

# Package ‘fcaR’

April 27, 2023

**Title** Formal Concept Analysis

**Version** 1.2.1

**Maintainer** Domingo Lopez Rodriguez <dominlopez@uma.es>

**Description** Provides tools to perform fuzzy formal concept analysis, presented in Wille (1982) <[doi:10.1007/978-3-642-01815-2\\_23](https://doi.org/10.1007/978-3-642-01815-2_23)> and in Ganter and Obiedkov (2016) <[doi:10.1007/978-3-662-49291-8](https://doi.org/10.1007/978-3-662-49291-8)>. It provides functions to load and save a formal context, extract its concept lattice and implications. In addition, one can use the implications to compute semantic closures of fuzzy sets and, thus, build recommendation systems.

**License** GPL-3

**URL** <https://github.com/Malaga-FCA-group/fcaR>

**BugReports** <https://github.com/Malaga-FCA-group/fcaR/issues>

**Depends** R (>= 3.1)

**Imports** forcats, fractional, glue, grDevices, Matrix, methods, POSetR, R6, Rcpp, registry, settings, stringr, tibble, tikzDevice, magrittr, purrr

**Suggests** arules, covr, hasseDiagram, knitr, markdown, rmarkdown, testthat (>= 2.1.0), tictoc, parallel

**LinkingTo** Rcpp

**VignetteBuilder** knitr

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.2.3

**NeedsCompilation** yes

**Author** Domingo Lopez Rodriguez [aut, cre]  
(<<https://orcid.org/0000-0002-0172-1585>>),  
Angel Mora [aut],  
Jesus Dominguez [aut],  
Ana Villalon [aut],  
Ian Johnson [ctb]

Repository CRAN

Date/Publication 2023-04-27 19:50:06 UTC

## R topics documented:

as_Set	2
as_vector	3
cobre32	4
cobre61	5
Concept	6
ConceptLattice	7
ConceptSet	12
equivalencesRegistry	14
fcaR	15
fcaR_options	16
FormalContext	17
ImplicationSet	27
parse_implication	32
parse_implications	33
planets	33
scalingRegistry	34
Set	35
vegas	37
%&%	38
%entails%	38
%==%	39
%-%	40
%holds_in%	41
%<=%	41
%or%	42
%respects%	43
%~%	43
<b>Index</b>	<b>45</b>

---

as\_Set

*Convert Named Vector to Set*

---

### Description

Convert Named Vector to Set

### Usage

as\_Set(A)

**Arguments**

A                    A named vector or matrix to build a new Set.

**Value**

A Set object.

**Examples**

```
A <- c(a = 0.1, b = 0.2, p = 0.3, q = 0)
as_Set(A)
```

---

as\_vector

*Convert Set to vector*

---

**Description**

Convert Set to vector

**Usage**

```
as_vector(v)
```

**Arguments**

v                    A Set to convert to vector.

**Value**

A vector.

**Examples**

```
A <- c(a = 0.1, b = 0.2, p = 0.3, q = 0)
v <- as_Set(A)
A2 <- as_vector(v)
all(A == A2)
```

---

cobre32

*Data for Differential Diagnosis for Schizophrenia*

---

### Description

A subset of the COBRE dataset has been retrieved, by querying SchizConnect for 105 patients with neurological and clinical symptoms, collecting also their corresponding diagnosis.

### Usage

cobre32

### Format

A matrix with 105 rows and 32 columns. Column names are related to different scales for depression and Schizophrenia:

**COSAS\_n** The *Simpson-Angus Scale*, 7 items to evaluate Parkinsonism-like alterations, related to schizophrenia, in an individual.

**FICAL\_n** The *Calgary Depression Scale for Schizophrenia*, 9 items (attributes) assessing the level of depression in schizophrenia, differentiating between positive and negative aspects of the disease.

**SCIDII\_n** The *Structured Clinical Interview for DSM-III-R Personality Disorders*, with 14 variables related to the presence of signs affecting personality.

**dx\_ss** if TRUE, the diagnosis is strict schizophrenia.

**dx\_other** if TRUE, the diagnosis is other than schizophrenia, including schizoaffective, bipolar disorder and major depression.

In summary, the dataset consists in the previous 30 attributes related to signs or symptoms, and 2 attributes related to diagnosis (these diagnoses are mutually exclusive, thus only one of them is assigned to each patient). This makes a dataset with 105 objects (patients) and 32 attributes to explore. The symptom attributes are multi-valued.

Thus, according to the specific scales used, all attributes are fuzzy and graded. For a given attribute (symptom), the available grades range from *absent* to *extreme*, with *minimal*, *mild*, *moderate*, *moderate severe* and *severe* in between.

These fuzzy attributes are mapped to values in the interval [0, 1].

### Source

Aine, C. J., Bockholt, H. J., Bustillo, J. R., Cañive, J. M., Caprihan, A., Gasparovic, C., ... & Liu, J. (2017). Multimodal neuroimaging in schizophrenia: description and dissemination. *Neuroinformatics*, 15(4), 343-364. <https://pubmed.ncbi.nlm.nih.gov/26142271/>

---

cobre61

*Data for Differential Diagnosis for Schizophrenia*

---

### Description

A subset of the COBRE dataset has been retrieved, by querying SchizConnect for 105 patients with neurological and clinical symptoms, collecting also their corresponding diagnosis.

### Usage

cobre61

### Format

A matrix with 105 rows and 61 columns. Column names are related to different scales for depression and Schizophrenia:

**COSAS\_n** The *Simpson-Angus Scale*, 7 items to evaluate Parkinsonism-like alterations, related to schizophrenia, in an individual.

**FIPAN\_n** The *Positive and Negative Syndrome Scale*, a set of 29 attributes measuring different aspects and symptoms in schizophrenia.

**FICAL\_n** The *Calgary Depression Scale for Schizophrenia*, 9 items (attributes) assessing the level of depression in schizophrenia, differentiating between positive and negative aspects of the disease.

**SCIDII\_n** The *Structured Clinical Interview for DSM-III-R Personality Disorders*, with 14 variables related to the presence of signs affecting personality.

**dx\_ss** if TRUE, the diagnosis is strict schizophrenia.

**dx\_other** if TRUE, the diagnosis is other than schizophrenia, including schizoaffective, bipolar disorder and major depression.

In summary, the dataset consists in the previous 59 attributes related to signs or symptoms, and 2 attributes related to diagnosis (these diagnoses are mutually exclusive, thus only one of them is assigned to each patient). This makes a dataset with 105 objects (patients) and 61 attributes to explore. The symptom attributes are multi-valued.

Thus, according to the specific scales used, all attributes are fuzzy and graded. For a given attribute (symptom), the available grades range from *absent* to *extreme*, with *minimal*, *mild*, *moderate*, *moderate severe* and *severe* in between.

These fuzzy attributes are mapped to values in the interval [0, 1].

### Source

Aine, C. J., Bockholt, H. J., Bustillo, J. R., Cañive, J. M., Caprihan, A., Gasparovic, C., ... & Liu, J. (2017). Multimodal neuroimaging in schizophrenia: description and dissemination. *Neuroinformatics*, 15(4), 343-364. <https://pubmed.ncbi.nlm.nih.gov/26142271/>

---

 Concept

*R6 class for a fuzzy concept with sparse internal representation*


---

## Description

This class implements the data structure and methods for fuzzy concepts.

## Methods

### Public methods:

- [Concept\\$new\(\)](#)
- [Concept\\$get\\_extent\(\)](#)
- [Concept\\$get\\_intent\(\)](#)
- [Concept\\$print\(\)](#)
- [Concept\\$to\\_latex\(\)](#)
- [Concept\\$clone\(\)](#)

**Method** `new()`: Creator for objects of class `Concept`

*Usage:*

`Concept$new(extent, intent)`

*Arguments:*

`extent` (Set) The extent of the concept.

`intent` (Set) The intent of the concept.

*Returns:* An object of class `Concept`.

**Method** `get_extent()`: Internal Set for the extent

*Usage:*

`Concept$get_extent()`

*Returns:* The Set representation of the extent.

**Method** `get_intent()`: Internal Set for the intent

*Usage:*

`Concept$get_intent()`

*Returns:* The Set representation of the intent.

**Method** `print()`: Prints the concept to console

*Usage:*

`Concept$print()`

*Returns:* A string with the elements of the set and their grades between brackets .

**Method** `to_latex()`: Write the concept in LaTeX format

*Usage:*

```
Concept$to_latex(print = TRUE)
```

*Arguments:*

print (logical) Print to output?

*Returns:* The fuzzy concept in LaTeX.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
Concept$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
# Build a formal context and find its concepts
fc_planets <- FormalContext$new(planets)
fc_planets$find_concepts()

# Print the first three concepts
fc_planets$concepts[1:3]

# Select the first concept:
C <- fc_planets$concepts$sub(1)

# Get its extent and intent
C$get_extent()
C$get_intent()
```

---

ConceptLattice

*R6 class for a concept lattice*

---

## Description

This class implements the data structure and methods for concept lattices.

## Super class

```
fcaR::ConceptSet -> ConceptLattice
```

## Methods

### Public methods:

- `ConceptLattice$new()`
- `ConceptLattice$plot()`
- `ConceptLattice$sublattice()`
- `ConceptLattice$top()`

- `ConceptLattice$bottom()`
- `ConceptLattice$join_irreducibles()`
- `ConceptLattice$meet_irreducibles()`
- `ConceptLattice$decompose()`
- `ConceptLattice$supremum()`
- `ConceptLattice$infimum()`
- `ConceptLattice$subconcepts()`
- `ConceptLattice$superconcepts()`
- `ConceptLattice$lower_neighbours()`
- `ConceptLattice$upper_neighbours()`
- `ConceptLattice$clone()`

**Method** `new()`: Create a new `ConceptLattice` object.

*Usage:*

`ConceptLattice$new(extents, intents, objects, attributes, I = NULL)`

*Arguments:*

`extents` (`dgMatrix`) The extents of all concepts

`intents` (`dgMatrix`) The intents of all concepts

`objects` (character vector) Names of the objects in the formal context

`attributes` (character vector) Names of the attributes in the formal context

`I` (`dgMatrix`) The matrix of the formal context

*Returns:* A new `ConceptLattice` object.

**Method** `plot()`: Plot the concept lattice

*Usage:*

`ConceptLattice$plot(object_names = TRUE, to_latex = FALSE, ...)`

*Arguments:*

`object_names` (logical) If TRUE, plot object names, otherwise omit them from the diagram.

`to_latex` (logical) If TRUE, export the plot as a `tikzpicture` environment that can be included in a LaTeX file.

... Other parameters to be passed to the `tikzDevice` that renders the lattice in LaTeX, or for the figure caption. See Details.

*Details:* Particular parameters that control the size of the `tikz` output are: `width`, `height` (both in inches), and `pointsize` (in points), that should be set to the font size used in the `documentclass` header in the LaTeX file where the code is to be inserted.

If a caption is provided, the whole `tikz` picture will be wrapped by a `figure` environment and the caption set.

*Returns:* If `to_latex` is FALSE, it returns nothing, just plots the graph of the concept lattice. Otherwise, this function returns the LaTeX code to reproduce the concept lattice.

**Method** `sublattice()`: Sublattice

*Usage:*

`ConceptLattice$sublattice(...)`

*Arguments:*

. . . See Details.

*Details:* As argument, one can provide both integer indices or Concepts, separated by commas. The corresponding concepts are used to generate a sublattice.

*Returns:* The generated sublattice as a new ConceptLattice object.

**Method top():** Top of a Lattice

*Usage:*

```
ConceptLattice$top()
```

*Returns:* The top of the Concept Lattice

*Examples:*

```
fc <- FormalContext$new(planets)
fc$find_concepts()
fc$concepts$top()
```

**Method bottom():** Bottom of a Lattice

*Usage:*

```
ConceptLattice$bottom()
```

*Returns:* The bottom of the Concept Lattice

*Examples:*

```
fc <- FormalContext$new(planets)
fc$find_concepts()
fc$concepts$bottom()
```

**Method join\_irreducibles():** Join-irreducible Elements

*Usage:*

```
ConceptLattice$join_irreducibles()
```

*Returns:* The join-irreducible elements in the concept lattice.

**Method meet\_irreducibles():** Meet-irreducible Elements

*Usage:*

```
ConceptLattice$meet_irreducibles()
```

*Returns:* The meet-irreducible elements in the concept lattice.

**Method decompose():** Decompose a concept as the supremum of meet-irreducible concepts

*Usage:*

```
ConceptLattice$decompose(C)
```

*Arguments:*

C A list of Concepts

*Returns:* A list, each field is the set of meet-irreducible elements whose supremum is the corresponding element in C.

**Method** supremum(): Supremum of Concepts

*Usage:*

ConceptLattice\$supremum(...)

*Arguments:*

... See Details.

*Details:* As argument, one can provide both integer indices or Concepts, separated by commas. The corresponding concepts are used to compute their supremum in the lattice.

*Returns:* The supremum of the list of concepts.

**Method** infimum(): Infimum of Concepts

*Usage:*

ConceptLattice\$infimum(...)

*Arguments:*

... See Details.

*Details:* As argument, one can provide both integer indices or Concepts, separated by commas. The corresponding concepts are used to compute their infimum in the lattice.

*Returns:* The infimum of the list of concepts.

**Method** subconcepts(): Subconcepts of a Concept

*Usage:*

ConceptLattice\$subconcepts(C)

*Arguments:*

C (numeric or SparseConcept) The concept to which determine all its subconcepts.

*Returns:* A list with the subconcepts.

**Method** superconcepts(): Superconcepts of a Concept

*Usage:*

ConceptLattice\$superconcepts(C)

*Arguments:*

C (numeric or SparseConcept) The concept to which determine all its superconcepts.

*Returns:* A list with the superconcepts.

**Method** lower\_neighbours(): Lower Neighbours of a Concept

*Usage:*

ConceptLattice\$lower\_neighbours(C)

*Arguments:*

C (SparseConcept) The concept to which find its lower neighbours

*Returns:* A list with the lower neighbours of C.

**Method** upper\_neighbours(): Upper Neighbours of a Concept

*Usage:*

```
ConceptLattice$upper_neighbours(C)
```

*Arguments:*

C (SparseConcept) The concept to which find its upper neighbours

*Returns:* A list with the upper neighbours of C.

**Method clone():** The objects of this class are cloneable with this method.

*Usage:*

```
ConceptLattice$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## Examples

```
# Build a formal context
fc_planets <- FormalContext$new(planets)

# Find the concepts
fc_planets$find_concepts()

# Find join- and meet- irreducible elements
fc_planets$concepts$join_irreducibles()
fc_planets$concepts$meet_irreducibles()

# Get concept support
fc_planets$concepts$support()

## -----
## Method `ConceptLattice$top`
## -----

fc <- FormalContext$new(planets)
fc$find_concepts()
fc$concepts$top()

## -----
## Method `ConceptLattice$bottom`
## -----

fc <- FormalContext$new(planets)
fc$find_concepts()
fc$concepts$bottom()
```

---

 ConceptSet

*R6 class for a set of concepts*


---

### Description

This class implements the data structure and methods for concept sets.

### Methods

#### Public methods:

- `ConceptSet$new()`
- `ConceptSet$size()`
- `ConceptSet$is_empty()`
- `ConceptSet$extents()`
- `ConceptSet$intents()`
- `ConceptSet$print()`
- `ConceptSet$to_latex()`
- `ConceptSet$to_list()`
- `ConceptSet$[]()`
- `ConceptSet$sub()`
- `ConceptSet$support()`
- `ConceptSet$clone()`

**Method** `new()`: Create a new ConceptLattice object.

*Usage:*

`ConceptSet$new(extents, intents, objects, attributes, I = NULL)`

*Arguments:*

`extents` (`dgCMatrix`) The extents of all concepts

`intents` (`dgCMatrix`) The intents of all concepts

`objects` (character vector) Names of the objects in the formal context

`attributes` (character vector) Names of the attributes in the formal context

`I` (`dgCMatrix`) The matrix of the formal context

*Returns:* A new ConceptLattice object.

**Method** `size()`: Size of the Lattice

*Usage:*

`ConceptSet$size()`

*Returns:* The number of concepts in the lattice.

**Method** `is_empty()`: Is the lattice empty?

*Usage:*

`ConceptSet$is_empty()`

*Returns:* TRUE if the lattice has no concepts.

**Method** extents(): Concept Extents

*Usage:*

ConceptSet\$extents()

*Returns:* The extents of all concepts, as a dgCMatrix.

**Method** intents(): Concept Intents

*Usage:*

ConceptSet\$intents()

*Returns:* The intents of all concepts, as a dgCMatrix.

**Method** print(): Print the Concept Set

*Usage:*

ConceptSet\$print()

*Returns:* Nothing, just prints the concepts

**Method** to\_latex(): Write in LaTeX

*Usage:*

ConceptSet\$to\_latex(print = TRUE, ncols = 1, numbered = TRUE, align = TRUE)

*Arguments:*

print (logical) Print to output?

ncols (integer) Number of columns of the output.

numbered (logical) Number the concepts?

align (logical) Align objects and attributes independently?

*Returns:* The LaTeX code to list all concepts.

**Method** to\_list(): Returns a list with all the concepts

*Usage:*

ConceptSet\$to\_list()

*Returns:* A list of concepts.

**Method** [(): Subsets a ConceptSet

*Usage:*

ConceptSet\$[(indices)

*Arguments:*

indices (numeric or logical vector) The indices of the concepts to return as a list of Concepts.

It can be a vector of logicals where TRUE elements are to be retained.

*Returns:* Another ConceptSet.

**Method** sub(): Individual Concepts

*Usage:*

ConceptSet\$sub(index)

*Arguments:*

index (numeric) The index of the concept to return.

*Returns:* The Concept.

**Method** support(): Get support of each concept

*Usage:*

```
ConceptSet$support()
```

*Returns:* A vector with the support of each concept.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
ConceptSet$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

**Examples**

```
# Build a formal context
fc_planets <- FormalContext$new(planets)

# Find the concepts
fc_planets$find_concepts()

# Find join- and meet- irreducible elements
fc_planets$concepts$join_irreducibles()
fc_planets$concepts$meet_irreducibles()
```

---

equivalencesRegistry *Equivalence Rules Registry*

---

**Description**

Equivalence Rules Registry

**Usage**

```
equivalencesRegistry
```

**Format**

An object of class `equivalence_registry` (inherits from `registry`) of length 6.

**Details**

This is a registry that stores the equivalence rules that can be applied using the `apply_rules()` method in an `ImplicationSet`.

One can obtain the list of available equivalence operators by: `equivalencesRegistry$get_entry_names()`

## Description

The aim of this package is to provide tools to perform fuzzy formal concept analysis (FCA) from within R. It provides functions to load and save a Formal Context, extract its concept lattice and implications. In addition, one can use the implications to compute semantic closures of fuzzy sets and, thus, build recommendation systems.

## Details

The fcaR package provides data structures which allow the user to work seamlessly with formal contexts and sets of implications. More explicitly, three main classes are implemented, using the R6 object-oriented-programming paradigm in R:

- `FormalContext` encapsulates the definition of a formal context  $(G, M, I)$ , being  $G$  the set of objects,  $M$  the set of attributes and  $I$  the (fuzzy) relationship matrix, and provides methods to operate on the context using FCA tools.
- `ImplicationSet` represents a set of implications over a specific formal context.
- `ConceptLattice` represents the set of concepts and their relationships, including methods to operate on the lattice.

Two additional helper classes are implemented:

- `Set` is a class solely used for visualization purposes, since it encapsulates in sparse format a (fuzzy) set.
- `Concept` encapsulates internally both extent and intent of a formal concept as `Set`. Since fcaR is an extension of the data model in the arules package, most of the methods and classes implemented interoperate with the main S4 classes in arules (transactions and rules).

## References

- Guigues J, Duquenne V (1986). "Familles minimales d'implications informatives résultant d'un tableau de données binaires." *Mathématiques et Sciences humaines*, 95, 5-18.
- Ganter B, Wille R (1999). *Formal concept analysis : mathematical foundations*. Springer. ISBN 3540627715.
- Cordero P, Enciso M, Mora Á, Pérez de Guzman I (2002). "SLFD Logic: Elimination of Data Redundancy in Knowledge Representation." *Advances in Artificial Intelligence - IBERAMIA 2002*, 2527, 141-150. doi: 10.1007/3-540-36131-6\_15 (URL: [http://doi.org/10.1007/3-540-36131-6\\_15](http://doi.org/10.1007/3-540-36131-6_15)).
- Belohlavek R (2002). "Algorithms for fuzzy concept lattices." In *Proc. Fourth Int. Conf. on Recent Advances in Soft Computing*. Nottingham, United Kingdom, 200-205.
- Hahsler M, Grun B, Hornik K (2005). "arules - a computational environment for mining association rules and frequent item sets." *J Stat Softw*, 14, 1-25.
- Mora A, Cordero P, Enciso M, Fortes I, Aguilera G (2012). "Closure via functional dependence simplification." *International Journal of Computer Mathematics*, 89(4), 510-526. Belohlavek R,

Cordero P, Enciso M, Mora Á, Vychodil V (2016). “Automated prover for attribute dependencies in data with grades.” *International Journal of Approximate Reasoning*, 70, 51-67.

### Examples

```
# Build a formal context
fc_planets <- FormalContext$new(planets)

# Find its concepts and implications
fc_planets$find_implications()

# Print the extracted implications
fc_planets$implications
```

---

fcaR_options	<i>Set or get options for fcaR</i>
--------------	------------------------------------

---

### Description

Set or get options for fcaR

### Usage

```
fcaR_options(...)
```

### Arguments

... Option names to retrieve option values or [key]=[value] pairs to set options.

### Supported options

The following options are supported

- `decimal_places(numeric;2)` The number of decimal places to show when printing or exporting to  $\LaTeX$  sets, implications, concepts, etc.
- `latex_size(character;"normalsize")` Size to use when exporting to LaTeX.
- `reduced\_lattice(logical;TRUE)` Plot the reduced concept lattice?

---

FormalContext	<i>R6 class for a formal context</i>
---------------	--------------------------------------

---

## Description

This class implements the data structure and methods for formal contexts.

## Public fields

I The table of the formal context as a matrix.  
attributes The attributes of the formal context.  
objects The objects of the formal context.  
grades\_set The set of degrees (in  $[0, 1]$ ) the whole set of attributes can take.  
expanded\_grades\_set The set of degrees (in  $[0, 1]$ ) each attribute can take.  
concepts The concept lattice associated to the formal context as a [ConceptLattice](#).  
implications A set of implications on the formal context as an [ImplicationSet](#).

## Methods

### Public methods:

- [FormalContext\\$new\(\)](#)
- [FormalContext\\$is\\_empty\(\)](#)
- [FormalContext\\$scale\(\)](#)
- [FormalContext\\$get\\_scales\(\)](#)
- [FormalContext\\$background\\_knowledge\(\)](#)
- [FormalContext\\$dual\(\)](#)
- [FormalContext\\$intent\(\)](#)
- [FormalContext\\$uparrow\(\)](#)
- [FormalContext\\$extent\(\)](#)
- [FormalContext\\$downarrow\(\)](#)
- [FormalContext\\$closure\(\)](#)
- [FormalContext\\$obj\\_concept\(\)](#)
- [FormalContext\\$att\\_concept\(\)](#)
- [FormalContext\\$is\\_concept\(\)](#)
- [FormalContext\\$is\\_closed\(\)](#)
- [FormalContext\\$clarify\(\)](#)
- [FormalContext\\$reduce\(\)](#)
- [FormalContext\\$standardize\(\)](#)
- [FormalContext\\$find\\_concepts\(\)](#)
- [FormalContext\\$find\\_implications\(\)](#)
- [FormalContext\\$to\\_transactions\(\)](#)

- `FormalContext$save()`
- `FormalContext$load()`
- `FormalContext$dim()`
- `FormalContext$print()`
- `FormalContext$to_latex()`
- `FormalContext$incidence()`
- `FormalContext$subcontext()`
- `FormalContext$[()]`
- `FormalContext$plot()`
- `FormalContext$use_logic()`
- `FormalContext$get_logic()`
- `FormalContext$use_connection()`
- `FormalContext$get_connection()`
- `FormalContext$clone()`

**Method** `new()`: Creator for the Formal Context class

*Usage:*

```
FormalContext$new(I, filename, remove_const = FALSE)
```

*Arguments:*

`I` (numeric matrix) The table of the formal context.

`filename` (character) Path of a file to import.

`remove_const` (logical) If TRUE, remove constant columns. The default is FALSE.

*Details:* Columns of `I` should be named, since they are the names of the attributes of the formal context.

If no `I` is used, the resulting `FormalContext` will be empty and not usable unless for loading a previously saved one. In this case, one can provide a `filename` to import. Only RDS, CSV and CXT files are currently supported.

*Returns:* An object of the `FormalContext` class.

**Method** `is_empty()`: Check if the `FormalContext` is empty

*Usage:*

```
FormalContext$is_empty()
```

*Returns:* TRUE if the `FormalContext` is empty, that is, has not been provided with a matrix, and FALSE otherwise.

**Method** `scale()`: Scale the context

*Usage:*

```
FormalContext$scale(attributes, type, ...)
```

*Arguments:*

`attributes` The attributes to scale

`type` Type of scaling.

...

*Details:* The types of scaling are implemented in a registry, so that `scalingRegistry$get_entries()` returns all types.

*Returns:* The scaled formal context

*Examples:*

```
filename <- system.file("contexts", "aromatic.csv", package = "fcaR")
fc <- FormalContext$new(filename)
fc$scale("nitro", "ordinal", comparison = `>=` , values = 1:3)
fc$scale("OS", "nominal", c("O", "S"))
fc$scale(attributes = "ring", type = "nominal")
```

**Method** `get_scales()`: Scales applied to the formal context

*Usage:*

```
FormalContext$get_scales(attributes = names(private$scales))
```

*Arguments:*

`attributes` (character) Name of the attributes for which scales (if applied) are returned.

*Returns:* The scales that have been applied to the specified attributes of the formal context. If no attributes are passed, then all applied scales are returned.

*Examples:*

```
filename <- system.file("contexts", "aromatic.csv", package = "fcaR")
fc <- FormalContext$new(filename)
fc$scale("nitro", "ordinal", comparison = `>=` , values = 1:3)
fc$scale("OS", "nominal", c("O", "S"))
fc$scale(attributes = "ring", type = "nominal")
fc$get_scales()
```

**Method** `background_knowledge()`: Background knowledge of a scaled formal context

*Usage:*

```
FormalContext$background_knowledge()
```

*Returns:* An `ImplicationSet` with the implications extracted from the application of scales.

*Examples:*

```
filename <- system.file("contexts", "aromatic.csv", package = "fcaR")
fc <- FormalContext$new(filename)
fc$scale("nitro", "ordinal", comparison = `>=` , values = 1:3)
fc$scale("OS", "nominal", c("O", "S"))
fc$scale(attributes = "ring", type = "nominal")
fc$background_knowledge()
```

**Method** `dual()`: Get the dual formal context

*Usage:*

```
FormalContext$dual()
```

*Returns:* A `FormalContext` where objects and attributes have interchanged their roles.

**Method** `intent()`: Get the intent of a fuzzy set of objects

*Usage:*

FormalContext\$intent(S)

*Arguments:*

S (Set) The set of objects to compute the intent for.

*Returns:* A Set with the intent.

**Method** uparrow(): Get the intent of a fuzzy set of objects

*Usage:*

FormalContext\$uparrow(S)

*Arguments:*

S (Set) The set of objects to compute the intent for.

*Returns:* A Set with the intent.

**Method** extent(): Get the extent of a fuzzy set of attributes

*Usage:*

FormalContext\$extent(S)

*Arguments:*

S (Set) The set of attributes to compute the extent for.

*Returns:* A Set with the intent.

**Method** downarrow(): Get the extent of a fuzzy set of attributes

*Usage:*

FormalContext\$downarrow(S)

*Arguments:*

S (Set) The set of attributes to compute the extent for.

*Returns:* A Set with the intent.

**Method** closure(): Get the closure of a fuzzy set of attributes

*Usage:*

FormalContext\$closure(S)

*Arguments:*

S (Set) The set of attributes to compute the closure for.

*Returns:* A Set with the closure.

**Method** obj\_concept(): Object Concept

*Usage:*

FormalContext\$obj\_concept(object)

*Arguments:*

object (character) Name of the object to compute its associated concept

*Returns:* The object concept associated to the object given.

**Method** att\_concept(): Attribute Concept

*Usage:*

FormalContext\$att\_concept(attribute)

*Arguments:*

attribute (character) Name of the attribute to compute its associated concept

*Returns:* The attribute concept associated to the attribute given.

**Method** is\_concept(): Is a Concept?

*Usage:*

FormalContext\$is\_concept(C)

*Arguments:*

C A Concept object

*Returns:* TRUE if C is a concept.

**Method** is\_closed(): Testing closure of attribute sets

*Usage:*

FormalContext\$is\_closed(S)

*Arguments:*

S A Set of attributes

*Returns:* TRUE if the set S is closed in this formal context.

**Method** clarify(): Clarify a formal context

*Usage:*

FormalContext\$clarify(copy = FALSE)

*Arguments:*

copy (logical) If TRUE, a new FormalContext object is created with the clarified context, otherwise the current one is overwritten.

*Returns:* The clarified FormalContext.

**Method** reduce(): Reduce a formal context

*Usage:*

FormalContext\$reduce(copy = FALSE)

*Arguments:*

copy (logical) If TRUE, a new FormalContext object is created with the clarified and reduced context, otherwise the current one is overwritten.

*Returns:* The clarified and reduced FormalContext.

**Method** standardize(): Build the Standard Context

*Usage:*

FormalContext\$standardize()

*Details:* All concepts must be previously computed.

*Returns:* The standard context using the join- and meet- irreducible elements.

**Method** `find_concepts()`: Use Ganter Algorithm to compute concepts

*Usage:*

```
FormalContext$find_concepts(verbose = FALSE)
```

*Arguments:*

`verbose` (logical) TRUE will provide a verbose output.

*Returns:* A list with all the concepts in the formal context.

**Method** `find_implications()`: Use modified Ganter algorithm to compute both concepts and implications

*Usage:*

```
FormalContext$find_implications(save_concepts = TRUE, verbose = FALSE)
```

*Arguments:*

`save_concepts` (logical) TRUE will also compute and save the concept lattice. FALSE is usually faster, since it only computes implications.

`verbose` (logical) TRUE will provide a verbose output.

*Returns:* Nothing, just updates the internal fields concepts and implications.

**Method** `to_transactions()`: Convert the formal context to object of class transactions from the arules package

*Usage:*

```
FormalContext$to_transactions()
```

*Returns:* A transactions object.

**Method** `save()`: Save a FormalContext to RDS or CXT format

*Usage:*

```
FormalContext$save(filename = tempfile(fileext = ".rds"))
```

*Arguments:*

`filename` (character) Path of the file where to store the FormalContext.

*Details:* The format is inferred from the extension of the filename.

*Returns:* Invisibly the current FormalContext.

**Method** `load()`: Load a FormalContext from a file

*Usage:*

```
FormalContext$load(filename)
```

*Arguments:*

`filename` (character) Path of the file to load the FormalContext from.

*Details:* Currently, only RDS, CSV and CXT files are supported.

*Returns:* The loaded FormalContext.

**Method** `dim()`: Dimensions of the formal context

*Usage:*

```
FormalContext$dim()
```

*Returns:* A vector with (number of objects, number of attributes).

**Method** `print()`: Prints the formal context

*Usage:*

```
FormalContext$print()
```

*Returns:* Prints information regarding the formal context.

**Method** `to_latex()`: Write the context in LaTeX format

*Usage:*

```
FormalContext$to_latex(table = TRUE, label = "", caption = "")
```

*Arguments:*

`table` (logical) If TRUE, surrounds everything between `\begin{table}` and `\end{table}`.

`label` (character) The label for the table environment.

`caption` (character) The caption of the table.

`fraction` (character) If none, no fractions are produced. Otherwise, if it is `frac`, `dfrac` or `sfrac`, decimal numbers are represented as fractions with the corresponding LaTeX type-setting.

*Returns:* A table environment in LaTeX.

**Method** `incidence()`: Incidence matrix of the formal context

*Usage:*

```
FormalContext$incidence()
```

*Returns:* The incidence matrix of the formal context

*Examples:*

```
fc <- FormalContext$new(planets)
fc$incidence()
```

**Method** `subcontext()`: Subcontext of the formal context

*Usage:*

```
FormalContext$subcontext(objects, attributes)
```

*Arguments:*

`objects` (character array) Name of the objects to keep.

`attributes` (character array) Names of the attributes to keep.

*Details:* A warning will be issued if any of the names is not present in the list of objects or attributes of the formal context.

If `objects` or `attributes` is empty, then it is assumed to represent the whole set of objects or attributes of the original formal context.

*Returns:* Another `FormalContext` that is a subcontext of the original one, with only the objects and attributes selected.

*Examples:*

```
fc <- FormalContext$new(planets)
fc$subcontext(attributes = c("moon", "no_moon"))
```

**Method** `[()]:` Subcontext of the formal context

*Usage:*

```
FormalContext$[(objects, attributes)]
```

*Arguments:*

`objects` (character array) Name of the objects to keep.

`attributes` (character array) Names of the attributes to keep.

*Details:* A warning will be issued if any of the names is not present in the list of objects or attributes of the formal context.

If `objects` or `attributes` is empty, then it is assumed to represent the whole set of objects or attributes of the original formal context.

*Returns:* Another `FormalContext` that is a subcontext of the original one, with only the objects and attributes selected.

*Examples:*

```
fc <- FormalContext$new(planets)
fc[, c("moon", "no_moon")]
```

**Method** `plot():` Plot the formal context table

*Usage:*

```
FormalContext$plot(to_latex = FALSE, ...)
```

*Arguments:*

`to_latex` (logical) If TRUE, export the plot as a `tikzpicture` environment that can be included in a LaTeX file.

... Other parameters to be passed to the `tikzDevice` that renders the lattice in LaTeX, or for the figure caption. See *Details*.

*Details:* Particular parameters that control the size of the `tikz` output are: `width`, `height` (both in inches), and `pointsize` (in points), that should be set to the font size used in the `documentclass` header in the LaTeX file where the code is to be inserted.

If a caption is provided, the whole `tikz` picture will be wrapped by a `figure` environment and the caption set.

*Returns:* If `to_latex` is FALSE, it returns nothing, just plots the graph of the formal context. Otherwise, this function returns the LaTeX code to reproduce the formal context plot.

**Method** `use_logic():` Sets the logic to use

*Usage:*

```
FormalContext$use_logic(name = available_logics())
```

*Arguments:*

`name` The name of the logic to use. To see the available names, run `available_logics()`.

**Method** `get_logic():` Gets the logic used

*Usage:*

```
FormalContext$get_logic()
```

*Returns:* A string with the name of the logic.

**Method** `use_connection()`: Sets the name of the Galois connection to use

*Usage:*

```
FormalContext$use_connection(connection)
```

*Arguments:*

`connection` The name of the Galois connection. Available connections are "standard" (anti-tone), "benevolent1" and "benevolent2" (isotone)

**Method** `get_connection()`: Gets the name of the Galois connection

*Usage:*

```
FormalContext$get_connection()
```

*Returns:* A string with the name of the Galois connection

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
FormalContext$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## References

Guigues J, Duquenne V (1986). "Familles minimales d'implications informatives résultant d'un tableau de données binaires." *Mathématiques et Sciences humaines*, 95, 5-18.

Ganter B, Wille R (1999). *Formal concept analysis : mathematical foundations*. Springer. ISBN 3540627715.

Belohlavek R (2002). "Algorithms for fuzzy concept lattices." In *Proc. Fourth Int. Conf. on Recent Advances in Soft Computing*. Nottingham, United Kingdom, 200-205.

Hahsler M, Grun B, Hornik K (2005). "arules - a computational environment for mining association rules and frequent item sets." *J Stat Softw*, 14, 1-25.

## Examples

```
# Build and print the formal context
fc_planets <- FormalContext$new(planets)
print(fc_planets)

# Define a set of attributes
S <- Set$new(attributes = fc_planets$attributes)
S$assign(moon = 1, large = 1)

# Compute the closure of S
Sc <- fc_planets$closure(S)
# Is Sc a closed set?
fc_planets$is_closed(Sc)

# Clarify and reduce the formal context
fc2 <- fc_planets$reduce(TRUE)
```

```

# Find implications
fc_planets$find_implications()

# Read a formal context from CSV
filename <- system.file("contexts", "airlines.csv", package = "fcaR")
fc <- FormalContext$new(filename)

# Read a formal context from a CXT file
filename <- system.file("contexts", "lives_in_water.cxt", package = "fcaR")
fc <- FormalContext$new(filename)

## -----
## Method `FormalContext$scale`
## -----

filename <- system.file("contexts", "aromatic.csv", package = "fcaR")
fc <- FormalContext$new(filename)
fc$scale("nitro", "ordinal", comparison = `>=` , values = 1:3)
fc$scale("OS", "nominal", c("0", "S"))
fc$scale(attributes = "ring", type = "nominal")

## -----
## Method `FormalContext$get_scales`
## -----

filename <- system.file("contexts", "aromatic.csv", package = "fcaR")
fc <- FormalContext$new(filename)
fc$scale("nitro", "ordinal", comparison = `>=` , values = 1:3)
fc$scale("OS", "nominal", c("0", "S"))
fc$scale(attributes = "ring", type = "nominal")
fc$get_scales()

## -----
## Method `FormalContext$background_knowledge`
## -----

filename <- system.file("contexts", "aromatic.csv", package = "fcaR")
fc <- FormalContext$new(filename)
fc$scale("nitro", "ordinal", comparison = `>=` , values = 1:3)
fc$scale("OS", "nominal", c("0", "S"))
fc$scale(attributes = "ring", type = "nominal")
fc$background_knowledge()

## -----
## Method `FormalContext$incidence`
## -----

fc <- FormalContext$new(planets)
fc$incidence()

## -----
## Method `FormalContext$subcontext`

```

```
## -----
fc <- FormalContext$new(planets)
fc$subcontext(attributes = c("moon", "no_moon"))

## -----
## Method `FormalContext$`
## -----

fc <- FormalContext$new(planets)
fc[, c("moon", "no_moon")]
```

---

ImplicationSet	<i>R6 Class for Set of implications</i>
----------------	---

---

## Description

This class implements the structure needed to store implications and the methods associated.

## Methods

### Public methods:

- `ImplicationSet$new()`
- `ImplicationSet$get_attributes()`
- `ImplicationSet$[]()`
- `ImplicationSet$to_arules()`
- `ImplicationSet$add()`
- `ImplicationSet$cardinality()`
- `ImplicationSet$is_empty()`
- `ImplicationSet$size()`
- `ImplicationSet$closure()`
- `ImplicationSet$recommend()`
- `ImplicationSet$apply_rules()`
- `ImplicationSet$to_basis()`
- `ImplicationSet$print()`
- `ImplicationSet$to_latex()`
- `ImplicationSet$get_LHS_matrix()`
- `ImplicationSet$get_RHS_matrix()`
- `ImplicationSet$filter()`
- `ImplicationSet$support()`
- `ImplicationSet$clone()`

**Method** `new()`: Initialize with an optional name

*Usage:*

ImplicationSet\$new(...)

*Arguments:*

... See Details.

*Details:* Creates and initialize a new ImplicationSet object. It can be done in two ways: initialize(name, attributes, lhs, rhs) or initialize(rules)

In the first way, the only mandatory argument is attributes, (character vector) which is a vector of names of the attributes on which we define the implications. Optional arguments are: name (character string), name of the implication set, lhs (a dgCMatrx), initial LHS of the implications stored and the analogous rhs.

The other way is used to initialize the ImplicationSet object from a rules object from package arules.

*Returns:* A new ImplicationSet object.

**Method** get\_attributes(): Get the names of the attributes

*Usage:*

ImplicationSet\$get\_attributes()

*Returns:* A character vector with the names of the attributes used in the implications.

**Method** [()]: Get a subset of the implication set

*Usage:*

ImplicationSet[(idx)

*Arguments:*

idx (integer or logical vector) Indices of the implications to extract or remove. If logical vector, only TRUE elements are retained and the rest discarded.

*Returns:* A new ImplicationSet with only the rules given by the idx indices (if all idx > 0 and all but idx if all idx < 0).

**Method** to\_arules(): Convert to arules format

*Usage:*

ImplicationSet\$to\_arules(quality = TRUE)

*Arguments:*

quality (logical) Compute the interest measures for each rule?

*Returns:* A rules object as used by package arules.

**Method** add(): Add a precomputed implication set

*Usage:*

ImplicationSet\$add(...)

*Arguments:*

... An ImplicationSet object, a rules object, or a pair lhs, rhs of Set objects or dgCMatrx. The implications to add to this formal context.

*Returns:* Nothing, just updates the internal implications field.

**Method** cardinality(): Cardinality: Number of implications in the set

*Usage:*

```
ImplicationSet$cardinality()
```

*Returns:* The cardinality of the implication set.

**Method** `is_empty()`: Empty set

*Usage:*

```
ImplicationSet$is_empty()
```

*Returns:* TRUE if the set of implications is empty, FALSE otherwise.

**Method** `size()`: Size: number of attributes in each of LHS and RHS

*Usage:*

```
ImplicationSet$size()
```

*Returns:* A vector with two components: the number of attributes present in each of the LHS and RHS of each implication in the set.

**Method** `closure()`: Compute the semantic closure of a fuzzy set with respect to the implication set

*Usage:*

```
ImplicationSet$closure(S, reduce = FALSE, verbose = FALSE)
```

*Arguments:*

S (a Set object) Fuzzy set to compute its closure. Use class Set to build it.

reduce (logical) Reduce the implications using simplification logic?

verbose (logical) Show verbose output?

*Returns:* If reduce == FALSE, the output is a fuzzy set corresponding to the closure of S. If reduce == TRUE, a list with two components: closure, with the closure as above, and implications, the reduced set of implications.

**Method** `recommend()`: Generate a recommendation for a subset of the attributes

*Usage:*

```
ImplicationSet$recommend(S, attribute_filter)
```

*Arguments:*

S (a vector) Vector with the grades of each attribute (a fuzzy set).

attribute\_filter (character vector) Names of the attributes to get recommendation for.

*Returns:* A fuzzy set describing the values of the attributes in attribute\_filter within the closure of S.

**Method** `apply_rules()`: Apply rules to remove redundancies

*Usage:*

```
ImplicationSet$apply_rules(
  rules = c("composition", "generalization"),
  batch_size = 25000L,
  parallelize = FALSE,
  reorder = FALSE
)
```

*Arguments:*

`rules` (character vector) Names of the rules to use. See details.

`batch_size` (integer) If the number of rules is large, apply the rules by batches of this size.

`parallelize` (logical) If possible, should we parallelize the computation among different batches?

`reorder` (logical) Should the rules be randomly reordered previous to the computation?

*Details:* Currently, the implemented rules are "generalization", "simplification", "reduction" and "composition".

*Returns:* Nothing, just updates the internal matrices for LHS and RHS.

**Method** `to_basis()`: Convert Implications to Canonical Basis*Usage:*

```
ImplicationSet$to_basis()
```

*Returns:* The canonical basis of implications obtained from the current ImplicationSet

**Method** `print()`: Print all implications to text*Usage:*

```
ImplicationSet$print()
```

*Returns:* A string with all the implications in the set.

**Method** `to_latex()`: Export to LaTeX*Usage:*

```
ImplicationSet$to_latex(
  print = TRUE,
  ncols = 1,
  numbered = TRUE,
  numbers = seq(self$cardinality())
)
```

*Arguments:*

`print` (logical) Print to output?

`ncols` (integer) Number of columns for the output.

`numbered` (logical) If TRUE (default), implications will be numbered in the output.

`numbers` (vector) If numbered, use these elements to enumerate the implications. The default is to enumerate 1, 2, ..., but can be changed.

*Returns:* A string in LaTeX format that prints nicely all the implications.

**Method** `get_LHS_matrix()`: Get internal LHS matrix*Usage:*

```
ImplicationSet$get_LHS_matrix()
```

*Returns:* A sparse matrix representing the LHS of the implications in the set.

**Method** `get_RHS_matrix()`: Get internal RHS matrix*Usage:*

```
ImplicationSet$get_RHS_matrix()
```

*Returns:* A sparse matrix representing the RHS of the implications in the set.

**Method** filter(): Filter implications by attributes in LHS and RHS

*Usage:*

```
ImplicationSet$filter(  
  lhs = NULL,  
  not_lhs = NULL,  
  rhs = NULL,  
  not_rhs = NULL,  
  drop = FALSE  
)
```

*Arguments:*

lhs (character vector) Names of the attributes to filter the LHS by. If NULL, no filtering is done on the LHS.

not\_lhs (character vector) Names of the attributes to not include in the LHS. If NULL (the default), it is not considered at all.

rhs (character vector) Names of the attributes to filter the RHS by. If NULL, no filtering is done on the RHS.

not\_rhs (character vector) Names of the attributes to not include in the RHS. If NULL (the default), it is not considered at all.

drop (logical) Remove the rest of attributes in RHS?

*Returns:* An ImplicationSet that is a subset of the current set, only with those rules which has the attributes in lhs and rhs in their LHS and RHS, respectively.

**Method** support(): Compute support of each implication

*Usage:*

```
ImplicationSet$support()
```

*Returns:* A vector with the support of each implication

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
ImplicationSet$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

## References

- Ganter B, Obiedkov S (2016). Conceptual Exploration. Springer. <https://doi.org/10.1007/978-3-662-49291-8>
- Hahsler M, Grun B, Hornik K (2005). “arules - a computational environment for mining association rules and frequent item sets.” *J Stat Softw*, 14, 1-25.
- Belohlavek R, Cordero P, Enciso M, Mora Á, Vychodil V (2016). “Automated prover for attribute dependencies in data with grades.” *International Journal of Approximate Reasoning*, 70, 51-67.
- Mora A, Cordero P, Enciso M, Fortes I, Aguilera G (2012). “Closure via functional dependence simplification.” *International Journal of Computer Mathematics*, 89(4), 510-526.

**Examples**

```
# Build a formal context
fc_planets <- FormalContext$new(planets)

# Find its implication basis
fc_planets$find_implications()

# Print implications
fc_planets$implications

# Cardinality and mean size in the ruleset
fc_planets$implications$cardinality()
sizes <- fc_planets$implications$size()
colMeans(sizes)

# Simplify the implication set
fc_planets$implications$apply_rules("simplification")
```

---

parse_implication	<i>Parses a string into an implication</i>
-------------------	--

---

**Description**

Parses a string into an implication

**Usage**

```
parse_implication(string, attributes)
```

**Arguments**

string	(character) The string to be parsed
attributes	(character vector) The attributes' names

**Value**

Two vectors as sparse matrices representing the LHS and RHS of the implication

---

parse_implications	<i>Parses several implications given as a string</i>
--------------------	--

---

**Description**

Parses several implications given as a string

**Usage**

```
parse_implications(input)
```

**Arguments**

input (character) The string with the implications or a file containing the implications

**Details**

The format for the input file is:

- Every implication in its own line or separated by semicolon (;)
- Attributes are separated by commas (,)
- The LHS and RHS of each implication are separated by an arrow (->)

**Value**

An ImplicationSet

**Examples**

```
input <- system.file("implications", "ex_implications", package = "fcaR")
imps <- parse_implications(input)
```

---

planets	<i>Planets data</i>
---------	---------------------

---

**Description**

This dataset records some properties of the planets in our solar system.

**Usage**

```
planets
```

**Format**

A matrix with 9 rows (the planets) and 7 columns, representing additional features of the planets:

**small** 1 if the planet is small, 0 otherwise.

**medium** 1 if the planet is medium-sized, 0 otherwise.

**large** 1 if the planet is large, 0 otherwise.

**near** 1 if the planet belongs in the inner solar system, 0 otherwise.

**far** 1 if the planet belongs in the outer solar system, 0 otherwise.

**moon** 1 if the planet has a natural moon, 0 otherwise.

**no\_moon** 1 if the planet has no moon, 0 otherwise.

**Source**

Wille R (1982). "Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts." In *Ordered Sets*, pp. 445–470. Springer.

---

scalingRegistry	<i>Scaling Registry</i>
-----------------	-------------------------

---

**Description**

Scaling Registry

**Usage**

scalingRegistry

**Format**

An object of class `scaling_registry` (inherits from `registry`) of length 6.

**Details**

This is a registry that stores the implemented scales that can be applied using the `scale()` method in an `FormalContext`.

One can obtain the list of available equivalence operators by: `scalingRegistry$get_entry_names()`

Set

*R6 class for a fuzzy set with sparse internal representation***Description**

This class implements the data structure and methods for fuzzy sets.

**Methods****Public methods:**

- `Set$new()`
- `Set$assign()`
- `Set$[()`
- `Set$cardinal()`
- `Set$get_vector()`
- `Set$get_attributes()`
- `Set$length()`
- `Set$print()`
- `Set$to_latex()`
- `Set$clone()`

**Method** `new()`: Creator for objects of class Set

*Usage:*

```
Set$new(attributes, M = NULL, ...)
```

*Arguments:*

`attributes` (character vector) Names of the attributes that will be available in the fuzzy set.

`M` (numeric vector or column Matrix) Values (grades) to be assigned to the attributes.

`...` `key = value` pairs, where the value `value` is assigned to the key attribute name.

*Details:* If `M` is omitted and no pair `key = value`, the fuzzy set is the empty set. Later, one can use the `assign` method to assign grades to any of its attributes.

*Returns:* An object of class Set.

**Method** `assign()`: Assign grades to attributes in the set

*Usage:*

```
Set$assign(attributes = c(), values = c(), ...)
```

*Arguments:*

`attributes` (character vector) Names of the attributes to assign a grade to.

`values` (numeric vector) Grades to be assigned to the previous attributes.

`...` `key = value` pairs, where the value `value` is assigned to the key attribute name.

*Details:* One can use both of: `S$assign(A = 1, B = 0.3)` `S$assign(attributes = c(A, B), values = c(1, 0.3))`.

**Method** `[]()`: Get elements by index

*Usage:*

`Set$[(indices)`

*Arguments:*

`indices` (numeric, logical or character vector) The indices of the elements to return. It can be a vector of logicals where TRUE elements are to be retained.

*Returns:* A Set but with only the required elements.

**Method** `cardinal()`: Cardinal of the Set

*Usage:*

`Set$cardinal()`

*Returns:* the cardinal of the Set, counted as the sum of the degrees of each element.

**Method** `get_vector()`: Internal Matrix

*Usage:*

`Set$get_vector()`

*Returns:* The internal sparse Matrix representation of the set.

**Method** `get_attributes()`: Attributes defined for the set

*Usage:*

`Set$get_attributes()`

*Returns:* A character vector with the names of the attributes.

**Method** `length()`: Number of attributes

*Usage:*

`Set$length()`

*Returns:* The number of attributes that are defined for this fuzzy set.

**Method** `print()`: Prints the set to console

*Usage:*

`Set$print(eol = TRUE)`

*Arguments:*

`eol` (logical) If TRUE, adds an end of line to the output.

*Returns:* A string with the elements of the set and their grades between brackets .

**Method** `to_latex()`: Write the set in LaTeX format

*Usage:*

`Set$to_latex(print = TRUE)`

*Arguments:*

`print` (logical) Print to output?

*Returns:* The fuzzy set in LaTeX.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`Set$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

**Examples**

```
S <- Set$new(attributes = c("A", "B", "C"))
S$assign(A = 1)
print(S)
S$to_latex()

S <- Set$new(c("A", "B", "C"), C = 1, B = 0.5)
S
```

vegas

*Data for Tourist Destination in Las Vegas***Description**

The dataset `vegas` is the binary translation of the Las Vegas Strip dataset (@moro2017stripping), which records more than 500 TripAdvisor reviews of hotels in Las Vegas Strip. The uninformative attributes (such as the user continent or the weekday of the review) are removed.

**Usage**

```
vegas
```

**Format**

A matrix with 504 rows and 25 binary columns. Column names are related to different features of the hotels:

**Period of Stay** 4 categories are present in the original data, which produces as many binary variables: Period of stay=Dec-Feb, Period of stay=Mar-May, Period of stay=Jun-Aug and Period of stay=Sep-Nov.

**Traveler type** Five binary categories are created from the original data: Traveler type=Business, Traveler type=Couples, Traveler type=Families, Traveler type=Friends and Traveler type=Solo.

**Pool, Gym, Tennis court, Spa, Casino, Free internet** Binary variables for the services offered by each destination hotel

**Stars** Five binary variables are created, according to the number of stars of the hotel, Stars=3, Stars=3.5, Stars=4, Stars=4.5 and Stars=5.

**Score** The score assigned in the review, from Score=1 to Score=5.

**Source**

Moro, S., Rita, P., & Coelho, J. (2017). Stripping customers' feedback on hotels through data mining: The case of Las Vegas Strip. *Tourism Management Perspectives*, 23, 41-52.

%%

*Intersection (Logical AND) of Fuzzy Sets***Description**

Intersection (Logical AND) of Fuzzy Sets

**Usage**

S1 %% S2

**Arguments**

S1	A Set
S2	A Set

**Details**

Both S1 and S2 must be Sets.

**Value**

Returns the intersection of S1 and S2.

**Examples**

```
# Build two sparse sets
S <- Set$new(attributes = c("A", "B", "C"))
S$assign(A = 1, B = 1)
T <- Set$new(attributes = c("A", "B", "C"))
T$assign(A = 1, C = 1)

# Intersection
S %% T
```

%entails%

*Entailment between implication sets***Description**

Entailment between implication sets

**Usage**

imps %entails% imps2

**Arguments**

`imps` (ImplicationSet) A set of implications.  
`imps2` (ImplicationSet) A set of implications which is tested to check if it follows semantically from `imps`.

**Value**

A logical vector, where element `k` is TRUE if the `k`-th implication in `imps2` follows from `imps`.

**Examples**

```
fc <- FormalContext$new(planets)
fc$find_implications()
imps <- fc$implications[1:4]$clone()
imps2 <- fc$implications[3:6]$clone()
imps %entails% imps2
```

**Description**

Equality in Sets and Concepts

**Usage**

```
C1 %==% C2
```

**Arguments**

`C1` A Set or Concept  
`C2` A Set or Concept

**Details**

Both `C1` and `C2` must be of the same class.

**Value**

Returns TRUE if `C1` is equal to `C2`.

**Examples**

```
# Build two sparse sets
S <- Set$new(attributes = c("A", "B", "C"))
S$assign(A = 1)
T <- Set$new(attributes = c("A", "B", "C"))
T$assign(A = 1)

# Test whether S and T are equal
S %==% T
```

---

*%-%**Difference in Sets*

---

**Description**

Difference in Sets

**Usage**

S1 %-% S2

**Arguments**

S1	A Set
S2	A Set

**Details**

Both S1 and S2 must be Sets.

**Value**

Returns the difference S1 - S2.

**Examples**

```
# Build two sparse sets
S <- Set$new(attributes = c("A", "B", "C"))
S$assign(A = 1, B = 1)
T <- Set$new(attributes = c("A", "B", "C"))
T$assign(A = 1)

# Difference
S %-% T
```

---

%holds\_in%                      *Implications that hold in a Formal Context*

---

**Description**

Implications that hold in a Formal Context

**Usage**

```
imps %holds_in% fc
```

**Arguments**

imps                      (ImplicationSet) The set of implications to test if hold in the formal context.  
fc                         (FormalContext) A formal context where to test if the implications hold.

**Value**

A logical vector, indicating if each implication holds in the formal context.

**Examples**

```
fc <- FormalContext$new(planets)  
fc$find_implications()  
imps <- fc$implications$clone()  
imps %holds_in% fc
```

---

%<=%                              *Partial Order in Sets and Concepts*

---

**Description**

Partial Order in Sets and Concepts

**Usage**

```
C1 %<=% C2
```

**Arguments**

C1                         A Set or Concept  
C2                         A Set or Concept

**Details**

Both C1 and C2 must be of the same class.

**Value**

Returns TRUE if concept C1 is subconcept of C2 or if set C1 is subset of C2.

**Examples**

```
# Build two sparse sets
S <- Set$new(attributes = c("A", "B", "C"))
S$assign(A = 1)
T <- Set$new(attributes = c("A", "B", "C"))
T$assign(A = 1, B = 1)

# Test whether S is subset of T
S %<=% T
```

---

 %or%

---

*Union (Logical OR) of Fuzzy Sets*


---

**Description**

Union (Logical OR) of Fuzzy Sets

**Usage**

```
S1 %|% S2
```

**Arguments**

S1	A Set
S2	A Set

**Details**

Both S1 and S2 must be Sets.

**Value**

Returns the union of S1 and S2.

**Examples**

```
# Build two sparse sets
S <- Set$new(attributes = c("A", "B", "C"))
S$assign(A = 1, B = 1)
T <- Set$new(attributes = c("A", "B", "C"))
T$assign(C = 1)

# Union
S %|% T
```

---

%respects%

*Check if Set or FormalContext respects an ImplicationSet*

---

**Description**

Check if Set or FormalContext respects an ImplicationSet

**Usage**

```
set %respects%imps
```

**Arguments**

set	(list of Sets, or a FormalContext) The sets of attributes to check whether they respect the ImplicationSet.
imps	(ImplicationSet) The set of implications to check.

**Value**

A logical matrix with as many rows as Sets and as many columns as implications in the ImplicationSet. A TRUE in element (i, j) of the result means that the i-th Set respects the j-th implication of the ImplicationSet.

**Examples**

```
fc <- FormalContext$new(planets)
fc$find_implications()
imps <- fc$implications$clone()
fc %respects%imps
```

---

%~%

*Equivalence of sets of implications*

---

**Description**

Equivalence of sets of implications

**Usage**

```
imps %~%imps2
```

**Arguments**

imps	A ImplicationSet.
imps2	Another ImplicationSet.

**Value**

TRUE if and only if `imps` and `imps2` are equivalent, that is, if every implication in `imps` follows from `imps2` and viceversa.

**Examples**

```
fc <- FormalContext$new(planets)
fc$find_implications()
imps <- fc$implications$clone()
imps2 <- imps$clone()
imps2$apply_rules(c("simp", "rsimp"))
imps %~% imps2
imps %~% imps2[1:9]
```

# Index

- \* **datasets**
  - cobre32, [4](#)
  - cobre61, [5](#)
  - equivalencesRegistry, [14](#)
  - planets, [33](#)
  - scalingRegistry, [34](#)
  - vegas, [37](#)
- %-%, [40](#)
- %<=%, [41](#)
- %==%, [39](#)
- %&&%, [38](#)
- %~%, [43](#)
- %entails%, [38](#)
- %holds\_in%, [41](#)
- %or%, [42](#)
- %respects%, [43](#)
  
- as\_Set, [2](#)
- as\_vector, [3](#)
  
- cobre32, [4](#)
- cobre61, [5](#)
- Concept, [6](#)
- ConceptLattice, [7](#), [17](#)
- ConceptSet, [12](#)
  
- equivalencesRegistry, [14](#)
  
- fcaR, [15](#)
- fcaR::ConceptSet, [7](#)
- fcaR\_options, [16](#)
- FormalContext, [17](#)
  
- ImplicationSet, [17](#), [27](#)
  
- parse\_implication, [32](#)
- parse\_implications, [33](#)
- planets, [33](#)
  
- scalingRegistry, [34](#)
- Set, [35](#)
  
- vegas, [37](#)