

# Package ‘fcaR’

January 12, 2026

**Title** Formal Concept Analysis

**Version** 1.3.0

**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. Matrix factorization is provided by the GreConD+ algorithm (Belohlavek and Trneckova, 2024 <[doi:10.1109/TFUZZ.2023.3330760](https://doi.org/10.1109/TFUZZ.2023.3330760)>).

**License** GPL-3

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

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

**Depends** R (>= 4.1)

**Imports** dplyr, forcats, ggplot2, ggraph, glue, grDevices, igraph,  
Matrix, methods, R6, rlang, Rcpp, registry, settings, stringr,  
tibble, tidyr, magrittr, purrr, yaml, cli, rstudioapi

**Suggests** arules, covr, DT, fractional, knitr, markdown, miniUI,  
rmarkdown, shiny, testthat (>= 2.1.0), tictoc, tikzDevice,  
tinytex, parallel

**LinkingTo** Rcpp, BH

**VignetteBuilder** knitr

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.3

**NeedsCompilation** yes

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

**Maintainer** Domingo Lopez Rodriguez <dominlopez78@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-01-12 10:30:02 UTC

## Contents

as_Set . . . . .	3
as_vector . . . . .	3
calculate_density . . . . .	4
calculate_grades . . . . .	4
calculate_separation . . . . .	5
calculate_stability . . . . .	5
cobre32 . . . . .	6
cobre61 . . . . .	7
compute_labels_and_colors . . . . .	8
Concept . . . . .	8
ConceptLattice . . . . .	10
conceptRegistry . . . . .	16
ConceptSet . . . . .	16
equivalencesRegistry . . . . .	19
export_to_tikz . . . . .	19
fcaR_options . . . . .	20
fetch_context . . . . .	21
FormalContext . . . . .	21
get_fcarepository_contexts . . . . .	32
ImplicationSet . . . . .	33
lattice_plot . . . . .	39
parse_implication . . . . .	40
parse_implications . . . . .	40
planets . . . . .	41
print_repo_details . . . . .	42
RandomContext . . . . .	42
RandomDistributiveContext . . . . .	43
randomize_context . . . . .	44
save_tikz . . . . .	45
scalingRegistry . . . . .	45
select_repository_context . . . . .	46
Set . . . . .	46
vegas . . . . .	48
%&% . . . . .	49
%entails% . . . . .	50
%==% . . . . .	51
%-% . . . . .	51
%holds_in% . . . . .	52
%<=% . . . . .	53
%or% . . . . .	54
%respects% . . . . .	54

<code>as_Set</code>	3
---------------------	---

<code>%~%</code> . . . . .	55
----------------------------	----

<b>Index</b>	56
--------------	----

---

<code>as_Set</code>	<i>Convert Named Vector to Set</i>
---------------------	------------------------------------

---

**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)
```

---

<code>as_vector</code>	<i>Convert Set to vector</i>
------------------------	------------------------------

---

**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)
```

---

calculate_density	<i>Calculate Fuzzy Density</i>
-------------------	--------------------------------

---

**Description**

Calculates the density of each concept in the original matrix I.

**Usage**

```
calculate_density(extents, intents, I)
```

**Arguments**

extents	A SparseSet or CsparseMatrix of extents.
intents	A SparseSet or CsparseMatrix of intents.
I	The original numeric matrix (FormalContext\$I).

**Value**

A numeric vector.

---

calculate_grades	<i>Calculate Concept Grades (Levels)</i>
------------------	--

---

**Description**

Calculates the grade (level) of each concept using the longest path from the bottom element. This is a fast C++ implementation.

**Usage**

```
calculate_grades(concept_ids, edge_from, edge_to)
```

**Arguments**

concept_ids	A vector of concept IDs (integers).
edge_from	A vector of source concept IDs from the cover relation (Hasse diagram).
edge_to	A vector of target concept IDs from the cover relation (Hasse diagram).

**Value**

An integer vector of the calculated grade for each concept ID.

---

calculate_separation	<i>Calculate Concept Separation</i>
----------------------	-------------------------------------

---

**Description**

Computes the separation of each concept. Separation is defined as the number of objects in a concept's extent that are NOT covered by any of its *immediate* subconcepts (children).

**Usage**

```
calculate_separation(lattice)
```

**Arguments**

lattice	A ConceptLattice object.
---------	--------------------------

**Value**

A numeric vector of separation values.

---

calculate_stability	<i>Calculate Concept Stability</i>
---------------------	------------------------------------

---

**Description**

Calculates the intensional stability of each concept.

Calculates the intensional stability of each concept in the lattice directly from the sparse matrix representation. Stability measures the probability that a concept is preserved when a random subset of objects is removed.

**Usage**

```
calculate_stability(extents)
```

```
calculate_stability(extents)
```

**Arguments**

extents	A SparseSet object or a sparse matrix (CsparseMatrix) representing concept extents (columns are concepts).
---------	--

**Value**

A numeric vector.

A numeric vector with stability values in  $[0, 1]$ .

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/>

---

compute_labels_and_colors
<i>Compute Labels and Colors for Lattice Nodes</i>

---

**Description**

Internal function to calculate node labels based on the selected mode.

**Usage**

```
compute_labels_and_colors(  
  nodes_df,  
  cover_edges,  
  extents,  
  intents,  
  obj_names,  
  att_names,  
  mode  
)
```

**Arguments**

nodes_df	Data frame with 'id'.
cover_edges	Data frame with 'from', 'to'.
extents	List of numeric vectors.
intents	List of numeric vectors.
obj_names	Character vector.
att_names	Character vector.
mode	Character: "full", "attributes", "reduced", "empty".

**Value**

A data frame extending nodes\_df with labels and colors.

---

Concept	<i>R6 class for a fuzzy concept with sparse internal representation</i>
---------	---

---

**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	<i>R6 class for a concept lattice</i>
----------------	---------------------------------------

---

**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$stability()`

- `ConceptLattice$separation()`
- `ConceptLattice$density()`
- `ConceptLattice$is_distributive()`
- `ConceptLattice$is_modular()`
- `ConceptLattice$is_semimodular()`
- `ConceptLattice$is_atomic()`
- `ConceptLattice$clone()`

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

*Usage:*

```
ConceptLattice$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** `plot()`: Plot the concept lattice

*Usage:*

```
ConceptLattice$plot(
  object_names = TRUE,
  to_latex = FALSE,
  method = c("sugiyama", "force"),
  mode = NULL,
  ...
)
```

*Arguments:*

`object_names` (logical) Deprecated. Use `mode` instead. If `TRUE` (default), implies `mode = "reduced"` or similar depending on heuristics. Kept for backward compatibility.

`to_latex` (logical) If `TRUE`, exports the plot as TikZ code (LaTeX) instead of drawing it. Returns an object of class `tikz_code` that prints the LaTeX code to console.

`method` (character) The layout algorithm to use. Options are:

- `"sugiyama"` (default): A hierarchical layout that minimizes edge crossings and centers nodes (similar to `ConExp` or `hasseDiagram`).
- `"force"`: A force-directed (spring) layout, useful for large or non-hierarchical lattices.

`mode` (character) The labeling mode for the nodes. If `NULL` (default), a heuristic based on lattice size is used. Options are:

- `"reduced"`: Standard FCA labeling. Nodes are labeled with an attribute (or object) only if they are the supreme (or infimum) of that attribute (or object).
- `"full"`: Each node shows its complete extent and intent.
- `"attributes"`: Nodes show only their intent (attributes).

- "empty": Nodes are drawn as points without labels. Recommended for very large lattices (>50 concepts).
- ... Other parameters passed to the internal plotting function (e.g., graphical parameters for ggraph).

*Returns:* If `to_latex` is FALSE, it returns (invisibly) the ggplot2 object representing the graph. If `to_latex` is TRUE, it returns a `tikz_code` object containing the LaTeX code.

**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** stability(): Computes the stability of each concept.

*Usage:*

ConceptLattice\$stability()

*Returns:* A numeric vector with the stability of each concept.

**Method** separation(): Computes the separation of each concept. Separation is the number of objects covered by the concept but not by any of its immediate subconcepts.

*Usage:*

ConceptLattice\$separation()

*Returns:* A numeric vector with the separation of each concept.

**Method** density(): Computes the fuzzy density of each concept.

*Usage:*

ConceptLattice\$density(I = NULL)

*Arguments:*

I (Optional) The original incidence matrix. If NULL, it tries to access it from the parent FormalContext if linked.

*Returns:* A numeric vector with the density of each concept.

**Method** is\_distributive(): Check if the lattice is distributive. A lattice is distributive if  $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$  for all elements.

*Usage:*

ConceptLattice\$is\_distributive()

*Returns:* Logical.

**Method** is\_modular(): Check if the lattice is modular. A lattice is modular if  $x \leq z \implies x \vee (y \wedge z) = (x \vee y) \wedge z$ . Distributive lattices are always modular.

*Usage:*

ConceptLattice\$is\_modular()

*Returns:* Logical.

**Method** is\_semimodular(): Check if the lattice is upper semimodular. A lattice is upper semimodular if for every  $x, y$ : if  $x$  covers  $x \wedge y$ , then  $x \vee y$  covers  $y$ .

*Usage:*

ConceptLattice\$is\_semimodular()

*Returns:* Logical.

**Method** is\_atomic(): Check if the lattice is atomic. A lattice is atomic if for every element  $x > \perp$ , there exists an atom  $a$  such that  $a \leq x$ . Atoms are elements that cover the bottom element.

*Usage:*

ConceptLattice\$is\_atomic()

*Returns:* Logical.

**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()
```

---

conceptRegistry	<i>Concept Miners Registry</i>
-----------------	--------------------------------

---

### Description

Concept Miners Registry

### Usage

conceptRegistry

### Format

An object of class `concept_miner_registry` (inherits from `registry`) of length 3.

### Details

This is a registry that stores the concept miners that can be applied using the `find_concepts()` method in an `FormalConcept`.

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

---

ConceptSet	<i>R6 class for a set of concepts</i>
------------	---------------------------------------

---

### 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$support()`
- `ConceptSet$stability()`
- `ConceptSet$clone()`

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

*Usage:*

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

*Arguments:*

`extents` (dgCMatrx) The extents of all concepts

`intents` (dgCMatrx) 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` (dgCMatrx) 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 dgCMatrx.

**Method** `intents()`: Concept Intents

*Usage:*

`ConceptSet$intents()`

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

**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** `stability()`: Compute the stability of each concept

*Usage:*

`ConceptSet$stability()`

*Returns:* A numeric vector with the stability 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	<i>Equivalence Rules Registry</i>
----------------------	-----------------------------------

---

## 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()

---

export_to_tikz	<i>Export Layout to TikZ (LaTeX)</i>
----------------	--------------------------------------

---

## Description

Generates TikZ code for the concept lattice.

**Usage**

```
export_to_tikz(
  plot_data,
  edges_df,
  width = 12,
  height = 12,
  standalone = FALSE,
  caption = NULL,
  ...
)
```

**Arguments**

plot_data	Data frame with node info.
edges_df	Data frame with edge info.
width	Numeric. Target width in cm (default: 12).
height	Numeric. Target height in cm (default: 12).
standalone	Logical. If TRUE, wraps code in a documentclass to be compiled directly.
caption	Character. Optional caption for the figure.
...	Additional arguments (currently unused).

**Value**

An object of class 'tikz\_code'.

---

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  $\text{\LaTeX}$  sets, implications, concepts, etc.

`latex_size` (character;"normalsize") Size to use when exporting to LaTeX.

`reduced\_lattice` (logical;TRUE) Plot the reduced concept lattice?

---

fetch_context	<i>Fetch a Formal Context from the FCA Repository</i>
---------------	---

---

**Description**

Downloads a context file from the [fcarepository.org](http://fcarepository.org) (via GitHub mirror), parses it, and returns a FormalContext object.

**Usage**

```
fetch_context(filename, verbose = TRUE)
```

**Arguments**

`filename` Character string. The ID/filename of the context (e.g., "animals\_en.cxt").

`verbose` Logical. If TRUE, prints metadata and progress messages using cli/glue.

**Value**

A FormalContext object.

---

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](#).

`description` An optional description of the dataset

## 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$factorize()`
- `FormalContext$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.

If the file is not present, the `fcarepository.org` is looked for coincidences. If so, the corresponding context is loaded.

*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.

In the dots argument, the user can supply the value for `bg` (logical), which, if set to TRUE, indicates to compute background knowledge as implications on the scales; if FALSE, no implications will be computed on the scales.

*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(method = "InClose", verbose = FALSE)`

*Arguments:*

`method` (string) The name of a method for the computation of concepts. Available options can be listed with `conceptRegistry$get_entries()`.

`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** `factorize()`: Factorize the formal context using Boolean/Fuzzy Matrix Factorization algorithms.

*Usage:*

```
FormalContext$factorize(method = "GreConD", ...)
```

*Arguments:*

`method` (character) The algorithm to use. Currently supported: "GreConD", "ASSO".

... Additional arguments:

- For GreConD: `w` (weight, default 1.0), `stop_threshold_ratio` (error tolerance, default 0.0).
- For ASSO: `threshold` (confidence threshold, default 0.7), `w_pos` (reward), `w_neg` (penalty).

*Returns:* A list with two `FormalContext` objects:

- `object_factor`: The context mapping Objects to Factors (Matrix A).
- `factor_attribute`: The context mapping Factors to Attributes (Matrix B).

**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")]

```

---

```
get_fcarepository_contexts
```

*Get Metadata from the FCA Repository*

---

## Description

Connects to the official FCA Repository (<https://fcarepository.org>) and downloads the metadata for all available datasets.

## Usage

```
get_fcarepository_contexts()
```

## Value

A list containing the metadata for each context (title, dimensions, description, source).

## Examples

```

## Not run:
meta <- get_fcarepository_contexts()

## End(Not run)

```



---

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$to_direct_optimal()`
- `ImplicationSet$print()`
- `ImplicationSet$to_latex()`
- `ImplicationSet$get_LHS_matrix()`
- `ImplicationSet$get_RHS_matrix()`
- `ImplicationSet$filter()`
- `ImplicationSet$support()`
- `ImplicationSet$use_logic()`
- `ImplicationSet$get_logic()`
- `ImplicationSet$use_hedge()`
- `ImplicationSet$get_hedge()`
- `ImplicationSet$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 dgCMatrix), 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 dgCMatrix. 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** `to_direct_optimal()`: Compute the Direct Optimal Basis using optimized C++ algorithms.

*Usage:*

```
ImplicationSet$to_direct_optimal(
  method = c("direct_optimal", "final_ts", "monotonic", "priority"),
  verbose = FALSE
)
```

*Arguments:*

`method` (character) The specific algorithm to run:

- "direct\_optimal": (Default) The Direct Optimal Saturation-Pruning algorithm.
- "final\_ts": Computes Transitive Closure then Prunes (Standard approach).
- "monotonic": Incremental algorithm maintaining monotonicity.
- "priority": Priority-based refinement algorithm.

`verbose` (logical) Print verbose output from the C++ backend.

*Returns:* Nothing, updates the ImplicationSet in place with the new basis.

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

*Usage:*

```
ImplicationSet$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:*

```
ImplicationSet$get_logic()
```

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

**Method** `use_hedge()`: Sets the hedge to use when computing closures

*Usage:*

```
ImplicationSet$use_hedge(name = c("globalization", "identity"))
```

*Arguments:*

name The name of the hedge to use. Only "globalization" and "identity" are allowed.

**Method** `get_hedge()`: Gets the hedge used to compute closures

*Usage:*

```
ImplicationSet$get_hedge()
```

*Returns:* A string with the name of the hedge

**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")

```

lattice\_plot

*Plot Concept Lattice***Description**

Visualization of the concept lattice using 'ggraph'.

**Usage**

```

lattice_plot(
  nodes_df,
  cover_matrix,
  method = "sugiyama",
  mode = NULL,
  objects = NULL,
  attributes = NULL,
  object_names = TRUE,
  to_latex = FALSE,
  extents = NULL,
  intents = NULL,
  ...
)

```

**Arguments**

nodes_df	Data frame with 'id'.
cover_matrix	Sparse matrix.
method	Layout method ("sugiyama", "force").
mode	Labeling mode ("reduced", "full", "empty").
objects	Character vector.
attributes	Character vector.
object_names	Logical (Deprecated).
to_latex	Logical.

extents	List of extents.
intents	List of intents.
...	Extra args.

---

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

*Planets data*

---

**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.

---

print_repo_details	<i>Print Details of Repository Contexts</i>
--------------------	---

---

### Description

Prints a formatted summary of the contexts available in the FCA Repository to the console. It displays the filename, title, dimensions (objects x attributes), and description for each entry.

### Usage

```
print_repo_details(meta)
```

### Arguments

meta	A list of metadata objects, typically obtained via <a href="#">get_fcarepository_contexts</a> .
------	---

### Value

Prints the summary to the console. Returns NULL invisibly.

### Examples

```
## Not run:
meta <- get_fcarepository_contexts()
print_repo_details(meta)

## End(Not run)
```

---

RandomContext	<i>Generate Random Formal Contexts</i>
---------------	--

---

### Description

Functions to generate synthetic formal contexts using advanced statistical distributions. These methods allow creating datasets that mimic real-world properties (non-uniform density) or randomizing existing contexts while preserving their structural properties.

### Usage

```
RandomContext(
  n_objects,
  n_attributes,
  density = 0.1,
  distribution = "uniform",
  alpha = 1,
  ...
)
```

**Arguments**

<code>n_objects</code>	(integer) Number of objects.
<code>n_attributes</code>	(integer) Number of attributes.
<code>density</code>	(numeric) Expected density of the context (proportion of 1s). Used for uniform distribution.
<code>distribution</code>	(character) The distribution to use for generating the context. <ul style="list-style-type: none"> <li>• "uniform": Each cell is 1 with probability density.</li> <li>• "dirichlet": The number of attributes per object follows a categorical distribution derived from a Dirichlet distribution. This creates "clumpy" or "sparse" rows typical of real data.</li> </ul>
<code>alpha</code>	(numeric) Concentration parameter for the Dirichlet distribution. Low values (e.g., 0.1) produce very skewed distributions (some objects have few attributes, others many). High values produce more uniform row sums. Default is 1.0.
<code>...</code>	Additional arguments passed to internal methods.

**Value**

A FormalContext object.

**Examples**

```
# 1. Uniform Random Context
fc_uni <- RandomContext(10, 5, density = 0.2)
print(fc_uni)

# 2. Dirichlet Random Context (Mimicking real data structure)
# Objects will have varying 'sizes' (number of attributes)
fc_dir <- RandomContext(10, 5, distribution = "dirichlet", alpha = 0.5)
print(fc_dir)
```

---

RandomDistributiveContext

*Generate a Random Distributive Context*

---

**Description**

Generates a random formal context that is guaranteed to produce a **Distributive Concept Lattice**.

It relies on Birkhoff's Representation Theorem: The lattice of order ideals of a Poset is always distributive. The context is constructed such that objects and attributes are the elements of the poset, and the incidence relation is  $gIm \iff \neg(g \geq m)$ .

**Usage**

```
RandomDistributiveContext(n_elements, density = 0.1)
```

**Arguments**

n_elements	Number of elements in the underlying Poset.
density	Probability of an order relation $a \leq b$ .

**Value**

A FormalContext.

---

randomize_context	<i>Randomize an Existing Formal Context</i>
-------------------	---

---

**Description**

Modifies the incidence matrix of a formal context to create a random variation while preserving certain statistical properties. This is essential for statistical significance testing in FCA (e.g., "is this concept structure random?").

**Usage**

```
randomize_context(fc, method = "swap", iterations = NULL)
```

**Arguments**

fc	(FormalContext) The context to randomize.
method	(character) The randomization strategy: <ul style="list-style-type: none"> <li>• "swap": Edge Swapping (Curveball algorithm). Preserves exact row sums and column sums (marginal distributions). The structure changes, but the statistics of objects and attributes remain identical.</li> <li>• "rewire": Edge Rewiring. Preserves only the global density (total number of 1s). Row and column sums may change.</li> </ul>
iterations	(integer) Number of swap/rewire operations to perform. Default is $10 \times$ number of 1s, which is usually sufficient for mixing.

**Value**

A new FormalContext object with the randomized incidence.

**Examples**

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

# 1. Edge Swapping (Preserves degree distribution)
# Useful for null-model testing
fc_rand_swap <- randomize_context(fc, method = "swap")

# Verify marginals are preserved
```

```
colSums(fc$incidence())
colSums(fc_rand_swap$incidence())

# 2. Rewiring (Preserves only density)
fc_rand_rewire <- randomize_context(fc, method = "rewire")
```

---

save\_tikz

*Save TikZ Code to File*


---

### Description

Exports the generated TikZ code to a .tex file.

### Usage

```
save_tikz(x, file, ...)
```

### Arguments

x	An object of class 'tikz_code'.
file	Character. Path to the output file.
...	Additional arguments.

---

scalingRegistry

*Scaling Registry*


---

### 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()

---

```
select_repository_context
```

*GUI to select and download a context from the repository*

---

### Description

GUI to select and download a context from the repository

### Usage

```
select_repository_context(meta)
```

### Arguments

meta                      A list of metadata objects (obtained via `get_fcarepository_contexts`).

---

```
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%	<i>Entailment between implication sets</i>
-----------	--

---

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
```

---

%==%

*Equality in Sets and Concepts*

---

### 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%	<i>Implications that hold in a Formal Context</i>
------------	---

---

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%	<i>Union (Logical OR) of Fuzzy Sets</i>
------	---

---

**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%	<i>Check if Set or FormalContext respects an ImplicationSet</i>
------------	---

---

**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 of 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, [6](#)
- cobre61, [7](#)
- conceptRegistry, [16](#)
- equivalencesRegistry, [19](#)
- planets, [41](#)
- scalingRegistry, [45](#)
- vegas, [48](#)

%-, [51](#)

%<=, [53](#)

%==, [51](#)

%&%, [49](#)

%~%, [55](#)

%entails%, [50](#)

%holds\_in%, [52](#)

%or%, [54](#)

%respects%, [54](#)

as\_Set, [3](#)

as\_vector, [3](#)

calculate\_density, [4](#)

calculate\_grades, [4](#)

calculate\_separation, [5](#)

calculate\_stability, [5](#)

cobre32, [6](#)

cobre61, [7](#)

compute\_labels\_and\_colors, [8](#)

Concept, [8](#)

ConceptLattice, [10](#), [21](#)

conceptRegistry, [16](#)

ConceptSet, [16](#)

equivalencesRegistry, [19](#)

export\_to\_tikz, [19](#)

fcaR::ConceptSet, [10](#)

fcaR\_options, [20](#)

fetch\_context, [21](#)

FormalContext, [21](#)

get\_fcarepository\_contexts, [32](#), [42](#)

ImplicationSet, [21](#), [33](#)

lattice\_plot, [39](#)

parse\_implication, [40](#)

parse\_implications, [40](#)

planets, [41](#)

print\_repo\_details, [42](#)

RandomContext, [42](#)

RandomDistributiveContext, [43](#)

randomize\_context, [44](#)

save\_tikz, [45](#)

scalingRegistry, [45](#)

select\_repository\_context, [46](#)

Set, [46](#)

vegas, [48](#)