Package 'epicmodel'

November 8, 2024

Title Causal Modeling in Epidemiology

Version 0.1.1

Description Create causal models for use in epidemiological studies, including sufficient-component cause models as introduced by Rothman (1976) <doi:10.1093/oxfordjournals.aje.a112335>.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Imports checkmate, cli, dagitty, DiagrammeR, dplyr, DT, ggplot2, gtools, magrittr, methods, prompter, purrr, rlang, shiny, shinyalert, shinyjs, shinythemes, spsUtil, stringr, tibble, tidyr

Suggests ggdag, ggforce, ggraph, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Depends R (>= 3.5.0)

LazyData true

VignetteBuilder knitr

URL https://forsterepi.github.io/epicmodel/, https://github.com/forsterepi/epicmodel

BugReports https://github.com/forsterepi/epicmodel/issues

NeedsCompilation no

Author Felix Forster [aut, cre, cph] (<https://orcid.org/0000-0002-3670-9244>)

Maintainer Felix Forster <felix.forster@med.uni-muenchen.de>

Repository CRAN

Date/Publication 2024-11-08 14:30:01 UTC

Contents

`are_sufficient` *Check if a certain set of component causes is suffcient*

Description

Provide a SCC model and a set of component causes and evaluate if the provided set of causes fulfills any sufficient cause, i.e., is sufficient for the outcome to occur based on the provided SCC model. Fulfilling a sufficient cause means that all component causes of a certain sufficient cause are in the provided set of causes. Unknown causes are ignored by this function.

Usage

```
are_sufficient(scc, causes = NULL, type = c("status", "binary"))
```

Arguments

`scc`	An object of class `epicmodel_scc`.
`causes`	NULL (default) or a character vector containing IDs of a set of component causes. If NULL, prints a list of all available component causes.
`type`	Either "status" (default) or "binary". If "status", returns one of "always", "depends", "never". If "binary", returns TRUE or FALSE.

Details

Depending on the value of type, the following values are possible:

- type = "status": If the provided set of causes contains all component causes of a sufficient cause with status "always", returns "always". If the provided set of causes only fulfills sufficient cause with status "depends" or "depends (potential order implausibilities)", returns "depends". If no sufficient causes are fulfilled, returns "never".
- type = "binary": If the returned status would have been "always" or "depends", TRUE is returned. If the returned status would have been "never", returns FALSE.

Value

For type = "binary, returns TRUE if all component causes for at least one sufficient cause are in causes and FALSE otherwise. For type = status, returns "always" if at least one sufficient cause with sufficiency status "always" is fulfilled. If not, returns "depends" if at least one sufficient cause with sufficiency status "depends" or "depends (potential order implausibilities)" is fulfilled. If no sufficient cause is fulfilled, returns "never".

Examples

```
# Create some SCC model
steplist_checked <- check_steplist(steplist_rain)
scc_model <- create_scc(steplist_checked)

# Check sufficiency for a certain set of component causes
are_sufficient(scc_model, c("THENa1","THENa5"), type = "status")
are_sufficient(scc_model, c("THENa1","THENa5"), type = "binary")
```

---

check_steplist *Check* epicmodel_steplist *class objects*

Description

Check if epicmodel_steplist class objects fulfill the conditions for being inputed in create_scc().

Usage

```
check_steplist(steplist)
```

Arguments

steplist	An object of class epicmodel_steplist.

Details

The following checks are conducted:

Errors:

- Correct ID format in WHAT segments
- No duplicated IDs in WHAT segments
- Correct ID format in DOES segments
- No duplicated IDs in DOES segments
- Correct ID format in WHERE segments
- No duplicated IDs in WHERE segments
- Correct ID format in Modules
- No duplicated IDs in Modules
- Correct ID format in ICC
- No duplicated IDs in ICC
- All WHAT segments used in data.frame `step` must be listed in data.frame `what`
- All DOES segments used in data.frame `step` must be listed in data.frame `does`
- All WHERE segments used in data.frame `step` must be listed in data.frame `where`
- All modules used in data.frame `step` must be listed in data.frame `modules`
- Either all steps or no steps have modules specified in data.frame `step`
- All step IDs used in ICC definition must be specified in data.frame `step`
- Starting steps, i.e., steps without IF condition, must not have `end_step == 1` in data.frame `step`
- A steplist must contain component causes
- In case there are two steps with identical THEN statements, they cannot have both `end_step == 1` and `end_step == 0` in data.frame `step`
- THEN statements used in IF/IFNOT conditions must be available for chaining, i.e., there must be a step with this statement as its THEN part and this step must not be defined as end step
- For all steps, their THEN statement must be available in data.frame `then`
- A step must not have identical IF and IFNOT conditions
- A step's THEN statement must not be part of its own IF/IFNOT condition
- All steps used in the outcome definition must be in data.frame `step` with `end_step == 1`

Warnings:

- No duplicated keywords in WHAT segments
- No duplicated keywords in DOES segments
- No duplicated keywords in WHERE segments
- No duplicated keywords in Modules
- All WHAT segments in data.frame `what` should be used in data.frame `step`
- All DOES segments in data.frame `does` should be used in data.frame `step`
- All WHERE segments in data.frame `where` should be used in data.frame `step`
- All modules in data.frame `modules` should be used in data.frame `step`
- All steps should have references

- There should not be any steps with identical THEN statements
- All steps with `end_step == 1`in data.frame `step` should be used in the outcome definition
- Outcome definitions should not be contained in each other, e.g., for outcome definition `(A and B) or (A and B and C)`, `A and B` is contained, i.e., a subset of, `A and B and C`, which makes `A and B and C` redundant

Value

Prints information about successful and unsuccessful checks in the console. Returns the input steplist. If checks were successful, returns a steplist of class `epicmodel_steplist_checked` that can be used in building SCC models.

Examples

```
steplist_checked <- check_steplist(steplist_rain)
```

`create_scc` *Creating SCC models*

Description

Creates a sufficient-components cause (SCC) model from a steplist, which is a list of IF/THEN statements describing the causal mechanism behind an outcome of interest. The steplist needs to meet certain structural requirements. Therefore, for steplist creation, use the Steplist Creator `shiny` app launched by `launch_steplist_creator()`.

Usage

```
create_scc(steplist)
```

Arguments

`steplist`	An object of class `epicmodel_steplist_checked`.

Details

The following algorithm is used to create a sufficient-component cause (SCC) model from a steplist.

- Check inputs: The steplist needs to be checked by `check_steplist()` before input
- Are modules used: Evaluate if the steplist contains modules
- Process steplist: Process steplist and outcome definition so that they can be used by the procedure
- Get all combinations of component causes in the steplist: Component causes are steps, which themselves have no IF condition but appear in IF conditions of other steps (and maybe additionally in IFNOT conditions). Interventions are not considered to be component causes. Interventions are as well steps without IF condition, but they only appear in IFNOT conditions of other steps. Invalid combinations of component causes as specified in the ICC part of the steplist are excluded, as well as every component cause being absent.

- Check sufficiency: Sufficiency is checked for every combination of component causes. First, based on a specific set of component causes, it is derived, which steps can be caused by this set, i.e., which IF conditions are fulfilled. For this, a current set of included steps is defined, which in the beginning includes only the corresponding set of component causes. Then, it is iteratively checked, for which other steps with IF condition (i.e., excluding non-selected component causes and interventions) this IF condition is fulfilled. These steps are added to the current set of included steps and the process is repeated until for no new steps the IF condition is fulfilled. Second, this final list of steps is compared against the outcome definitions. If it is fulfilled, the set of component causes is sufficient.

- Check IFNOT conditions: Please note that IFNOT conditions were ignored up to this point. Now, all sets of component causes that were found to be sufficient previously, are re-checked for IFNOT conditions. First, it is checked if there are any IFNOT conditions in the final list of steps derived above and if those are fulfilled based only on the other steps in this list. If no, checking is complete and the corresponding set of component causes is always sufficient. If yes, further checking is required. In these cases, sufficiency depends on the order in which individual steps occur. In principle, a step with both IF and IFNOT conditions fulfilled, occurs if the IF condition is fulfilled before the IFNOT condition, similar to how I do not care if a door is closed if I already went through it when it was still open. Please note that this approach extends SCC models by an additional time component. Sufficiency is therefore re-checked for all possible sequences of IF and IFNOT conditions of all steps that include IFNOT conditions that can be fulfilled by the final set of steps. It is possible to have component causes with IFNOT conditions. Since they do not have an IF condition, the THEN statement is used instead. For every sequence, it is evaluated if the IF (or THEN for component causes) occurs before or after the IFNOT. If IF/THEN occur after the corresponding IFNOT, this step is removed from the final list of steps. Sufficieny is now re-checked based on the updated list. If some orderings do not fulfill the outcome definition, the sufficiency status of the corresponding set of component causes is changed to "depends", as it depends on the sequence of events. Please note that currently, all sequences are checked even though some of them might be implausible, e.g., when two steps with IFNOT conditions are chained together. In this case, there will be a warning displayed, but the user ultimately needs to check plausibility of the sequence of events.

- Minimize: Sufficient causes must be minimal by definition, i.e., every component cause must be necessary within its sufficient cause, i.e., the absence of one component cause of a sufficient set means that the outcome does not occur anymore. Therefore, the list of sufficient (both always and depends) sets of component causes is reduced to minimal ones.

- Add unknown causes: It is possible/likely that unknown causes, both component causes and sufficient causes, are not part of the model yet. Therefore, every sufficient cause gets an additional individual (i.e., a different one for each sufficient cause) unknown component cause representing additional unknown components, and one unknown sufficient cause is added to the model consisting of a single unknown component cause and representing all unknown sufficient causes. If relevant, the user can decide in functions with the SCC model as input if unknown causes should be included or not.

- Output preparation: Combines all outputs to an object of class `epicmodel_scc` for further analysis.

Value

An object of class `epicmodel_scc`. If no sufficient causes are found, no object is returned but instead a corresponding message is displayed in the console.

References

Rothman KJ (1976): Causes. American Journal of Epidemiology 104 (6): 587–592.

See Also

- `SCC models` for information on `epicmodel_scc` objects
- `Steplist` for information on `epicmodel_steplist` objects

Examples

```
# First, create a steplist in the shiny app
# Launch the app with launch_steplist_creator()
# Then load your steplist using readRDS()
# In this example we use the built-in steplist_rain

# Check the steplist before running create_scc()
steplist_checked <- check_steplist(steplist_rain)

# Use the checked steplist in create_scc()
scc_model <- create_scc(steplist_checked)
```

`effect_size` *Determine standardized effect size of component causes*

Description

SCC models teach us that effect strength, e.g., a risk ratio, is no natural constant but depends on the prevalence of component causes and, therefore, differs between populations. However, even without any population, this function derives effect sizes for every component cause by comparing how many sets of component causes with and without a certain cause are sufficient to cause the outcome of interest.

Usage

```
effect_size(scc, depends = TRUE, output = c("nice", "table"))
```

Arguments

`scc`	An object of class `epicmodel_scc`.
`depends`	TRUE (default) or FALSE. If FALSE, only includes sufficient causes with sufficiency status "always".
`output`	A single element of type character, either "nice" (default) or "table". If "table", returns a data.frame. If "nice", a nicely formated output is printed in the console.

Details

The following algorithm is used to derive effect sizes from SCC models:

- The effect size is derived for one specific component cause. The following steps are repeated for all of them.
- Get all potential combinations of component causes
- Remove combinations that contain incompatible component causes (ICC), as specified in the steplist
- Split the set of possible combinations of component causes into two parts: Sets, in which the component cause of interest is present & sets, in which the component cause of interest is absent. The numbers are recorded and returned in the output table (output = "table") as variables `num_combos_true` (cause is present) and `num_combos_false` (cause is absent). If there are no incompatible component causes (ICC), both values should be the same.
- Check for all possible combinations of component causes, if they are sufficient for the outcome to occur. The number of sufficient combinations are counted separately for combinations with the component cause of interest present and combinations with the component cause of interest absent. The numbers are recorded and returned in the output table (output = "table") as variables `suff_true` (cause is present) and `suff_false` (cause is absent).
- A ratio is calculated using the following formula: `(suff_true / num_combos_true) / (suff_false / num_combos_false)`. In the output table (output = "table"), this value is stored in variable `ratio`. In the nice output (output = "nice"), it is reported in the column `Impact`, which shows: `ratio [suff_true/num_combos_true vs. suff_false/num_combos_false]`
- There are two special cases when calculating the `ratio`. When `suff_true > 0` but `suff_false == 0`, the outcome only occurs if the corresponding component cause is present. The `ratio` then gets value `necessary`. When `suff_true == 0` and `suff_false == 0`, the `ratio` gets value `not a cause`.

Value

Either a dataframe (`output = "table"`) with one row for every component cause and with variables `id` (step ID), `desc` (step description), `suff_true`, `suff_false`, `num_combos_true`, `num_combos_false`, and `ratio`, or a nicely formated output in the console (`output = "nice"`). See Details for more information.

Examples

```
# Create some SCC model
steplist_checked <- check_steplist(steplist_rain)
scc_model <- create_scc(steplist_checked)

# Use the SCC model in effect_size()
effect_size(scc_model)
```

`export_mechanism` *Export mechanisms*

Description

Exports one or all sufficient cause mechanisms as PNG, PDF, SVG, or PostScript using `DiagrammeR::export_graph()`.

Usage

```
export_mechanism(
  mechanism,
  sc = NULL,
  file_name = NULL,
  file_type = "png",
  title = NULL,
  ...
)
```

Arguments

`mechanism`	An object of class `epicmodel_mechanism`.
`sc`	A single integer value (can be specified as numeric, e.g., 2 instead of 2L). If provided, a graph is only exported for the specified sufficient cause, e.g., for SC2 if sc = 2. If sc = NULL (default), graphs for all sufficient causes are exported.
`file_name`	The name of the exported file (including it's extension).
`file_type`	The type of file to be exported. Options for graph files are: `png`, `pdf`, `svg`, and `ps`.
`title`	An optional title for the output graph.
`...`	Arguments passed on to `DiagrammeR::export_graph`
	`width` Output width in pixels or `NULL` for default. Only useful for export to image file formats `png`, `pdf`, `svg`, and `ps`.
	`height` Output height in pixels or `NULL` for default. Only useful for export to image file formats `png`, `pdf`, `svg`, and `ps`.

Value

Saves the mechanisms as PNG, PDF, SVG, or PostScript.

See Also

- `DiagrammeR::export_graph()`
- `mechanism()` for information on sufficient cause mechanisms

Examples

```
# Derive mechanisms
mech <- mechanism(scc_rain)

# Export mechanism plot of sufficient cause (sc) 1
if(interactive()){
tmp <- tempfile(fileext = ".png")
export_mechanism(mech, sc = 1, file_name = tmp, title = "Sufficient Cause 1")
unlink(tmp) # delete saved file
}
```

`intervene` *Explore effect of interventions*

Description

Interventions are steps without IF condition (start steps) that only appear in other IFNOT conditions, i.e., that can only prevent steps but not cause them. Interventions are not considered when creating SCC models using `create_scc()`. `intervene()` evaluates their impact in two directions: 1) which sufficient causes can be prevented by certain (sets of) interventions and 2) which set of interventions is at least needed to prevent the outcome in an individual with a given set of component causes.

Usage

```
intervene(scc, causes = NULL, intervention = NULL, output = c("nice", "table"))
```

Arguments

`scc`	An object of class `epicmodel_scc`.
`causes`	A character vector containing step IDs of component causes. If "all", investigates all sufficient causes, i.e., all minimally sufficient sets of component causes. If NULL (default), prints a list of all available component causes in the console. If a set of step IDs is specified, only the specified set is investigated.
`intervention`	A character vector containing step IDs of interventions. If "all", investigates all possible combinations of available interventions. If NULL (default), prints a list of all available interventions in the console. If a set of step IDs is specified, investigates all possible combinations of the specified interventions.
`output`	Either "nice" (default) or "table". If "nice", prints a nicely formatted summary in the console. If "table", returns a list of several elements described in detail in section "Value" below.

Details

The following algorithm is used to evaluate the effect of interventions:

- Derive the list of intervention sets to evaluate
- Derive the list of sets of component causes to evaluate

- Evaluate sufficiency without intervention for every set of component causes
- Evaluate sufficiency for every combination of intervention set and set of componen causes: First, check which steps are prevented by the corresponding set of interventions, i.e., for which steps the IFNOT condition is fulfilled by the intervention set. These steps are removed from the list of available steps. Second, evaluate sufficiency based on the remaining steps similar to `create_scc()` (Check sufficiency & Check IFNOT conditions).
- Evaluate, which intervention sets are minimal, i.e., at least necessary to prevent the outcome

Value

Output:

If `output = "nice"` (default), prints a nicely formatted output in the console. If `output = "table"`, returns a list with the following elements:

`cause_set` A list of character vectors with one element for every investigated set of component causes. The character vectors contain the step IDs of the component causes that are part of the corresponding set. Sets are named in a format similar to cc1, cc2, etc.

`intv` A list of character vectors with one element for every investigated set of interventions. The character vectors contain the step IDs of the interventions that are part of the corresponding set. Sets are named as intv1, intv2, etc.

`status` A data.frame with one row per set of component causes and one column per set of intervention. In addition, contains one column representing no interventions (`intv0`). Each cell contains the sufficiency status of the corresponding set of component causes when the corresponding set of interventions is applied. Possible values are "always", "depends", and "never". See below for an interpretation.

`minimal` A data.frame with one row per set of component causes and one column per set of intervention. Each cell is either TRUE or FALSE indicating if the set of interventions is minimal. For non-minimal sets of interventions, a smaller set which is contained within the corresponding set exists and has the same preventive power. Minimality is defined separately for every set of component causes. If both the larger non-minimal and the smaller minimal set sometimes prevent the outcome (status "depends" in `status` (see above)), the non-minimal set might actually prevent more sufficient orders of occurrence than the minimal set. In this case, please inspect and compare element `order` (see next), for all minimal and non-minimal sets of interventions with status "depends".

`order` A 2-level list, i.e., a list with one element per intervention set, for which each element is another list with one element per evaluated set of component causes. Each intervention/component causes combination contains a data.frame, similar to the data.frames in the `sc_order` element of `epicmodel_scc` objects, if the corresponding status is "depends", or is NA otherwise (for "always" or "never"). The data.frames contain two columns, which are called "order" and "suff" (short for "sufficient"), and one row for every order of occurrence. The order of occurrence is summarized in "order" (as character), while "suff" is either TRUE or FALSE indicating if the corresponding order of occurrence is sufficient, i.e., leads to the outcome, or not. Please note that the prevented orders of occurrence have `suff == FALSE`.

How to interpret `status`:

If the sufficiency status for a certain intervention in column `intv0` is `always`, the three sufficiency status options for a certain intervention have the following interpretations:

- always: The corresponding set of inteventions never prevents the outcome, because after applying the intervention, the corresponding set of component causes is still always sufficient.
- depends: The corresponding set of interventions sometimes prevents the outcome, because after applying the intervention, sufficiency for the corresponding set of component causes depends on the order of occurrence.
- never: The corresponding set of interventions always prevents the outcome, because after applying the intervention, the corresponding set of component causes is never sufficient.

If the sufficiency status for a certain intervention in column intv0 is depends, the sufficiency status options for a certain intervention have the following interpretations:

- depends: The corresponding set of interventions sometimes or never prevents the outcome, because after applying the intervention, sufficiency for the corresponding set of component causes depends on the order of occurrence. Further inspection and comparison of sufficient orders of occurrence is necessary to determine if the intervention actually prevents anything.
- never: The corresponding set of interventions always prevents the outcome, because after applying the intervention, the corresponding set of component causes is never sufficient.

If the sufficiency status for a certain intervention in column intv0 is never, no intervention is necessary, because the corresponding set of component causes is never sufficient.

Examples

```
# Derive SCC model
scc_model <- scc_rain

# Inspect the effect of interventions
intervene(scc_model, causes = "all", intervention = "all")
intv <- intervene(scc_model, causes = "all", intervention = "all", output = "table")
```

---

launch_steplist_creator

Launch steplist creator shiny *app*

Description

Run this function to start the Steplist Creator shiny app.

Usage

```
launch_steplist_creator()
```

Value

The launch_steplist_creator function is used for the side effect of starting the Steplist Creator shiny app.

Examples

```
if(interactive()){
launch_steplist_creator()
}
```

---

mechanism *Investigate mechanisms*

Description

Creates graphs that visiualize the mechanisms behind each sufficient cause using the DiagrammeR package.

new_mechanism() and validate_mechanism() define the epicmodel_mechanism S3 class, which is created by mechanism().

plot() renders the graphs in the RStudio Viewer.

print() prints the legend for node labels in the console.

Usage

```
mechanism(scc, modules = TRUE, module_colors = NULL)

new_mechanism(x = list())

validate_mechanism(x)

## S3 method for class 'epicmodel_mechanism'
plot(x, reverse = TRUE, ...)

## S3 method for class 'epicmodel_mechanism'
print(x, ...)
```

Arguments

scc	For mechanism(), an object of class epicmodel_scc.
modules	For mechanism(), TRUE (default) or FALSE, indicating if nodes in the same module should be colored equally (TRUE) or if all nodes have white background (FALSE). Colors are only applied, if modules have actually been specified in the epicmodel_steplist. If modules are considered by mechanism(), the module keywords are added to the legend (accessable via print()).
module_colors	For mechanism(), if nodes are colored by module, colors can be provided via this argument. Colors must be provided as a character vector. Both named colors and hexadecimal color codes are allowed. The function has 8 colors stored internally. If module_colors = NULL (default), these colors are used. If the model has more than 8 modules, module_colors must be specified. If more colors than necessary are specified, the function takes as many as necessary from the start of the vector.

x	x is used in several functions: • new_mechanism(): A list to be converted to class epicmodel_mechanism. • validate_mechanism(): An object of class epicmodel_mechanism to be validated. • plot.epicmodel_mechanism(): An object of class epicmodel_mechanism. • print.epicmodel_mechanism(): An object of class epicmodel_mechanism.
reverse	For plot.epicmodel_mechanism(), TRUE or FALSE indicating if the output should be displayed in reverse order. Since graphs rendered later show up first in the viewer pane, reverse = T leads to SC1 being the last rendered and the one displayed on top.
...	Additional arguments for generics print() and plot().

Details

The graphs:

One graph per sufficient cause is created. The graphs display steps as nodes and IF/IFNOT relations as edges. Nodes will not be labeled with their IDs or descriptions due to limited space, but with newly created labels. These labels are based on the type of node and are listed together with the step description in the legend (accessed by print()). Step descriptions are also accessible via tooltips in the graph. Just put your cursor on the node labels.

There are 4 different types of nodes:

- Component causes: Labeled "CC", squares, gray border
- Interventions: Labeled "I", triangles, gray border
- End steps: Labeled "E", circles, black border
- Other steps: Labeled "S", circles, gray border

There are 2 types of edges:

- IF conditions: gray arrows
- IFNOT conditions: red and T-shaped

epicmodel_mechanism objects:

epicmodel_mechanism objects are created by mechanism(). They are lists containing 2 elements:

legend A data.frame with up to 3 variables:

- Label: Contains the labels used in the graphs.
- Module: Contains the name of the module to which this step belongs. Only available if modules = TRUE in mechanism() and if the SCC model actually uses modules (specified in element sc_use_modules of epicmodel_scc objects).
- Step: A description of the corresponding step.

graph A list of length equal to the number of sufficient causes. Each element contains another list with 2 elements:

- ndf: A data.frame containing information about nodes in the graph (see DiagrammeR::node_aes()).
- edf: A data.frame containing information about edges in the graph (see DiagrammeR::edge_aes()).

ndf Data.frames containing the following variables:

- id: Node ID used internally by DiagrammeR to define edges (from and to in edf data.frames).

- type: Type of node as defined by epicmodel. Possible options are: cc (component cause), int (intervention), end (step that is part of an outcome definition), other (all other steps).
- label: The label displayed in the graph and listed in variable Label of legend.
- tooltip: The text displayed when putting the cursor on top of the node label. Corresponds to the step descriptions in variable Step of legend.
- shape: The shape of the node. square for type cc, triangle for type int, and circle for types end and other.
- color: Color of the node border. Gray for types cc, int, and other, and black for type end.
- fillcolor: Color of the background, which is similar for all steps in the same module. If modules are not considered, fillcolor is white for all nodes.
- fontcolor: Color of the node label. Always black.

edf Data.frames containing the following variables:

- id: Edge ID used internally by DiagrammeR.
- from: Node ID of the node from which the edge starts.
- to: Node ID of the node at which the edge ends.
- rel: Type of edge as defined by epicmodel. Possible options are: if (from node is in IF condition of to node), ifnot (from node is in IFNOT condition of to node).
- arrowhead: Type of arrow. normal for rel if and tee for rel ifnot.
- arrowsize: Size of arrow. 1 for rel if and 1.2 for rel ifnot.
- color: Color of arrow. Gray for rel if and #A65141 for rel ifnot.

Value

- mechanism(): An object of class epicmodel_mechanism. Use plot() to plot the graphs in the RStudio Viewer. Use print() to print the legend in the console. Use export_mechanism() to save the graphs as PNG, PDF, SVG, or PostScript.
- new_mechanism(): An object of class epicmodel_mechanism.
- validate_mechanism(): An object of class epicmodel_mechanism that has been checked to have the correct structure.
- plot.epicmodel_mechanism(): Renders the graphs in the RStudio Viewer.
- print.epicmodel_mechanism(): Prints the legend of the epicmodel_mechanism object in the console.

See Also

- export_mechanism() for saving the plots
- DiagrammeR::node_aes() for a list of node-related variables in DiagrammeR
- DiagrammeR::edge_aes() for a list of edge-related variables in DiagrammeR

Examples

```
# Create some SCC model
steplist_checked <- check_steplist(steplist_rain)
```

```
scc_model <- create_scc(steplist_checked)

# Derive mechanisms
mech <- mechanism(scc_model)

# new_mechanism() and validate_mechanism() are used inside mechanism()
# nonetheless, you can check its structure using validate_mechanism()
validate_mechanism(mech)

# Plot the mechanisms
plot(mech)

# Print the legend
print(mech)
mech
```

---

new_scc *SCC model objects*

Description

The S3 class `epicmodel_scc` is used to store information on sufficient-component cause (SCC) models created by `create_scc()`.

`new_scc()`, `validate_scc()`, and `empty_scc()` define the S3 class.

`print()` prints a summary of SCC models in the console. `summary()` and `print()` are identical.

`plot()` creates the familiar causal pie charts from an object of class `epicmodel_scc`.

Usage

```
new_scc(x = list())

validate_scc(x)

empty_scc()

## S3 method for class 'epicmodel_scc'
print(x, ...)

## S3 method for class 'epicmodel_scc'
summary(object, ...)

## S3 method for class 'epicmodel_scc'
plot(
  x,
  remove_sc = NULL,
  sc_label = NULL,
```

```
  unknown = TRUE,
  names = TRUE,
  text_color = NULL,
  pie_color = NULL,
  border_color = NULL,
  ...
)
```

Arguments

x	x is used in several functions: • new_scc(): A list to be converted to class epicmodel_scc. • validate_scc(): An object of class epicmodel_scc to be validated. • print.epicmodel_scc(): An object of class epicmodel_scc. • plot.epicmodel_scc(): An object of class epicmodel_scc.
...	Additional arguments for generics print(), summary(), and plot().
object	For summary.epicmodel_scc(), an object of class epicmodel_scc.
remove_sc	For plot.epicmodel_scc(), a vector of integerish numbers, i.e., integers that can be specified as numeric, i.e., 1 and 1L are both possible. Removes the sufficient cause (SC) with the specified index from the plot, i.e., for remove_sc = 2, removes SC 2, and for remove_sc = c(2,3), removes SC 2 and SC 3. If there are x sufficient causes in the model, x is the highest allowed value. At least one sufficient cause needs to remain, i.e., not all sufficient causes can be removed. If NULL (default), all sufficient causes are plotted.
sc_label	For plot.epicmodel_scc(), a character vector with the labels written above the pies, i.e., sufficient causes. If NULL (default), "Sufficient Cause 1", "Sufficient Cause 2", etc. are used. If specified, try to provide as many labels as there are pies in the plot. Duplicates are not allowed.
unknown	For plot.epicmodel_scc(), TRUE (default) or FALSE. If TRUE, unknown causes are added to the SCC model: every sufficient cause gets an additional individual unknown component cause representing additional unknown components; an unknown sufficient cause is added to the model consisting of a single unknown component cause and representing all unknown sufficient causes.
names	For plot.epicmodel_scc(), TRUE (default) or FALSE. If TRUE, includes the translation of pie segment names to descriptions of component causes in the plot.
text_color	For plot.epicmodel_scc(), a single element of type character, which is a valid color description. Valid color descriptions can be named colors ("white") or hexadecimal color codes ("#FFFFFF"). text_color will be used for the pie segment names. If NULL (default), "white" is used.
pie_color	For plot.epicmodel_scc(), a character vector of length 3 containing valid color descriptions. Valid color descriptions can be named colors ("white") or hexadecimal color codes ("#FFFFFF"). The first element of pie_color is used to color sufficient causes, which are always sufficient. The second element is used to color sufficient causes, for which sufficiency depends on the order of occurrence. The third element is used to color the unknown sufficient cause,

which is present if unknown is TRUE. If NULL (default), the following colors are used: "#B1934A", "#A65141", "#394165"

border_color For plot.epicmodel_scc(), a single element of type character, which is a valid color description. Valid color descriptions can be named colors ("white") or hexadecimal color codes ("#FFFFFF"). border_color will be used for all pie borders apart from the unknown sufficient cause. Therefore, only specify border_color if unknown is FALSE. If NULL (default), "white" is used. (Borders for the unknown sufficient cause have the same color as the pie.)

Details

epicmodel_scc **objects:**

epicmodel_scc objects are lists containing 10 elements. These elements are described below:

sc_cc A data.frame with one column for every component cause and one row for every sufficient cause. Colnames are the step IDs from the corresponding steplist. Rownames are sufficient cause IDs (see below). Each cell contains either TRUE or FALSE indicating if the component cause in the column is part of a set of component causes described by the row.

sc_status A named character vector with one element for every sufficient cause. The names are sufficient cause IDs (see below). The elements contain the status of the sufficient cause (see below). Here, only "always", "depends", and "depends (potential order implausibilities)" appear.

sc_steps A list of character vectors with one list element for every sufficient cause. The list is named using sufficient cause IDs (see below). Every character vector contains the step IDs of all steps that are part of the corresponding sufficient cause, i.e., that can be caused by the corresponding set of component causes.

sc_order A list with one list element for every sufficient cause. The list is named using sufficient cause IDs (see below). List elements are either NA (if a sufficient cause's status is "always") or a data.frame (if a sufficient cause's status is "depends" or "depends (potential order implausibilities)". Data.frames contain two columns, which are called "order" and "suff" (short for "sufficient"), and one row for every order of occurrence. The order of occurrence is summarized in "order" (as character), while "suff" is either TRUE or FALSE indicating if the corresponding order of occurrence is sufficient, i.e., leads to the outcome, or not.

sc_implausibilities A named vector of TRUE and FALSE with length equal to the number of sufficient causes. The names are sufficient cause IDs (see below). Is TRUE if for the corresponding sufficient cause there are potential order implausibilities, i.e., if its status is "depends (potential order implausibilities)", and is FALSE otherwise.

sc_implausibilities_detail A list with one list element for every sufficient cause. The list is named using sufficient cause IDs (see below). List elements are either NA (if the corresponding element in sc_implausibilities is FALSE) or a character vector (if the corresponding element in sc_implausibilities is TRUE) with the THEN statements of the steps that might be involved in implausible orders of occurrence.

sc_use_modules Either TRUE or FALSE indicating if modules have been specified in the steplist.

unknown_cc Similar to sc_cc but includes unknown component causes and an unknown sufficient cause (see "Unknown causes" below). It therefore additionally contains:

- one column to the right for every sufficient cause with name "Urownumber" (U1, U2, etc.) and all values equal to FALSE appart from row rownumber, which is TRUE

- one additional column to the right with name "USC" and all values equal to FALSE for all sufficient causes
- one additional row with name "cc0" and all values equal to FALSE apart from column "USC", which is TRUE

`unknown_status` Similar to `sc_status` but has one additional element with value "unknown" and name "cc0" (see "Unknown causes" below).

`steplist` The object of class `epicmodel_steplist_checked` that has been the input to function `create_scc()`, from which the `epicmodel_scc` object has been created.

Other details::

`Sufficient cause IDs` `create_scc()` checks every combination of component causes for sufficiency. Every combination is assigned an ID of the format "`ccnumber`" (cc1, cc2, etc.). `epicmodel_scc` only contains information about minimally sufficient combinations of component causes, but the initial IDs are kept. The IDs are used throughout the different elements of `epicmodel_scc` to link information that belongs to the same sufficient cause. The unknown sufficient cause used in elements `unknown_cc` and `unknown_status` has ID cc0.

`Unknown causes` Since many causes might be unknown, it is reasonable for some applications to include these unknown causes in a SCC model (see, e.g., Rothman et al. (2008)). They are also useful to remind us of our limited knowledge. In a suffcient-component cause model, unknown causes come in two flavors:

- `Unknown component causes`: These are additional component causes within a sufficient cause, which are necessary for sufficiency. Please note that each sufficient cause has its own set of unknown component causes. In `unknown_cc`, unknown component causes are called U1, U2, etc.
- `Unknown sufficient causes`: There might be unknown mechanisms that lead to outcome occurrence. These sufficient causes are summarized in one additional sufficient cause, which has only a single component cause called `USC` in `unknown_cc`. This set of component causes has sufficient cause ID `cc0`.

Please note that in `plot_dag()` an ellipse represents a **determinative set** of sufficient causes, as suggested and defined by VanderWeele & Robins (2007). A determinative set contains all sufficient causes and, therefore, in most cases, an unknown sufficient cause is necessary to at least achieve a theoretical determinative set. Determinative sets are important for creating causal diagrams (in the form of directed acyclic graphs) from SCC models. VanderWeele and Robins (2007) write (p. 1099, D refers to the outcome):
"To ensure that the DAG with the sufficient causation structure is itself a causal DAG, it is important that the set of sufficient causes for D on the graph be a determinative set of sufficient causes — that is, that the sufficient causes represent all of the pathways by which the outcome D may occur. Otherwise certain nodes may have common causes which are not on the graph, and the graph will then not be a causal DAG."
It can of course be argued that an unknown sufficient cause in the described form is hardly of any use when creating a causal graph (as a DAG) from a SCC model. Nonetheless, it can be, as mentioned, a placeholder and reminder of limited knowledge.

`Sufficiency status` The sufficiency status describes under which circumstances a certain set of component causes is sufficient. There are 5 possible values:

- `always`: The set of component causes is always sufficient.
- `depends`: The set of component causes is sometimes sufficient and sufficiency depends on the order of occurrence of the involved steps, because some of them contain IFNOT

conditions. However, if an IFNOT condition prevents the step from happening depends on the order of occurrence: if the IF condition is fulfilled before the IFNOT condition, the step (usually) occurs anyways, similar to how I do not care if a door is closed if I already went through it when it was still open.

- `depends (potential order implausibilities)`: Same as "depends", but in the list of potential orders of occurrence of the involved steps, there might be some that do not make sense in practice, e.g., when two steps with IFNOT conditions are chained together: Imagine Step1 having IF condition If1 and IFNOT condition Ifnot1, and Step2 having IF condition If2 and IFNOT condition Step1. The order Step1 -> Ifnot1 -> If1 -> If2 is not plausible because Ifnot1 occurred before If1 and therefore Step1 did never occur. The user needs to discard these orders of occurrence (as I am currently not confident to correctly remove only implausible ones with code).
- `never`: The set of component causes is never sufficient. This status is not used in `epicmodel_scc`. It's only used when investigating the effect of interventions (see `intervene()`).
- `unknown`: This is the status of the unknown sufficient cause, which is added to the SCC model. It's only used in element `unknown_status` of `epicmodel_scc` objects.

Value

- `new_scc()`: An object of class `epicmodel_scc`.
- `validate_scc()`: An object of class `epicmodel_scc` that has been checked to have the correct structure.
- `empty_scc()`: A (realtively) empty object of class `epicmodel_scc` with correct structure.
- `print.epicmodel_scc()`: Prints a summary of the object of class `epicmodel_scc` in the console.
- `summary.epicmodel_scc()`: Same as `print.epicmodel_scc()`.
- `plot.epicmodel_scc()`: A `ggplot` object.

References

- Rothman KJ, Greenland S, Poole C, Lash TL (2008): Causation and Causal Inference. In: Rothman KJ, Greenland S, Lash TL (Ed.): Modern epidemiology. Third edition. Philadelphia, Baltimore, New York: Wolters Kluwer Health Lippincott Williams & Wilkins, pp. 5–31.
- VanderWeele TJ, Robins JM (2007): Directed acyclic graphs, sufficient causes, and the properties of conditioning on a common effect. American Journal of Epidemiology 166 (9): 1096–1104.

See Also

- `create_scc()` for information on the algorithm for creating SCC models
- `plot_dag()` for how determinative sets of component causes are displayed in a DAG
- `intervene()` for the use of sufficiency status "never"

Examples

```
# epicmodel_scc object are created by create_scc()

# first, check your steplist of choice
steplist_checked <- check_steplist(steplist_rain)
# then, use it in create_scc()
scc_model <- create_scc(steplist_checked)

# new_scc() and validate_scc() are used inside create_scc()
# nonetheless, you can check its structure with validate_scc()
validate_scc(scc_model)

# print() and summary() both summarize the model in the console
print(scc_model)
scc_model
summary(scc_model)

# plot causal pies with plot()
plot(scc_model)
```

---

new_steplist *Steplist objects*

Description

The S3 classes `epicmodel_steplist` and `epicmodel_steplist_checked` store the input information for SCC model creation. They are created from the Steplist Creator `shiny` app, which can be launched with `launch_steplist_creator()`.

`new_steplist()`, `validate_steplist()`, and `empty_steplist()` define the S3 class.

`print()` prints a summary of the steplist entries in the console.

`summary()` prints a list of steps sorted by type of step in the console.

`plot()` renders a graph of the complete network of mechanisms in the RStudio Viewer.

Usage

```
new_steplist(x = list())

validate_steplist(x)

empty_steplist()

## S3 method for class 'epicmodel_steplist'
print(x, ...)

## S3 method for class 'epicmodel_steplist_checked'
print(x, ...)
```

```
## S3 method for class 'epicmodel_steplist'
summary(object, ...)

## S3 method for class 'epicmodel_steplist_checked'
summary(object, ...)

## S3 method for class 'epicmodel_steplist'
plot(x, ...)

## S3 method for class 'epicmodel_steplist_checked'
plot(x, modules = TRUE, module_colors = NULL, render = TRUE, ...)
```

Arguments

x	x is used in several functions: • new_steplist(): A list to be converted to class epicmodel_steplist. • validate_steplist(): An object of class epicmodel_steplist or epicmodel_steplist_checke to be validated. • print.epicmodel_steplist(): An object of class epicmodel_steplist. • print.epicmodel_steplist_checked(): An object of class epicmodel_steplist_checked. • plot.epicmodel_steplist(): An object of class epicmodel_steplist. • plot.epicmodel_steplist_checked(): An object of class epicmodel_steplist_checked.
...	Additional arguments for generics print(), summary(), and plot().
object	For summary.epicmodel_steplist(), an object of class epicmodel_steplist. For summary.epicmodel_steplist_checked(), an object of class epicmodel_steplist_checked.
modules	For plot.epicmodel_steplist_checked, TRUE (default) or FALSE, indicating if nodes in the same module should be colored equally (TRUE) or if all nodes have white background (FALSE). Colors are only applied, if modules have actually been specified in the epicmodel_steplist.
module_colors	For plot.epicmodel_steplist_checked, if nodes are colored by module, colors can be provided via this argument. Colors must be provided as a character vector. Both named colors and hexadecimal color codes are allowed. The function has 8 colors stored internally. If module_colors = NULL (default), these colors are used. If the model has more than 8 modules, module_colors must be specified. If more colors than necessary are specified, the function takes as many as necessary from the start of the vector.
render	For plot.epicmodel_steplist_checked, if TRUE (default), graph is directly rendered. IF FALSE, the output contains the non-rendered graph.

Details

epicmodel_steplist **objects:**

epicmodel_steplist objects are lists containing 8 data.frames. These data.frames are described below:

`what` A list of subjects and objects (WHAT segments) appearing in the step descriptions, e.g., cells, interleukins, symptoms, etc., with the following variables:

- id_what: Automatically created ID for WHAT segments. Starts with "a" followed by a number, e.g., a1. Used in creating automatic step IDs.
- key_what: Keyword describing the WHAT segment. Used in steplist_creator shiny app dropdown menus.
- desc_what: Text used in step descripiton.
- plural_what: Indicates if plural (1) or singular (0) version of the DOES description should be used, if this WHAT segment is used as subject, i.e., WHAT segment before the DOES segment.

`does` A list of actions or verbs (DOES segments), with which the WHAT segments interact, e.g., is present, produce, migrate, exposed to, with the following variables:

- id_does: Automatically created ID for DOES segments. Starts with "d" followed by a number, e.g., d1. Used in creating automatic step IDs.
- key_does: Keyword describing the DOES segment. Used in steplist_creator shiny app dropdown menus.
- subject_singular_does: Description used if subject (WHAT segment in front) has been specified as singular (plural_what=0).
- subject_plural_does: Description used if subject (WHAT segment in front) has been specified as plural (plural_what=1).
- no_subject_does: Description used if no subject (WHAT segment in front) has been specified.
- then_object_does: Indicates if the object for this DOES segment is a WHAT segment (0) or a THEN statement (1).

`where` A list of locations (WHERE segments), where the specified actions take place, e.g., in the airways, with the following variables:

- id_where: Automatically created ID for WHERE segments. Starts with "e" followed by a number, e.g., e1. Used in creating automatic step IDs.
- key_where: Keyword describing the WHERE segment. Used in steplist_creator shiny app dropdown menus.
- desc_where: Text used in step descripiton. Please include the corresponding preposition, e.g., 'in', 'into', 'on', etc.

`then` A list of combinations of WHAT, DOES and WHERE segments (THEN statements). A THEN statement can contain up to 4 segments: WHAT (subject), DOES, WHAT (object), WHERE. Not all 4 of them need to be specified. For some DOES segments, the corresponding object is not a WHAT segment but a THEN statement (see `then_object_does`). In general, all combinations are possible, although only DOES, only WHERE, and WHAT WHAT do not make a lot of sense. `then` exists to store the THEN statements that are later used in IF and IFNOT conditions. It contains the following variables:

- id_then: Automatically created ID based on segment IDs, e.g., a4, a1d5a15e9, d2a3.
- desc_then: Automatically created description based on segment descriptions.

`module` Modules are groups, into which the steps are sorted, e.g., immune system, lung, etc., as it is sometimes of interest to see which groups are involved in the sufficient causes. It contains the following variables:

- id_module: Automatically created ID for modules. Starts with "m" followed by a number, e.g., m1.

- key_module: Keyword describing the module.
- desc_module: Module description.

`step` Main table of interest and the one further processed to create sufficient-component cause models. It contains the following variables:

- id_step: Automatically created step ID based on IDs of included THEN statements, e.g., IFd6a10IFNOTd6a18+d1a8THENa11d3a12.
- desc_step: Automatically created step description based on descriptions of included THEN statements.
- end_step: Indicator variable that describes if this step is at the end of a certain sub-mechanism, e.g., symptom x occured.
- module_step: Module, i.e., group, into which this step has been sorted.
- note_step: Additional notes that are important for future users, e.g., if there are conflicting results or if the result is from a mouse model.
- ref_step: References on which this step is based.

`icc` ICC is short for incompatibel component causes. It contains pairs of component causes, i.e., steps without IF or IFNOT condition, that are not compatible with each other, i.e., cannot appear in the same sufficient cause. It contains the following variables:

- id_icc: Automatically created ID for ICC pairs. Starts with "i" followed by a number, e.g., i1.
- id1: Step ID of first component cause.
- id2: Step ID of second component cause.
- desc1: Step description of first component cause.
- desc2: Step description of second component cause.

`outc` A list that contains conditions under which the outcome of interest is assumed to occur. Each line might contain one or more THEN statements, that have been marked as end steps by setting step$end_step to 1. If more than one THEN statement is selected, they are combined with AND logic. All lines in this table are combined with OR logic, i.e., any of the specified conditions is assumed to represent outcome occurrence. The table contains the following variables:

- id_outc: Automatically created ID for outcome definitions as a combination of the THEN statement IDs connected by '+'.
- desc_outc: Automatically created description for the outcome definitions as a combination of the THEN statement descriptions.

`epicmodel_steplist_checked` **objects:**

Before using `epicmodel_steplist` object for SCC model creation in `create_scc()`, they need to be checked for any structures that might make SCC model creation impossible. Checking is performed by `check_steplist()` and if successful, the returned object is of type `epicmodel_steplist_checked`. When changing the steplist in the Steplist Creator `shiny` app or by functions `remove_all_modules()`, `remove_na()`, or `remove_segment()`, the steplist is "un-checked" and returned as class `epicmodel_steplist`. Apart from that, both classes have similar structure, which can be validated by `validate_steplist()`.

Value

- `new_steplist()`: An object of class `epicmodel_steplist`.

- validate_steplist(): An object of class epicmodel_steplist or epicmodel_steplist_checked, that has been checked to have the correct structure.
- empty_steplist(): An empty object of class epicmodel_steplist object with correct structure.
- print.epicmodel_steplist(): Prints the number of entries in each data.frame in the console and the information that the steplist is unchecked.
- print.epicmodel_steplist_checked(): Same as print.epicmodel_steplist() but with the information the the steplist has been checked successfully.
- summary.epicmodel_steplist(): Prints an allert that the steplist needs to be checked with check_steplist() before using summary().
- summary.epicmodel_steplist_checked(): Prints a list of steps by type of step in the console.
- plot.epicmodel_steplist(): Prints an allert that the steplist needs to be checked with check_steplist() before using plot().
- plot.epicmodel_steplist_checked(): Prints a graph of the complete network of mechanisms in the RStudio Viewer and the corresponding legend in the console.

Examples

```
# Create steplists in the Steplist Creator `shiny` app
if(interactive()){
launch_steplist_creator()
}

# Download the steplist from the `shiny` app
# Load the steplist into R
path <- system.file("extdata", "steplist_rain.rds", package = "epicmodel")
steplist <- readRDS(path)

# new_steplist(), validate_steplist(), and empty_steplist() are used in the `shiny` app
# nonetheless, you can check steplist structures with validate_steplist()
validate_steplist(steplist)

# print() provides a summary of steplist entries and if it's checked or unchecked
print(steplist)

# Check steplist before using `summary()` and `plot()`
steplist_checked <- check_steplist(steplist)
summary(steplist_checked)
plot(steplist_checked)
```

---

plot_dag *Plot DAG*

Description

Creates a ggplot from a dagitty object, using packages dagitty and ggdag. Mimics format and colors used on the dagitty homepage https://www.dagitty.net. Please note the recommendation in argument label_shift below: Getting the values for label_shift right can be an iterative and slightly tedious procedure. It is highly recommended to evaluate the result of the current values already in the saved plot using, e.g., ggsave and not in the RStudio Viewer.

Usage

```
plot_dag(
  dag,
  node_outc = NULL,
  node_expo = NULL,
  node_adj = NULL,
  node_latent = NULL,
  path_causal = NULL,
  path_biased = NULL,
  label = NULL,
  label_shift = NULL,
  label_size = 2.5,
  node_size = 7,
  node_stroke = 1,
  e_w = 0.4,
  cap_mm = 4,
  scc = FALSE,
  scc_size = c(0.1, 0.35),
  scc_shift = c(0, 0),
  scc_angle = 0
)
```

Arguments

dag	An object of class dagitty. Can be created by using dagitty::dagitty('[model_code]') or scc_to_dag(). If your DAG has been created by scc_to_dag(), make sure to pass only the first element (named dag) to plot_dag.
node_outc	A single element of type character or NULL (default). If the outcome has not yet been specified in dag, it can be done here by specifiying the name of the corresponding node.
node_expo	A single element of type character or NULL (default). If the exposure has not yet been specified in dag, it can be done here by specifiying the name of the corresponding node.
node_adj	A character vector or NULL (default). Specify the names of nodes that should be defined as "adjusted".
node_latent	A character vector or NULL (default). Specify the names of nodes that should be defined as "latent".
path_causal	A character vector or NULL (default). Specify the names of the paths in format "V1->V2" that should be defined as "causal".

path_biased — A character vector or NULL (default). Specify the names of the paths in format "V1->V2" that should be defined as "biased".

label — A named character vector or NULL (default). Change the name of nodes in the graph, i.e., labels. The vector elements correspond to the new names, the vector names correspond to the old node names, i.e., `label = c(old_name = "new_name")`.

label_shift — A named list (with all elements being numerical vectors of length 2) or NULL (default). Numerical values are used to move the labels of the corresponding nodes in x and y direction, respectively. The list names correspond to the nodes to which the values apply. Possible list names are the node names (initial names prior to changing them via `label`), node types, i.e., `outcome`, `exposure`, `adjusted`, `latent`, and `other`, as well as `all`, which applies to all nodes. If a node is addressed by several entries, e.g., its name and `all`, all entries are summed up. See the example below. Getting the values for `label_shift` right can be an iterative and slightly tedious procedure. It is highly recommended to evaluate the result of the current values already in the saved plot using, e.g., `ggsave` and not in the RStudio Viewer.

label_size — A single numeric value, which controls the font size of the label. Default is 2.5.

node_size — A single numeric value, which controls the size of the circle that represents the node. Default is 7.

node_stroke — A single numeric value, which controls the size of the black border around the node circles. Default is 1.

e_w — A single numeric value, which controls edge width. Default is 0.4.

cap_mm — A single numeric value, which controls the distance, i.e., white space, between when the node ends and the edge begins/ the edge ends and the node begins. Higher values correspond to shorter edges/arrows. Default is 4.

scc — TRUE or FALSE (default). Only applies to DAGs that are based on sufficient-component cause (SCC) models. If TRUE, an ellipse is added to the DAG, which should surround all sufficient cause variables, if they are a determinative set of sufficient causes, as suggested by VanderWeele and Robins (2007). If the DAG is not based on a SCC, leave `scc` at FALSE.

scc_size — A numeric vector of length 2, which controls the size of the ellipse. Default is c(0.1, 0.35).

scc_shift — A numeric vector of length 2, which controls the shift of the complete ellipse in x and y direction. Default is c(0, 0).

scc_angle — A single numeric value, which controls rotiation of the ellipse in degree units. Default is 0.

Value

A `ggplot` object.

References

VanderWeele TJ, Robins JM (2007): Directed acyclic graphs, sufficient causes, and the properties of conditioning on a common effect. American Journal of Epidemiology 166 (9): 1096–1104.

See Also

- dagitty::dagitty()
- scc_to_dag() for creating DAGs from SCC models
- SCC models for more information on SCC models

Examples

```
# Transform SCC model into a DAG
dag <- scc_to_dag(scc_rain)[["dag"]]

# Plot DAG
plot_dag(dag, label_shift = list(all = c(0,0.15), outcome = c(0.05,0)))

# plot_dag() works also with dagitty objects created in other ways
dag_to_plot <- dagitty::dagitty('dag {
bb="-2.628,-2.412,2.659,2.378"
V1 [pos="-2.128,-1.912"]
V2 [pos="-0.031,0.035"]
V3 [pos="2.159,1.878"]
V1 -> V2
V2 -> V3
}')
plot_dag(dag_to_plot, node_outc = "V3", node_expo = "V1", label = c(V3 = "outcome"))
```

---

remove_all_modules *Remove all modules*

Description

Removes all entries in data.frame module from an epicmodel_steplist object. Also turns all values of variable module_step in data.frame step from an epicmodel_steplist to empty strings.

Usage

```
remove_all_modules(steplist)
```

Arguments

steplist	An epicmodel_steplist or epicmodel_steplist_checked object.

Value

An epicmodel_steplist object with empty data.frame module and empty strings in variable module_step in data.frame step. When continuing with this steplist, SCC models cannot be inspected by module. If you made any changes, you need to call check_steplist() again.

Examples

```
x <- remove_all_modules(steplist_party)
```

remove_na *Removing NA in* icc *and* outc

Description

Remove any entries that only consist of NA from data.frames `icc` (Incompatible Component Causes) and `outc` (outcome definition) from an `epicmodel_steplist`.

Usage

```
remove_na(steplist)
```

Arguments

`steplist`	An `epicmodel_steplist` or `epicmodel_steplist_checked` object.

Value

An `epicmodel_steplist` object without entries in data.frame `icc`, which contain 'NA' in either `id1` or `id2` as well as entries in data.frame `outc` that contain 'NA' in `id_outc`. If you made any changes, you need to call `check_steplist()` again.

Examples

```
x <- remove_na(steplist_party)
```

remove_segment *Remove segments*

Description

Removes individual entries from data.frames `what`, `does`, `where`, `module`, or `icc`.

Usage

```
remove_segment(steplist, id)
```

Arguments

`steplist`	An `epicmodel_steplist` or `epicmodel_steplist_checked` object.
`id`	A single non-missing element of type character describing the ID of the entry you want deleted.

Value

An `epicmodel_steplist` class object. If you made any changes, you need to call `check_steplist()` again.

Examples

```
steplist_party <- remove_segment(steplist_party, "d4")
```

`scc_cause_sets` *Extracting component causes from SCC model*

Description

Extracting component causes by sufficient cause from an `epicmodel_scc` object.

Usage

```
scc_cause_sets(
  scc,
  output = c("id", "desc", "desc_no_start", "all"),
  depends = TRUE,
  unknown = FALSE
)
```

Arguments

`scc`	An object of class `epicmodel_scc`.
`output`	A single element of type character, which determines the type of output. Options are "id", "desc", "desc_no_start", and "all". See returns-part below for description.
`depends`	TRUE (default) or FALSE. If FALSE, only includes sufficient causes with sc_status "always".
`unknown`	TRUE or FALSE (default). If TRUE, unknown causes are added to the SCC model: every sufficient cause gets an additional individual unknown component cause representing additional unknown components; an unknown sufficient cause is added to the model consisting of a single unknown component cause and representing all unknown sufficient causes.

Value

A named list but its content depends on parameter "output". The names correspond to the component cause set IDs, i.e., `cc[[:digit:]]+`.

- id: Returns a named list of character vectors. Each vector contains the step IDs of its component causes.
- desc: Returns a named list of character vectors. Each vector contains the step descriptions of its component causes.
- desc_no_start: Returns a named list of character vectors. Each vector contains the step descriptions of its component causes, but with the "Start: " in the beginning removed.
- all: A named list of the three lists above. The names correspond to the corresponding option for parameter "output".

Examples

```
# Create some SCC model
steplist_checked <- check_steplist(steplist_rain)
scc_model <- create_scc(steplist_checked)

# Get sets of component causes that form the sufficient causes
scc_cause_sets(scc_model, output = "all")
```

`scc_rain` *Rain example SCC model*

Description

An example SCC model created from `steplist_rain`.

Usage

```
scc_rain
```

Format

An object of class `epicmodel_scc`, which is a list of 10 elements. See `new_scc()` for the detailed structure of `epicmodel_scc` class objects.

`scc_to_dag` *Transform SCC to DAG*

Description

Creates an object of class `dagitty` (`dagitty` package) from a SCC model, following VanderWeele and Robins (2007).

Usage

```
scc_to_dag(scc, unknown = TRUE)
```

Arguments

`scc`	An object of class `epicmodel_scc`.
`unknown`	TRUE (default) or FALSE. If TRUE, unknown causes are added to the SCC model: every sufficient cause gets an additional individual unknown component cause representing additional unknown components; an unknown sufficient cause is added to the model consisting of a single unknown component cause and representing all unknown sufficient causes.

Value

A list of length 2 containing an object of class dagitty (named dag) and a data.frame containing the information, which label in the DAG corresponds to which component cause (named legend).

References

VanderWeele TJ, Robins JM (2007): Directed acyclic graphs, sufficient causes, and the properties of conditioning on a common effect. American Journal of Epidemiology 166 (9): 1096–1104.

See Also

- dagitty::dagitty()
- SCC models for more information on unknown causes and SCC models in general
- plot_dag() to create a ggplot object from dagitty model code

Examples

```
# Create some SCC model
steplist_checked <- check_steplist(steplist_rain)
scc_model <- create_scc(steplist_checked)

# Transform it into a DAG
scc_to_dag(scc_model)
```

---

sc_contain_steps *Do steps appear in sufficient causes?*

Description

Extracts from a SCC model, if certain steps are part of the mechanism of sufficient causes. If you want a list of all steps, ignore argument steps.

Usage

```
sc_contain_steps(scc, steps = NULL, output = c("nice", "table"))
```

Arguments

scc	An object of class epicmodel_scc.
steps	A character vector containing step IDs. IF NULL (default), provides a list of all steps.
output	A single element of type character, either "nice" (default) or "table". If "table", returns a list (or data.frame if steps = NULL). If "nice", a nicely formated output is printed in the console.

Value

Either a list (output = "table") with length equal to the number of sufficient causes and each element being a named vector of TRUE/FALSE with the variables in steps as names and TRUE indicating that the step appears in the corresponding sufficient cause, or a nicely formated output in the console (output = "nice"). If steps = NULL and output = "table", returns a data.frame, which contains variables id_step and desc_step from the epicmodel_steplist_checked data.frame step.

Examples

```
# Create some SCC model
steplist_checked <- check_steplist(steplist_rain)
scc_model <- create_scc(steplist_checked)

# Check if one or more steps are part of the mechanism for each sufficient cause
sc_contain_steps(scc_model, c("THENa1","THENa5"))
```

---

show_steps *Show all steps of a SCC model*

Description

Prints all steps that are part of a sufficient-component cause model. The function wraps sc_contain_steps() with steps = NULL.

Usage

```
show_steps(scc, output = c("nice", "table"))
```

Arguments

scc	An object of class epicmodel_scc.
output	A single element of type character, either "nice" (default) or "table". If "table", returns a data.frame. If "nice", a nicely formated output is printed in the console.

Value

Either a data.frame (output = "table") with variables id_step (step ID) and desc_step (step description) and one row for every step in the model, i.e., from the epicmodel_steplist_checked data.frame step, or a nicely formated output in the console (output = "nice").

Examples

```
# Create some SCC model
steplist_checked <- check_steplist(steplist_rain)
scc_model <- create_scc(steplist_checked)

# Show all steps
show_steps(scc_model)
```

steplist_party *Birthday party example steplist*

Description

An example steplist, which contains the steps that tell Clara, under which conditions her birthday party will be a success.

Usage

```
steplist_party
```

Format

An object of class `epicmodel_steplist`, which is a list of 8 data.frames. See `new_steplist()` for the detailed structure of `epicmodel_steplist` class objects.

steplist_rain *Rain example steplist*

Description

An example steplist, which contains a rain-themed example to illustrate intervention functions.

Usage

```
steplist_rain
```

Format

An object of class `epicmodel_steplist`, which is a list of 8 data.frames. See `new_steplist()` for the detailed structure of `epicmodel_steplist` class objects.

Index