ...)
## S3 method for class 'set'
lines(x, ...)
## S3 method for class 'tuple'
lines(x, col = 1, universe = NULL, ...)
## S3 method for class 'charfun_generator'
lines(x, universe = NULL, ...)
```

Arguments

`x`	For a method for class *foo*, an object of class *foo*.
`type`	Same as the `type` argument of `plot`. For `plot.gset` and `plot.cset`, `"barplot"` can also be used.
`universe`	Universal set used for setting up the plot region. By default, this is deduced from the object(s) to be plotted.
`col`	Character or integer vector specifying the color of the object(s) to be plotted.
`continuous`	Logical indicating whether zero membership degrees “inside” the graph should be ignored.
`xlab, ylab`	Character labels for the axes.
`ylim`	Double vector of length 2 defining the range of the y axis.
`...`	Further arguments passed to the default plot methods.

Value

The main argument (invisibly).

See Also

`set`, `gset`, and `tuple` for the set types, and `fuzzy_normal` for available characteristic functions.

Examples

```
## basic plots
plot(gset(1:3, 1:3/3))
plot(gset(1:3, 1:3/3, universe = 0:4))
plot(gset(c("a", "b"), list(1:2/2, 0.3)))

## characteristic functions
plot(fuzzy_normal)
plot(tuple(fuzzy_normal, fuzzy_bell), col = 1:2)
plot(fuzzy_pi3_gset(min = 2, max = 15))

## superposing plots using lines()
x <- fuzzy_normal_gset()
y <- fuzzy_trapezoid_gset(corners = c(5, 10, 15, 17), height = c(0.7, 1))
plot(tuple(x, y))
```

```
lines(x | y, col = 2)
lines(x & y, col = 3)

## another example using gset_mean
x <- fuzzy_two_normals_gset(sd = c(2, 1))
y <- fuzzy_trapezoid_gset(corners = c(5, 9, 11, 15))
plot(tuple(x, y))
lines(tuple(gset_mean(x, y),
            gset_mean(x, y, "geometric"),
            gset_mean(x, y, "harmonic")),
      col = 2:4)

## creating a sequence of sets
plot(fuzzy_tuple(fuzzy_cone, 10), col = gray.colors(10))
```

---

set *Sets*

Description

Creation and manipulation of sets.

Usage

```
set(...)
as.set(x)
make_set_with_order(x)
is.set(x)

set_is_empty(x)
set_is_subset(x, y)
set_is_proper_subset(x, y)
set_is_equal(x, y)
set_contains_element(x, e)

set_union(...)
set_intersection(...)
set_symdiff(...)
set_complement(x, y)
set_cardinality(x)
## S3 method for class 'set'
length(x)
## S3 method for class 'set'
lengths(x, use.names = TRUE)
set_power(x)
set_cartesian(...)
set_combn(x, m)
```

Arguments

x	For as.set() and is.set(): an R object. A set object otherwise.
y	A set object.
e	An R object.
m	Number of elements to choose.
use.names	logical; should the names of x be used in the result?
...	For set(): R objects, and set objects otherwise.

Details

These functions represent basic infrastructure for handling sets of general (R) objects. The set_is_*foo*() predicates are vectorized. In addition to the methods defined, one can use the following operators: | for the union, - for the difference (or complement), & for the intersection, %D% for the symmetric difference, * and ^*n* for the (n-fold) cartesian product, 2^ for the power set, %e% for the element-of predicate, < and <= for the (proper) subset predicate, > and >= for the (proper) superset predicate, and == and != for (in)equality. The length method for sets gives the cardinality. The lengths method coerces the set to a list before applying the length method on its elements. set_combn returns the set of all subsets of specified length. The Summary methods do also work if defined for the set elements. The mean and median methods try to convert the object to a numeric vector before calling the default methods.

Because set elements are unordered, it is not allowed to use *positional* indexing. However, it is possible to do indexing using element labels or simply the elements themselves (useful, e.g., for subassignment). In addition, it is possible to iterate over *all* elements using for and lapply/sapply.

Note that converting objects to sets may change the internal order of the elements, so that iterating over the original data might give different results than iterating over the corresponding set. The permutation can be obtained using the generic function make_set_with_order, returning both the set and the ordering. as.set simply calls make_set_with_order internally and strips the order information, so user-defined methods for coercion have to be provided for the latter and not for as.set.

Note that set_union, set_intersection, and set_symdiff accept any number of arguments. The n-ary symmetric difference of sets contains just elements which are in an odd number of the sets.

set_contains_element is vectorized in e, that is, if e is an atomic vector or list, the is-element operation is performed element-wise, and a logical vector returned. Note that, however, objects of class tuple are taken as atomic objects to correctly handle sets of tuples.

Value

For the predicate functions, a logical. For make_set_with_order, a list with two components "set" and "order". For set_cardinality and the length method, an integer value. For the lengths method, an integer vector. For all others, a set.

References

D. Meyer and K. Hornik (2009), Generalized and customizable sets in R, *Journal of Statistical Software* **31**(2), 1–27. http://www.jstatsoft.org/v31/i02/

See Also

set_outer, gset for generalized sets, and tuple for tuples (“vectors”).

Examples

```
## constructor
s <- set(1L, 2L, 3L)
s

## named elements
snamed <- set(one = 1, 2, three = 3)
snamed

## indexing by label
snamed[["one"]]

## subassignment
snamed[c(2,3)] <- c("a","b")
snamed

## a more complex set
set(c, "test", list(1, 2, 3))

## converter
s2 <- as.set(2:5)
s2

## converter with order
make_set_with_order(5:1)

## set of sets
set(set(), set(1))

## cartesian product
s * s2
s * s
s ^ 2 # same as above
s ^ 3

## power set
2 ^ s

## tuples
s3 <- set(tuple(1,2,3), tuple(2,3,4))
s3

## Predicates:

## element
1:2 %e% s
tuple(1,2,3) %e% s3
```

```
## subset
s <= s2
s2 >= s # same

## proper subset
s < s

## complement, union, intersection, symmetric difference:
s - 1L
s + set("a") # or use: s | set("a")
s & s
s %D% s2
set(1,2,3) - set(1,2)
set_intersection(set(1,2,3), set(2,3,4), set(3,4,5))
set_union(set(1,2,3), set(2,3,4), set(3,4,5))
set_symdiff(set(1,2,3), set(2,3,4), set(3,4,5))

## subsets:
set_combn(as.set(1:3),2)

## iterators:
sapply(s, sqrt)
for (i in s) print(i)

## Summary methods
sum(s)
range(s)

## mean / median
mean(s)
median(s)

## cardinality
s <- set(1, list(1, 2))
length(s)
lengths(s)
```

`similarity` *Similarity and Dissimilarity Functions*

Description

Similarities and dissimilarities for (generalized) sets.

Usage

```
set_similarity(x, y, method = "Jaccard")
gset_similarity(x, y, method = "Jaccard")
cset_similarity(x, y, method = "Jaccard")
```

```
set_dissimilarity(x, y,
                  method = c("Jaccard", "Manhattan", "Euclidean",
                             "L1", "L2"))
gset_dissimilarity(x, y,
                   method = c("Jaccard", "Manhattan", "Euclidean",
                              "L1", "L2"))
cset_dissimilarity(x, y,
                   method = c("Jaccard", "Manhattan", "Euclidean",
                              "L1", "L2"))
```

Arguments

`x, y`	Two (generalized/customizable) sets.
`method`	Character string specifying the proximity method (see below).

Details

For two generalized sets X and Y, the `Jaccard` similarity is $|X \cap Y|/|X \cup Y|$ where $|\cdot|$ denotes the cardinality for generalized sets (sum of memberships). The `Jaccard` dissimilarity is 1 minus the similarity.

The `L1` (or `Manhattan`) and `L2` (or `Euclidean`) dissimilarities are defined as follows. For two fuzzy multisets A and B on a given universe X with elements x, let $M_A(x)$ and $M_B(x)$ be functions returning the memberships of an element x in sets A and B, respectively. The memberships are returned in *standard form*, i.e. as an infinite vector of decreasing membership values, e.g. $(1, 0.3, 0, 0, \ldots)$. Let $M_A(x)_i$ and $M_B(x)_i$ denote the ith components of these membership vectors. Then the L1 distance is defined as:

$$d_1(A,B) = \sum_{x \in X} \sum_{i=1} \infty |M_A(x)_i - M_B(x)_i|$$

and the L2 distance as:

$$d_2(A,B) = \sqrt{\sum_{x \in X} \sum_{i=1} \infty |M_A(x)_i - M_B(x)_i|^2}$$

Value

A numeric value (similarity or dissimilarity, as specified).

Source

T. Matthe, R. De Caluwe, G. de Tre, A. Hallez, J. Verstraete, M. Leman, O. Cornelis, D. Moelants, and J. Gansemans (2006), Similarity Between Multi-valued Thesaurus Attributes: Theory and Application in Multimedia Systems, *Flexible Query Answering Systems*, Lecture Notes in Computer Science, Springer, 331–342.

K. Mizutani, R. Inokuchi, and S. Miyamoto (2008), Algorithms of Nonlinear Document Clustering Based on Fuzzy Multiset Model, *International Journal of Intelligent Systems*, **23**, 176–198.

See Also

set.

Examples

```
A <- set("a", "b", "c")
B <- set("c", "d", "e")
set_similarity(A, B)
set_dissimilarity(A, B)

A <- gset(c("a", "b", "c"), c(0.3, 0.7, 0.9))
B <- gset(c("c", "d", "e"), c(0.2, 0.4, 0.5))
gset_similarity(A, B, "Jaccard")
gset_dissimilarity(A, B, "Jaccard")
gset_dissimilarity(A, B, "L1")
gset_dissimilarity(A, B, "L2")

A <- gset(c("a", "b", "c"), list(c(0.3, 0.7), 0.1, 0.9))
B <- gset(c("c", "d", "e"), list(0.2, c(0.4, 0.5), 0.8))
gset_similarity(A, B, "Jaccard")
gset_dissimilarity(A, B, "Jaccard")
gset_dissimilarity(A, B, "L1")
gset_dissimilarity(A, B, "L2")
```

---

tuple *Tuples*

Description

Creation and manipulation of tuples.

Usage

```
tuple(...)
as.tuple(x)
is.tuple(x)
singleton(...)
pair(...)
triple(...)
tuple_is_singleton(x)
tuple_is_pair(x)
tuple_is_triple(x)
tuple_is_ntuple(x, n)
```

Arguments

x	An R object.
n	A non-negative integer.

... Possibly named R objects (for `singleton`, `pair`, and `triple` exactly one, two, and three, respectively.)

Details

These functions represent basic infrastructure for handling tuples of general (R) objects. Class `tuple` is used in particular to correctly handle cartesian products of sets. Although tuple objects should behave like "ordinary" vectors, some operations might yield unexpected results since tuple objects are in fact list objects internally. The `Summary` methods do work if defined for the set elements. The `mean` and `median` methods try to convert the object to a numeric vector before calling the default methods.

See Also

`set`.

Examples

```
## Constructor.
tuple(1,2,3, TRUE)
triple(1,2,3)
pair(Name = "David", Height = 185)
tuple_is_triple(triple(1,2,3))
tuple_is_ntuple(tuple(1,2,3,4), 4)

## Converter.
as.tuple(1:3)

## Operations.
c(tuple("a","b"), 1)
tuple(1,2,3) * tuple(2,3,4)
rep(tuple(1,2,3), 2)
min(tuple(1,2,3))
sum(tuple(1,2,3))
```

Index

∗Topic **datasets**
fuzzydocs, 11
∗Topic **math**
canonicalize set and mapping, 2
closure, 3
cset, 4
fuzzy, 8
fuzzyfuns, 12
fuzzyinference, 15
gset, 17
interval, 22
labels, 26
options, 27
outer, 28
plot, 29
set, 31
similarity, 34
tuple, 36
.I. (fuzzy), 8
.N. (fuzzy), 8
.S. (fuzzy), 8
.T. (fuzzy), 8
%..% (interval), 22
%<=% (interval), 22
%<% (interval), 22
%>=% (interval), 22
%>% (interval), 22
%D% (gset), 17
%e% (gset), 17
%is% (fuzzyinference), 15

as.character.interval (interval), 22
as.cset (cset), 4
as.double.interval (interval), 22
as.gset (gset), 17
as.interval (interval), 22
as.list.interval (interval), 22
as.set (set), 31
as.tuple (tuple), 36

binary_closure (closure), 3
binary_reduction (closure), 3

canonicalize set and mapping, 2
canonicalize_set_and_mapping (canonicalize set and mapping), 2
charfun_generator (fuzzyfuns), 12
closure, 3
cset, 4, *28*
cset_bound (cset), 4
cset_cardinality (cset), 4
cset_cartesian (cset), 4
cset_charfun (cset), 4
cset_combn (cset), 4
cset_complement (cset), 4
cset_concentrate (cset), 4
cset_contains_element (cset), 4
cset_core (cset), 4
cset_defuzzify (cset), 4
cset_difference (cset), 4
cset_dilate (cset), 4
cset_dissimilarity (similarity), 34
cset_has_missings (cset), 4
cset_height (cset), 4
cset_intersection (cset), 4
cset_is_crisp (cset), 4
cset_is_empty (cset), 4
cset_is_equal (cset), 4
cset_is_fuzzy_multiset (cset), 4
cset_is_fuzzy_set (cset), 4
cset_is_multiset (cset), 4
cset_is_proper_subset (cset), 4
cset_is_set (cset), 4
cset_is_set_or_fuzzy_set (cset), 4
cset_is_set_or_multiset (cset), 4
cset_is_subset (cset), 4
cset_matchfun (cset), 4
cset_matchfun<- (cset), 4
cset_mean (cset), 4

cset_memberships (cset), 4
cset_normalize (cset), 4
cset_orderfun (cset), 4
cset_orderfun<- (cset), 4
cset_outer, *7*
cset_outer (outer), 28
cset_peak (cset), 4
cset_power (cset), 4
cset_product (cset), 4
cset_similarity (similarity), 34
cset_sum (cset), 4
cset_support (cset), 4
cset_symdiff (cset), 4
cset_transform_memberships (cset), 4
cset_union (cset), 4
cset_universe (cset), 4
cut.cset (cset), 4
cut.gset (gset), 17

dnorm, *14*

e (gset), 17

format, *27*
fuzzy, 8
fuzzy_bell (fuzzyfuns), 12
fuzzy_bell_gset (fuzzyfuns), 12
fuzzy_cone (fuzzyfuns), 12
fuzzy_cone_gset (fuzzyfuns), 12
fuzzy_docs (fuzzydocs), 11
fuzzy_inference (fuzzyinference), 15
fuzzy_logic, *16*
fuzzy_logic (fuzzy), 8
fuzzy_normal, *30*
fuzzy_normal (fuzzyfuns), 12
fuzzy_normal_gset (fuzzyfuns), 12
fuzzy_partition (fuzzyinference), 15
fuzzy_pi3 (fuzzyfuns), 12
fuzzy_pi3_gset (fuzzyfuns), 12
fuzzy_pi4 (fuzzyfuns), 12
fuzzy_pi4_gset (fuzzyfuns), 12
fuzzy_rule (fuzzyinference), 15
fuzzy_sigmoid (fuzzyfuns), 12
fuzzy_sigmoid_gset (fuzzyfuns), 12
fuzzy_system (fuzzyinference), 15
fuzzy_trapezoid (fuzzyfuns), 12
fuzzy_trapezoid_gset (fuzzyfuns), 12
fuzzy_triangular (fuzzyfuns), 12
fuzzy_triangular_gset (fuzzyfuns), 12
fuzzy_tuple, *16*
fuzzy_tuple (fuzzyfuns), 12
fuzzy_two_normals (fuzzyfuns), 12
fuzzy_two_normals_gset (fuzzyfuns), 12
fuzzy_variable (fuzzyinference), 15
fuzzydocs, 11
fuzzyfuns, 12
fuzzyinference, 15

gset, *4*, *7*, *15*, *16*, 17, *24*, *30*, *33*
gset_bound (gset), 17
gset_cardinality (gset), 17
gset_cartesian (gset), 17
gset_charfun (gset), 17
gset_combn (gset), 17
gset_complement (gset), 17
gset_concentrate (gset), 17
gset_contains_element (gset), 17
gset_core (gset), 17
gset_defuzzify (gset), 17
gset_difference (gset), 17
gset_dilate (gset), 17
gset_dissimilarity (similarity), 34
gset_has_missings (gset), 17
gset_height (gset), 17
gset_intersection (gset), 17
gset_is_crisp (gset), 17
gset_is_empty (gset), 17
gset_is_equal (gset), 17
gset_is_fuzzy_multiset (gset), 17
gset_is_fuzzy_set (gset), 17
gset_is_multiset (gset), 17
gset_is_proper_subset (gset), 17
gset_is_set (gset), 17
gset_is_set_or_fuzzy_set (gset), 17
gset_is_set_or_multiset (gset), 17
gset_is_subset (gset), 17
gset_mean (gset), 17
gset_memberships (gset), 17
gset_normalize (gset), 17
gset_outer, *21*
gset_outer (outer), 28
gset_peak (gset), 17
gset_power (gset), 17
gset_product (gset), 17
gset_similarity (similarity), 34
gset_sum (gset), 17
gset_support (gset), 17
gset_symdiff (gset), 17

gset_transform_memberships (gset), 17
gset_union (gset), 17
gset_universe (gset), 17

identical, 7
integers (interval), 22
integers2reals (interval), 22
interval, 22
interval_complement (interval), 22
interval_contains_element (interval), 22
interval_difference (interval), 22
interval_division (interval), 22
interval_domain (interval), 22
interval_intersection (interval), 22
interval_is_bounded (interval), 22
interval_is_closed (interval), 22
interval_is_countable (interval), 22
interval_is_degenerate (interval), 22
interval_is_empty (interval), 22
interval_is_equal (interval), 22
interval_is_finite (interval), 22
interval_is_greater_than (interval), 22
interval_is_greater_than_or_equal (interval), 22
interval_is_half_bounded (interval), 22
interval_is_left_bounded (interval), 22
interval_is_left_closed (interval), 22
interval_is_left_open (interval), 22
interval_is_left_unbounded (interval), 22
interval_is_less_than (interval), 22
interval_is_less_than_or_equal (interval), 22
interval_is_proper (interval), 22
interval_is_proper_subinterval (interval), 22
interval_is_right_bounded (interval), 22
interval_is_right_closed (interval), 22
interval_is_right_open (interval), 22
interval_is_right_unbounded (interval), 22
interval_is_subinterval (interval), 22
interval_is_unbounded (interval), 22
interval_is_uncountable (interval), 22
interval_measure (interval), 22
interval_power (interval), 22
interval_product (interval), 22
interval_sum (interval), 22
interval_symdiff (interval), 22
interval_union (interval), 22
is.charfun_generator (fuzzyfuns), 12
is.cset (cset), 4
is.gset (gset), 17
is.interval (interval), 22
is.set (set), 31
is.tuple (tuple), 36
is_element (gset), 17

LABEL (labels), 26
LABELS (labels), 26
labels, 26
length.cset (cset), 4
length.gset (gset), 17
length.set (set), 31
lengths.cset (cset), 4
lengths.gset (gset), 17
lengths.set (set), 31
lines.charfun_generator (plot), 29
lines.cset (plot), 29
lines.gset (plot), 29
lines.set (plot), 29
lines.tuple (plot), 29

make.unique, 27
make_set_with_order (set), 31
match, 6, 7
matchfun (cset), 4
max.interval (interval), 22
mean, 7, 20, 32, 37
mean.cset (cset), 4
mean.gset (gset), 17
mean.interval (interval), 22
median, 20, 32, 37
median.cset (cset), 4
median.gset (gset), 17
min.interval (interval), 22

naturals (interval), 22
naturals0 (interval), 22

options, 27
order, 7, 20
outer, 28, 29

pair (tuple), 36
plot, 29, 30
plot.charfun_generator (plot), 29
plot.cset (plot), 29

plot.gset, *15*
plot.gset (plot), 29
plot.set (plot), 29
plot.tuple (plot), 29
prod.interval (interval), 22

range.interval (interval), 22
reals (interval), 22
reals2integers (interval), 22
reduction (closure), 3

set, *2*, *4*, *7*, *15*, *16*, *21*, *24*, *27*, *29*, *30*, 31, *36*, *37*
set_cardinality (set), 31
set_cartesian (set), 31
set_combn (set), 31
set_complement (set), 31
set_contains_element (set), 31
set_dissimilarity (similarity), 34
set_intersection (set), 31
set_is_empty (set), 31
set_is_equal (set), 31
set_is_proper_subset (set), 31
set_is_subset (set), 31
set_outer, *33*
set_outer (outer), 28
set_power (set), 31
set_similarity (similarity), 34
set_symdiff (set), 31
set_union (set), 31
sets_options (options), 27
similarity, 34
singleton (tuple), 36
sum.interval (interval), 22

triple (tuple), 36
tuple, *7*, *15*, *21*, *27*, *29*, *30*, *33*, 36
tuple_is_ntuple (tuple), 36
tuple_is_pair (tuple), 36
tuple_is_singleton (tuple), 36
tuple_is_triple (tuple), 36
tuple_outer (outer), 28