Package: SSPICE (via r-universe)

November 13, 2026

Type Package

Title Systematic Skip-Pattern Integrity and Consistency Engine

Version 0.1.0

Description Provides a graph-based framework for representing, analyzing, and validating structured statistical data governed by explicit logical dependencies, including questionnaire routing and other rule-based data structures. It supports the construction and structural analysis of directed acyclic graphs, preparation of record-level data, detection and localization of logical and skip-pattern inconsistencies, and generation of variable domains and formal compatibility rules. It also provides tools for evaluating empirical closure and minimality properties of the resulting Horn theory and for supporting the correction of inconsistent statistical records.

License GPL (>= 3)

Encoding UTF-8

Language en-US

Depends R (>= 4.2.0)

Imports gmp, cli, doParallel, dplyr, foreach, jsonlite, parallel, pdftools, readr, stringi, stringr, tibble, withr (>= 2.5.0)

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

Config/testthat/edition 3

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation no

Author Pierpaolo Massoli [aut, cre]

Maintainer Pierpaolo Massoli <pimassol@istat.it>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-13 17:53:49 UTC

RemoteUrl https://github.com/cran/SSPICE

RemoteRef HEAD

RemoteSha 7e6cab926ec5a39553402e2c695e80056ded4b9d

Contents

build_DAG 2
cached_routing_check 3
calculate_hits 4
correct_data 5
correction_control 8
correction_quality_report 10
count_admissible_patterns 12
find_isolated_nodes 13
find_longest_path 13
prepare_data 14
print.sspice_dag 15
print.sspice_prepared_data 15
routing_check 16
validate_extended_elog 17
validate_horn_theory 18

Index **20**

`build_DAG` *Build and analyze a directed acyclic graph*

Description

Builds the SSPICE graph representation from routing metadata. Logical OR and XOR groups are coarsened before constructing the transposed incidence matrix and the adjacency matrix. No record-level data are required.

Usage

```
build_DAG(
  routing,
  separator = ";",
  na_strings = c("", ".", "NA", "NULL"),
  analyze = TRUE,
  stop_on_cycle = TRUE,
  max_paths_per_partition = 1e+06,
  start_node = 1L,
  end_node = NULL
)
```

Arguments

`routing`	Either the path to a routing CSV file or a data frame containing the routing metadata.
`separator`	Field separator used when `routing` is a file path.
`na_strings`	Character values interpreted as missing values when reading a routing file.

analyze	Logical value indicating whether structural diagnostics should be calculated.
stop_on_cycle	Logical value indicating whether to stop when the resulting graph is not acyclic.
max_paths_per_partition	Maximum estimated number of paths allowed in each graph partition.
start_node	Index of the starting node used for path diagnostics.
end_node	Optional index of the destination node used for path diagnostics. The last graph node is used by default.

Value

An object of class sspice_dag. It contains the original and transformed routing metadata, node names, the transposed incidence matrix Bt, the adjacency matrix A, and optional node-level and graph-level diagnostics.

Examples

```
routing <- data.frame(
  V1 = c("Q1", "Q2"),
  V2 = c(1, 1),
  V3 = c(1, 1),
  V4 = c(1, 1),
  V5 = c(0, 0),
  V6 = c(0, 0)
)
dag <- build_DAG(routing)
dag$Bt
dag$A
```

---

cached_routing_check *Inspect or build a cached routing-check artifact*

Description

The cache is reused only when its manifest matches the current input files, package version, cache schema and routing-check parameters. Input files are fingerprinted with MD5 checksums.

Usage

```
cached_routing_check(
  data,
  dag,
  record_id,
  logical_edit_names = NULL,
  items_mapping_file = NULL,
  cache_file,
  input_files,
```

```
  cache_policy = c("auto", "refresh", "readonly", "off"),
  check_function = NULL
)
```

Arguments

data	Record-level data passed to routing_check.
dag	Extended SSPICE DAG.
record_id	Name of the record identifier.
logical_edit_names	Optional logical-edit node names.
items_mapping_file	Optional items-mapping file.
cache_file	RDS file containing the cached routing-check result.
input_files	Named character vector of files on which the artifact depends.
cache_policy	One of "auto", "refresh", "readonly", or "off".
check_function	Optional routing-check implementation. Intended for package testing; by default SSPICE's routing_check() is used.

Value

A list containing result, manifest, and cache diagnostics.

---

calculate_hits *Calculate HITS scores*

Description

Calculates hub and authority scores for the nodes of a directed graph by applying the HITS iteration to its adjacency matrix.

Usage

```
calculate_hits(A, max_iter = 100L, tol = 1e-06)
```

Arguments

A	A square numeric adjacency matrix. Rows are origin nodes and columns are destination nodes.
max_iter	Maximum number of iterations.
tol	Convergence tolerance for the largest absolute score change.

Value

A list containing authority, hub, iterations, and converged.

`correct_data` *Correct questionnaire records using MCMC simulated annealing*

Description

Correction is restricted to the local neighbourhood of reported routing or logical-edit violations. Every fill or flip samples a mode from the node domain. Immutable donor matrices estimate the conditional distribution of each variable given its immediate predecessor and strata, including when the predecessor is missing, with predecessor-only and marginal fallbacks. Complete donor records are never copied or updated with corrected records.

Usage

```
correct_data(
  data,
  dag,
  record_id = NULL,
  donor_data = NULL,
  donor_data_validated = FALSE,
  donor_cache = NULL,
  vaux_metadata = NULL,
  logical_edit_metadata = NULL,
  immutable_variables = NULL,
  strata = NULL,
  imputation_method = c("donor", "user_prior", "uniform"),
  imputation_priors = NULL,
  domains = NULL,
  external_data_dir = ".",
  items_mapping_file = NULL,
  control = correction_control(),
  batch_output_dir = NULL
)
```

Arguments

`data`	Data frame to correct, already aligned with `dag`.
`dag`	An SSPICE DAG or extended DAG.
`record_id`	Record identifier. If omitted, `dag$record_id` is used.
`donor_data`	Optional data frame containing validated records.
`donor_data_validated`	Whether `donor_data` has already passed the complete DAG+ routing and logical-edit check.
`donor_cache`	Optional immutable cache built from the original valid donor records. Corrected records are never added to this cache.
`vaux_metadata`	Optional nine-column VAUX metadata.

`logical_edit_metadata`
: Optional nine-column LEDIT metadata.

`immutable_variables`
: Base variables that must never be changed. Sampled VAUX, mode-locked VAUX, and logical edits are classified automatically and must not be supplied here.

`strata`
: Variables used in addition to active DAG predecessors.

`imputation_method`
: `donor`, `user_prior`, `uniform`, or a named vector.

`imputation_priors`
: Optional named list of named probability vectors.

`domains`
: Optional named list overriding domains derived from routing.

`external_data_dir`
: Directory containing external VAUX sources.

`items_mapping_file`
: Optional complete path to `items_mapping.csv`.

`control`
: An object produced by `correction_control`.

`batch_output_dir`
: Optional directory for atomic per-batch checkpoints. Domains, donor distributions, caches, and the routing oracle are prepared once before batching and are never rebuilt between batches.

Details

`corrected_data` contains the supplied rows after correction in the prepared representation. `record_summary` contains per-record diagnostics and `resolved`; `unresolved_records` is its unresolved subset. `final_routing_check` contains the final violation reports. `donor_diagnostics` gives supplied and retained donor counts. `control`, `strata` and `immutable_variables` record settings; `parallel` and `batch_manifest` describe execution and checkpoints.

Random generation uses deterministic per-record seeds. The caller's random state and RNG kind are restored on exit, including on errors. Results do not depend on worker count or batch boundaries. Random sequences differ from the initial CRAN submission; checkpoints from that submission cannot be reused.

The operational Step 10, rather than this function, performs the initial EXACT/ERRONEOUS partition and pilot selection. A finite stochastic search does not guarantee a feasible correction or a global minimum-change result. See the overview vignette for a complete standalone example.

Value

An object of class `sspice_correction_result`.

Territorial strata in the example

In the supplied example, STRATUM identifies five geographical divisions of Italy: North-West (1), North-East (2), Centre (3), South (4), and Islands (5). It is used for stratification during correction. With freeze_strata = TRUE, stratum membership remains unchanged.

Preparation before the first batch

Before record correction begins, SSPICE prepares or reuses conditional distributions, donor information, variable domains and other supporting structures. Parallel execution also requires worker initialization. Step 10 performs an initial routing check unless a valid cached result is available.

Preparation time depends on the number of variables and records, the number of categories and strata, and the routing structure. Computing conditional distributions can take appreciable time before the first batch starts.

The first batch-completion message appears only after preparation and processing of that batch. A delay before this message does not by itself indicate that execution has stalled.

Distinguish total script elapsed time from the correction time reported in the output. Their difference is not a complete measure of initialization time, because some preparation may occur within the timed correction call. Measuring individual preparation stages requires separate timing.

Examples

```
local({
  routes <- read.table(text = paste(
    "ID;1;1;9999;0;0;.;.",
    "GATE;2;1;2;0;1;1;END",
    "DETAIL;2;1;2;0;0;.;.",
    "END;1;1;1;0;0;.;.", sep = "\n"),
    sep = ";", header = FALSE, na.strings = ".")
  dag <- SSPICE::build_DAG(routes, analyze = FALSE)
  raw <- data.frame(ID = 1:3, GATE = c(1, 2, 1),
                    DETAIL = c(NA, 1, 1), END = c(1, 1, 1))
  prepared <- SSPICE::prepare_data(raw, dag)

  # The current checker requires a mapping file, even with no recoding.
  mapping <- tempfile(fileext = ".csv")
  on.exit(unlink(mapping), add = TRUE)
  write.csv(data.frame(varname = character(), item_route = numeric(),
                       item_dataset = numeric()), mapping, row.names = FALSE)
  check <- SSPICE::routing_check(
    prepared$data, dag$routes_numeric, dag$Bt, "ID",
    basename(mapping), dirname(mapping)
  )
  print(check$full_report)
  stopifnot(setequal(unique(check$full_report$ID), 3))

  # Only the erroneous row is sent to correction. Donors are fixed.
  result <- SSPICE::correct_data(
    data = prepared$data[3, , drop = FALSE], dag = dag, record_id = "ID",
    donor_data = prepared$data[1:2, , drop = FALSE],
    items_mapping_file = mapping,
    control = SSPICE::correction_control(
      seed = 123L, max_iterations = 20L, use_parallel = FALSE
    )
  )
  print(result$corrected_data)
  print(result$record_summary)
```

```
  print(result$unresolved_records)
  stopifnot(identical(result$corrected_data$ID, prepared$data$ID[3]))
  # Resolution is read from the final checks, not assumed from termination.
  print(result$final_routing_check$full_report)
})
```

`correction_control` *Configure SSPICE data correction*

Description

Configure SSPICE data correction

Usage

```
correction_control(
  routing_weight = 1,
  logical_edit_weight = 1,
  vaux_weight = 1,
  lambda_change = 2,
  future_weight = 0.5,
  penalty_fill = 1,
  penalty_blank = 2,
  penalty_flip = 3,
  initial_temperature = NULL,
  final_temperature = 1e-04,
  cooling_fraction = 0.9,
  max_iterations = NULL,
  top_k = 10L,
  beta_fill = 1,
  max_nodes_per_iteration = 1L,
  logical_repair_max_evaluations = 1000L,
  logical_repair_max_changes = 2L,
  logical_repair_beam_width = 10L,
  minimum_donors = 1L,
  allow_unconditional_donor = TRUE,
  branch_fill_threshold = 0.5,
  freeze_strata = TRUE,
  seed = 1234L,
  use_parallel = FALSE,
  parallel_workers = NULL,
  batch_size = NULL,
  resume_batches = FALSE
)
```

Arguments

`routing_weight`, `logical_edit_weight`, `vaux_weight`
: Non-negative energy weights.

`lambda_change`
: Weight assigned to minimum-change distance.

`future_weight`
: Weight assigned to the local look-ahead cost.

`penalty_fill`, `penalty_blank`, `penalty_flip`
: Costs of elementary changes.

`initial_temperature`
: Initial SA temperature. NULL selects the maximum of final_temperature, the floored initial constraint score, and lambda_change times the largest elementary change penalty.

`final_temperature`
: Positive final SA temperature.

`cooling_fraction`
: Fraction of the chain used for linear cooling. The remaining iterations use the constant final temperature.

`max_iterations`
: Maximum iterations per record. NULL uses the number of DAG nodes; the search can terminate earlier.

`top_k`
: Maximum number of proposed modes per node.

`beta_fill`
: Compatibility parameter retained in the control object. It is validated and stored, but the current proposal implementation does not use it.

`max_nodes_per_iteration`
: Maximum local nodes explored at each MCMC iteration. The default reproduces the original node-by-node strategy.

`logical_repair_max_evaluations`
: Maximum candidate assignments checked by the generic local repair search for one violated logical edit.

`logical_repair_max_changes`
: Maximum number of base variables changed by one generic logical-edit repair proposal.

`logical_repair_beam_width`
: Maximum retained values per variable when the repair search must consider more than one simultaneous change.

`minimum_donors`
: Minimum stratum observations before backing off.

`allow_unconditional_donor`
: Whether the original marginal donor fallback may be used after conditional pools fail.

`branch_fill_threshold`
: Filled share above which a gate-flip proposal is generated for a skipped segment.

`freeze_strata`
: Whether stratum variables are fixed.

`seed`
: Random seed.

`use_parallel`
: Whether records are corrected in parallel with a PSOCK cluster registered through `doParallel`. The default preserves the sequential correction path.

parallel_workers
: Optional positive integer number of workers. When NULL and use_parallel = TRUE, SSPICE uses max(1, parallel::detectCores(logical = TRUE) - 2), capped at the number of records being corrected. This argument is mainly useful for testing or for explicitly limiting memory use.

batch_size
: Optional positive integer number of records collected per parallel batch. NULL preserves the previous all-at-once behaviour.

resume_batches
: Whether compatible completed batch checkpoints may be reused when correct_data() receives a batch_output_dir.

Value

A list of class sspice_correction_control.

Examples

```
control <- correction_control(seed = 123L, use_parallel = FALSE)
control$max_iterations
control$final_temperature
```

---

correction_quality_report

Build a quality report for an SSPICE correction run

Description

Produces read-only diagnostics describing the input data, violations before and after correction, changed cells, correction invasiveness, execution characteristics, categorical post-hoc uncertainty, and presence/absence actions on quantitative variables. It never changes the correction result or any input record.

Usage

```
correction_quality_report(
  original_data,
  correction_result,
  dag,
  corrected_data = NULL,
  initial_check = NULL,
  final_check = NULL,
  donor_data = NULL,
  record_id = NULL,
  output_dir = NULL,
  elapsed_seconds = NA_real_,
  donor_origin = NA_character_
)
```

Arguments

original_data	Data frame before correction.
correction_result	Object returned by correct_data.
dag	SSPICE DAG used for correction.
corrected_data	Optional corrected data frame. By default, correction_result$corrected_data is used.
initial_check	Optional routing-check object for the input data.
final_check	Optional routing-check object for the corrected data. By default, correction_result$final_routing_c is used.
donor_data	Optional immutable donor data used to estimate post-hoc uncertainty for filled or changed base variables.
record_id	Record identifier; defaults to dag$record_id or the first column of correction_result$record_summar
output_dir	Optional directory in which CSV, RDS, and HTML report files are written.
elapsed_seconds	Optional total execution time when it is not already represented in external run metadata.
donor_origin	Optional label describing whether donors were obtained from input records passing the initial check or supplied separately.

Details

Variable measurement type is determined without heuristic interpretation: textual columns are STRING; non-textual routing nodes with nitems == 1 and minval < maxval are QUANTITATIVE; all other variables are QUALITATIVE.

The result contains summary (metric/value pairs), variable_profile, record_summary, unresolved_records, violations_before, violations_after, changes, their record/variable summaries, uncertainty and quantitative_structural_changes.

The editable-cell denominator is the number of distinct processed IDs times the number of non-ID columns not marked immutable. It is not limited to active questions or base variables. Changed-cell percentage is 100 times the number of changes divided by this denominator, or NA if it is zero.

Categorical entropy uses natural logarithms; normalized entropy divides by the logarithm of positive support size (zero for support size at most one). These are post-hoc distributions at the final record, not confidence intervals or a replay of sampled proposals. User priors and domain overrides are not restored by this reconstruction. Inspect status and distribution source. No independent final validation is run: the supplied or stored final check is reused. See the quality vignette for interpretation.

Value

An object of class sspice_correction_quality_report.

`count_admissible_patterns` *Count the admissible data patterns induced by a routing DAG*

Description

Uses dynamic programming and arbitrary-precision integers to count all record patterns compatible with the modal transitions represented by `Bt`.

Usage

```
count_admissible_patterns(
  dag,
  root_variables = NULL,
  max_memo_states = 5000000L,
  verbose = TRUE
)
```

Arguments

`dag`	A DAG object produced by the SSPICE DAG-building pipeline.
`root_variables`	Optional root-variable names or indices. By default, roots are inferred from `Bt`.
`max_memo_states`	Maximum number of memoized states.
`verbose`	Whether to print a short execution summary.

Details

Inspect `n_admissible_patterns` and `n_all_unrestricted_patterns`, stored as arbitrary-precision integers. Use `as.character()` to display their full values without conversion to floating point. `n_memo_states` describes the memoized computation. Counts concern structural patterns represented by the DAG, not all possible numeric or textual values in the original questionnaire.

Value

An object of class `sspice_admissible_pattern_count`.

Examples

```
routes <- data.frame(V1 = c("ID", "Q", "END"),
  V2 = c(1, 2, 1), V3 = c(1, 1, 1), V4 = c(9999, 2, 1),
  V5 = 0, V6 = 0, V7 = NA, V8 = NA)
dag <- build_DAG(routes, analyze = FALSE)
x <- data.frame(ID = 1:2, Q = c(1, 2), END = c(1, 1))
counts <- count_admissible_patterns(dag, verbose = FALSE)
as.character(counts$n_admissible_patterns)
```

find_isolated_nodes *Find isolated graph nodes*

Description

Identifies nodes having neither incoming nor outgoing edges.

Usage

```
find_isolated_nodes(A)
```

Arguments

A	A square numeric adjacency matrix. Rows are origin nodes and columns are destination nodes.

Value

An integer vector containing the indices of isolated nodes.

find_longest_path *Find a longest path in a DAG*

Description

Finds a longest directed path between two selected nodes of a directed acyclic graph.

Usage

```
find_longest_path(A, vnames = NULL, start_node = 1L, end_node = nrow(A))
```

Arguments

A	A square numeric adjacency matrix. Rows are origin nodes and columns are destination nodes.
vnames	Optional vector of node names. When omitted, matrix row names are used, followed by integer node indices when row names are unavailable.
start_node	Index of the starting node.
end_node	Index of the destination node.

Value

A list containing `path`, `length`, `edge_length`, and `exists_path`. For backward compatibility, `length` is the number of nodes in the path; `edge_length` is the number of traversed edges.

prepare_data *Prepare record-level data for SSPICE validation*

Description

Creates auxiliary variables, coarsens logical OR and XOR groups, converts the resulting variables to the numeric representation used by SSPICE, and verifies their alignment with a graph created by build_DAG.

Usage

```
prepare_data(
  data,
  dag,
  vaux = NULL,
  external_data_dir = file.path("data-raw", "external"),
  data_separator = ",",
  metadata_separator = ";",
  na_strings = c("", ".", "NA", "NULL")
)
```

Arguments

data	Either a data frame or the path to a CSV file containing the record-level data.
dag	Either an object created by build_DAG or the path to an RDS file containing that object.
vaux	Either NULL, a data frame containing VAUX rules, or the path to a delimited VAUX metadata file.
external_data_dir	Directory containing external CSV files referenced by the VAUX metadata.
data_separator	Field separator used when data is a file path.
metadata_separator	Field separator used when vaux is a file path.
na_strings	Character values interpreted as missing values.

Details

data_original retains the input; data_with_VAUX, data_with_VOR, and data_with_VOR_and_VXOR retain intermediate representations. vaux_metadata holds auxiliary rules and varnames gives the aligned names. Use result$data for checking. Text can become presence indicators and logical-group members can become canonical nodes; the final data are not a lossless copy of the questionnaire.

Value

An object of class sspice_prepared_data. Its data element is the final numeric data frame aligned with the DAG. Intermediate results are retained in the remaining elements.

Examples

```
routes <- data.frame(V1 = c("ID", "Q", "END"),
  V2 = c(1, 2, 1), V3 = c(1, 1, 1), V4 = c(9999, 2, 1),
  V5 = 0, V6 = 0, V7 = NA, V8 = NA)
dag <- build_DAG(routes, analyze = FALSE)
x <- data.frame(ID = 1:2, Q = c(1, 2), END = c(1, 1))
prepared <- prepare_data(x, dag)
prepared$data
```

---

print.sspice_dag *Print an SSPICE DAG*

Description

Print an SSPICE DAG

Usage

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

Arguments

x	An object created by build_DAG.
...	Additional arguments currently ignored.

Value

x, invisibly.

---

print.sspice_prepared_data *Print prepared SSPICE data*

Description

Print prepared SSPICE data

Usage

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

Arguments

x	An object created by prepare_data.
...	Additional arguments currently ignored.

Value

x, invisibly.

---

routing_check *Check record conformity with an SSPICE routing structure*

Description

Runs the path-based routing check and optionally adds an exhaustive scan of logical-edit variables. The latter reports every LEDIT equal to NA, even when an earlier path violation prevents that node from being reached.

Usage

```
routing_check(
  dset_data,
  routes,
  Bt,
  varkey,
  items.maps,
  path.items.maps,
  logical_edit_names = NULL
)
```

Arguments

dset_data	Record-level data aligned with the routing DAG.
routes	Processed numeric routing table.
Bt	Transposed incidence matrix.
varkey	Name of the record identifier column.
items.maps	File name of the mapping CSV. Use a header-only file when no recoding is needed. Columns are variable, routing code, dataset code.
path.items.maps	Directory containing the item-mapping file.
logical_edit_names	Optional character vector naming logical-edit variables. NULL preserves the original routing-only behaviour.

Details

full_report has one row per reported violation, with the record key, variable labels, error code and message. summary_by_rec groups by identifier and message, so multiple rows can represent one record. summary_by_var groups by variable and message. Consistent records are absent from these summaries. violation_source is added to the full report only when logical-edit names are supplied. Item mapping translates dataset codes to structural routing codes without modifying the input records. See the routing guide for the mapping contract.

Value

A list containing `full_report`, `summary_by_rec`, `summary_by_var`, `record_id`, and `logical_edit_names`.

Examples

```
routes <- data.frame(V1 = c("ID", "Q", "END"),
  V2 = c(1, 2, 1), V3 = c(1, 1, 1), V4 = c(9999, 2, 1),
  V5 = 0, V6 = 0, V7 = NA, V8 = NA)
dag <- build_DAG(routes, analyze = FALSE)
x <- data.frame(ID = 1:2, Q = c(1, 2), END = c(1, 1))
mapping <- tempfile(fileext = ".csv")
write.csv(data.frame(varname = character(), item_route = numeric(),
                     item_dataset = numeric()), mapping, row.names = FALSE)
checked <- routing_check(x, dag$routes_numeric, dag$Bt, "ID",
                         basename(mapping), dirname(mapping))
checked$full_report
unlink(mapping)
```

---

validate_extended_elog

Validate an extended canonical representation with ELOG variables

---

Description

Selects empirical semantic edits, constructs their ELOG variables with accepted domain `1` and `2`, and verifies localization against the induced semantic ground truth.

Usage

```
validate_extended_elog(
  data,
  routing,
  id_col = "ID_RECORD",
  seed = 123L,
  min_outcome_count = 5L,
  max_search = 50000L,
  min_non_missing_rate = 0.25,
  max_levels = 12L
)
```

Arguments

`data`	A routing-consistent record-level data frame.
`routing`	Original routing metadata.
`id_col`	Name of the record identifier column.
`seed`	Random seed.

`min_outcome_count`
: Minimum required frequency for every ELOG outcome.

`max_search`
: Maximum number of candidate searches per logical form.

`min_non_missing_rate`
: Minimum eligible-variable completeness.

`max_levels`
: Maximum number of levels for an eligible variable.

Value

An object of class `sspice_elog_validation`.

`validate_horn_theory` *Validate the Horn theory induced by a routing DAG*

Description

Builds the elementary Horn basis induced by the modal rows of the transposed incidence matrix, verifies empirical closure equivalence, and evaluates rule indispensability through a deletion test.

Usage

```
validate_horn_theory(
  dag,
  rules_file = NULL,
  items_mapping_file = NULL,
  n_random = 5000L,
  max_seed_size = 8L,
  seed = 123L,
  add_bridges = TRUE
)
```

Arguments

`dag`
: A DAG object produced by the SSPICE DAG-building pipeline.

`rules_file`
: Optional path to a `RULES.dat` file to compare with the elementary rules generated directly from `Bt`.

`items_mapping_file`
: Optional path to the item-mapping CSV file used to express modal values in the same coding adopted by `RULES.dat`.

`n_random`
: Number of random multi-seed closure tests.

`max_seed_size`
: Maximum size of a random seed set.

`seed`
: Random seed used for the empirical closure tests.

`add_bridges`
: Whether to add technical VALUE-to-FILLED implications.

Details

Inspect `summary`, `canonical_tests`, `closure_tests`, `closure_failures`, `minimality_tests`, and `stored_rule_comparison`. Rule tables are retained in `elementary_rules`, `closure_basis` and `complete_rule_set`. Random multi-seed closure checks provide empirical evidence, not an exhaustive proof over all subsets. The deletion tests assess rule indispensability.

Value

An object of class `sspice_horn_validation`.

Index

build_DAG, 2, *14, 15*

cached_routing_check, 3
calculate_hits, 4
correct_data, 5, *11*
correction_control, *6*, 8
correction_quality_report, 10
count_admissible_patterns, 12

find_isolated_nodes, 13
find_longest_path, 13

prepare_data, 14, *15*
print.sspice_dag, 15
print.sspice_prepared_data, 15

routing_check, *4*, 16

validate_extended_elog, 17
validate_horn_theory, 18