

Package ‘CORAtool’

October 5, 2026

Type Package

Title Combinational Regularity Analysis

Version 0.1.2

Description Searches configurational data for causes that are each an insufficient but non-redundant part of an unnecessary but sufficient (INUS) condition for their effect, so that cause-effect relations are marked by conjunctivity and disjunctivity. The method, Combinational Regularity Analysis (CORA), borrows its Boolean minimisation algorithms from switching circuit analysis. Truth tables are minimised either with the classical Quine-McCluskey algorithm over positive and don't care terms or with McCluskey's modified algorithm over positive and negative terms, and the resulting prime implicant charts are solved with Petrick's method. Multi-value conditions and structures with simple as well as complex effects are supported, together with a configurational data-mining search and two-level logic diagrams. The package is an R port of the 'Python' packages 'CORA' and 'LOGIGRAM' described in Sebechlebská, Mkrtychyan and Thiem (2023) <doi:10.21105/joss.05019>; it computes in plain R and requires no 'Python' installation. It is an independent implementation and is not endorsed by the authors of the original packages.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports graphics, stats, utils

Suggests testthat (>= 3.0.0), reticulate, knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

URL <https://github.com/youngchanresearcher/CORAtool>

BugReports <https://github.com/youngchanresearcher/CORAtool/issues>

NeedsCompilation no

RoxygenNote 7.3.1

Author Young Chan [aut, cre, cph] (Author of the R implementation),
Zuzana Sebechlebská [cph] (Copyright holder of the original Python
implementation),
Lusine Mkrtchyan [cph] (Copyright holder of the original Python
implementation),
Alrik Thiem [cph] (Copyright holder of the original Python
implementation)

Maintainer Young Chan <youngchanresearcher@gmail.com>

Repository CRAN

Date/Publication 2026-10-05 16:00:09 UTC

Contents

bergschlosser	2
cora_compare_python	3
cora_context	4
cora_coverage_score	5
cora_data_mining	6
cora_describe	7
cora_dnf	8
cora_irredundant_sums	9
cora_irredundant_systems	10
cora_logigram	11
cora_petrick	12
cora_pi_chart	13
cora_pi_details	14
cora_prime_implicants	14
cora_python_available	15
cora_recode	15
cora_solutions	16
cora_system_details	17
cora_truth_table	17
gross_carvin	18
mccluskey	19
swiss_minaret	19
Index	20

bergschlosser	<i>Berg-Schlosser and De Meur's praetorianism data</i>
---------------	--

Description

Multi-value data on 48 African countries with three outcome columns, AUTH, DEM and PRAET.

Usage

```
bergschlosser
```

Format

A data frame with 48 rows: the case label Case, the conditions AGRPOP, PARCL, APROG, PS, RQ and LRC, and the outcomes AUTH, DEM and PRAET.

Source

Shipped with the Python CORA package, <https://github.com/PoliUniLu/cora>.

Examples

```
ctx <- cora_context(bergschlosser, "PRAET",
  input_labels = c("PS", "RQ", "LRC", "AUTH"),
  inc_score1 = 0.6, case_col = "Case")
cora_irredundant_sums(ctx)
```

cora_compare_python	<i>Cross-check a result against the Python implementation</i>
---------------------	---

Description

Runs the same analysis through the original Python cora package and compares the prime implicants and the irredundant solutions. Results are compared as sets: the two implementations enumerate solutions in different orders, so the running numbers of the solutions need not line up.

Usage

```
cora_compare_python(ctx)
```

Arguments

ctx A `cora_context()`.

Value

An object of class `cora_comparison`: a list holding the two summaries (`r`, `python`), a logical `agrees`, and for each of `prime_implicants` and `solutions` a list of `agree`, `r_only` and `python_only`. Nothing is written to the console while it is computed; printing the object gives a short report of what agrees and what does not.

Examples

```
## Not run:
## Needs a Python installation carrying the original `cora` package, so it
## is not run automatically. See `cora_python_available()`.
df <- data.frame(A = c(1, 0, 1, 0), B = c(1, 0, 0, 1),
                 C = c(0, 1, 1, 0), OUT = c(1, 1, 0, 1))
if (cora_python_available()) {
  cora_compare_python(cora_context(df, "OUT"))
}

## End(Not run)
```

cora_context	<i>Create an optimisation context</i>
--------------	---------------------------------------

Description

Bundles the data and the analytical choices that drive a Combinational Regularity Analysis. The context is evaluated lazily: the truth table, the prime implicants and the irredundant solutions are each computed on first request and cached afterwards.

Usage

```
cora_context(
  data,
  output_labels,
  input_labels = NULL,
  case_col = NULL,
  n_cut = 1,
  inc_score1 = 1,
  inc_score2 = NULL,
  U = NULL,
  rename_columns = FALSE,
  algorithm = c("ON-DC", "ON-OFF")
)
```

Arguments

data	A data frame. Input columns must hold non-negative integers coded from zero upwards; output columns must be binary unless their analysed values are declared in output_labels.
output_labels	Character vector naming the outcome columns. A multi-value outcome declares the values that count as positive in curly brackets, e.g. "OUT{1,2}".
input_labels	Character vector naming the input columns. Defaults to every column that is neither an outcome nor the case column.
case_col	Name of the column holding case identifiers, or NULL.

n_cut	Minimum number of cases below which a truth table row is declared a don't care.
inc_score1	Minimum sufficiency inclusion score for an output function value of 1.
inc_score2	Maximum sufficiency inclusion score for an output function value of 0, or NULL.
U	Either 0 or 1; required when inc_score2 is given.
rename_columns	If TRUE, input columns are renamed to single letters in alphabetical order.
algorithm	"ON-DC" (the classical Quine-McCluskey algorithm over positive and don't care terms) or "ON-OFF" (McCluskey's modified algorithm over positive and negative terms).

Value

An object of class cora_context.

See Also

[cora_truth_table\(\)](#), [cora_prime_implicants\(\)](#), [cora_irredundant_sums\(\)](#), [cora_irredundant_systems\(\)](#)

Examples

```
df <- data.frame(A = c(1, 0, 1, 0), B = c(1, 0, 0, 1),
                 C = c(0, 1, 1, 0), OUT = c(1, 1, 0, 1))
ctx <- cora_context(df, output_labels = "OUT")
cora_prime_implicants(ctx)
```

cora_coverage_score	<i>Sufficiency statistics of a prime implicant</i>
---------------------	--

Description

The coverage score is the share of the cases showing the outcome that the prime implicant covers.
The inclusion score is the share of the cases the prime implicant covers that show the outcome.

Usage

```
cora_coverage_score(x, ...)

cora_inclusion_score(x, ...)
```

Arguments

x A prime implicant, as returned by [cora_prime_implicants\(\)](#).
... Unused.

Value

A single number, or NaN when the denominator is empty.

Examples

```
df <- data.frame(A = c(1, 0, 1, 0), B = c(1, 0, 0, 1),
                 C = c(0, 1, 1, 0), OUT = c(1, 1, 0, 1))
pis <- cora_prime_implicants(cora_context(df, "OUT"))
cora_coverage_score(pis[[1]])
cora_inclusion_score(pis[[1]])
```

cora_data_mining

Configurational data mining

Description

Analyses every n-tuple of input variables in search of tuples that generate a solution, which amounts to a configurational version of Occam's razor: it keeps the number of inputs required for a solution at a minimum.

Usage

```
cora_data_mining(
  data,
  output_labels,
  len_of_tuple,
  input_labels = NULL,
  case_col = NULL,
  n_cut = 1,
  inc_score1 = 1,
  inc_score2 = NULL,
  U = NULL,
  algorithm = c("ON-DC", "ON-OFF"),
  automatic = FALSE
)
```

Arguments

data	A data frame.
output_labels	Character vector naming the outcome columns, in the notation accepted by cora_context() .
len_of_tuple	Number of input variables to combine.
input_labels	Character vector naming the input columns to draw from. Defaults to every column that is neither an outcome nor the case column.
case_col	Name of the column holding case identifiers, or NULL.

n_cut	Minimum number of cases below which a truth table row is declared a don't care.
inc_score1	Minimum sufficiency inclusion score for an output function value of 1.
inc_score2	Maximum sufficiency inclusion score for an output function value of 0, or NULL.
U	Either 0 or 1; required when inc_score2 is given.
algorithm	"ON-DC" or "ON-OFF".
automatic	If TRUE, the search widens the tuple length until a non-zero solution is found.

Value

A data frame with one row per tuple, holding the number of irredundant solutions and the best inclusion, coverage and combined score across them.

Note

A tuple whose only solution is the tautology 1 explains nothing, and is reported with zero solutions and zero scores. The Python implementation intends the same but never reaches the case, because it compares the solution against "1" after the tautology has already been marked essential and renamed to "#1".

Examples

```
data <- data.frame(A = c(1, 1, 1, 0), B = c(0, 1, 0, 1),
                  C = c(1, 1, 0, 0), D = c(0, 1, 0, 1))
cora_data_mining(data, "0", len_of_tuple = 2)
```

cora_describe	<i>Descriptive rendering of a solution</i>
---------------	--

Description

Renders a solution as a sufficiency, necessity or equivalence statement, depending on whether its inclusion and coverage scores clear the thresholds of the analysis.

Usage

```
cora_describe(x, cov = 1, ...)
```

Arguments

x	A solution from cora_irredundant_sums() or cora_irredundant_systems() .
cov	Minimum coverage score for the relation to be read as necessary as well as sufficient.
...	Unused.

Value

A character string.

Examples

```
df <- data.frame(A = c(1, 0, 1, 0), B = c(1, 0, 0, 1),
                 C = c(0, 1, 1, 0), OUT = c(1, 1, 0, 1))
sums <- cora_irredundant_sums(cora_context(df, "OUT"))
cora_describe(sums[[1]])
```

cora_dnf	<i>Disjunctive normal form of a solution</i>
----------	--

Description

Renders a solution in the "A*B+c<=>F" notation that [cora_logigram\(\)](#) draws.

Usage

```
cora_dnf(x, ...)
```

Arguments

- x A solution from [cora_irredundant_sums\(\)](#) or [cora_irredundant_systems\(\)](#).
- ... Unused.

Value

A character vector with one entry per outcome.

Examples

```
df <- data.frame(A = c(1, 0, 1, 0), B = c(1, 0, 0, 1),
                 C = c(0, 1, 1, 0), OUT = c(1, 1, 0, 1))
cora_dnf(cora_irredundant_sums(cora_context(df, "OUT"))[[1]])
```

cora_irredundant_sums *Irredundant sums of a single-outcome analysis*

Description

Solves the prime implicant chart with Petrick's method and returns every irredundant sum of prime implicants.

Usage

```
cora_irredundant_sums(
  ctx,
  max_depth = NULL,
  search = c("bounded", "exhaustive")
)
```

Arguments

ctx	A cora_context() with exactly one outcome.
max_depth	Optional upper bound on the number of prime implicants a solution may contain. The restriction applies to the call, never to the context: the next call without it still sees every solution.
search	How max_depth is applied. "bounded", the default, stops Petrick's method from building solutions longer than the bound in the first place, which is often the difference between an answer and no answer at all; solutions are numbered from 1 within the restricted set. "exhaustive" solves the chart in full and then filters, so each solution keeps the number it has in the unrestricted set and a restricted call can return M2 and M5. Both return the same solutions. Ignored when max_depth is not given.

Value

A list of solutions, of class cora_systems.

Note

The number of irredundant sums can grow exponentially with the size of the prime implicant chart. A chart of a few dozen prime implicants can have tens of thousands of solutions, which is a sign that the analysis has too many conditions or too loose a threshold rather than a result to report; max_depth is the way to ask a narrower question.

Examples

```
df <- data.frame(A = c(1, 0, 1, 0), B = c(1, 0, 0, 1),
                  C = c(0, 1, 1, 0), OUT = c(1, 1, 0, 1))
cora_irredundant_sums(cora_context(df, "OUT"))
```

```
## Only the solutions built from at most one prime implicant.
cora_irredundant_sums(cora_context(df, "OUT"), max_depth = 1)
```

cora_irredundant_systems

Irredundant systems of a multi-outcome analysis

Description

Solves the prime implicant chart of every outcome and combines the single-outcome solutions into irredundant systems. Individual functions inside a system need not be irredundant, but the system as a whole is.

Usage

```
cora_irredundant_systems(
  ctx,
  max_depth = NULL,
  search = c("bounded", "exhaustive")
)
```

Arguments

ctx	A cora_context() with more than one outcome.
max_depth	Optional upper bound on the number of distinct prime implicants a system may contain, as in cora_irredundant_sums() .
search	How max_depth is applied, as in cora_irredundant_sums() . "bounded" also bounds each outcome's own chart, which is what makes a system of several outcomes over a large chart solvable at all.

Value

A list of systems, of class cora_systems.

Examples

```
df <- data.frame(A = c(1, 1, 0, 0), B = c(2, 1, 2, 2), C = c(0, 1, 1, 2),
                 D = c(1, 0, 0, 0), OUT1 = c(1, 2, 0, 1),
                 OUT2 = c(2, 0, 1, 1), OUT3 = c(1, 0, 2, 1))
## B is coded 1 and 2 here, which CORA does not accept.
df <- cora_recode(df, "B")
ctx <- cora_context(df, c("OUT1{1,2}", "OUT2{1}", "OUT3{1,0}"),
                  algorithm = "ON-OFF")
cora_irredundant_systems(ctx)
```

cora_logigram	<i>Draw a two-level logic diagram</i>
---------------	---------------------------------------

Description

Renders a Boolean or multi-value function in disjunctive normal form as a two-level logic diagram: conjunctions become AND gates, the disjunction over them an OR gate. This is an R implementation of LOGIGRAM.

Usage

```
cora_logigram(x, ...)

## Default S3 method:
cora_logigram(
  x,
  color_or = "lightblue",
  color_and = "lemonchiffon",
  notation = c("case", "prime"),
  title = NULL,
  subtitle = NULL,
  show_terms = FALSE,
  ...
)

## S3 method for class 'cora_system'
cora_logigram(x, title = NULL, subtitle = NULL, ...)

## S3 method for class 'cora_system_multi'
cora_logigram(x, title = NULL, subtitle = NULL, ...)

## S3 method for class 'cora_context'
cora_logigram(x, ...)
```

Arguments

x	A character vector of functions in disjunctive normal form, such as "A*B+c*A+b<=>F" or "A{1}*B{2}+C{0}<=>F", one entry per outcome. Square brackets are read as curly ones, so the "A[1]*B[2]" notation of the QCA package is accepted too. A cora_context() or a solution from cora_irredundant_sums() or cora_irredundant_systems() is accepted directly and converted first.
...	Passed to methods.
color_or	Fill colour of the OR gates.
color_and	Fill colour of the AND gates.
notation	"case" when a negated literal is written in lower case, "prime" when it is written with a trailing apostrophe.

title	Text printed above the diagram, one line per element. The default, NULL, prints the expression the diagram stands for; for a solution object that is the solution itself, essential prime implicants still marked with #. Pass NA to print no title.
subtitle	A single line printed under the title. The default, NULL, prints the coverage and inclusion scores when the object drawn carries them, and nothing otherwise. Pass NA to print no subtitle.
show_terms	Print each conjunction next to the gate that forms it. Useful when the diagram is read on its own, away from the solution.

Value

The parsed diagram, invisibly. Called for the plot it draws.

Examples

```
cora_logigram("A*B+c*A+b<=>F")
cora_logigram("A{1}*B{2}+C{0}<=>F")

df <- data.frame(A = c(1, 0, 1, 0), B = c(1, 0, 0, 1),
                 C = c(0, 1, 1, 0), OUT = c(1, 1, 0, 1))
sol <- cora_irredundant_sums(cora_context(df, "OUT"))[[1]]

## The solution and its scores are written above the diagram.
cora_logigram(sol)

## Each conjunction next to its own gate, and no header.
cora_logigram(sol, title = NA, subtitle = NA, show_terms = TRUE)
```

cora_petrick

Solve a prime implicant chart with Petrick's method

Description

Solve a prime implicant chart with Petrick's method

Usage

```
cora_petrick(coverages, max_depth = NULL)
```

Arguments

coverages	A list with one integer vector per prime implicant giving the rows that implicant covers.
max_depth	Optional upper bound on the number of prime implicants a sum may contain. The bound is applied while the products are being multiplied out rather than to the finished list, which is what makes a large chart solvable at all; the sums returned are the same either way.

Value

A list with `essential`, the indices of the prime implicants that are the only cover of some row, and `sums`, a list of integer vectors holding the prime implicant indices of every irredundant sum. Indices are one-based positions in coverages.

Examples

```
cora_petrick(list(c(1, 2), c(2, 3), c(3, 4)))

## Only the sums built from at most two prime implicants.
cora_petrick(list(c(1, 2), c(2, 3), c(3, 4)), max_depth = 2)
```

cora_pi_chart	<i>Prime implicant chart</i>
---------------	------------------------------

Description

Prime implicant chart

Usage

```
cora_pi_chart(ctx)
```

Arguments

`ctx` A `cora_context()`.

Value

A data frame with one row per prime implicant and one column per covered truth table row; an entry is 1 when the prime implicant covers that row and 0 otherwise.

Examples

```
df <- data.frame(A = c(1, 0, 1, 0), B = c(1, 0, 0, 1),
                 C = c(0, 1, 1, 0), OUT = c(1, 1, 0, 1))
cora_pi_chart(cora_context(df, "OUT"))
```

cora_pi_details	<i>Statistical overview of the prime implicants</i>
-----------------	---

Description

Statistical overview of the prime implicants

Usage

```
cora_pi_details(ctx, max_solutions = 50)
```

Arguments

ctx	A <code>cora_context()</code> .
max_solutions	Largest number of solution columns to build. A chart with many prime implicants can have tens of thousands of solutions, and one column each is a table nobody can read; the first max_solutions are kept and a message says how many were left out. Inf keeps all of them.

Value

A data frame with one row per prime implicant holding its coverage score (Cov.r), its inclusion score (Inc.) and, for every solution, the share of the outcome that the prime implicant covers uniquely within that solution. NA marks a prime implicant absent from a solution.

Examples

```
df <- data.frame(A = c(1, 0, 1, 0), B = c(1, 0, 0, 1),
                  C = c(0, 1, 1, 0), OUT = c(1, 1, 0, 1))
cora_pi_details(cora_context(df, "OUT"))
```

cora_prime_implicants	<i>Prime implicants of an optimisation context</i>
-----------------------	--

Description

Minimises the truth table with the algorithm chosen in the context and returns the resulting prime implicants.

Usage

```
cora_prime_implicants(ctx)
```

Arguments

ctx	A <code>cora_context()</code> .
-----	---------------------------------

Value

A list of prime implicants, of class `cora_implicants`. Every literal is printed as `CONDITION{value}`, a term joins its literals with `*`, and an essential prime implicant is prefixed with `#`.

Examples

```
df <- data.frame(A = c(1, 0, 1, 0), B = c(1, 0, 0, 1),
                 C = c(0, 1, 1, 0), OUT = c(1, 1, 0, 1))
cora_prime_implicants(cora_context(df, "OUT"))
```

<code>cora_python_available</code>	<i>Is the Python CORA package reachable?</i>
------------------------------------	--

Description

Answering the question means asking 'reticulate' for a module, which starts Python. On a machine where no interpreter has been configured, recent versions of 'reticulate' provision one at that moment, which can take half a minute and reach the network. The example is therefore not run automatically; call it yourself when you want the answer.

Usage

```
cora_python_available()
```

Value

TRUE when both 'reticulate' and the Python cora module are available, FALSE otherwise.

Examples

```
## Not run:
cora_python_available()

## End(Not run)
```

<code>cora_recode</code>	<i>Recode conditions onto 0, 1, 2, ...</i>
--------------------------	--

Description

CORA reads a condition's values as the levels of a factor coded from zero upwards. Data seldom arrives that way: `as.integer()` on a factor numbers the levels from one, and rating scales are usually stored as they were collected. This function maps each named condition onto 0, 1, 2, ..., keeping the order of its values, and leaves every other column alone.

Usage

```
cora_recode(data, conditions = NULL)
```

Arguments

<code>data</code>	A data frame.
<code>conditions</code>	Character vector naming the columns to recode. Defaults to every integer-valued column that is not already coded from zero.

Value

data with the named columns recoded.

Examples

```
df <- data.frame(A = c(2, 1, 2, 1), B = c(1, 2, 1, 2), OUT = c(1, 0, 1, 1))
cora_recode(df, c("A", "B"))

## A rating scale collected as 1-5 becomes 0-4.
cora_recode(data.frame(score = c(3, 1, 5, 1)), "score")

## Left to itself it recodes exactly the columns that need it.
cora_recode(df)
```

<code>cora_solutions</code>	<i>Solution summary table</i>
-----------------------------	-------------------------------

Description

Solution summary table

Usage

```
cora_solutions(ctx, max_solutions = 50)
```

Arguments

<code>ctx</code>	A <code>cora_context()</code> .
<code>max_solutions</code>	Largest number of solutions to lay out. A chart with many prime implicants can have tens of thousands, so the first <code>max_solutions</code> are kept and a message says how many were left out. Inf keeps all of them.

Value

A data frame with one column per prime implicant marking, with a 1, the solutions the prime implicant belongs to. In the multi-outcome case each system contributes one row per outcome, and the Output and System columns identify them.

Examples

```
df <- data.frame(A = c(1, 0, 1, 0), B = c(1, 0, 0, 1),
                 C = c(0, 1, 1, 0), OUT = c(1, 1, 0, 1))
cora_solutions(cora_context(df, "OUT"))
```

cora_system_details	<i>Statistical overview of a solution</i>
---------------------	---

Description

Statistical overview of a solution

Usage

```
cora_system_details(ctx)
```

Arguments

ctx A `cora_context()`.

Value

A one-row data frame with the coverage and inclusion score of the first irredundant solution.

Examples

```
df <- data.frame(A = c(1, 0, 1, 0), B = c(1, 0, 0, 1),
                 C = c(0, 1, 1, 0), OUT = c(1, 1, 0, 1))
cora_system_details(cora_context(df, "OUT"))
```

cora_truth_table	<i>Truth table of an optimisation context</i>
------------------	---

Description

Aggregates the cases into configurations, applies the frequency cut-off and the inclusion cut-offs, and returns the resulting truth table.

Usage

```
cora_truth_table(ctx, raw = FALSE)
```

Arguments

ctx A `cora_context()`.
 raw If TRUE, also return the case counts, the case labels and the raw inclusion scores of every configuration.

Value

A data frame.

Examples

```
df <- data.frame(A = c(1, 0, 1, 1, 1), B = c(0, 1, 1, 1, 1),
                 C = c(0, 0, 1, 1, 1), O = c(1, 1, 0, 1, 1))
cora_truth_table(cora_context(df, "O", inc_score1 = 0.5))
```

gross_carvin

Tort liability of highway authorities

Description

Multi-value data on tort claims against highway authorities, used as the single-outcome example of the Python CORA package.

Usage

```
gross_carvin
```

Format

A data frame with 18 rows: the case label Case, the conditions PRIC, LENG, UPSI, DOSI, RISK, FRFL and MIMA, and the outcome TORT.

Source

Shipped with the Python CORA package, <https://github.com/PoliUniLu/cora>.

Examples

```
ctx <- cora_context(gross_carvin, "TORT", case_col = "Case")
cora_prime_implicants(ctx)
```

mccluskey

McCluskey's two-output switching function

Description

The textbook two-output switching function used to illustrate multi-output Boolean minimisation.

Usage

```
mccluskey
```

Format

A data frame with 16 rows: the conditions A, B, C and D, and the outcomes F1 and F2.

Source

Shipped with the Python CORA package, <https://github.com/PoliUniLu/cora>.

Examples

```
cora_data_mining(mccluskey, c("F1", "F2"), len_of_tuple = 2)
```

swiss_minaret

Swiss minaret referendum

Description

Cantonal data on the 2009 Swiss referendum on the construction of minarets, used as the multi-outcome example of the Python CORA package.

Usage

```
swiss_minaret
```

Format

A data frame with 11 rows and 6 columns: the conditions A, L, S and T, and the outcomes X and M.

Source

Shipped with the Python CORA package, <https://github.com/PoliUniLu/cora>.

Examples

```
ctx <- cora_context(swiss_minaret, c("X", "M"), algorithm = "ON-OFF")
cora_irredundant_systems(ctx)
```

Index

* datasets

- bergschlosser, [2](#)
- gross_carvin, [18](#)
- mccluskey, [19](#)
- swiss_minaret, [19](#)

bergschlosser, [2](#)

cora_compare_python, [3](#)
cora_context, [4](#)
cora_context(), [3](#), [6](#), [9–11](#), [13](#), [14](#), [16](#), [17](#)
cora_coverage_score, [5](#)
cora_data_mining, [6](#)
cora_describe, [7](#)
cora_dnf, [8](#)
cora_inclusion_score
 (cora_coverage_score), [5](#)
cora_irredundant_sums, [9](#)
cora_irredundant_sums(), [5](#), [7](#), [8](#), [10](#), [11](#)
cora_irredundant_systems, [10](#)
cora_irredundant_systems(), [5](#), [7](#), [8](#), [11](#)
cora_logigram, [11](#)
cora_logigram(), [8](#)
cora_petrick, [12](#)
cora_pi_chart, [13](#)
cora_pi_details, [14](#)
cora_prime_implicants, [14](#)
cora_prime_implicants(), [5](#)
cora_python_available, [15](#)
cora_recode, [15](#)
cora_solutions, [16](#)
cora_system_details, [17](#)
cora_truth_table, [17](#)
cora_truth_table(), [5](#)

gross_carvin, [18](#)

mccluskey, [19](#)

swiss_minaret, [19](#)