Package 'fsr'

July 5, 2022

Type Package

Title Handling Fuzzy Spatial Data

Version 1.0.2

URL https://accarniel.github.io/fsr/, https://github.com/accarniel/fsr

BugReports https://github.com/accarniel/fsr/issues

Depends R (>= 3.6.0)

Imports rlang (>= 0.4.11), methods (>= 2.0.0), sf (>= 0.9.4), FuzzyR (>= 2.3.0), dplyr (>= 1.0.6), ggplot2 (>= 3.3.5), stringr (>= 1.4.0), tibble (>= 3.0.1), pso (>= 1.0.3), e1071 (>= 1.7.3), utils (>= 3.6.3), lwgeom (>= 0.2.6)

Description Support for fuzzy spatial objects, their operations, and fuzzy spatial inference models based on Spatial Plateau Algebra.
It employs fuzzy set theory and fuzzy logic as foundation to deal with spatial fuzziness.
It implements underlying concepts defined in the following research papers:
(i) ``Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types'' <doi:10.1109/FUZZ-IEEE.2018.8491565>;
(ii) ``A Systematic Approach to Creating Fuzzy Region Objects from Real Spatial Data Sets'' <doi:10.1109/FUZZ-IEEE.2019.8858878>;
(iii) ``Fuzzy Inference on Fuzzy Spatial Objects (FIFUS) for Spatial Decision Support Systems'' <doi:10.1109/FUZZ-IEEE.2017.8015707>.

License GPL-3

RoxygenNote 7.2.0

NeedsCompilation no

Encoding UTF-8

Collate 'fsr_base.R' 'builder_functions.R' 'fsi_functions.R' 'utility_functions.R' 'spa_functions.R'

Author Anderson Carniel [rth, aut, cre] (<https://orcid.org/0000-0002-8297-9894>),
Felippe Galdino [rtm, aut] (<https://orcid.org/0000-0003-2594-9733>),
Juliana Philippsen [rtm, aut],
Markus Schneider [rth]

Maintainer Anderson Carniel <accarniel@ufscar.br>

Repository CRAN

Date/Publication 2022-07-05 02:50:02 UTC

R topics documented:

as_tibble.pgeometry *Converting a* pgeometry *object into tabular data*

Description

We can convert a pgeometry object into tabular data, such as a tibble or data.frame object, where the components of the pgeometry object compose the rows of the table.

Usage

```
## S3 method for class 'pgeometry'
as.data.frame(x, ...)

## S3 method for class 'pgeometry'
as_tibble(x, ...)
```

Arguments

x	A pgeometry object.
...	<dynamic-dots> Unused.

Details

This function is an interface for the S3 generic as_tibble. Here, it turns a pgeometry object into a tibble, which is a data frame with class tbl_df. This allows us to get the internal components of the pgeometry object (i.e., spatial features objects and membership degrees) as a data frame with two separate columns - called md (*membership degree*) and geometry (an sfc object).

For each component of the pgeometry object, as_tibble gets the md and geometry values and allocates them into a row of the new created tibble, in separated columns. Therefore, each row of this tibble represents a component of the original pgeometry object.

It is also possible to call the S3 method as.data.frame to convert a pgeometry object into tabular data.

Value

A tibble object of size n x 2 where n is the number of components of the pgeometry object and two columns in the format (md, geometry).

Examples

```
library(sf)

# Creating components for our plateau point object
v1 <- rbind(c(1,2), c(3,4))
v2 <- rbind(c(1,4), c(2,3),c(4,4))
```

```
md1 <- 0.2
md2 <- 0.1
md3 <- 0.4
pts1 <- rbind(c(1, 2), c(3, 2))
pts2 <- rbind(c(1, 1), c(2, 3), c(2, 1))
pts3 <- rbind(c(2, 2), c(3, 3))

comp1 <- component_from_sfg(st_multipoint(pts1), md1)
comp2 <- component_from_sfg(st_multipoint(pts2), md2)
comp3 <- component_from_sfg(st_multipoint(pts3), md3)

# Creating the plateau point object as a pgeometry object with 3 components

plateau_point <- create_pgeometry(list(comp1, comp2, comp3), "PLATEAUPOINT")

# Converting the pgeometry object into a tibble object
plateau_point_tibble <- as_tibble(plateau_point)

plateau_point_tibble
```

`component-class` *An S4 Class for representing a component of a spatial plateau object*

Description

An S4 Class for representing a component of a spatial plateau object

Details

A `component` object is composed of two attributes. The first one is a crisp spatial object and the second one a membership degree in]0, 1] of this `component`.

Slots

`obj` An `sfg` object.

`md` The membership degree of the component.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

create_empty_pgeometry

Creation of an empty pgeometry *object*

Description

This function builds an empty pgeometry object of a specific type.

Usage

```
create_empty_pgeometry(type)
```

Arguments

type	A character value indicating the data type of the pgeometry object. It can be either "PLATEAUPOINT", "PLATEAULINE" or "PLATEAUREGION".

Details

The create_empty_pgeometry creates a new pgeometry object with no components. To add new components to this object, you should use spa_add_component. The components added to this object must be of same type of the empty pgeometry object.

Value

A pgeometry object.

Examples

```

# Creating an Empty Plateau Point object
empty_plateau_point <- create_empty_pgeometry("PLATEAUPOINT")

# Creating an Empty Plateau Line object
empty_plateau_line <- create_empty_pgeometry("PLATEAULINE")

# Creating an Empty Plateau Region object
empty_plateau_region <- create_empty_pgeometry("PLATEAUREGION")
```

create_pgeometry *Creation of a* pgeometry *object with components*

Description

This function creates a pgeometry object from a data.frame or a list of components.

Usage

```
create_pgeometry(components, type)
```

Arguments

components	A list of component objects or a data.frame. The type of each component must be the same for all components.
type	A character value that indicates the type of the desired pgeometry object. It should be either "PLATEAUPOINT", "PLATEAULINE", or "PLATEAUREGION". It must be compatible with the components given in components parameter.

Details

The create_pgeometry function creates a pgeometry object of a given type. This object is built by using either a list of component objects or a dataframe (or tibble). If a dataframe is given, it must have two columns: the first one is a sfc object and second one indicates the membership degree of each respective object of the sfc column.

Value

A pgeometry object.

Examples

```
library(sf)
# Example 1 - Creating an `PLATEAUPOINT` object.

# Creating components for the plateau point object
v1 <- rbind(c(1,2), c(3,4))
v2 <- rbind(c(1,4), c(2,3),c(4,4))

md1 <- 0.2
md2 <- 0.1
md3 <- 0.4
pts1 <- rbind(c(1, 2), c(3, 2))
pts2 <- rbind(c(1, 1), c(2, 3), c(2, 1))
pts3 <- rbind(c(2, 2), c(3, 3))

comp1 <- component_from_sfg(st_multipoint(pts1), md1)
comp2 <- component_from_sfg(st_multipoint(pts2), md2)
```

```
comp3 <- component_from_sfg(st_multipoint(pts3), md3)

# Creating the plateau point object as a pgeometry object with 3 components

plateau_point_pgeom <- create_pgeometry(list(comp1, comp2, comp3), "PLATEAUPOINT")

# Example 2 - Creating an `PLATEAULINE` object.

lpts1 <- rbind(c(0, 0), c(1, 1))
lpts2 <- rbind(c(1, 1), c(1.2, 1.9), c(2, 1))
lpts3 <- rbind(c(2, 1), c(1.5, 0.5))

comp4 <- component_from_sfg(st_linestring(lpts1), 0.4)
comp5 <- component_from_sfg(st_linestring(lpts2), 1)
comp6 <- component_from_sfg(st_linestring(lpts3), 0.7)

plateau_line <- create_pgeometry(list(comp4, comp5, comp6), "PLATEAULINE")
```

`fsi_add_cs` *Adding the consequent to an FSI model*

Description

This function adds the consequent to a fuzzy spatial inference (FSI) model. It consists of a set of membership functions labeled with linguistic values.

Usage

```
fsi_add_cs(fsi, lvar, lvals, mfs, bounds)
```

Arguments

`fsi`	The FSI model instantiated with the `fsi_create` function.
`lvar`	A character value that represents a linguistic variable of the consequent.
`lvals`	A character vector that represents linguistic values of the linguistic variable of the consequent.
`mfs`	A vector of functions created by the `genmf` of the FuzzyR package.
`bounds`	A numeric vector that represents the lower and upper bounds of the consequent domain.

Details

Each linguistic value defined at the `lvals` parameter has a membership function defined at the `mfs` parameter. `lvals` is a character vector containing the names of linguistic values and `mfs` is vector containing its corresponding membership functions. Thus, the vectors defined for these two parameters must have the same length. For instance, the first value of `lvals` is the linguistic value for the first membership function in `mfs`. In bounds, the lower and upper values correspond to the first and second parameter, respectively.

Value

An FSI model populated with a consequent.

Examples

```
library(FuzzyR)

# Create the fsi_model:
fsi <- fsi_create("To visit or not to visit, that is the question",
                  default_conseq = genmf("trimf", c(10, 30, 60)))

# Create the vector with the linguistic values of the linguistic variable "visiting experience":
lvals_visiting_exp <- c("awful", "average", "great")

# Define the membership function for each linguistic value:
awful_mf <- genmf("trimf", c(0, 0, 20))
average_mf <- genmf("trimf", c(10, 30, 60))
great_mf <- genmf("trapmf", c(40, 80, 100, 100))

# Add the consequent to the FSI model:
fsi <- fsi_add_cs(fsi, "visiting experience", lvals_visiting_exp,
                 c(awful_mf, average_mf, great_mf), c(0, 100))
```

fsi_add_fsa *Adding an antecedent to an FSI model*

Description

This function adds a fuzzy spatial antecedent to a fuzzy spatial inference (FSI) model. A fuzzy spatial antecedent corresponds to a layer of fuzzy spatial objects that describe the different characteristics of the problem. The antecedent has a linguistic variable and its fuzzy spatial objects have linguistic values so that they are used in the IF part of fuzzy rules.

Usage

```
fsi_add_fsa(fsi, lvar, tbl)
```

Arguments

fsi	The FSI model instantiated with the `fsi_create` function.
lvar	A character value that represents a linguistic variable of the antecedent.
tbl	A tibble with spatial plateau objects annotated with linguistic values of the linguistic variable specified by the above `lvar` parameter.

Details

The fuzzy spatial antecedent added by the fsi_add_fsa function is composed of a linguistic variable and its corresponding pgeometry objects annotated by linguistic values. The format of the tbl parameter is the same as the output of the function spa_creator, allowing the user to directly provides plateau region objects as input when designing FSI models.

Value

An FSI model populated with a fuzzy spatial antecedent.

Examples

```
library(FuzzyR)
library(tibble)

# Create spatial plateau objects for the linguistic variable accomodation_price
lvals_accom_price <- c("cut-rate", "affordable", "expensive")
cut_rate_mf <- genmf("trapmf", c(0, 0, 10, 48))
affordable_mf <- genmf("trapmf", c(10, 48, 80, 115))
expensive_mf <- genmf("trapmf", c(80, 115, 10000, 10000))

# Example of dataset
accom_price <- tibble(
                      `longitude` = c(-74.0, -74.0, -74.0),
                      `latitude` = c(40.8, 40.7, 40.7),
                      `price` = c(150, 76, 60)
)

accom_price_layer <- spa_creator(accom_price, classes = lvals_accom_price,
                         mfs = c(cut_rate_mf, affordable_mf, expensive_mf))

# Create the fsi_model:
fsi <- fsi_create("To visit or not to visit, that is the question",
                  default_conseq = genmf("trimf", c(10, 30, 60)))

# Add the fuzzy spatial antecedent to the fsi_model:
fsi <- fsi_add_fsa(fsi, "accommodation price", accom_price_layer)
```

fsi_add_rules *Adding fuzzy rules to an FSI model*

Description

This function adds the fuzzy rules set to a fuzzy spatial inference (FSI) model. A fuzzy rule must contain only linguistic variables and values employed by the added antecedent parts and consequent.

Usage

```
fsi_add_rules(fsi, rules, weights = rep(1, length(rules)))
```

Arguments

`fsi`	An FSI model instantiated with the function `fsi_create`.
`rules`	A character vector containing the rules defined by the user. It follows a specific format, as detailed below.
`weights`	A numeric vector of weight values for each rule. Default values are 1.

Details

The definition of a fuzzy rule is user-friendly since users can write it by using the *linguistic variables* and *linguistic values* previously defined and added to the FSI model. A fuzzy rule has the format `IF A THEN B`, where `A` is called the antecedent and `B` the consequent of the rule such that `A` implies `B`. Further, `A` and `B` are statements that combine fuzzy propositions by using logical connectives like `AND` or `OR`. Each fuzzy proposition has the format `LVar is LVal` where `LVal` is a linguistic value in the scope of the linguistic variable `LVar`. To avoid possible contradictions keep in mind the following items when specifying the rules:

- the order of the statements in the antecedent is not relevant;
- each linguistic variable has to appear at most one time in each fuzzy rule;

Value

An FSI model populated with fuzzy rules set.

Examples

```
# Creating the FSI model from an example implemented with the visitation function:
fsi <- visitation()

# Creating a vector of fuzzy rules;
## note that we make use of the linguistic variables and linguistic values previously defined:
rules <- c(
  "IF accommodation review is reasonable AND food safety is low
  THEN visiting experience is awful",
 "IF accommodation price is expensive AND accommodation review is reasonable
   THEN visiting experience is awful",
 "IF accommodation price is affordable AND accommodation review is good AND food safety is medium
   THEN visiting experience is average",
 "IF accommodation price is affordable AND accommodation review is excellent
                                                                AND food safety is high
   THEN visiting experience is great",
 "IF accommodation price is cut-rate AND accommodation review is excellent AND food safety is high
   THEN visiting experience is great")

# Adding these rules to the FSI model previously instantiated:
fsi <- fsi_add_rules(fsi, rules)
```

fsi_create *Creation of an empty fuzzy spatial inference model*

Description

This function builds a fuzzy spatial inference (FSI) model without elements of the data source component (i.e., spatial plateau objects, fuzzy rules set, and fuzzy sets).

Usage

```
fsi_create(name, and_method = "min", or_method = "max",
           imp_method = "min", agg_method = "max",
           defuzz_method = "centroid", default_conseq = NULL)
```

Arguments

name	A character value that specifies the name of the FSI model.
and_method	A character value that defines the operator name for the logical connective AND. Default value is "min".
or_method	A character value that defines the operator for the logical connective OR. Default value is "max".
imp_method	A character value that defines the operator for the implication operator. Default value is "min".
agg_method	A character value that defines the operator for the aggregation operator. Default value is "max".
defuzz_method	A character value that determines the defuzzification technique. Default value is the centroid technique.
default_conseq	This parameter is a membership function generated by the function genmf of the FuzzyR package.

Details

The FSI model created with the function fsi_create and its default parameter values will implement a model using Mamdani's method. The possible values for the parameters and_method and imp_method are: "min", "prod". Other t-norms from the FuzzyR package are also conceivable. The possible value for the parameters or_method and agg_method is: "max". Other t-conorms from the FuzzyR package are also conceivable. The possible values for the parameter defuzz_method include other defuzzification techniques from the FuzzyR package. The parameter default_conseq defines the default behavior of the FSI model when there is no fuzzy rule with a degree of fulfillment greater than 0 returned by the FSI model.

After creating an empty FSI model, you have to call the functions fsi_add_fsa, fsi_add_cs, and fsi_add_rules to fulfill the FSI model.

Value

An empty named FSI model that is ready to be populated with fuzzy rules representing the antecedents and the consequent.

Examples

```
library(FuzzyR)
# Creating the FSI model
fsi <- fsi_create("To visit or not to visit, that is the question",
                  default_conseq = genmf("trimf", c(10, 30, 60)))
```

`fsi_eval` *Evaluating an FSI model for a given point location*

Description

This function executes the reasoning process of a fuzzy spatial inference (FSI) model for a given point location (i.e., `sfg` object of the type `POINT`).

Usage

```
fsi_eval(fsi, point, ...)
```

Arguments

`fsi`	An FSI model built with the function `fsi_create` and populated by the functions `fsi_add_fsa`, `fsi_add_cs`, and `fsi_add_rules`.
`point`	An `sfg` object of geometry type `point`, which is created through the function `st_point` of the sf package.
`...`	<`dynamic-dots`> Informs the `fsi_eval` how the elements of the resulting fuzzy set should be discretized if the user does not want the default configuration (see below). Default values: `discret_by` is 0.5 and `discret_length` is NULL.

Details

This function evaluates an FSI model populated with its fuzzy spatial antecedent, consequent, and fuzzy rules set on a specific point location. This evaluation is based on the algorithm specified by FIFUS.

The default behavior of the function `fsi_eval` in the parameter `...` is to consider a discrete interval of values with an increment of 0.5 between lower and upper values for the consequent domain (i.e., defined at `fsi_add_cs` function with the parameter `bounds`).

The user can modify the default behavior by using one of the following two ways:

- define a value for the parameter `discret_by` by changing the incremental value.
- define a desired length for the sequence of values domain of the consequent `discret_length`.

Value

A numeric value that belongs to the domain of the consequent (i.e., as specified by `fsi_add_cs`) and represents the result of the reasoning process in a particular point location.

References

Carniel, A. C.; Schneider, M. Fuzzy inference on fuzzy spatial objects (FIFUS) for spatial decision support systems. In Proceedings of the 2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2017), pp. 1-6, 2017.

Examples

```
library(sf)
# Creating the FSI model from an example implemented with the visitation function:
fsi <- visitation()

# Creating a vector of fuzzy rules;
## note that we make use of the linguistic variables and linguistic values previously defined:
rules <- c(
  "IF accommodation review is reasonable AND food safety is low
  THEN visiting experience is awful",
 "IF accommodation price is expensive AND accommodation review is reasonable
   THEN visiting experience is awful",
 "IF accommodation price is affordable AND accommodation review is good AND food safety is medium
   THEN visiting experience is average",
 "IF accommodation price is affordable AND accommodation review is excellent
                                                                 AND food safety is high
   THEN visiting experience is great",
 "IF accommodation price is cut-rate AND accommodation review is excellent AND food safety is high
   THEN visiting experience is great")

# Adding these rules to the FSI model previously instantiated:
fsi <- fsi_add_rules(fsi, rules)

# Using the default configuration:
res <- fsi_eval(fsi, st_point(c(-74.0, 40.7)))

# Change the default discretization by modifying the default step value:
res <- fsi_eval(fsi, st_point(c(-74.0, 40.7)), discret_by=0.8)

# Change the default discretization by choosing the quantity of values
## between the lower and upper values for the consequent domain:
res <- fsi_eval(fsi, st_point(c(-74.0, 40.7)), discret_length=200)
```

`fsi_qw_eval` *Evaluating a query window inference*

Description

This function implements two approaches for evaluating the query window inference on a fuzzy spatial inference (FSI) model. Given a query window (i.e., a rectangular object), it returns a set of inferred points inside this window that satisfy a specific condition (e.g., target linguistic value, or maximum/minimum inferred values).

Usage

```
fsi_qw_eval(fsi, qw, approach = "discretization", ...)
```

Arguments

`fsi` An FSI model built with the `fsi_create` function that is populated by the following functions `fsi_add_fsa`, `fsi_add_cs`, and `fsi_add_rules`.

`qw` An `sfg` object storing the query window that is supposed to be used as input for the inference. It has to be an axis-aligned rectangle represented by a simple polygon object of 5 points (since the last coordinate pair closes the external ring of the rectangle).

`approach` Defines which approach is employed to perform the query window inference: “`discretization`” or “`pso`”. Default value is ‘"discretization"“

`...` <`dynamic-dots`> Different set of parameters required depending on the chosen approach (see more in details below).

Details

For the *discretization* approach, two additional parameters are needed and they have to be informed by using the three-dots parameter `...`:

- `target_lval`: A character value that indicates the target linguistic value from the linguistic variable of the consequent.
- `k`: A numeric value that defines the number of points that will be captured from the query window and evaluated by the function `fsi_eval`. Its square root has to an integer value. Alternatively, you can inform the number of columns and rows of the regular grid to be performed on the query window by informing numeric values for `n_col` and `n_row`, respectively. Thus, these parameters can be given instead of the number `k`.

For the *pso* approach, it is necessary to set the following parameters:

- `what`: A character value that defines the user's goal, which can be either **maximize** or **minimize** inferred values. Thus, this parameter can be “`max`” and “`min`”, respectively. The default value is “`max`”.
- `max_depth`: A numeric value that refers to the number of times the user wants to split the query window. The default value is equal to 2. For instance, a `max_depth = 2` results in the query window split into four sub quadrants, where the particle swarm optimization (PSO) algorithm will be applied to each one as its search space. In addition, the PSO algorithm has its own set of parameters:
- `maxit`: A numeric value that defines the maximum number of iterations. Default value is 50.
- `population`: A numeric value that defines the number of particles. Default value is 10.

Value

A tibble in the format (points, inferred_values), where points is an sfc object (i.e., a list of sfg objects of geometry type POINT) and inferred_values are inferred values in the domain of the consequent of the FSI model.

Examples

```
library(sf)
# Creating the FSI model from an example implemented with the visitation function:
fsi <- visitation()

# Creating a vector of fuzzy rules;
## note that we make use of the linguistic variables and linguistic values previously defined:
rules <- c(
  "IF accommodation review is reasonable AND food safety is low
  THEN visiting experience is awful",
 "IF accommodation price is expensive AND accommodation review is reasonable
   THEN visiting experience is awful",
 "IF accommodation price is affordable AND accommodation review is good AND food safety is medium
   THEN visiting experience is average",
 "IF accommodation price is affordable AND accommodation review is excellent
                                                                 AND food safety is high
   THEN visiting experience is great",
 "IF accommodation price is cut-rate AND accommodation review is excellent AND food safety is high
   THEN visiting experience is great")

# Adding these rules to the FSI model previously instantiated:
fsi <- fsi_add_rules(fsi, rules)

# Defining the query window that is defined over an application domain
pts_qw1 <- rbind(c(-73.92, 40.68527), c(-73.75, 40.68527),
                 c(-73.75, 40.75), c(-73.92, 40.75), c(-73.92, 40.68527))
qw1 <- st_polygon(list(pts_qw1))

# Example using the discretization approach:
dis_res <- fsi_qw_eval(fsi, qw1, approach = "discretization", target_lval = "great", k = 25)

## Example using the pso approach in two levels:
## Not run:
pso_res <- fsi_qw_eval(fsi, qw1, approach = "pso", max_depth = 2)

## End(Not run)
```

fsr_components *Creation of a component*

Description

There are two functions that build a component from coordinate pairs or a single `sfg` object labeled with a membership degree. This component can be added to a spatial plateau object. A component consists of an `sfg` object and an associated membership degree. A component can be built in two different ways. By using the function `create_component`, the component is formed by the means of a numeric vector, list or matrix that represents a pair of coordinates. By using the function `component_from_sfg`, the component is created from an `sfg` object.

Usage

```
create_component(raw_obj, md, type)

component_from_sfg(sfg, md)
```

Arguments

`raw_obj`	A vector, list or matrix containing the pairs of coordinates to create the `sfg` object of the component.
`md`	A numeric value indicating the membership degree of the component. It has to be a value in $]0,1]$.
`type`	A character value that indicates the type of the desired `sfg` object. It should be either `"POINT"`, `"LINE"`, or `"REGION"`.
`sfg`	An `sfg` object. It should be either `POINT`, `MULTIPOINT`, `LINESTRING`, `MULTILINESTRING`, `POLYGON` or `MULTIPOLYGON` type. Other types of spatial objects are not allowed.

Details

These functions create a `component` object, which is a pair of an `sfg` object and a membership degree in $]0,1]$.

The function `create_component` receives three parameters: `raw_obj`, `md` and `type`. The use of `raw_obj` is similar to the parameter of the family of functions of the `sf` package (`st` family) that creates spatial objects from a numeric vector, matrix or list (e.g., the functions `st_point`, `st_multipoint`, etc.). The spatial data type (i.e., the type of the `sfg` object) indicated by the parameter `type` represents simple and complex objects. For instance, `"POINT"` may refer to simple or complex point objects (internally, we can create a POINT or MULTIPOINT object).

The `component_from_sfg` builds a `component` object by using the specification of two parameters that directly represents the pair of an `sfg` object and its corresponding membership degree (i.e., `md` value).

Value

A `component` object that can be added to a spatial plateau object (i.e., a `pgeometry` object).

Examples

```
# Creating two components of the type POINT
v1 = rbind(c(1,2), c(3,4))
```

```
v2 = rbind(c(1,4), c(2,3),c(4,4))

md1 = 0.2
md2 = 0.1

comp1 <- create_component(v1, md1, type="POINT")
comp2 <- create_component(v2, md2, type="POINT")

# Creating two components of the type LINE

md3 = 0.45
md4 = 0.32

v3 = rbind(c(2,2), c(3,3))
v4 = rbind(c(1,1), c(3,2))

comp3 <- create_component(v3, md3, type="LINE")
comp4 <- create_component(v4, md4, type="LINE")

# Creating two components of the type REGION

p1 <- rbind(c(0,0), c(1,0), c(3,2), c(2,4), c(1,4), c(0,0))
p2 <- rbind(c(1,1), c(1,2), c(2,2), c(1,1))
list_pols_1 <- list(p1,p2)

p3 <- rbind(c(1,0), c(2,0), c(4,2), c(3,4), c(2,4), c(1,0))
p4 <- rbind(c(2,2), c(2,3), c(3,4), c(2,2))
list_pols_2 <- list(p3,p4)

comp_pol1 <- create_component(list_pols_1, 0.4, "REGION")
comp_pol2 <- create_component(list_pols_2, 0.6, "REGION")


# Creating components with an sfg object
library(sf)

# POINT
md1 <- 0.2
pts1 <- rbind(c(1, 2), c(3, 2))
comp1 <- component_from_sfg(st_multipoint(pts1), md1)

# LINE
md2 <- 0.1
pts2 <- rbind(c(2, 2), c(3, 3))
comp2 <- component_from_sfg(st_linestring(pts2), md2)

# REGION
md3 <- 0.4
matrix_object = matrix(c(1,1,8,1,8,8,1,8,1,1),ncol=2, byrow=TRUE)
pts3 = list(matrix_object)
comp3 = component_from_sfg(st_polygon(pts3), md3)
```

fsr_diff_operators *Fuzzy difference operators*

Description

Fuzzy difference operations are set operations that generalize Boolean difference operations. This family of functions implements some operators that help us to define different fuzzy difference operations. These operators receive two numerical values in [0, 1] as input and calculates another numerical value in [0, 1] as output.

Usage

```
f_diff(x, y)

f_bound_diff(x, y)

f_symm_diff(x, y)

f_abs_diff(x, y)
```

Arguments

x	A numerical vector whose values are in [0, 1].
y	A numerical vector whose values are in [0, 1].

Details

These functions calculate the resulting membership degree of a fuzzy difference operator applied on two numerical values in the interval [0, 1]. The following fuzzy difference operators are available:

- `f_diff`: The standard *fuzzy set difference* operator defined as the intersection of `x` and the complement of `y`, that is, `min(x, 1 - y)`.
- `f_bound_diff`: The *fuzzy bounded difference* operator defined as `x` minus `y` with upper bound equal to 0, that is, `max(0, x - y)`.
- `f_symm_diff`: The *fuzzy symmetric difference* operator defined as the union of the difference of `x` and `y` and the difference of `y` and `x`, that is, `max(f_diff(x, y), f_diff(y, x))`.
- `f_abs_diff`: The *fuzzy absolute difference* operator defined as the absolute difference of `x` and `y`, that is, `abs(x - y)`.

These operators are useful to process the function `spa_difference` since one of them can be informed as a parameter for this function.

Value

A numerical vector.

Examples

```
x <- c(0.1, 0.3, 0.6, 0.8)
y <- c(0.9, 0.7, 0.4, 0.2)

f_diff(x, y)
f_bound_diff(x, y)
f_symm_diff(x, y)
f_abs_diff(x, y)
```

`fsr_eval_modes` *Evaluation modes*

Description

This family of functions implements evaluation modes that returns a Boolean value for a given degree in [0, 1] obtained from a membership function of a linguistic value.

Usage

```
soft_eval(degree)

strict_eval(degree)

alpha_eval(degree, alpha)

soft_alpha_eval(degree, alpha)
```

Arguments

`degree`	A numerical vector whose values are in [0, 1].
`alpha`	A single numeric value in [0, 1].

Details

These functions yield a Boolean value that express the meaning of a degree returning from an evaluation of a membership function. That is, the parameter `degree` is a value in [0, 1] resulting from evaluation a value in a membership degree. Then, an evaluation mode "translate" the meaning of this degree of truth as a Boolean value.

There some different ways to make this kind of translation:

- `soft_eval`: It returns `TRUE` if `degree` is greater than 0.
- `strict_eval`: It returns `TRUE` if `degree` is equal to 0.
- `alpha_eval`: It returns `TRUE` if `degree` is greater than or equal to another value (named `alpha`).

- soft_alpha_eval: It returns TRUE if degree is greater than another value (named alpha).

These operators are employed to process the evaluation modes of fuzzy topological relationships that are processed as Boolean predicates.

Value

A Boolean vector.

Examples

```
x <- c(0.1, 0.3, 0.6, 0.8)

soft_eval(x)
strict_eval(x)
alpha_eval(x, 0.3)
soft_alpha_eval(x, 0.3)
```

fsr_geometric_operations

Fuzzy geometric set operations

Description

Fuzzy geometric set operations are given as a family of functions that implements spatial plateau set operations. These functions yield a spatial plateau object from a specific combination of other two spatial plateau objects, such as the intersection of two plateau region objects.

Usage

```
spa_intersection(pgo1, pgo2, itype = "min")

spa_union(pgo1, pgo2, utype = "max")

spa_difference(pgo1, pgo2, dtype = "f_diff")

spa_common_points(pline1, pline2, itype = "min")
```

Arguments

pgo1	A pgeometry object of any type.
pgo2	A pgeometry object of the same type of pgo1.
itype	A character value that indicates the name of a function implementing a t-norm. The default value is "min", which is the standard operator of the intersection.
utype	A character value that refers to a t-conorm. The default value is "max", which is the standard operator of the union.

`dtype` A character value that indicates the name of a difference operator. The default value is `"f_diff"`, which implements the standard fuzzy difference.

`pline1` A `pgeometry` object of the type `PLATEAULINE`.

`pline2` A `pgeometry` object of the type `PLATEAULINE`.

Details

These functions implement geometric operations of the spatial plateau algebra. They receive two `pgeometry` objects of the *same type* together with an operation as inputs and yield another `pgeometry` object as output. The output object has *the same* type of the inputs. The family of fuzzy geometric set operations consists of the following functions:

- `spa_intersection` computes the geometric intersection of two spatial plateau objects. The membership degree of common points are calculated by using a t-norm operator given by the parameter `itype`. Currently, it can assume `"min"` (default) or `"prod"`.
- `spa_union` computes the geometric union of two spatial plateau objects. The membership degree of common points are calculated by using a t-conorm operator given by the parameter `utype`. Currently, it can assume `"max"` (default).
- `spa_difference` computes the geometric difference of two spatial plateau objects. The membership degree of common points are calculated by using a diff operator given by the parameter `dtype`. Currently, it can assume `"f_diff"` (default fuzzy difference), `"f_bound_diff"` (fuzzy bounded difference), `"f_symm_diff"` (fuzzy symmetric difference), and `"f_abs_diff"` (fuzzy absolute difference).

Another related geometric function is:

- `spa_common_points` which gets the common points of two plateau line objects by using a t-norm to compute their membership degrees. It is different from the other functions since it gets two plateau line objects as input and yields a plateau point object as output.

Other t-norms, t-conorms, and diff operators can be implemented and given as values for the `"itype"`, `"utype"`, and `"dtype"`, respectively. For this, the following steps should be performed:

1 - implement your function that accepts two numeric values as inputs and yields another numeric value as output. All values should be between 0 and 1. Recall that t-norms and t-conorms must have some specific properties according to the fuzzy set theory. 2 - use the name of your function as the character value of the corresponding `"itype"`, `"utype"`, or `"dtype"`.

An example of operator is the source code of `f_bound_diff`:

```
f_bound_diff <- function(x, y) { max(0, (x - y)) }
```

Value

A `pgeometry` object that is the result of the geometric manipulation between two spatial plateau objects.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(sf)

pts1 <- rbind(c(1, 2), c(3, 2))
pts2 <- rbind(c(1, 1), c(2, 3), c(2, 1))
pts3 <- rbind(c(2, 2), c(3, 3))

cp1 <- component_from_sfg(st_multipoint(pts1), 0.3)
cp2 <- component_from_sfg(st_multipoint(pts2), 0.6)
cp3 <- component_from_sfg(st_multipoint(pts3), 1)

pp1 <- create_pgeometry(list(cp1, cp2, cp3), "PLATEAUPOINT")

pts4 <- rbind(c(0, 0), c(1, 1))
pts5 <- rbind(c(2, 3), c(1.2, 1.9), c(2, 1))
pts6 <- rbind(c(3, 1), c(1.5, 0.5))

cp4 <- component_from_sfg(st_multipoint(pts4), 0.4)
cp5 <- component_from_sfg(st_multipoint(pts5), 1)
cp6 <- component_from_sfg(st_multipoint(pts6), 0.7)

pp2 <- create_pgeometry(list(cp4, cp5, cp6), "PLATEAUPOINT")

pp1
pp2

spa_intersection(pp1, pp2)
spa_intersection(pp1, pp2, itype = "prod") #changing the t-norm
spa_union(pp1, pp2)
spa_difference(pp1, pp2)
```

---

fsr_is_empty *Checking whether a* pgeometry *object is empty*

Description

This function checks whether a pgeometry object is empty (i.e., if it does not contain components).

Usage

```
fsr_is_empty(pgo)
```

Arguments

pgo A pgeometry object.

Details

It checks if a pgeometry object has any component or not. If the number of components of a `pgeometry` object is equal to 0, then it returns `TRUE`. Otherwise, it returns `FALSE`.

Value

A Boolean value that indicates if a `pgeometry` is empty.

Examples

```
# Creating an empty pgeometry object
pgo1 <- create_empty_pgeometry("PLATEAULINE")

# Checking if it is empty
fsr_is_empty(pgo1)

# Creating a component to populate the pgeometry object

library(sf)
md <- 0.4
pts <- rbind(c(1, 1), c(2, 3), c(2, 1))

comp <- component_from_sfg(st_multipoint(pts), md)

# Adding the component to the pgeometry object
pgo1 <- spa_add_component(pgo1, comp)

# Checking if it is still empty
fsr_is_empty(pgo1)
```

---

fsr_numerical_operations

Fuzzy numerical operations

Description

Fuzzy numerical operations are given as a family of functions that implements spatial plateau metric operations. These functions extract metric properties from spatial plateau objects, such as the area of a plateau region object and the length of a plateau line object.

Usage

```
spa_avg_degree(pgo)

spa_ncomp(pgo)
```

```
spa_area(pr)

spa_perimeter(pr)

spa_length(pl)
```

Arguments

pgo	A pgeometry object of any type.
pr	A pgeometry object of the type PLATEAUREGION. It throws an error if a different type is given.
pl	A pgeometry object of the type PLATEAULINE. It throws an error if a different type is given.

Details

These functions calculate numerical properties from spatial plateau objects (i.e., pgeometry objects). Some of them are *type-independent*. This means that the parameter can be a pgeometry object of any type. The type-independent functions are:

- spa_avg_degree calculates the average membership degree of a spatial plateau object.
- spa_ncomp returns the number of components of a spatial plateau object.

The remaining functions are *type-dependent*. This means that the parameter have to be of a specific type. The type-dependent functions are:

- spa_area computes the area of a plateau region object. Thus, its parameter has to be a PLATEAUREGION object.
- spa_perimeter computes the perimeter of a plateau region object. Thus, its parameter has to be a PLATEAUREGION object.
- spa_length computes the length of a plateau line object. Thus, its parameter has to be a PLATEAULINE object.

Value

A numerical value.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(sf)
library(tibble)
```

```
pts1 <- rbind(c(1, 2), c(3, 2))
comp1 <- component_from_sfg(st_multipoint(pts1), 0.2)
comp2 <- component_from_sfg(st_point(c(1, 5)), 0.8)

pp <- create_pgeometry(list(comp1, comp2), "PLATEAUPOINT")

# calculating the average degree and number of components of pp

spa_avg_degree(pp)
spa_ncomp(pp)

# calculating the area and perimeter

set.seed(345)

# some random points to create plateau region objects by using the function spa_creator
tbl = tibble(x = runif(10, min= 0, max = 20),
             y = runif(10, min = 0, max = 30),
             z = runif(10, min = 0, max = 100))

#getting the convex hull on the points to clip the construction of plateau region objects
pts <- st_as_sf(tbl, coords = c(1, 2))
ch <- st_convex_hull(do.call(c, st_geometry(pts)))

pregions <- spa_creator(tbl, fuzz_policy = "fcp", k = 2, base_poly = ch)

spa_area(pregions$pgeometry[[1]])
spa_area(pregions$pgeometry[[2]])

spa_perimeter(pregions$pgeometry[[1]])
spa_perimeter(pregions$pgeometry[[2]])

# calculating the length of a plateau line object

lpts1 <- rbind(c(0, 0), c(1, 1))
lpts2 <- rbind(c(1, 1), c(1.2, 1.9), c(2, 1))
lpts3 <- rbind(c(2, 1), c(1.5, 0.5))

cp1 <- component_from_sfg(st_linestring(lpts1), 0.4)
cp2 <- component_from_sfg(st_linestring(lpts2), 1)
cp3 <- component_from_sfg(st_linestring(lpts3), 0.7)

pline <- create_pgeometry(list(cp1, cp2, cp3), "PLATEAULINE")

spa_length(pline)
```

fsr_topological_relationships

Fuzzy topological relationships

Description

Fuzzy topological relationships are given as a family of functions that implements spatial plateau topological relationships. A fuzzy topological relationship expresses a particular relative position of two spatial plateau objects. Since the spatial objects are fuzzy, their topological relationships are also fuzzy. Hence, a fuzzy topological relationship determines the degree to which a relation holds for any two spatial plateau objects by a real value in the interval [0, 1]. The key idea of these relationships is to consider point subsets resulting from the combination of spatial plateau set operations and spatial plateau metric operations on the spatial plateau objects for computing the resulting degree. The resulting degree can be also interpreted as a linguistic value.

Usage

```
spa_overlap(pgo1, pgo2, itype = "min", ret = "degree", ...)

spa_meet(pgo1, pgo2, itype = "min", ret = "degree", ...)

spa_disjoint(pgo1, pgo2, itype = "min", ret = "degree", ...)

spa_equal(pgo1, pgo2, utype = "max", ret = 'degree', ...)

spa_inside(pgo1, pgo2, utype = "max", ret = 'degree', ...)

spa_contains(pgo1, pgo2, utype = "max", ret = 'degree', ...)
```

Arguments

pgo1	A pgeometry object of the type PLATEAUREGION.
pgo2	A pgeometry object of the type PLATEAUREGION.
itype	A character value that indicates the name of a function implementing a t-norm. The default value is "min", which is the standard operator of the intersection.
ret	A character value that indicates the return type of the fuzzy topological relationship. The default value is "degree" and other possible values are "list" and "bool".
...	<dynamic-dots> If ret = "bool", two additional parameters have to be informed, as described below.
utype	A character value that indicates the name of a function implementing a t-conorm. The default value is "max", which is the standard operator of the union.

Details

These functions implement topological relationships of the spatial plateau algebra. They receive two pgeometry objects of the type PLATEAUREGION together with some additional parameters (as detailed below). The family of fuzzy topological relationships consists of the following functions:

- spa_overlap computes the overlapping degree of two plateau region objects. Since it uses the intersection operation, a t-norm operator can be given by the parameter itype. Currently, it can assume "min" (default) or "prod".

- spa_meet computes the meeting degree of two plateau region objects. Similarly to spa_overlap, a t-norm operator can be given by the parameter itype.
- spa_disjoint computes the disjointedness degree of two plateau region objects. Similarly to spa_overlap and spa_meet, a t-norm operator can be given by the parameter itype.
- spa_equal - computes how equal are two plateau region objects. Since it uses the union operation, a t-conorm operator can be given by the parameter utype. Currently, it can assume "max" (default).
- spa_inside - computes the containment degree of pgo1 in pgo2. Similarly to spa_equal, a t-conorm operator can be given by the parameter utype.
- spa_contains - it is the same of spa_inside but changing the order of the operands pgo1 and pgo2.

The parameter ret determines the returning value of a fuzzy topological relationship. The default value is the following:

- "degree" (default) - it indicates that the function will return a value in [0, 1] that represents the degree of truth of a given topological relationships.

For the remainder possible values, the functions make use of a set of linguistic values that characterize the different situations of topological relationships. Each linguistic value has an associated membership function defined in the domain [0, 1]. The fsr package has a default set of linguistic values. You can use the function spa_set_classification to change this set of linguistic values.

The remainder possible values for the parameter ret are:

- "list" - it indicates that the function will return a named list containing how much the result of the predicate belongs to each linguistic value (i.e., it employs the membership functions of the linguistic values).
- "bool" - it indicates that the function will return a Boolean value indicating whether the degree returned by the topological relationship matches a given linguistic value according to an *evaluation mode*. The evaluation mode and the linguistic values have to be informed by using the parameters eval_mode and lval, respectively. The possible values for eval_mode are: "soft_eval", "strict_eval", "alpha_eval", and "soft_alpha_eval". They have different behavior in how computing the Boolean value from the membership function of a linguistic value. See their documentations for more details. Note that the parameter lval only accept a character value belonging to the set of linguistic values that characterize the different situations of topological relationships.

Value

The returning value is determined by the parameter ret, as described above.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(tibble)
library(sf)
library(FuzzyR)

set.seed(456)

# some random points to create pgeometry objects by using the function spa_creator
tbl = tibble(x = runif(10, min= 0, max = 30),
             y = runif(10, min = 0, max = 30),
             z = runif(10, min = 0, max = 50))

#getting the convex hull on the points to clipping the construction of plateau region objects
pts <- st_as_sf(tbl, coords = c(1, 2))
ch <- st_convex_hull(do.call(c, st_geometry(pts)))

pregions <- spa_creator(tbl, base_poly = ch, fuzz_policy = "fcp", k = 2)

# Showing the different types of returning values
spa_overlap(pregions$pgeometry[[1]], pregions$pgeometry[[2]])
spa_overlap(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")
spa_overlap(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "bool",
           eval_mode = "soft_eval", lval = "mostly")

## Examples for evaluating the other fuzzy topological relationships
## Not run:
spa_meet(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")
spa_disjoint(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")
spa_equal(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")
spa_inside(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")
spa_contains(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")

## End(Not run)
```

`pgeometry-class` *An S4 Class for representing a spatial plateau object*

Description

An S4 Class for representing a spatial plateau object

Details

A `pgeometry` object is composed of a list of `component` objects, an `sfg` object that represents the union of all crisp spatial objects of its components (i.e., the support), and its data type, which can be either `PLATEAUPOINT`, `PLATEAULINE`, or `PLATEAUREGION`.

Slots

`component` A list of components.

`supp` An `sfg` object that stores the union of the spatial objects of the components of the spatial plateau object.

`type` The data type of the spatial plateau object.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

`plot` *Visualization of* `pgeometry` *objects*

Description

This function plots a `pgeometry` object.

Usage

```
## S4 method for signature 'pgeometry,missing'
plot(x, y, ...)

fsr_plot(pgo, base_poly = NULL, add_base_poly = TRUE,
           low = "white", high = "black", crs = NA, ...)
```

Arguments

`x` A `pgeometry` object of any type.

`y` Not applicable.

`...` <`dynamic-dots`> Optional parameters. They can be the same as the parameters of `geom_sf` function.

`pgo` A `pgeometry` object of any type.

`base_poly` An `sfg` object of the type `POLYGON` or `MULTIPOLYGON`. It can also be an `sfc` object with only one element of the type `POLYGON` or `MULTIPOLYGON`.

`add_base_poly` A Boolean value that indicates whether `base_poly` will added to the visualization.

`low` A character value that indicates the color for the lower `mds` limit value (0). Default is `"white"`.

`high` A character value that indicates the color for the higher `mds` limit value (1). Default is `"black"`.

`crs` A numerical value that denotes the coordinate reference system (i.e., EPSG code) of the visualization. Default is `NA`.

Details

The fsr_plot uses a ggplot method to construct the plot. It receives a pgeometry object (if it is empty, an empty graphics in obtained).

The low and high parameters are the colors for the minimum and maximum limits of the membership degrees. The default colors are "white" and "black", respectively. Other colors can be given in the same way that colors are informed to visualizations produced by the ggplot package.

It is possible to clip the geometric format of the components by using the parameter base_poly. The boundaries of this object can also be included in the visualization if the parameter add_base_poly is TRUE.

Value

A ggplot object.

Examples

```
library(sf)

### Example 1

# Creating components for the plateau point object
v1 <- rbind(c(1,2), c(3,4))
v2 <- rbind(c(1,4), c(2,3),c(4,4))

md1 <- 0.2
md2 <- 0.1
md3 <- 0.4
pts1 <- rbind(c(1, 2), c(3, 2))
pts2 <- rbind(c(1, 1), c(2, 3), c(2, 1))
pts3 <- rbind(c(2, 2), c(3, 3))

comp1 <- component_from_sfg(st_multipoint(pts1), md1)
comp2 <- component_from_sfg(st_multipoint(pts2), md2)
comp3 <- component_from_sfg(st_multipoint(pts3), md3)

# Creating the plateau point object as a pgeometry object with 3 components

ppoint <- create_pgeometry(list(comp1, comp2, comp3), "PLATEAUPOINT")

fsr_plot(ppoint) # with default colors
fsr_plot(ppoint, low="blue",high = "red") # with custom limit colors

# Example 2 - PLATEAULINE PLOT

lpts1 <- rbind(c(0, 0), c(1, 1))
lpts2 <- rbind(c(1, 1), c(1.2, 1.9), c(2, 1))
lpts3 <- rbind(c(2, 1), c(1.5, 0.5))

comp4 <- component_from_sfg(st_linestring(lpts1), 0.4)
comp5 <- component_from_sfg(st_linestring(lpts2), 1)
```

```
comp6 <- component_from_sfg(st_linestring(lpts3), 0.7)

pline <- create_pgeometry(list(comp4, comp5, comp6), ”PLATEAULINE”)

fsr_plot(pline) # Default values
fsr_plot(pline, low=”green”, high=”blue”) # Custom colors ...


# Example 3 - PLATEAUREGION PLOT

p1 <- rbind(c(0,0), c(1,0), c(3,2), c(2,4), c(1,4), c(0,0))
p2 <- rbind(c(1,1), c(1,2), c(2,2), c(1,1))
pol1 <-st_polygon(list(p1,p2))
p3 <- rbind(c(3,0), c(4,0), c(4,1), c(3,1), c(3,0))
p4 <- rbind(c(3.3,0.3), c(3.8,0.3), c(3.8,0.8), c(3.3,0.8), c(3.3,0.3))[5:1,]
pol2 <- st_polygon(list(p3,p4))
pol3 <- st_polygon(list(rbind(c(3,3), c(4,2), c(4,3), c(3,3))))

comp1 <- component_from_sfg(pol1, 0.2)
comp2 <- component_from_sfg(pol2, 0.4)
comp3 <- component_from_sfg(pol3, 0.7)

pregion <- create_pgeometry(list(comp1, comp2, comp3), ”PLATEAUREGION”)
fsr_plot(pregion)
fsr_plot(pregion, low = ”blue”, high = ”red”)
```

PWKT *The PWKT of a spatial plateau object*

Description

This function gives the Plateau Well-Known Text (PWKT) representation of a `pgeometry` object.

Usage

```
spa_pwkt(pgo)

## S3 method for class 'pgeometry'
format(x, ...)

## S4 method for signature 'pgeometry'
show(object)

## S4 method for signature 'pgeometry'
as.character(x, ...)
```

Arguments

pgo	A pgeometry object of any type.
x	A pgeometry object of any type.
...	<dynamic-dots> Unused.
object	A pgeometry object of any type.

Details

It gives the textual representation for a pgeometry object, combining the Well-Known Text (WKT) representation for crisp vector geometry objects and the formal definitions of the tree spatial plateau data types. (i.e. PLATEAUPOINT, PLATEAULINE, PLATEAUREGION).

Value

A character value with the textual representation of a given pgeometry object.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(sf)

# For a `PLATEAUPOINT` object.
pts1 <- rbind(c(1, 2), c(3, 2))
comp1 <- component_from_sfg(st_multipoint(pts1), 0.2)
comp2 <- component_from_sfg(st_point(c(1, 5)), 0.8)

ppoint <- create_pgeometry(list(comp1, comp2), "PLATEAUPOINT")

spa_pwkt(ppoint)

# For a `PLATEAULINE` object.

lpts1 <- rbind(c(0, 0), c(1, 1))
lpts2 <- rbind(c(1, 1), c(1.2, 1.9), c(2, 1))
lpts3 <- rbind(c(2, 1), c(1.5, 0.5))

comp4 <- component_from_sfg(st_linestring(lpts1), 0.4)
comp5 <- component_from_sfg(st_linestring(lpts2), 1)
comp6 <- component_from_sfg(st_linestring(lpts3), 0.7)

pline <- create_pgeometry(list(comp4, comp5, comp6), "PLATEAULINE")

spa_pwkt(pline)
```

```
# For a `PLATEAUREGION` object.

p1 <- rbind(c(0,0), c(1,0), c(3,2), c(2,4), c(1,4), c(0,0))
p2 <- rbind(c(1,1), c(1,2), c(2,2), c(1,1))
pol1 <-st_polygon(list(p1,p2))

comp1 <- component_from_sfg(pol1, 0.2)

pregion <- create_pgeometry(list(comp1), "PLATEAUREGION")

spa_pwkt(pregion)
```

spa_add_component *Adding components to a* pgeometry *object*

Description

This function adds components to a spatial plateau object (i.e., pgeometry object). The crisp spatial object of the component must be compatible with the type of the plateau spatial object. For instance, a pgeometry object of the type PLATEAUREGION accepts only components containing polygons (e.g., POLYGON or MULTIPOLYGON).

Usage

```
spa_add_component(pgo, components)
```

Arguments

pgo	A pgeometry object of any type.
components	A component object or a list of component objects.

Details

This function implements the $\odot$ operator defined by Spatial Plateau Algebra. The goal of this function is to insert a component or a list of components into a pgeometry object. This insertion is based on the membership degree of the component (e.g., created by component_from_sfg). Thus, it preserves the properties of a spatial plateau object. However, this function assumes that a component is compatible with the pgeometry object and its geometric format is valid (i.e., it does not overlap with existing components).

Value

A pgeometry object containing the component objects.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(sf)

pts1 <- rbind(c(1, 2), c(3, 2))
comp1 <- component_from_sfg(st_multipoint(pts1), 0.2)
comp2 <- component_from_sfg(st_point(c(1, 5)), 0.8)

# appending these components into an empty pgeometry object

pp <- create_empty_pgeometry("PLATEAUPOINT")
pp <- spa_add_component(pp, list(comp1, comp2))
pp
```

---

spa_boundary_pregion *Capturing the fuzzy boundary of a plateau region object*

Description

This function yields a specific part of the fuzzy boundary of a plateau region object.

Usage

```
spa_boundary_pregion(pregion, bound_part = "region")
```

Arguments

pregion	A pgeometry object of the type PLATEAUREGION. It throws an error if a different type is given.
bound_part	A character value that indicates the part of the fuzzy boundary to be returned. It can be "region" or "line". See below for more details.

Details

It employs the definition of *fuzzy boundary* of a fuzzy region object in the context of spatial plateau algebra (as defined in the references). The *fuzzy boundary* of a fuzzy region object A has a heterogeneous nature since it consists of two parts:

- a fuzzy line object that corresponds to the boundary of the core of A.
- a fuzzy region object that comprises all points of A with a membership degree greater than 0 and less than 1.

This means that the function spa_boundary_pregion can yield one specific part of the fuzzy boundary of a plateau region object (the argument pgeometry). If boundary = "line", then the function returns the boundary plateau line of pgeometry (i.e., returns a pgeometry object of the type PLATEAULINE). Else if boundary = "region" (the default value), then the function returns the boundary plateau region of pgeometry (i.e., returns a pgeometry object of the type PLATEAUREGION).

Value

A pgeometry object that represents a specific part of the fuzzy boundary of pgeometry object given as input.

References

- Carniel, A. C.; Schneider, M. A Conceptual Model of Fuzzy Topological Relationships for Fuzzy Regions. In Proceedings of the 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016), pp. 2271-2278, 2016.
- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(tibble)
library(FuzzyR)

set.seed(123)

# some random points to create pgeometry objects by using the function spa_creator
tbl = tibble(x = runif(10, min= 0, max = 30),
             y = runif(10, min = 0, max = 50),
             z = runif(10, min = 0, max = 100))

classes <- c("category-1", "category-2")
mf1 <- genmf("trapmf", c(0, 5, 20, 35))
mf2 <- genmf("trimf", c(20, 80, 100))

pregions <- spa_creator(tbl, classes = classes, mfs = c(mf1, mf2))
pregions$pgeometry[[1]]
pregions$pgeometry[[2]]

# capturing and showing the boundary plateau line of each pgeometry object previously created
(spa_boundary_pregion(pregions$pgeometry[[1]], bound_part = "line"))
(spa_boundary_pregion(pregions$pgeometry[[2]], bound_part = "line"))
# the last boundary is empty because there is no core!

# capturing and showing the boundary plateau region (this is the default behavior)
(spa_boundary_pregion(pregions$pgeometry[[1]]))
(spa_boundary_pregion(pregions$pgeometry[[2]]))
```

spa_contour *Capturing the frontier of a plateau region object*

Description

This function extracts the frontier (i.e., linear boundary) of a plateau region object by maintaining its membership degrees.

Usage

```
spa_contour(pregion)
```

Arguments

pregion	A pgeometry object of the type PLATEAUREGION. It throws an error if a different type is given.

Details

It employs the definition of *fuzzy frontier* of a fuzzy region object in the context of spatial plateau algebra (as defined in the references). The *fuzzy frontier* of a fuzzy region object A collects all single points of A, preserving its membership degrees, that are not in the interior of its support.

IMPORTANT NOTE: Fuzzy frontier is different from fuzzy boundary (see spa_boundary_region).

Value

A pgeometry object of the type PLATEAULINE that represents the contour (i.e. frontier) of a plateau region object given as input.

References

- Carniel, A. C.; Schneider, M. A Conceptual Model of Fuzzy Topological Relationships for Fuzzy Regions. In Proceedings of the 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016), pp. 2271-2278, 2016.
- Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(tibble)
library(sf)
library(FuzzyR)

set.seed(123)

# some random points to create pgeometry objects by using the function spa_creator
```

```
tbl = tibble(x = runif(10, min= 0, max = 30),
             y = runif(10, min = 0, max = 50),
             z = runif(10, min = 0, max = 100))

classes <- c("category-1", "category-2")
mf1 <- genmf("trapmf", c(0, 5, 20, 35))
mf2 <- genmf("trimf", c(35, 80, 100))

#getting the convex hull on the points to clipping the construction of plateau region objects
pts <- st_as_sf(tbl, coords = c(1, 2))
ch <- st_convex_hull(do.call(c, st_geometry(pts)))

pregions <- spa_creator(tbl, classes = classes, mfs = c(mf1, mf2), base_poly = ch)

# capturing and showing the frontier of each pgeometry object previously created
frontier_pregion1 <- spa_contour(pregions$pgeometry[[1]])
frontier_pregion2 <- spa_contour(pregions$pgeometry[[2]])

plot(pregions$pgeometry[[1]])
plot(frontier_pregion1)

plot(pregions$pgeometry[[2]])
plot(frontier_pregion2)
```

spa_core *Capturing the core of a* pgeometry *object*

Description

This function yields a crisp spatial object (as an sfg object) that corresponds to the core of a pgeometry object given as input.

Usage

```
spa_core(pgo)
```

Arguments

pgo	A pgeometry object of any type.

Details

It employs the classical definition of *core* from the fuzzy set theory in the context of spatial plateau algebra. The *core* only comprises the points with membership degree equal to 1. Hence, this operation returns the sfg object that represents the component labeled with membership degree equal to 1 of the pgeometry object given as input. If the pgeometry object has no core, then an empty sfg object is returned (i.e., a crisp spatial object without points).

Value

An sfg object that represents the core of pgo. It can be an empty object if pgo does not have a component with membership degree 1.

References

Carniel, A. C.; Schneider, M. A Conceptual Model of Fuzzy Topological Relationships for Fuzzy Regions. In Proceedings of the 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016), pp. 2271-2278, 2016.

Examples

```
library(sf)

pts1 <- rbind(c(1, 2), c(3, 2))
pts2 <- rbind(c(1, 1), c(2, 3), c(2, 1))
pts3 <- rbind(c(2, 2), c(3, 3))

cp1 <- component_from_sfg(st_multipoint(pts1), 0.3)
cp2 <- component_from_sfg(st_multipoint(pts2), 0.6)
cp3 <- component_from_sfg(st_multipoint(pts3), 1.0)

pp <- create_pgeometry(list(cp1, cp2, cp3), "PLATEAUPOINT")
pp

pp_core <- spa_core(pp)
pp_core

#Creating a pgeometry object without core
pp2 <- create_pgeometry(list(cp1, cp2), "PLATEAUPOINT")
pp2

spa_core(pp2)
```

spa_creator *Building* pgeometry *objects from a point dataset*

Description

This function builds a set of spatial plateau objects from a given point dataset assigned with domain-specific numerical values.

Usage

```
spa_creator(tbl, fuzz_policy = "fsp", const_policy = "voronoi", ...)
```

Arguments

`tbl` A data.frame or tibble with the following format: (x, y, z).

`fuzz_policy` The fuzzification policy to be employed by the algorithm. See details below.

`const_policy` The construction policy to be used by the algorithm. See details below.

`...` <`dynamic-dots`> Parameters for the chosen policies. See details below.

Details

It follows the two-stage construction method described in the research paper of reference.

The input `tbl` is a point dataset where each point represents the location of a phenomenon treated by the application. Further, each point is annotated with numerical data that describe its meaning in the application. Therefore, `tbl` must have three columns: (*x*, *y*, *z*). The columns *x*, *y* are the pair of coordinates, and *z* is the column containing domain-specific numeric values.

`fuzz_policy` refers to the method used by the **fuzzification stage**. This stage aims to assign membership degrees to each point of the dataset. It accepts three possible values only: `"fsp"` (default), or `"fcp"`.

`"fsp"` stands for *fuzzy set policy* and requires two parameters that should be informed in `...`:

- `classes`: A character vector containing the name of classes
- `mfs`: A vector of membership functions generated by the function `genmf` of `FuzzyR` package. Each membership function *i* represents the class *i*, where *i* in `length(classes)`

`"fcp"` stands for *fuzzy clustering policy* and requires the `e1071` package. Its possible parameters, informed in `...`, are:

- `k`: A numeric value that refers to the number of groups to be created
- `method`: A fuzzy clustering method of the package `e1071`, which can be either `"cmeans"` (default) or `"cshell"`
- `use_coords`: A Boolean value to indicate whether the columns (x, y) should be used in the clustering algorithm (default is `FALSE`)
- `iter`: A numeric indicating the number of maximum iterations of the clustering algorithm (default is 100)

An optional and common parameter for both fuzzification stages is the `"digits"`. This is an integer value that indicates the number of decimal digits of the membership degrees calculated by the fuzzification stage. That is, it is used to **round** membership degrees to the specified number of decimal places. Be careful with this optional parameter! If you specify a low value for `"digits"` some membership degrees could be rounded to 0 and thus, some components would not be created.

`const_policy` refers to the method used by the **construction stage**. This stage aims to create polygons from the labeled point dataset and use them to build spatial plateau objects. It accepts two possible values only: either `"voronoi"` (default) or `"delaunay"`.

`"voronoi"` stands for *Voronoi diagram policy* and has one optional parameter that can be provided in `...`:

- `base_poly`: An `sfg` object that will be used to clip the generated polygons (optional argument). If this parameter is not provided, the Voronoi is created by using a bounding box (standard behavior of `sf`).

- d_tolerance: It refers to the parameter dTolerance employed by the function st_voronoi of the package sf.

"delaunay" stands for *Delaunay triangulation policy*, which accepts the following parameters in ...:

- base_poly: An sfg object that will be used to clip the generated triangles (optional argument).
- tnorm: A t-norm used to calculate the membership degree of the triangle. It should be the name of a vector function. Possible values are "min" (default), and "prod". Note that it is possible to use your own t-norms. A t-norm should has the following signature: FUN(x) where *x* is a numeric vector. Such a function should return a single numeric value.
- d_tolerance: It refers to the parameter dTolerance employed by the function st_triangulate of the package sf.

"convex_hull" stands for *Convex hull policy*, which accepts the following parameters in ...:

- M: A numeric vector containing the membership degrees that will be used to create the components. The default is defined by seq(0.05, 1, by = 0.05).
- d: A numeric value representing the tolerance distance to compute the membership degree between the elements of M and the membership degrees of the points. The default is 0.05.
- base_poly: An sfg object that will be used to clip the generated polygons (optional argument).

Value

A tibble in the format (class, pgeometry), where class is a character column and pgeometry is a list of pgeometry objects. This means that a spatial plateau object is created for representing a specific class of the point dataset.

References

Carniel, A. C.; Schneider, M. A Systematic Approach to Creating Fuzzy Region Objects from Real Spatial Data Sets. In Proceedings of the 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2019), pp. 1-6, 2019.

Examples

```
library(tibble)
library(FuzzyR)

set.seed(7)
tbl = tibble(x = runif(10, min= 0, max = 30),
             y = runif(10, min = 0, max = 50),
             z = runif(10, min = 0, max = 100))
classes <- c("cold", "hot")
cold_mf <- genmf("trapmf", c(0, 10, 20, 35))
hot_mf <- genmf("trimf", c(35, 50, 100))

spa_creator(tbl, classes = classes, mfs = c(cold_mf, hot_mf))
spa_creator(tbl, fuzz_policy = "fcp", k = 4)

spa_creator(tbl, fuzz_policy = "fcp", k = 4, digits = 2)

spa_creator(tbl, fuzz_policy = "fcp", k = 3, const_policy = "delaunay")

spa_creator(tbl, fuzz_policy = "fcp", const_policy = "delaunay", k = 3, tnorm = "prod")

spa_creator(tbl, fuzz_policy = "fcp", k = 2, digits = 2,
            M = seq(0.1, 1, by = 0.1), d = 0.05, const_policy = "convex_hull")

spa_creator(tbl, classes = classes, mfs = c(cold_mf, hot_mf),
            digits = 2, const_policy = "convex_hull")
```

spa_eval *Capturing the membership degree of a point*

Description

This function evaluates the membership degree of a given point in a spatial plateau object of any type. It returns a value in [0, 1] that indicates to which extent the point belongs to the pgeometry object.

Usage

```
spa_eval(pgo, point)
```

Arguments

pgo	A pgeometry object of any type.
point	An sfg object of the type POINT.

Details

The goal of this function is to return the membership degree of a simple point object (i.e., sfg object) in a given spatial plateau object (i.e., pgeometry object). This evaluation depends on the following basic cases:

- if the simple point object belongs to the interior or boundary of *one* component of the spatial plateau object, it returns the membership degree of that component.
- if the simple point object intersects more components (e.g., boundaries of region components, or different line components), it returns the maximum membership degree of all intersected components.
- if the simple point object is disjoint to the support of the spatial plateau object, it returns 0.

Value

A numeric value between 0 and 1 that indicates the membership degree of a point (i.e., `sfg` object) in a spatial plateau object (i.e., `pgeometry` object).

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(tibble)
library(sf)
library(FuzzyR)

# some basic examples

pts1 <- rbind(c(1, 2), c(3, 2))
pts2 <- rbind(c(1, 1), c(2, 3), c(2, 1))
pts3 <- rbind(c(2, 2), c(3, 3))

cp1 <- component_from_sfg(st_multipoint(pts1), 0.3)
cp2 <- component_from_sfg(st_multipoint(pts2), 0.6)
cp3 <- component_from_sfg(st_multipoint(pts3), 1.0)

pp <- create_pgeometry(list(cp1, cp2, cp3), "PLATEAUPOINT")

spa_eval(pp, st_point(c(1, 2)))
spa_eval(pp, st_point(c(1, 3)))

# other examples with plateau regions

set.seed(345)

# some random points to create plateau region objects by using the function spa_creator
tbl = tibble(x = runif(10, min= 0, max = 20),
             y = runif(10, min = 0, max = 30),
             z = runif(10, min = 0, max = 100))

#getting the convex hull on the points to clipping the construction of plateau region objects
pts <- st_as_sf(tbl, coords = c(1, 2))
ch <- st_convex_hull(do.call(c, st_geometry(pts)))

pregions <- spa_creator(tbl, fuzz_policy = "fcp", k = 2, base_poly = ch)

# capturing the membership degree of a specific point in each object
spa_eval(pregions$pgeometry[[1]], st_point(c(5, 15)))
spa_eval(pregions$pgeometry[[2]], st_point(c(5, 15)))
```

spa_exact_equal *Check exact equality*

Description

This function checks whether two spatial plateau objects are exactly equal.

Usage

```
spa_exact_equal(pgo1, pgo2)
```

Arguments

pgo1	A pgeometry object of any type.
pgo2	A pgeometry object of any type.

Details

It is a Boolean function that checks *fuzzy equality* in the spatial plateau context. Two pgeometry objects are exactly equal if their components are equal. Two components are equal if they have the same membership degree and they are (spatially) equal (i.e., their sfg objects have the same geometric format - this means that the order of the points can be different).

Value

A Boolean value that indicates if two pgeometry objects are exactly equal.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(sf)

pts1 <- rbind(c(1, 2), c(3, 2))
pts2 <- rbind(c(1, 1), c(2, 3), c(2, 1))
pts3 <- rbind(c(2, 2), c(3, 3))

cp1 <- component_from_sfg(st_multipoint(pts1), 0.3)
cp2 <- component_from_sfg(st_multipoint(pts2), 0.6)
cp3 <- component_from_sfg(st_multipoint(pts3), 1.0)

pp1 <- create_pgeometry(list(cp1, cp2, cp3), "PLATEAUPOINT")
pp2 <- create_pgeometry(list(cp2, cp1), "PLATEAUPOINT")
```

```
spa_exact_equal(pp1, pp2)

spa_exact_equal(pp1, pp1)
```

spa_exact_inside *Check exact containment*

Description

This function checks whether a pgeometry object is completely inside of another pgeometry object.

Usage

```
spa_exact_inside(pgo1, pgo2)
```

Arguments

pgo1	A pgeometry object of any type.
pgo2	A pgeometry object of any type.

Details

It is a Boolean function that checks *fuzzy containment* in the spatial plateau context.

This Boolean function checks whether the components of pgo1 are contained in the components of pgo2 by considering their membership degrees and geographic positions. That is, it is follows the classical definition of fuzzy containment of the fuzzy set theory.

In other words, this function checks if the (standard) intersection of pgo1 and pgo2 is exactly equal to pgo1. The other of operands affects the result.

Value

A Boolean value that indicates if a pgeometry is completely and certainly inside pgo2.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(sf)

pts1 <- rbind(c(1, 2), c(3, 2))
pts2 <- rbind(c(1, 1), c(2, 3), c(2, 1))
pts3 <- rbind(c(2, 2), c(3, 3))

cp1 <- component_from_sfg(st_multipoint(pts1), 0.3)
cp2 <- component_from_sfg(st_multipoint(pts2), 0.6)
cp3 <- component_from_sfg(st_multipoint(pts3), 1.0)

# Creating two spatial plateau objects
pp1 <- create_pgeometry(list(cp1, cp2, cp3), "PLATEAUPOINT")
pp2 <- create_pgeometry(list(cp2, cp1), "PLATEAUPOINT")

# The other of operands after the result
# pp1 is not inside pp2 since it has one point that is not included in pp2
spa_exact_inside(pp1, pp2)

# on the other hand, pp2 is inside pp1
spa_exact_inside(pp2, pp1)
```

---

spa_set_classification

Setting a new classification for fuzzy topological relationships

---

Description

This functions configures a new set of linguistic values and their corresponding membership functions to be used by fuzzy topological relationships.

Usage

```
spa_set_classification(classes, mfs)
```

Arguments

`classes`	A character vector containing linguistic values that characterizes different situations of fuzzy topological relationships.
`mfs`	A vector containing membership functions generated by the function `genmf` of the FuzzyR package. Their domain have to be in [0, 1].

Details

This function replaces the default linguistic values employed by fuzzy topological relationships. Each membership function *i* of the parameter `mfs` represents the class *i* of the parameter `classes`. The length of these parameters have to be same.

Value

No return values, called for side effects.

References

Carniel, A. C.; Schneider, M. Spatial Plateau Algebra: An Executable Type System for Fuzzy Spatial Data Types. In Proceedings of the 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2018), pp. 1-8, 2018.

Examples

```
library(tibble)
library(sf)
library(FuzzyR)

set.seed(456)

# some random points to create pgeometry objects by using the function spa_creator
tbl = tibble(x = runif(10, min= 0, max = 30),
             y = runif(10, min = 0, max = 30),
             z = runif(10, min = 0, max = 50))

#getting the convex hull on the points to clipping the construction of plateau region objects
pts <- st_as_sf(tbl, coords = c(1, 2))
ch <- st_convex_hull(do.call(c, st_geometry(pts)))

pregions <- spa_creator(tbl, base_poly = ch, fuzz_policy = "fcp", k = 2)

# Showing the default list of classes
spa_overlap(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")

# Changing the default classification

classes <- c("small", "medium", "large")
small <- genmf("trapmf", c(0, 0.3, 0.4, 0.6))
medium <- genmf("trapmf", c(0.4, 0.6, 0.8, 1))
large <- genmf("trapmf", c(0.6, 0.8, 1, 1))

spa_set_classification(classes, c(small, medium, large))

spa_overlap(pregions$pgeometry[[1]], pregions$pgeometry[[2]], ret = "list")
```

spa_support *Capturing the support of a* pgeometry *object*

Description

This function yields a crisp spatial object (as an `sfg` object) that corresponds to the support of a `pgeometry` object given as input.

Usage

```
spa_support(pgo)
```

Arguments

`pgo`	A `pgeometry` object of any type.

Details

It employs the classical definition of *support* from the fuzzy set theory in the context of spatial plateau algebra. The *support* only comprises the points with membership degree greater than or equal to 1. Hence, this operation returns the `sfg` object that represents the total extent of the `pgeometry` given as input. If the `pgeometry` object has no components, then an empty `sfg` object is returned (i.e., a crisp spatial object without points).

Value

An `sfg` object that represents the support of `pgeometry`. It can be an empty object if `pgeometry` is empty.

References

Carniel, A. C.; Schneider, M. A Conceptual Model of Fuzzy Topological Relationships for Fuzzy Regions. In Proceedings of the 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016), pp. 2271-2278, 2016.

Examples

```
library(sf)

pts1 <- rbind(c(1, 2), c(3, 2))
pts2 <- rbind(c(1, 1), c(2, 3), c(2, 1))
pts3 <- rbind(c(2, 2), c(3, 3))

cp1 <- component_from_sfg(st_multipoint(pts1), 0.3)
cp2 <- component_from_sfg(st_multipoint(pts2), 0.6)
cp3 <- component_from_sfg(st_multipoint(pts3), 1.0)

pp <- create_pgeometry(list(cp1, cp2, cp3), "PLATEAUPOINT")
pp

pp_supp <- spa_support(pp)
pp_supp

pp_empty <- create_empty_pgeometry("PLATEAUPOINT")
```

```
pp_empty_supp <- spa_support(pp_empty)
pp_empty_supp
```

visitation *Visitation: An example of FSI model*

Description

This function provides an example, without rules, of a fuzzy spatial inference (FSI) model.

Usage

```
visitation()
```

Details

The FSI model provided by this function represents an FSI model to estimate the visiting experience based on prices and overall ratings of accommodations as well as sanitary conditions of restaurants. The output of such a model infers a value between 0 and 100 that indicates how attractive it is to visit a specific location. For this, the experience can be classified as *awful*, *average*, and *great*.

The linguistic variables and their linguistic values of this FSI model are listed below:

- *accommodation price*, with *cut-rate*, *affordable*, and *expensive* as linguistic values;
- *accommodation review* with *bad*, *good*, and *excellent* as linguistic values;
- *food safety* with *low*, *medium*, and *high* as linguistic values, which represent levels of sanitary conditions.

The usage of FSI models is subdivided into a *preparation phase* and an *evaluation phase*. The preparation phase is responsible for instantiating a new FSI model with the elements of the data source component of FIFUS. For this, the fsr package provides the following functions: `fsi_create`, `fsi_add_fsa`, and `fsi_add_cs`. These functions are employed by `visitation` so that users can add their own fuzzy set rules (by using `fsi_add_rules`) and perform the evaluation phase (by using the functions `fsi_eval` and/or `fsi_qw_eval`). In this sense, `visitation` performs the following internal actions to return an FSI model:

1. specify the linguistic variables and their corresponding linguistic values, which are in turn represented by membership functions generated by the function `genmf` of the FuzzyR package. These items are specified according to the context of the running example.
2. define small point datasets that represent each linguistic variable. Such datasets are `tibble` objects.
3. build spatial plateau objects by using the function `spa_creator` on the datasets. As a result, we get spatial plateau objects that represent each linguistic value.
4. create an FSI model with `fsi_create` function.
5. add fuzzy spatial antecedents with the `fsi_add_fsa` function. Recall that the antecedents are spatial plateau objects previously built.
6. define the linguistic variable and its linguistic values with membership functions for the consequent.
7. add the consequent to the FSI model by using the function `fsi_add_cs`.

Value

An example of an FSI model implemented without fuzzy rules set.

Examples

```
fsi <- visitation()
```

Index