Package: theoryforge (via r-universe)

November 22, 2026

Title Systematic Theory Development

Version 0.7.0

Description Provides a reproducible workflow for building, developing and testing scientific theories represented as versioned, machine-checkable objects. Functions read and validate theory objects, score them against a versioned rigour checklist, screen constructs for redundancy, derive the conditional independencies a causal theory implies, appraise amendments as progressive, degenerating or neutral, write preregistration and audit documents, compile 'lavaan' model syntax, simulate the construct network, map a theory onto the themes of a literature corpus and write diagram intermediate representations. A 'Python' package of the same name offers the same functions. Both follow one specification, so they produce the same verdicts and the same diagram intermediate representations.

License MIT + file LICENSE

URL https://github.com/pablobernabeu/theoryforge, https://pablobernabeu.github.io/theoryforge/r/

BugReports https://github.com/pablobernabeu/theoryforge/issues

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports jsonlite, stats, tools, utils, yaml (>= 2.3.8)

Suggests testthat (>= 3.1.7), knitr, rmarkdown, curl, dagitty, ggm, DiagrammeR, DiagrammeRsvg, htmltools, lavaan, Matrix, spelling

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 8.0.0

NeedsCompilation no

Author Pablo Bernabeu [aut, cre] (ORCID: <https://orcid.org/0000-0003-1083-2460>)

Maintainer Pablo Bernabeu <pcbernabeu@gmail.com>
Repository https://cran.r-universe.dev
Date/Publication 2026-11-22 00:20:02 UTC
RemoteUrl https://github.com/cran/theoryforge
RemoteRef HEAD
RemoteSha 3b853ec27a71d6b3fa6e36b787d075cc12973ba5

Contents

tf_add_alternative . . . 3
tf_add_assumption . . . 3
tf_add_construct . . . 4
tf_add_prediction . . . 5
tf_add_proposition . . . 6
tf_appraise_amendment . . . 7
tf_check . . . 8
tf_compile_sem . . . 9
tf_diagram . . . 11
tf_dossier . . . 13
tf_embedding_redundancy . . . 14
tf_example_names . . . 15
tf_example_path . . . 15
tf_fetch_corpus . . . 16
tf_implications . . . 17
tf_jaccard . . . 20
tf_landscape . . . 21
tf_lit_diagram . . . 22
tf_litmap . . . 23
tf_new_evidence_dois . . . 26
tf_osf_push . . . 27
tf_post_data_severity . . . 28
tf_preregister . . . 29
tf_read . . . 31
tf_read_corpus . . . 32
tf_redundancy_check . . . 32
tf_render_diagram . . . 33
tf_render_report . . . 34
tf_report . . . 35
tf_set_formal_model . . . 36
tf_severity . . . 37
tf_simulate . . . 38
tf_theory . . . 39
tf_tokens . . . 40
tf_validate . . . 41
tf_write . . . 42

Index **44**

tf_add_alternative *Add an alternative theory (BUILDING mode)*

Description

Add an alternative theory (BUILDING mode)

Usage

```
tf_add_alternative(theory, id, label, key_constructs = NULL)
```

Arguments

theory	A theory object (named list).
id, label	Alternative fields.
key_constructs	Optional character vector.

Value

The (mutated) theory object.

Examples

```
tf_theory("demo-1", "A demonstration theory") |>
  tf_add_alternative("alt1", "Cognitive appraisal account",
                     key_constructs = c("c_threat"))
```

tf_add_assumption *Add an auxiliary assumption (BUILDING mode)*

Description

Add an auxiliary assumption (BUILDING mode)

Usage

```
tf_add_assumption(theory, id, statement, added_for = NULL, protects = NULL)
```

Arguments

theory	A theory object (named list).
id, statement	Assumption fields.
added_for	Optional id of the prediction whose anomaly the assumption was added to answer. Leave it NULL for a core assumption. tf_appraise_amendment() counts an assumption added for an anomaly as ad hoc unless a prediction it protects that is new in the amended version, other than this one, is corroborated.
protects	Optional character vector of the ids of the predictions the assumption shields from refutation.

Value

The (mutated) theory object.

Examples

```
tf_theory("demo-1", "A demonstration theory") |>
  tf_add_assumption("a1", "Measurement error is negligible.")
```

`tf_add_construct` *Add a construct to a theory (BUILDING mode)*

Description

Appends a construct and a provenance entry, returning the mutated theory.

Usage

```
tf_add_construct(
  theory,
  id,
  label,
  definition,
  measurement = NULL,
  boundary_conditions = NULL
)
```

Arguments

`theory`	A theory object (named list).
`id`, `label`, `definition`	Construct fields.
`measurement`, `boundary_conditions`	Optional character vectors.

Value

The (mutated) theory object.

Examples

```
tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Physiological arousal",
                   "Bodily activation in response to a stressor.")
```

`tf_add_prediction` *Add a prediction to a theory (BUILDING mode)*

Description

Appends a prediction and a provenance entry, returning the mutated theory.

Usage

```
tf_add_prediction(
  theory,
  id,
  statement,
  type,
  derives_from = NULL,
  diagnostic_vs = NULL,
  severity = NULL,
  test = NULL
)
```

Arguments

`theory`	A theory object (named list).
`id`	The prediction's identifier.
`statement`	The claim in words.
`type`	The form of the claim, one of four. `"existence"` asserts that an effect or relation exists, without a direction. `"directional"` asserts a sign or an order, including comparisons, interactions, the invariance of a direction across groups and claims that an effect occurs only when a condition holds. `"interval"` asserts that a quantity lies in a stated range, the range the theory permits. `"point"` asserts one value, with the tolerance that measurement requires. The tolerance comes from measurement, and the theory itself permits only the one value. The label is self-declared, and no function checks it against the statement.
`derives_from, diagnostic_vs`	Optional character vectors.
`severity`	Optional declared pre-data severity from 0 to 1, stored only when given. The checklist's risk_severity item reads it, and reads the claim-form rubric's value (`tf_severity()`) for a prediction that declares none. `tf_dossier()` sets the two side by side.
`test`	Optional sampling plan for testing the prediction, a named list of any of `n_planned`, `unit`, `alpha`, `sided` (`"two"` or `"one"`), `sesoi` and `sesoi_scale` (the smallest effect of interest and its scale), `power` or `decision_probability` with an optional `power_lo` and `power_hi`, `method`, `design_ref` and `justification`. It is stored only when given. `tf_preregister()` lists it after the hypotheses, and nothing scores it.

Value

The (mutated) theory object.

Examples

```
tf_theory("demo-1", "A demonstration theory") |>
  tf_add_prediction("h1", "Arousal precedes threat appraisal.", "directional")

# A prediction with the sampling plan of its test.
tf_theory("demo-1", "A demonstration theory") |>
  tf_add_prediction("h1", "The switch cost lies between 20 and 60 ms.", "interval",
                    test = list(n_planned = 48, unit = "participants", alpha = 0.05,
                                sided = "two", power = 0.9))
```

---

tf_add_proposition *Add a proposition to a theory (BUILDING mode)*

Description

Add a proposition to a theory (BUILDING mode)

Usage

```
tf_add_proposition(theory, id, from, to, relation, mechanism = NULL)
```

Arguments

theory	A theory object (named list).
id, from, to, relation	Proposition fields. from is the source construct id (named from to match the schema field). relation is one of the directed relations "increases", "decreases", "causes", "mediates" (one link of a mediated path) and "moderates" (from modifies the effect of other causes on to), or "associates", covariance between from and to that the theory leaves unexplained.
mechanism	Optional mechanism string.

Value

The (mutated) theory object.

Examples

```
tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Arousal", "Bodily activation.") |>
  tf_add_construct("c_threat", "Perceived threat", "Appraised danger.") |>
  tf_add_proposition("p1", "c_arousal", "c_threat", "increases")
```

`tf_appraise_amendment` *Appraise an amendment as progressive, degenerating or neutral*

Description

Compares an amended theory `new` with its `prior` version by content, not by prediction ids, and returns a Lakatosian verdict with the evidence behind it.

Usage

```
tf_appraise_amendment(new, prior)
```

Arguments

`new`	The amended theory object (named list).
`prior`	The prior theory object (named list).

Details

A new prediction whose statement and type match a prediction the amendment dropped is that prediction renamed. A new prediction derived only from propositions the prior already held, and needing no new assumption, articulates old content. A prediction is corroborated when at least one test outcome passes it and none fails it. An assumption added for an anomaly (its `added_for` names the prediction it answers) is ad hoc unless a prediction it protects that is new in this version, other than the anomaly, is corroborated.

The verdict is `"progressive"` when a corroborated new prediction is neither an articulation nor underived, no assumption is ad hoc and no corroborated prediction is dropped. It is `"degenerating"` when an assumption is ad hoc and no such prediction exists, and `"neutral"` otherwise. Lakatos has only the first two. `"neutral"` is theoryforge's label for an amendment that meets neither rule, and the vectors returned tell its cases apart. Only `corroborated_new_registered` reads a test outcome's `registered`, and no date or `version` field is read, so which version is the prior is the caller's responsibility.

A test outcome whose `passed` is present and not `TRUE` or `FALSE`, in `new` (checked first) or in `prior`, is refused. Read as not passed, a quoted `"true"` could turn a progressive amendment into a degenerating one.

Value

A named list whose first element, `verdict`, is one of `"progressive"`, `"degenerating"` and `"neutral"`. `new_predictions`, `corroborated_new`, `ad_hoc_assumptions`, `articulated`, `underived`, `corroborated_new_registere` `dropped`, `dropped_corroborated`, `content_lost` and `new_anomalies` are ascending-sorted character vectors. `renamed` is a list of `list(prior, new)` pairs sorted by `new`, and `assumptions` holds one `list(id, added_for, class, independent)` per new assumption added for an anomaly, in file order. `independent` lists, each as `list(id, status)` and in the order of `protects`, the distinct predictions the assumption protects that are new in this version and differ from `added_for`, with their status in `new` (`"corroborated"`, `"refuted"`, `"mixed"` or `"untested"`). `class` is `"ad_hoc1"`

when there are none, `"ad_hoc2"` when none is corroborated and `"independently_corroborated"` otherwise, and `ad_hoc_assumptions` holds the ids of the first two classes. The elements come in the order `verdict`, `new_predictions`, `corroborated_new`, `ad_hoc_assumptions`, `articulated`, `underived`, `corroborated_new_registered`, `renamed`, `dropped`, `dropped_corroborated`, `content_lost`, `new_anomalies` and `assumptions`.

References

Lakatos, I. (1970). Falsification and the methodology of scientific research programmes. In I. Lakatos & A. Musgrave (Eds.), *Criticism and the growth of knowledge* (pp. 91-196). Cambridge University Press. doi:10.1017/cbo9781139171434.009

Meehl, P. E. (1990a). Appraising and amending theories: The strategy of Lakatosian defense and two principles that warrant it. *Psychological Inquiry*, 1(2), 108-141. doi:10.1207/s15327965pli0102_1

Examples

```
prior <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Arousal", "Bodily activation.") |>
  tf_add_construct("c_threat", "Perceived threat", "Appraised danger.") |>
  tf_add_proposition("p1", "c_arousal", "c_threat", "increases") |>
  tf_add_prediction("h1", "Threat rises with arousal.", "directional",
                    derives_from = "p1")
new <- prior |>
  tf_add_construct("c_avoid", "Avoidance", "Withdrawal from feared situations.") |>
  tf_add_proposition("p2", "c_threat", "c_avoid", "increases") |>
  tf_add_prediction("h2", "Avoidance rises 0.3 SD per SD of threat.", "point",
                    derives_from = "p2")
new$test_outcomes <- list(list(prediction_id = "h2", passed = TRUE))
tf_appraise_amendment(new, prior)$verdict
```

---

tf_check *Compute the rigour checklist report*

Description

Runs the full rigour checklist (12 items) over a theory object and returns a report, with the items in checklist order.

Usage

```
tf_check(theory)
```

Arguments

`theory`	A theory object (named list), such as one from `tf_read()`.

Details

Each item has a status (`"pass"`, `"warn"` or `"fail"`) and a score from 0 to 1. An item with nothing to assess has the status `"n/a"` and a `NULL` score: the redundancy screen with fewer than two constructs, the derivation chain with no prediction and parsimony when no auxiliary assumption was added in response to an anomaly. The aggregate score is the weighted mean of the applicable items' scores, times 100, and `coverage` is the share of the checklist's weight that was applicable.

Two values are refused before anything is scored, since no score built on them would be defensible. One is a test outcome whose `passed` is present and not `TRUE` or `FALSE`, such as the quoted string `"true"`. The other is a prediction `severity` that is present and not a finite number from 0 to 1. Both stop with an error, and `tf_validate()` with `full = TRUE` reports both.

Value

A named list with elements `theory_id`, `schema_version` (the theory's), `checklist_version` (the rigour checklist's, which is what the weights and thresholds came from), `maturity`, `aggregate_score` (0 to 100, to one decimal), `coverage`, `gate`, `n_blockers_failed` and `items`. `gate` is `"advisory"` at `draft` maturity, and otherwise `"blocked"` when an item whose `severity_if_fail` is `"blocker"` (falsifiability or derivation_chain) fails and `"pass"` when none does. `items` holds one list per checklist item, in checklist order, with `id`, `status`, `score`, `weight`, `severity_if_fail` (`"blocker"` or `"warning"`) and `citation`. An error is raised for a prediction severity that is not a finite number or is below 0 or above 1.

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Arousal", "Bodily activation.") |>
  tf_add_construct("c_threat", "Perceived threat", "Appraised danger.") |>
  tf_add_proposition("p1", "c_arousal", "c_threat", "causes",
                     mechanism = "Activation raises salience of threat cues.") |>
  tf_add_prediction("h1", "Arousal precedes threat appraisal.", "point")
report <- tf_check(theory)
report$aggregate_score
report$gate
```

`tf_compile_sem` *Compile a theory to lavaan model syntax*

Description

Compiles a theory's constructs and propositions into **lavaan** model syntax written for `lavaan::sem()`. Constructs with measurement indicators become a latent measurement model (`=~`), propositions become structural paths (`~`) and covariances (`~~`), and the covariances the theory rules out are fixed at zero. Comments say what to check before a fit. The output is deterministic.

Usage

```
tf_compile_sem(theory)
```

Arguments

`theory`	A theory object (named list), such as one from `tf_read()`.

Value

The lavaan model syntax as a single string (LF line endings, single trailing newline).

The structural model

Each proposition becomes a line in file order: a regression (`to ~ from`) for `causes`, `increases`, `decreases` and `mediates`, and a covariance (`from ~~ to`) for `associates`. A `moderates` proposition gives the moderator's main effect, `to ~ from`, unless a path relation already writes that line. A comment follows on the product term, which has to be added by hand: `to ~ x:from` for an observed predictor `x`, and `lavaan::sam()` or the modsem package when the variables are latent.

The propositions form the graph that `tf_implications()` reads. Two of its constructs that are both exogenous, with no proposition pointing into either, or both terminal, with neither pointing to another construct, are unrelated by the theory's account unless an association joins them. By default, `lavaan::sem()` frees the covariance of two exogenous latent variables and of two terminal variables, and takes that of two exogenous observed variables from the data. A fit with those defaults could not refute the claim. In the modality-switching example, two of the six independencies `tf_implications()` derives would go untested. The syntax therefore ends with a block that fixes each such covariance at zero (`a ~~ 0*b`), and deleting the block restores lavaan's defaults. A pair with a moderator keeps those defaults, because the product term changes the moderator's role.

lavaan takes the covariances of observed exogenous variables from the data only until a covariance line names one of them. It then treats that variable as random and fixes at zero each of its covariances that the syntax does not write. Where a zero line names such a variable, the block therefore ends by freeing each covariance between two observed exogenous variables that no association names, when either is a moderator (`x ~~ m`). The moderator then keeps lavaan's defaults.

Before fitting

The syntax is a starting point for a fit. Indicator names are the sanitised measurement entries and must match columns of the data, so entries written as column names, such as `heart_rate`, give syntax that fits as it stands. A comment lists each construct with a single indicator, whose residual variance `lavaan::sem()` fixes at zero, so the construct is treated as measured without error. A comment flags each feedback loop. A non-recursive model may not be identified (Bollen, 1989), and the panic-network example is not, so check identification before fitting. Some constructs have to be re-specified by hand. A manipulation is better entered as a coded observed variable and a categorical predictor with more than two levels as dummy variables. A covariate that a measurement entry only describes needs a variable of its own. Once the block names an observed predictor, a product term added by hand needs a free covariance with each observed exogenous variable that a covariance line names. lavaan fixes each one left unwritten at zero, and each takes a line of its own (`x:m ~~ x`).

Names

Every name is one lavaan reads as written. Construct ids that lavaan would refuse or misread, such as `self-efficacy`, `1arousal`, `NA` or its own keyword `efa`, are renamed. A comment after the

header records each renaming (`# renamed for lavaan: 'self-efficacy' -> self_efficacy`). An indicator is its measurement entry folded and lower-cased, with each run of other characters made one underscore. `i_` goes in front when the result would still be refused, so `7-point Likert rating` gives `i_7_point_likert_rating`. A comment writes each control character of an id, and each character outside the Basic Multilingual Plane, as `<U+XXXX>`, so that it stays one line that lavaan reads as a comment.

The function stops when two different ids would share a name, since lavaan would merge them. Such a pair is two constructs, two indicators of one construct or a construct and an indicator. A construct and an indicator collide when the construct is measured by its own name or named like an indicator of another construct. An indicator shared by two constructs, a cross-loading, is allowed.

References

Bollen, K. A. (1989). *Structural equations with latent variables*. Wiley. doi:10.1002/9781118619179

See Also

`tf_implications()` for the independencies the zero block encodes.

Examples

```
# Measurement entries written as column names give syntax that fits as it
# stands. Threat and avoidance both follow from arousal alone, so the
# syntax fixes the covariance of their disturbances at zero.
theory <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Arousal", "Bodily activation.",
                   measurement = c("heart_rate", "arousal_rating")) |>
  tf_add_construct("c_threat", "Perceived threat", "Appraised danger.",
                   measurement = "threat_rating") |>
  tf_add_construct("c_avoidance", "Avoidance", "Withdrawal from the trigger.",
                   measurement = c("avoidance_task", "avoidance_diary")) |>
  tf_add_proposition("p1", "c_arousal", "c_threat", "increases") |>
  tf_add_proposition("p2", "c_arousal", "c_avoidance", "increases")
cat(tf_compile_sem(theory))
```

`tf_diagram` *Render a diagram intermediate representation*

Description

Produces a deterministic diagram IR string for the requested type. The `engine` argument is accepted but has no effect, because the IR is engine independent (Graphviz DOT for the six digraph views, dagitty syntax for the causal DAG and SVG for the `venn`, `rigour` and `severity` views).

Usage

```
tf_diagram(theory, type = "nomological_net", engine = "graphviz")
```

Arguments

theory	A theory object (named list).
type	One of "nomological_net" (default), "provenance", "causal_dag", "development_roadmap", "pipeline", "context" (the theory, its scope and its rivals), "workflow" (the building-to-testing pipeline), "venn" (the scope overlap of the first three constructs in file order, each taken as the set of its boundary conditions, as an SVG), "rigour" (the checklist as a status grid, as an SVG) or "severity" (the claim-form riskiness of each prediction, the computed_severity of tf_severity(), as bars in an SVG).
engine	Rendering engine label, accepted but unused (default "graphviz").

Details

The causal DAG is read by **dagitty** and drawn by the apps with Graphviz. It writes a construct id bare only when the id is made of ASCII letters, digits and underscores, does not begin with a digit and is not a DOT keyword. The keywords are node, edge, graph, digraph, subgraph and strict, in any case. Any other id is double-quoted, with a double quote inside it escaped by a backslash, so self-efficacy stays one node, which dagitty would otherwise read as two. The view stops when it would write the id node or graph, which dagitty refuses even quoted.

The pipeline and workflow views give every test outcome a node of its own. The first outcome of a prediction is result_<id> (outcome_<id> in the workflow) and each later one result_<id>#<k> (outcome_<id>#<k>), where <k> counts the prediction's outcomes in file order, so a failed replication is not merged into an earlier pass. When a prediction has several outcomes, each label also carries the outcome's date.

The development_roadmap and rigour views score the checklist, so they stop, as tf_check() does, when a test outcome's passed is present and not TRUE or FALSE, or when a prediction's severity is not a finite number from 0 to 1.

Value

A single string ending in a newline: Graphviz DOT for the digraphs, dagitty syntax for the causal DAG and SVG for the venn, rigour and severity views.

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Arousal", "Bodily activation.") |>
  tf_add_construct("c_threat", "Perceived threat", "Appraised danger.") |>
  tf_add_proposition("p1", "c_arousal", "c_threat", "causes")
cat(tf_diagram(theory, "nomological_net"))
cat(tf_diagram(theory, "causal_dag"))
```

`tf_dossier` *Render a theory audit dossier (Markdown)*

Description

Assembles a reviewer-facing audit bundle: the header, the rigour-checklist table, the severity list, the test record, the evidence, the provenance list and the appended preregistration document. The output is deterministic, so the same theory always yields the same dossier.

Usage

```
tf_dossier(theory)
```

Arguments

`theory`	A theory object (named list), such as one from `tf_read()`.

Details

The header gives the checklist version, the aggregate score, the checklist coverage, the gate and the blockers that failed. The rigour table prints `n/a` for an item with nothing to assess. The severity list gives each prediction's claim-form rubric values and, when the prediction declares a severity, that value too, with a note when it exceeds the rubric by more than 0.2. The test record lists every test outcome in file order, with what it found and the observation, date and registration it records, and with the post-data severity of `tf_post_data_severity()` where it can be computed, an open bound written as `-inf` or `inf`. The evidence lists the prediction each entry supports, its direction and its DOI. The implied independencies appear once, in the preregistration.

It stops, as `tf_check()` does, when a test outcome's `passed` is present and not `TRUE` or `FALSE`, or when a prediction's `severity` is not a finite number from 0 to 1.

Value

The dossier Markdown as a single string (LF line endings, single trailing newline).

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Arousal", "Bodily activation.") |>
  tf_add_construct("c_threat", "Perceived threat", "Appraised danger.") |>
  tf_add_proposition("p1", "c_arousal", "c_threat", "causes",
                     mechanism = "Activation raises salience of threat cues.") |>
  tf_add_prediction("h1", "Effect is exactly 0.30.", "point")
cat(tf_dossier(theory))
```

`tf_embedding_redundancy` *Embedding-based pairwise construct-redundancy screen*

Description

For every unordered pair of constructs, embeds each definition with the supplied `embedder` and computes the cosine similarity (rounded to 6 decimals). The vectors of every compared pair must be of equal, nonzero length, or the pair is refused. Returns a data frame sorted by descending cosine then `(a, b)`, flagging pairs at or above `threshold` for review. This assistive screen complements the deterministic lexical `tf_redundancy_check()`, and its results are only as reproducible as the supplied `embedder`.

Usage

```
tf_embedding_redundancy(theory, embedder, threshold = NULL)
```

Arguments

`theory`	A theory object (named list), such as one from `tf_read()`.
`embedder`	A function mapping a definition string to a numeric vector.
`threshold`	Cosine threshold for the `"review"` flag. It defaults to the checklist's `embedding_similarity_max` (0.85). How high a cosine two unrelated definitions reach depends on the embedding model, so a threshold suited to one model may flag everything or nothing under another, and the value is best set for the model in use.

Value

A data frame with columns `a`, `b`, `cosine` and `flag`.

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Arousal", "bodily activation") |>
  tf_add_construct("c_threat", "Threat", "appraised danger")
# A toy deterministic embedder: bag-of-words counts over a fixed vocabulary.
vocab <- c("bodily", "activation", "appraised", "danger")
embedder <- function(def) {
  words <- strsplit(tolower(def), "\\s+")[[1]]
  vapply(vocab, function(w) sum(words == w), numeric(1))
}
tf_embedding_redundancy(theory, embedder)
```

tf_example_names *Names of the packaged example files*

Description

The bundled set is four example theories and two literature corpora, which `tf_read_corpus()` reads. `panic-corpus.yaml` is a small corpus designed to fall into four themes, and `openalex-panic-2026.corpus.yaml` holds 150 works fetched from OpenAlex and saved.

Usage

```
tf_example_names()
```

Value

A sorted character vector of the `.yaml` file names. The Python package's `example_names()` lists the same files.

Examples

```
tf_example_names()
```

tf_example_path *Path to a packaged example file*

Description

Path to a packaged example file

Usage

```
tf_example_path(name)
```

Arguments

`name`	File name, for example `"panic-network.theory.yaml"`. See `tf_example_names()`.

Value

The path to the installed file. A `name` that matches no installed file stops with an error that lists the names `tf_example_names()` returns.

Examples

```
tf_read(tf_example_path("panic-network.theory.yaml"))$id
```

`tf_fetch_corpus` *Build a corpus from the OpenAlex API (network call)*

Description

Assistive helper that builds a corpus by querying the OpenAlex works API (`https://api.openalex.org/works?search=..`. This is a network call to a live service whose results change over time, so it is not part of the package's deterministic core. Each work is mapped to `{id, doi, title, year, keywords, references}`, with the DOI as OpenAlex gives it (keywords falls back to the first five concepts when no keywords are present).

Usage

```
tf_fetch_corpus(
  query,
  per_page = 25,
  mailto = NULL,
  api_key = Sys.getenv("OPENALEX_API_KEY", ""),
  max_records = NULL
)
```

Arguments

`query`	Free-text search query.
`per_page`	Number of works to request in each page (default `25`). It may be 1 to 200, but OpenAlex supports pages of up to 100 and has deprecated larger ones.
`mailto`	Optional contact email. It is still sent, but OpenAlex now ignores it, since API keys replaced the polite pool it once selected.
`api_key`	An OpenAlex API key, by default the `OPENALEX_API_KEY` environment variable. It is sent only in an `Authorization: Bearer` header, never in the URL, the corpus or an error message. `""` or `NULL` sends no key.
`max_records`	Number of works to collect, paging as needed. `NULL` (the default) means `per_page`, which is one request.

Details

OpenAlex returns the works that match a search in pages of `per_page`, ranked by relevance, and a search usually matches far more works than one page holds. A `max_records` above `per_page` pages on through OpenAlex's cursor until that many works are collected or the results run out. Paging needs an API key: without one, OpenAlex serves the first page and refuses the next with HTTP 401, so the function stops and keeps none of the works already fetched. By default, `max_records` equals `per_page`, so one request is made, which works without a key. Each page is one request and costs USD 0.001. OpenAlex allows USD 0.10 a day without a key, about 100 one-page searches, and USD 1 with a free key. When OpenAlex refuses a request, as it does with HTTP 429 once the budget is spent and with HTTP 401 when a later page is asked for without a key, the function stops

with the status and, when the suggested **curl** package is installed, OpenAlex's own message. The request is not retried. Every request is abandoned after 30 seconds.

The arguments are checked before any request. `per_page` must be a whole number from 1 to 200 and `max_records` a positive whole number, and `api_key`, unless `NULL`, must be a single string that holds only visible ASCII characters once the spaces, tabs and line breaks around it are trimmed.

The corpus records where, when and how it was fetched in `source`. It gives the service, endpoint and query and the UTC time of retrieval (`retrieved`), then the number of works that matched (`total_count`), the number kept (`n_records`), the page size and the order (`sort`). The date matters because the keywords change. Since late September 2026, OpenAlex has written each work's keywords with a language model that reads its title, abstract and venue. It merges and splits that vocabulary over time. Works without keywords fall back to their concepts, a deprecated vocabulary with capitalised names that do not match the lower-case keywords in `tf_litmap()`. Save a fetched corpus and work from the saved file.

Value

A corpus object (named list) with `schema_version`, `id`, `source` and `records`.

Examples

```
## Not run:
# With OPENALEX_API_KEY set, for example in .Renviron, the key is sent in a
# request header.
corpus <- tf_fetch_corpus("panic disorder interoception",
                          per_page = 100, max_records = 400)
corpus$source
# null = "null" writes a missing DOI or count as null, where jsonlite would
# write an empty object.
path <- tempfile(fileext = ".json")
jsonlite::write_json(corpus, path, auto_unbox = TRUE, null = "null")
corpus <- tf_read_corpus(path)

## End(Not run)
```

`tf_implications` *Derive a theory's implied conditional independencies*

Description

Reads every proposition through the package's relation table. A directed relation (`"increases"`, `"decreases"`, `"causes"`, `"mediates"` or `"moderates"`) is an edge from `from` to `to`, and `"associates"` is a bidirected edge, covariance the theory leaves unexplained, as a latent common cause of the two constructs would. The constructs that a directed relation names are the vertices, and an association is an edge only between two of them. The function then takes every pair of vertices that no edge joins. The pair is stated independent given the parents of both when those m-separate it (Richardson, 2003), and otherwise given all the other ancestors of the two when those do. A pair that neither set separates is separated by no set of constructs (Richardson & Spirtes, 2002, Theorem 4.2), so the theory implies no independence for it, and it is listed under `inseparable`. These are the conditional

independencies the causal graph implies, each one a claim that data could refute. When every relation is directed and the graph is acyclic, they are the basis set of Pearl (1988) and Shipley (2000), from which every other independence the graph implies follows.

Usage

```
tf_implications(theory, cycles = "refuse")
```

Arguments

`theory`	A theory object (named list), such as one from `tf_read()`.
`cycles`	What to do with a graph whose directed edges form a cycle: `"refuse"` (the default) stops with an error, and `"sigma"` derives the statements by sigma-separation (see the section on feedback loops).

Details

Constructs that no directed relation names are left out, because silence about a construct is not a claim that it is independent of anything, and so are constructs without an id, which no proposition can name. A construct named only by associations would be a collider on every path through it, so leaving it out loses no statement about the others. A theory with no directed relations therefore comes back with no vertices and no error.

Value

A named list `list(theory_id, criterion, acyclic, constructs, n_edges, n_bidirected, feedback, implications, n_implications, inseparable)`. `criterion` names the separation criterion, `"m"` under `cycles = "refuse"` and `"sigma"` under `cycles = "sigma"`. `constructs` holds the vertices in file order. `n_edges` counts the directed edges the theory states and `n_bidirected` the bidirected ones, a pair stated twice counting once, before any acyclification. `feedback` lists the feedback loops, each a list of the ids of its members in file order and the loops in the order of their first members. A loop has two or more members, or one construct with an edge to itself. `acyclic` is `TRUE` when `feedback` is empty, which is always the case under `cycles = "refuse"`, and both are carried so that a serialised record states the verdict. Each entry of `implications` is a list `list(a, b, given, statement)`, where `statement` renders the claim as `a _||_ b | z1, z2`, and each entry of `inseparable` is `list(a, b)`. Pairs come in construct file order, as do the members of `given`.

Measurement

The statements concern constructs, and a study measures them with error. Error in a conditioning construct leaves part of the dependence that holding it fixed should remove. A conditional statement tested on observed scores is therefore rejected too often, and more often the larger the sample (Westfall & Yarkoni, 2016). Take a chain whose two paths have standardised coefficients of .5, with the middle construct measured at a reliability of .8. A partial-correlation test at the 5 per cent level then rejects the true statement in about 14, 29 and 51 per cent of studies of 200, 500 and 1,000 observations. Conditional statements are better tested with latent-variable models, which take the measurement error into account (Westfall & Yarkoni, 2016). The measurement model that `tf_compile_sem()` writes is one starting point, and each statement can be tested in it separately (Thoemmes et al., 2018).

Feedback loops

With `cycles = "sigma"`, a graph whose directed edges form a cycle is read by sigma-separation (Bongers et al., 2021), the criterion that holds when each feedback loop, a strongly connected component of the directed edges, has a unique equilibrium. The graph is replaced by its acyclification (Definition A.13): every member of a loop takes the parents of the whole loop from outside it, and the members of a loop are joined to each other by bidirected edges. Each pair the theory leaves unjoined is then tested in that graph as above, where m-separation equals sigma-separation in the theory's own graph (Proposition A.19). There, two members of one loop are always joined. So is a construct outside a loop to every member, when it enters one of them or when it or a member of its own loop is associated with one of them. No set separates such a pair, and when the theory itself leaves the pair unjoined, it is listed under `inseparable`. A theory that posits alternative stable states, such as a bistable network, has more than one equilibrium and violates the assumption, and its sigma statements are not guaranteed. On an acyclic graph, the two options give the same statements. Applying d-separation to a cyclic graph, as dagitty does with the `causal_dag` export, is valid only in special cases such as a linear model.

Refusals

The function stops in four cases. A value of `cycles` other than `"refuse"` or `"sigma"` is refused before the theory is read. Two constructs sharing an id would give one node two sets of parents, and a directed relation naming an undeclared construct would shrink the graph and so imply independencies the theory never claimed. With `cycles = "refuse"`, the default, a cycle among the directed edges is refused too, and the message names a cycle that was found and the `"sigma"` option.

References

Bongers, S., Forré, P., Peters, J., & Mooij, J. M. (2021). Foundations of structural causal models with cycles and latent variables. *The Annals of Statistics*, 49(5), 2885-2915. doi:10.1214/21-AOS2064

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann.

Richardson, T. (2003). Markov properties for acyclic directed mixed graphs. *Scandinavian Journal of Statistics*, 30(1), 145-157. doi:10.1111/14679469.00323

Richardson, T., & Spirtes, P. (2002). Ancestral graph Markov models. *The Annals of Statistics*, 30(4), 962-1030. doi:10.1214/aos/1031689015

Shipley, B. (2000). A new inferential test for path models based on directed acyclic graphs. *Structural Equation Modeling: A Multidisciplinary Journal*, 7(2), 206-218. doi:10.1207/S15328007SEM0702_4

Thoemmes, F., Rosseel, Y., & Textor, J. (2018). Local fit evaluation of structural equation models using graphical criteria. *Psychological Methods*, 23(1), 27-41. doi:10.1037/met0000147

Westfall, J., & Yarkoni, T. (2016). Statistically controlling for confounding constructs is harder than you think. *PLOS ONE*, 11(3), e0152719. doi:10.1371/journal.pone.0152719

See Also

`tf_diagram()` with `type = "causal_dag"`, which exports the same graph as dagitty syntax without reading it, and the methodological foundations article for the literature behind the causal-testability criterion.

Examples

```
# A mediated chain commits the theory to one thing it does not state
# directly: arousal and avoidance are independent once threat is held fixed.
theory <- tf_theory("mediation", "A mediated chain") |>
  tf_add_construct("c_arousal", "Arousal", "Bodily activation.") |>
  tf_add_construct("c_threat", "Perceived threat", "Appraised danger.") |>
  tf_add_construct("c_avoidance", "Avoidance", "Withdrawal from the trigger.") |>
  tf_add_proposition("p1", "c_arousal", "c_threat", "increases") |>
  tf_add_proposition("p2", "c_threat", "c_avoidance", "increases")

implied <- tf_implications(theory)
implied$n_implications
implied$implications[[1]]$statement

# Stating that arousal and avoidance also covary for reasons the theory
# leaves open withdraws that claim: the pair is joined by an edge.
covary <- tf_add_proposition(theory, "p3", "c_arousal", "c_avoidance", "associates")
tf_implications(covary)$n_implications

# Closing a feedback loop from threat back to arousal makes the graph cyclic.
# The default refuses it, and sigma-separation keeps the claim, since holding
# threat fixed still cuts the one way out of the loop towards avoidance.
loop <- tf_add_proposition(theory, "p3", "c_threat", "c_arousal", "causes")
try(tf_implications(loop))
implied <- tf_implications(loop, cycles = "sigma")
lapply(implied$feedback, unlist)
implied$implications[[1]]$statement
```

---

tf_jaccard *Jaccard similarity of two token sets*

Description

Returns 0.0 if both sets are empty and otherwise the size of the intersection divided by the size of the union, rounded to 3 decimals.

Usage

```
tf_jaccard(a, b)
```

Arguments

a, b	Character vectors of tokens (treated as sets).

Value

A numeric similarity in `[0, 1]`.

Examples

```
tf_jaccard(tf_tokens("arousal threat response"),
           tf_tokens("threat appraisal response"))
```

`tf_landscape` *Map a theory and its alternatives onto a literature landscape (deterministic)*

Description

Maps a theory's focal constructs and its registered alternatives onto the thematic structure of a corpus (computed by `tf_litmap()`). Each theme is tagged `"under_theorised"`, `"covered"` or `"crowded"`.

Usage

```
tf_landscape(
  theory,
  corpus,
  min_link = 2,
  max_token_share = 0.5,
  method = "components"
)
```

Arguments

`theory`	A theory object (named list), such as one from `tf_read()`.
`corpus`	A corpus object (named list), such as one from `tf_read_corpus()`.
`min_link`	Minimum co-occurrence count passed to `tf_litmap()` (default `2`). The co-citation map, which the landscape does not use, is not computed.
`max_token_share`	A number from 0 to 1 (default `0.5`). A word in the keywords of more than this share of the records is a field token and never matches.
`method`	How `tf_litmap()` builds the themes: `"components"` (the default) or `"simple_centres"`.

Details

A theme is matched by the words its keywords share with the focal theory's construct labels, or with an alternative's label and key constructs, and every match reports those words (`focal_terms` and `alternative_terms`). Three kinds of word never match. Field tokens, the words most of the corpus shares, are those in the keywords of more than `max_token_share` of the records. Phenomenon tokens are the words of the theory's title, which names the phenomenon that the focal theory and its rivals all explain. A construct word that also appears in the title does not match either. The third kind is a fixed list of words that name a kind of account: theory, model, account, hypothesis, framework and approach, with their plurals. One shared word is enough for a match.

The statuses count the registered accounts, the focal theory and its registered alternatives, that address a theme. A theme is `"under_theorised"` when none of them addresses it, which says nothing of accounts the theory does not register, `"covered"` when one does and `"crowded"` when two or more do. A crowded theme calls for predictions that discriminate between the accounts, and is not a finding of redundancy. `under_theorised_fronts` lists the ids of the under-theorised themes, and `redundancy_risk`, despite its name, the ids of the crowded ones.

The arguments are checked in the order `min_link`, `max_token_share`, `method`, then the corpus. The themes are those of `tf_litmap()`, built by `method` with its other defaults. With `"components"`, `tf_landscape()` gives the warning of `tf_litmap()` when one theme holds most of the linked keywords. With `"simple_centres"`, the result gains `method` after `theory_id`, and each theme its `centrality`, `density` and `quadrant`.

Value

A named list with elements `theory_id`, `max_token_share`, `field_tokens`, `phenomenon_tokens`, `themes`, `under_theorised_fronts` and `redundancy_risk`. Each theme holds `id`, `keywords`, `alternatives`, `focal`, `status`, `focal_terms` and `alternative_terms`, the last a list of `{id, terms}` in the order of `alternatives`.

Examples

```
theory <- tf_theory("demo-1", "A theory of panic") |>
  tf_add_construct("c_arousal", "Arousal", "Bodily activation.")
corpus <- list(
  schema_version = "1.0", id = "demo-corpus",
  records = list(
    list(id = "w1", keywords = list("arousal", "threat")),
    list(id = "w2", keywords = list("arousal", "threat")),
    list(id = "w3", keywords = list("avoidance", "exposure")),
    list(id = "w4", keywords = list("avoidance", "exposure"))
  )
)
tf_landscape(theory, corpus)
```

---

tf_lit_diagram *Render a literature-layer diagram intermediate representation*

Description

Produces a deterministic DOT string for the literature layer.

Usage

```
tf_lit_diagram(obj, type = "keyword_cooccurrence", max_edges = NULL)
```

Arguments

obj	A tf_litmap() result (for "keyword_cooccurrence" or "co_citation") or a tf_landscape() result (for "theme_landscape").
type	One of "keyword_cooccurrence" (default), "co_citation" or "theme_landscape".
max_edges	A positive integer that caps a "keyword_cooccurrence" or "co_citation" diagram at that many edges: the highest counts are kept, ties going to the earlier pair in (a, b) order, and only their endpoints are drawn. NULL (the default) draws every edge. "theme_landscape" ignores it.

Value

A single string ending in a newline.

Examples

```
corpus <- list(
  schema_version = "1.0", id = "demo-corpus",
  records = list(
    list(id = "w1", keywords = list("arousal", "threat")),
    list(id = "w2", keywords = list("arousal", "threat")),
    list(id = "w3", keywords = list("avoidance", "exposure")),
    list(id = "w4", keywords = list("avoidance", "exposure"))
  )
)
cat(tf_lit_diagram(tf_litmap(corpus), "keyword_cooccurrence"))

# On a large map, draw only the strongest edges.
fixture <- tf_read_corpus(system.file("fixtures", "panic-corpus.yaml",
                                      package = "theoryforge"))
cat(tf_lit_diagram(tf_litmap(fixture), "co_citation", max_edges = 1))
```

---

tf_litmap *Bibliometric map of a literature corpus (deterministic)*

Description

Computes keyword co-occurrence, thematic components and reference co-citation for a corpus. Records are read in file order.

Usage

```
tf_litmap(
  corpus,
  min_link = 2,
  method = "components",
  min_cocitation = NULL,
  min_theme_size = 2,
```

```
  max_theme_size = 10,
  max_df = 1
)
```

Arguments

corpus	A corpus object (named list), such as one from tf_read_corpus().
min_link	Minimum co-occurrence count for a keyword pair to be kept (default 2). A positive integer.
method	"components" (the default) or "simple_centres".
min_cocitation	Minimum count for a reference pair to be kept in co_citation. NULL (the default) uses min_link.
min_theme_size, max_theme_size	The smallest theme kept (default 2) and the largest a theme may grow (default 10) with "simple_centres". min_theme_size is a positive integer and max_theme_size an integer of at least 2 no smaller than min_theme_size.
max_df	A number from 0 to 1 (default 1). With "simple_centres", a keyword in more than this share of the records is a field term, left out of the map and listed in field_terms. The three theme settings are checked with either method, although only "simple_centres" reads them.

Details

The corpus is checked before anything is counted. It must hold a records list of mappings (an empty list is an empty corpus). A keyword or reference that is an integer becomes its decimal string, so an unquoted PubMed or Scopus id keys the same work in R and in Python. One of 2^{53} or more is refused, since R cannot hold every such integer exactly. A logical, a fraction or a nested value is refused: an unquoted NO (nitric oxide) or on in YAML is read as a logical, so quote it.

A reference that holds a DOI is keyed by the DOI, lower-cased and without its resolver prefix, as tf_new_evidence_dois() compares DOIs. One work cited as 10.1016/J.BRAT.2015.10.002 in one record and as https://doi.org/10.1016/j.brat.2015.10.002 in another is therefore one node, labelled 10.1016/j.brat.2015.10.002. Any other reference, such as a Scopus or OpenAlex identifier, is compared exactly as written, and so is every keyword.

Co-citation maps of real corpora are large. 200 OpenAlex records give about 11,000 reference pairs that share two or more citing records, so min_cocitation can be set above min_link, and tf_lit_diagram() can draw only the strongest edges.

method chooses how keywords are grouped into themes. With "components", the default, a theme is a connected component of the keyword map, and on a real corpus one or a few keywords shared by most records join nearly every keyword into one. When the largest theme holds more than half the linked keywords, a warning says so: such themes, and any landscape built on them, do not describe the field. The result is returned unchanged.

"simple_centres" is the co-word clustering of Coulter et al. (1998) and Cobo et al. (2011). Each link is weighted by the equivalence index $c^2/(df_a df_b)$, where c counts the records holding both keywords and df the records holding each. The strongest link between two unassigned keywords seeds a theme, which takes its strongest unassigned neighbour until it holds max_theme_size keywords, ties going to the keyword first in code-point order. A keyword whose links all reach keywords already in themes joins none. Themes smaller than min_theme_size are dropped. Each theme gains

its centrality (ten times the summed index of its links to the keywords of other themes) and its density (100 times the summed index of its internal links, over its size), the measures of Callon et al. (1991) as Cobo et al. (2011) scale them. It also gains its quadrant in the strategic diagram, split here at the median of each: `"motor"` (both high), `"basic"` (central but not dense), `"niche"` (dense but not central) or `"emerging_or_declining"` (both low). On real corpora, this gives bounded themes where components give one. Components remain the default for this release.

Value

A named list with elements `n_records`, `keywords` (the distinct keywords, sorted), `keyword_cooccurrence` (one `list(a, b, count)` per kept pair, `a` sorting before `b`, in `(a, b)` order), `themes` (one `list(id, keywords, size)` per theme, numbered `theme_1`, `theme_2`, ... in the order of their first keyword) and `co_citation` (as `keyword_cooccurrence`, over the reference keys). With `"simple_centres"`, each theme also holds `centrality`, `density` and `quadrant`, and the list ends with `method`, `parameters` (the five settings used) and `field_terms`.

References

Callon, M., Courtial, J. P., & Laville, F. (1991). Co-word analysis as a tool for describing the network of interactions between basic and technological research: The case of polymer chemistry. *Scientometrics, 22*(1), 155-205. doi:10.1007/BF02019280

Coulter, N., Monarch, I., & Konda, S. (1998). Software engineering as seen through its research literature: A study in co-word analysis. *Journal of the American Society for Information Science, 49*(13), 1206-1223. doi:10.1002/(SICI)10974571(1998)49:13<1206::AIDASI7>3.0.CO;2F

Cobo, M. J., López-Herrera, A. G., Herrera-Viedma, E., & Herrera, F. (2011). An approach for detecting, quantifying, and visualizing the evolution of a research field: A practical application to the Fuzzy Sets Theory field. *Journal of Informetrics, 5*(1), 146-166. doi:10.1016/j.joi.2010.10.002

Examples

```
corpus <- list(
  schema_version = "1.0", id = "demo-corpus",
  records = list(
    list(id = "w1", keywords = list("arousal", "threat")),
    list(id = "w2", keywords = list("arousal", "threat")),
    list(id = "w3", keywords = list("avoidance", "exposure")),
    list(id = "w4", keywords = list("avoidance", "exposure"))
  )
)
tf_litmap(corpus)

# Simple centres on the frozen OpenAlex corpus, where components give one
# theme.
openalex <- tf_read_corpus(tf_example_path("openalex-panic-2026.corpus.yaml"))
lm <- tf_litmap(openalex, method = "simple_centres")
table(vapply(lm$themes, function(th) th$quadrant, character(1)))
```

`tf_new_evidence_dois` *DOIs not already cited by a theory (deterministic)*

Description

Compares each DOI in `candidate_dois` against the theory's `evidence[].source_doi` and `alternatives[].source_doi` fields, by normalised form, so a fresh literature search, for example via OpenAlex, Scopus or any other source, can be checked against what the theory already engages with. The normalised form is the DOI itself, trimmed, lower-cased (ASCII letters only) and percent-decoded, wherever it sits in the text, so `doi: 10...`, `DOI 10...`, `doi.org/10...`, `https://www.doi.org/10...` and a URL with `%2F` all match the bare DOI, and trailing full stops, commas and semicolons are dropped. Returns the qualifying DOIs in their original form, deduplicated and sorted by normalised form. The function is deterministic and makes no network call. The search itself is left to whichever literature tool the caller prefers.

Usage

```
tf_new_evidence_dois(theory, candidate_dois)
```

Arguments

`theory`	A theory object (named list), such as one from `tf_read()`.
`candidate_dois`	Character vector of DOIs to check.

Value

A character vector of the candidate DOIs not already cited, deduplicated and sorted.

Examples

```
# The bundled panic theory cites one DOI as evidence and one for each of its
# two registered alternatives. All three count as already cited.
theory <- tf_read(system.file("fixtures", "panic-network.theory.yaml",
                              package = "theoryforge"))
tf_new_evidence_dois(theory, c(
  "10.1016/j.brat.2015.10.002",                   # cited as evidence
  "https://doi.org/10.1016/0005-7967(86)90011-2", # an alternative, in URL form
  "https://doi.org/10.1037/0033-2909.99.1.20"     # not yet cited
))
```

`tf_osf_push` *Deposit a theory's audit dossier to OSF storage*

Description

Builds (and optionally sends) a request to upload `tf_dossier(theory)` to OSF storage. With `dry_run = TRUE` (the default), the planned request is returned and nothing is sent. A live upload (`dry_run = FALSE`) requires both `token` and `node` (the OSF project id) and performs an authenticated `PUT` through the suggested **curl** package, which must be installed. Each request is abandoned after 30 seconds. The dossier is built first, so the function stops when `tf_check()` refuses the theory, with `dry_run = TRUE` as well.

Usage

```
tf_osf_push(
  theory,
  token = NULL,
  node = NULL,
  filename = NULL,
  dry_run = TRUE,
  base_url = .tf_OSF_BASE,
  overwrite = FALSE
)
```

Arguments

`theory`	A theory object (named list), such as one from `tf_read()`.
`token`	OSF personal access token (required when `dry_run = FALSE`).
`node`	OSF project (node) id, used to build the upload URL and required when `dry_run = FALSE`. An empty string counts as absent.
`filename`	Destination filename. It defaults to `<id>.dossier.md`, also when `NULL` or empty.
`dry_run`	When `TRUE` (default), return the planned request without sending it.
`base_url`	OSF storage base URL, by default `"https://files.osf.io/v1/resources/"`. Change it to reach another host. The node id is appended to it directly, so it must end in a slash.
`overwrite`	When `TRUE`, add a new version of an existing file of the same name instead of failing with HTTP 409. Default `FALSE`.

Details

OSF storage refuses to create a file whose name already exists in the project folder (HTTP 409), so depositing the same theory twice under the default filename fails. Pass a version-specific `filename`, or `overwrite = TRUE` to add a new version of the existing file: the folder is listed first and the dossier is sent to that file's upload link, the WaterButler route that records a new OSF version. When the folder holds no file of that name, `overwrite = TRUE` creates it as usual.

Value

When dry_run = TRUE, a list list(dry_run = TRUE, request = list(method, url, filename, content_bytes), note), with a lookup = list(method = "GET", url) entry before note when overwrite = TRUE. When dry_run = FALSE, list(dry_run = FALSE, status, filename) for the completed upload. A status outside 2xx stops with OSF upload failed with HTTP <status>, so a refused upload is never returned as a completed one.

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory")
tf_osf_push(theory)
tf_osf_push(theory, node = "abc12")$request$url
tf_osf_push(theory, node = "abc12", overwrite = TRUE)$lookup
```

---

tf_post_data_severity *Post-data severity of each bounded prediction*

Description

How severely each bounded claim passed its test, from the test outcome's estimate and standard error.

Usage

```
tf_post_data_severity(theory)
```

Arguments

theory	A theory object (named list), such as one from tf_read().

Details

A prediction states the region its claim allows in bounds, a mapping of lower and upper, either of which may be null (or infinite) for an open side. A test outcome records the estimate it observed and its standard error se. For an outcome with both, whose prediction declares bounds, the severity with which the claim passed is the smallest of $\Phi((estimate - lower)/se)$ and $\Phi((upper - estimate)/se)$ over the finite bounds, rounded to three decimals (Mayo & Spanos, 2006, 2011). It is the probability that the test would have given a result less in accord with the claim, had the claim been false at the bound. A one-sided claim, such as an effect above a smallest effect of interest, is a lower bound with the upper one open.

The computation is a normal approximation to the sampling distribution of the estimate. With two finite bounds, the value equals 1 minus the p value of the two one-sided tests (TOST) of equivalence. A point claim with lower equal to upper can pass with a severity of at most 0.5, so it needs a tolerance around its value. Unlike tf_severity(), which grades the form of a claim before any data, this value depends on the design and the data. The rigour checklist does not read it. tf_dossier() gives each value in its test record.

The function stops at the first outcome, in file order, that cannot be read, with one of these messages: `post_data_severity requires both estimate and se; test_outcome[<i>], ... requires a finite se greater th` and `... requires a finite estimate; ...`, and for its prediction's bounds `... requires bounds with numeric or nul` `... requires at least one finite bound; ...` and `... requires lower <= upper; ....` `tf_validate()` with `full = TRUE` reports each of these.

Value

A `data.frame` with one row per test outcome, in file order, that carries both an estimate and a standard error and whose prediction declares bounds. Its columns are `prediction_id`, `outcome` (the outcome's 0-based index among the test outcomes), `lower` and `upper` (`NA` for an open bound), `estimate`, `se` and `severity`.

References

Mayo, D. G., & Spanos, A. (2006). Severe testing as a basic concept in a Neyman-Pearson philosophy of induction. *The British Journal for the Philosophy of Science*, 57(2), 323-357. doi:10.1093/bjps/axl003

Mayo, D. G., & Spanos, A. (2011). Error statistics. In P. S. Bandyopadhyay & M. R. Forster (Eds.), *Philosophy of statistics* (pp. 153-198). Elsevier. doi:10.1016/B9780444518620.500058

Examples

```
theory <- tf_read(tf_example_path("modality-switching.theory.yaml"))
tf_post_data_severity(theory)

# A claim that |r| <= .10, tested with an estimate of 0 and a standard error
# of .10, passes with a severity of .841.
theory$predictions[[4]]$bounds <- list(lower = -0.10, upper = 0.10)
theory$test_outcomes <- list(list(prediction_id = "pred4", passed = TRUE,
                                  estimate = 0, se = 0.10))
tf_post_data_severity(theory)
```

---

tf_preregister *Render the hypotheses section of a preregistration*

Description

Produces a deterministic preregistration Markdown string for a theory and, if `path` is given, writes it (LF, single trailing newline).

Usage

```
tf_preregister(theory, path = NULL)
```

Arguments

theory	A theory object (named list), such as one from tf_read().
path	Optional destination path. When it is given, the Markdown is written with LF line endings.

Details

The header gives the theory id, the version when it has an id (with its parent, when one is named), the schema version, the maturity and whether every prediction's derivation references resolve: yes, no or not applicable (no predictions). The derivation line reports the checklist's reference check, which does not show that a prediction follows from the propositions it cites.

Each hypothesis is listed in file order with its claim type, statement and derivation, followed by the auxiliary assumptions that protect it (core or added for <id>), the registered rivals it discriminates and its test record. A hypothesis that a recorded outcome has already tested reads already tested: passed (<date>, registered <where>), one entry per outcome, and any other reads untested.

When at least one prediction states a sampling plan in its test block, a section headed Sampling plan (author-recorded; r follows, with one line per prediction in file order. A line gives what is recorded of the planned sample, alpha and sidedness, smallest effect of interest, power or decision probability, design analysis and justification. A prediction without a plan reads no sampling plan stated. Nothing scores the plan: power does not raise a claim's severity (Meehl, 1967), though it decides how informative a failed test is (Lakens, 2022). A theory without a plan has no such section.

A section then lists the conditional independencies that tf_implications() derives from the causal graph with its default settings, which are claims the theory makes without stating them, or says why there are none. The claim-form severity rubric closes the document.

The document is the part of a preregistration that the theory file holds. It is to be completed with the study design, the analysis plan, the inference criteria and whatever the sampling plan leaves out, the sections that a template such as PRP-QUANT asks for (Bosnjak et al., 2022).

It stops, as tf_check() does, when a test outcome's passed is present and not TRUE or FALSE, or when a prediction's severity is not a finite number from 0 to 1, and then writes nothing.

Value

The preregistration Markdown as a single string.

References

Bosnjak, M., Fiebach, C. J., Mellor, D., Mueller, S., O'Connor, D. B., Oswald, F. L., & Sokol, R. I. (2022). A template for preregistration of quantitative research in psychology: Report of the joint psychological societies preregistration task force. *American Psychologist*, 77(4), 602-615. doi:10.1037/amp0000879

Lakens, D. (2022). Sample size justification. *Collabra: Psychology*, 8(1), Article 33267. doi:10.1525/collabra.33267

Meehl, P. E. (1967). Theory-testing in psychology and physics: A methodological paradox. *Philosophy of Science*, 34(2), 103-115. doi:10.1086/288135

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_prediction("h1", "Effect is exactly 0.30.", "point")
cat(tf_preregister(theory))
```

`tf_read` *Read a theory object from a YAML or JSON file*

Description

Reads a theory object authored as YAML (or JSON, chosen by file extension) into a named list.

Usage

```
tf_read(path)
```

Arguments

`path`	Path to a `.yaml`, `.yml` or `.json` file.

Details

The file is read as the Python package's `theoryforge.read()` reads it. The text is UTF-8, and a byte-order mark is ignored. A YAML or JSON sequence is always a list, even with one element, so `tf_validate()` refuses `maturity: [draft]`. Unquoted `y`, `Y`, `n` and `N` stay strings, as do dates, while `yes`, `no`, `true`, `false`, `on` and `off` are logicals. Integers are decimal, octal or hexadecimal only, so `1:30` and `1_000` are strings, and so is a number written with a comma (`1,000`). An integer too large for an R integer is read as a double. A merge key (`<<`) lets the mapping's own keys win. A key repeated in any mapping, in YAML or JSON, stops with `(<path>) Duplicate map key: '<key>'`. A document that is not a mapping, such as a sequence or an empty YAML file, stops with `Theory data must be a mapping`.

Value

A named list holding the parsed theory object.

Examples

```
# Round-trip a theory through a temporary file.
theory <- tf_theory("demo-1", "A demonstration theory")
path <- tempfile(fileext = ".yaml")
tf_write(theory, path)
tf_read(path)
```

tf_read_corpus *Read a literature corpus from a YAML or JSON file*

Description

Reads a corpus object (`{schema_version, id, records}`) into a named list. The file extension sets the format: JSON for `.json` and YAML otherwise. The file is read by the rules that `tf_read()` follows, so an unquoted keyword such as `y` or `n` stays a string and a repeated key is refused. A document that is not a mapping stops with `Corpus data must be a mapping`. The records themselves are checked by `tf_litmap()` and `tf_landscape()`.

Usage

```
tf_read_corpus(path)
```

Arguments

`path`	Path to a `.yaml`, `.yml` or `.json` corpus file.

Value

A named list holding the parsed corpus object.

Examples

```
corpus <- list(
  schema_version = "1.0", id = "demo-corpus",
  records = list(
    list(id = "w1", keywords = list("arousal", "threat")),
    list(id = "w2", keywords = list("arousal", "threat"))
  )
)
path <- tempfile(fileext = ".json")
jsonlite::write_json(corpus, path, auto_unbox = TRUE)
tf_read_corpus(path)
```

tf_redundancy_check *Pairwise lexical similarity of construct definitions*

Description

Compares the definitions of every unordered pair of constructs and returns a data frame with one row per pair, sorted by descending similarity then `(a, b)` ascending. `similarity` is the Jaccard index of the two definitions' token sets and `overlap` their overlap coefficient, the shared tokens over the tokens of the shorter definition. `flag` is `"review"` for a near-duplicate, a similarity at or above the checklist's `redundancy_similarity_max`, and for a definition contained in the other, an overlap at or above `redundancy_overlap_max` when both definitions hold at least three tokens. It is `"ok"` otherwise.

Usage

```
tf_redundancy_check(theory)
```

Arguments

theory	A theory object (named list).

Details

The screen compares words, so it cannot detect empirical redundancy, two differently defined constructs that correlate almost perfectly once measurement error is corrected for (Le et al., 2010). Sibling constructs (Lawson & Robins, 2021) may share vocabulary without being redundant.

Value

A data frame with columns `a`, `b`, `similarity`, `overlap` and `flag`.

References

Le, H., Schmidt, F. L., Harter, J. K., & Lauver, K. J. (2010). The problem of empirical redundancy of constructs in organizational research: An empirical investigation. *Organizational Behavior and Human Decision Processes*, 112(2), 112-125. doi:10.1016/j.obhdp.2010.02.003

Lawson, K. M., & Robins, R. W. (2021). Sibling constructs: What are they, why do they matter, and how should you handle them? *Personality and Social Psychology Review*, 25(4), 344-366. doi:10.1177/10888683211047101

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Arousal",
                   "Bodily activation in response to a stressor.") |>
  tf_add_construct("c_threat", "Perceived threat",
                   "Appraised danger in response to a stressor.")
tf_redundancy_check(theory)
```

---

tf_render_diagram *Render a diagram in the viewer or as SVG*

Description

Renders a digraph view of a theory without leaving R. Where `tf_diagram()` returns the deterministic Graphviz DOT string, `tf_render_diagram()` passes that string to the DiagrammeR engine and returns either an interactive widget, which displays in the RStudio viewer and in R Markdown documents, or a standalone SVG string, ready to embed in a page or save to a file.

Usage

```
tf_render_diagram(x, type = "nomological_net", as = c("widget", "svg"))
```

Arguments

x	A theory object (named list), or a diagram IR string from tf_diagram() or tf_lit_diagram(), so literature diagrams render the same way.
type	The diagram type, as in tf_diagram(). Ignored when x is already an IR string.
as	Either "widget" (default), for display in the viewer and in R Markdown, or "svg", a standalone SVG string.

Details

The three chart views (venn, rigour and severity) are already SVG, so they are returned as-is under as = "svg" and wrapped for display under as = "widget". The causal_dag view emits dagitty syntax rather than DOT, so it is not rendered here. Paste it into a dagitty tool or the dagitty R package instead.

The DOT views need the suggested DiagrammeR package, and as = "svg" also needs DiagrammeRsvg. The chart views need neither, but showing one under as = "widget" needs htmltools. A missing package stops with a message naming it.

Value

With as = "widget", an htmlwidget for a DOT view, or a browsable htmltools HTML object for the venn, rigour and severity views. With as = "svg", a single SVG string.

See Also

tf_diagram() for the intermediate representation itself, which needs no optional packages and is identical in R and in Python.

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Arousal", "Bodily activation.") |>
  tf_add_construct("c_threat", "Perceived threat", "Appraised danger.") |>
  tf_add_proposition("p1", "c_arousal", "c_threat", "causes")
tf_render_diagram(theory, "nomological_net")
```

---

tf_render_report *Write a Quarto report for a theory*

Description

Writes a standalone Quarto report to path (forced to a .qmd suffix): a YAML header (title, format) followed by the deterministic tf_dossier(theory) body. Returns the written path. When render = TRUE, invokes quarto render, which requires a Quarto installation, and stops if that render fails. It stops before writing anything when tf_check() refuses the theory.

Usage

```
tf_render_report(theory, path, title = NULL, render = FALSE, to = "html")
```

Arguments

theory	A theory object (named list), such as one from tf_read().
path	Destination path. Any extension is replaced with .qmd.
title	Optional report title. It defaults to "theoryforge report: <title-or-id>", also when NULL, NA or empty. It is written as a YAML double-quoted scalar with backslashes, double quotes and control characters escaped, so a YAML parser reads back exactly the string passed. Quarto then reads the title as Markdown, so raw TeX such as \emph{} is dropped from HTML output and α becomes mathematics.
render	When TRUE, run quarto render on the written file and stop if it exits non-zero.
to	Quarto output format (default "html").

Value

The path of the written .qmd file.

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Arousal", "Bodily activation.")
path <- tempfile(fileext = ".qmd")
tf_render_report(theory, path)
```

---

tf_report *Render the rigour report as a string*

Description

Renders the result of tf_check() as a string. format = "json" returns valid, pretty-printed JSON. format = "html" returns an HTML fragment in which every interpolated value is escaped. Any other format stops, and so does a theory that tf_check() refuses.

Usage

```
tf_report(theory, format = "json")
```

Arguments

theory	A theory object (named list).
format	One of "json" (default) or "html".

Value

A single string.

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Arousal", "Bodily activation.") |>
  tf_add_prediction("h1", "Arousal precedes threat appraisal.", "point")
cat(tf_report(theory, format = "json"))
```

---

tf_set_formal_model *Set the formal model (BUILDING mode)*

Description

Set the formal model (BUILDING mode)

Usage

```
tf_set_formal_model(theory, type, spec_ref = NULL)
```

Arguments

theory	A theory object (named list).
type	Formal-model type, one of "ode", "abm", "network", "sem" or "none". Any but "none" passes the checklist's formalisation item.
spec_ref	Optional reference to the model specification.

Value

The (mutated) theory object.

Examples

```
tf_theory("demo-1", "A demonstration theory") |>
  tf_set_formal_model("ode", spec_ref = "models/panic.ode")
```

`tf_severity` *Per-prediction claim-form riskiness*

Description

A pre-data rubric of the form of each prediction's claim. `risk_score` is the base riskiness of the declared type: existence 0.1, directional 0.4, interval 0.7 and point 0.9. `computed_severity` discounts a directional prediction by 25 per cent, adds 0.1 when `diagnostic_vs` names a registered alternative and is capped at 1. The order of the types follows Popper (1959, sections 31-33) for nested claims about one quantity, and the directional penalty follows Meehl's (1967, 1990b) argument that a sign alone risks little. The base values, the discount and the bonus are the package's conventions, and neither source gives them.

Usage

```
tf_severity(theory)
```

Arguments

`theory`	A theory object (named list), such as one from `tf_read()`.

Details

How severely a claim is tested depends on the design and the data, which the rubric does not read. A prediction scores the same however it is tested, and a failed test leaves the value where it was.

Value

A `data.frame` with columns `prediction_id`, `type`, `risk_score` and `computed_severity`, with one row per prediction in file order.

References

Meehl, P. E. (1967). Theory-testing in psychology and physics: A methodological paradox. *Philosophy of Science*, 34(2), 103-115. doi:10.1086/288135

Meehl, P. E. (1990b). Why summaries of research on psychological theories are often uninterpretable. *Psychological Reports*, 66(1), 195-244. doi:10.2466/pr0.1990.66.1.195

Popper, K. R. (1959). *The logic of scientific discovery*. Hutchinson.

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_prediction("h1", "Effect is exactly 0.30.", "point") |>
  tf_add_prediction("h2", "Effect is positive.", "directional")
tf_severity(theory)
```

`tf_simulate` *Simulate a theory's construct network as a linear dynamical system*

Description

Treats each construct (in file order) as a state variable and each proposition whose relation is `increases`, `decreases`, `causes` or `mediates` as a signed linear coupling term. It then propagates `dX/dt = A X - damping * X` either with fixed-step (Euler) updates or exactly, with the matrix exponential of `(A - damping * I) * dt`. The result is deterministic. Construct ids must be unique, and duplicates are refused rather than assigned to an arbitrary state. A construct without an id is a state with no couplings, and two of them do not count as duplicates.

Usage

```
tf_simulate(
  theory,
  steps = 10,
  dt = 0.1,
  k = 1,
  damping = 0.5,
  init = 1,
  method = "euler"
)
```

Arguments

`theory`	A theory object (named list), such as one from `tf_read()`.
`steps`	Number of steps, a whole number of at least 0 (default `10`).
`dt`	Time between rows, a finite number greater than 0 (default `0.1`).
`k`	Coupling gain applied to each signed edge, a finite number (default `1.0`).
`damping`	Per-state linear decay, a finite number (default `0.5`).
`init`	Initial value for every state, a single finite number (default `1.0`).
`method`	`"euler"` (the default for this release) or `"exact"`.

Details

With `method = "exact"`, each step multiplies the state by the matrix exponential, computed by scaling and squaring a degree-18 Taylor polynomial (Moler and Van Loan, 2003, method 3), so the trajectory is the solution of the linear system at times `0, dt, 2 * dt, ...` whatever `dt` is.

With `method = "euler"`, the default for this release, the explicit step is stable only when $|1+dt\lambda| < 1$ for every eigenvalue λ of `A - damping * I`. A theory that decays at rate `damping` therefore grows without bound, alternating in sign, once `dt * damping` exceeds 2, and the steps turn a sustained oscillation into growth. Raising `damping` can cause a divergence as well as fail to stop one. The function warns when the Euler trajectory departs from the exact one by more than 5 per cent of `max(1, |exact|)` at any step. The default will change to `"exact"` in the next minor release.

When a state grows so large that a million times it is not a finite number, the function stops with a message naming the step and the state. The initial state is tested as step 0, so an `init` beyond about 1.8e302 in absolute value stops there.

The model is deliberately simple: one gain `k` for every coupling, `causes` and `mediates` taken as positive, `moderates` and `associates` coupling nothing, `functional_form` not read and a common initial value for every state. A linear system has one equilibrium, or a continuum of them when its matrix is singular, and never two separate ones, so it cannot show bistability.

Value

A named list `list(states, dt, steps, k, damping, init, method, ignored, opposed, trajectory)`, where `states` are the construct ids in file order and `trajectory` is a list of `steps + 1` numeric vectors (row 0 is the initial state), every value rounded to 6 decimals. The settings and the method are returned as given, because the trajectory cannot be reproduced without them. `ignored` lists the ids of the propositions that couple nothing (moderates, associates, a relation outside the schema's set or an endpoint that is not a declared construct), and `opposed` the `c(from, to)` pairs that carry couplings of both signs, which offset each other. An invalid setting is refused with an error.

References

Moler, C., & Van Loan, C. (2003). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review, 45*(1), 3-49. doi:10.1137/S00361445024180

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_construct("c_arousal", "Arousal", "Bodily activation.") |>
  tf_add_construct("c_threat", "Perceived threat", "Appraised danger.") |>
  tf_add_proposition("p1", "c_arousal", "c_threat", "increases")
sim <- tf_simulate(theory, steps = 5, method = "exact")
sim$states
sim$trajectory[[1]] # the common initial state
sim$trajectory[[length(sim$trajectory)]] # at time 5 * dt
```

`tf_theory` *Start a new, empty theory object (BUILDING mode entry point)*

Description

Seeds `schema_version = "1.0"` and a first provenance entry `{step:"1", action:"tf_theory", detail:<id>}`.

Usage

```
tf_theory(id, title, maturity = "building", theory_form = "network")
```

Arguments

id	Theory id.
title	Human-readable title.
maturity	Maturity stage, one of "draft", "building" (the default), "developing" or "testing". At "draft", the gate of tf_check() is advisory.
theory_form	Theory form, one of "variance", "network" (the default), "typology" or "process". No score depends on it.

Details

The builders do not check field values. tf_validate() reports a value outside its enum, except that it checks the formal-model type, an evidence entry's direction and a test block's sided only with full = TRUE.

Value

A theory object (named list).

Examples

```
tf_theory("demo-1", "A demonstration theory")
```

---

tf_tokens *Tokenise a string into a set of content tokens*

Description

Folds the text through the package's fold table (accented Latin letters to ASCII, Greek and Cyrillic letters to small unaccented ones) and lower-cases ASCII letters, then takes every maximal run of letters, marks and digits in any script (Unicode general categories L, M and N) as a token. Tokens shorter than 3 code points and the canonical English stopwords are dropped, and the unique set is returned. Scripts other than Latin, Greek and Cyrillic are compared as written, and text written without spaces, such as Chinese, gives one token per run.

Usage

```
tf_tokens(s)
```

Arguments

s	A single string (or NULL, treated as "").

Value

A character vector of unique tokens (possibly empty).

Examples

```
tf_tokens("The physiological arousal response to a threat")
# Accents and case are folded, so these give the same tokens.
tf_tokens("Na\u00efve \u00c9motion")
tf_tokens("naive emotion")
```

`tf_validate` *Validate a theory object*

Description

Checks a theory against the package's schema, `theory.schema.json`, without a JSON Schema engine, so that it runs wherever R does, webR included.

Usage

```
tf_validate(theory, full = FALSE)
```

Arguments

`theory`	A theory object (named list), such as one from `tf_read()`.
`full`	When `TRUE`, also check ids, cross-references and the rest of the schema.

Details

The default (`full = FALSE`) checks structure. It covers the required top-level fields, the `maturity` and `theory_form` enums, the top-level field names and the required fields and enums of each construct, proposition and prediction. It also checks that every collection is a list. A field that is absent, `NULL` or blank is reported as missing, and one that holds another type as such (`id must be a string`).

With `full = TRUE`, it also checks that every id is unique within its collection and that every cross-reference points to a declared id. The cross-references are the endpoints of a proposition, the derivations and diagnostics of a prediction and the target of an assumption, an evidence entry or a test outcome. It then checks the rest of the schema. That covers the required fields of assumptions, alternatives, evidence and test outcomes, a logical `passed`, the evidence direction and formal-model type enums, the version block and the `schema_version` pattern. It also covers numbers within [0, 1] where the schema asks for them, a prediction's `bounds` and `test` block and a test outcome's `estimate` and `se`. Last, it checks the type of every other field and of every entry of a string array.

Besides the ids and references, which the schema cannot express, the full check is stricter than a JSON Schema 2020-12 validator in three ways. A blank required string counts as missing. An entry of a string array must be a nonempty string. A prediction's `bounds` must hold a finite bound with `lower` not above `upper`, a test outcome's `estimate` and `se` and a `test` block's `sesoi` must be finite, that block's `n_planned` must lie within the range of a double and its `power_lo` must not be above `power_hi`. The full check is more lenient in three ways. A `NULL` optional field other than `theory_form` is absent, as is a single `NA`, which `tf_write()` writes as null. A single string stands

for a one-element array of strings. An empty `list()` stands for an empty mapping as well as an empty sequence.

Both checks are deterministic, and the Python package's `Theory.validate()` reports the same problems in the same order with the same text.

Value

`TRUE` (invisibly) on success. Otherwise, it stops with `"invalid theory object: "` followed by every problem found, joined by `"; "`.

Examples

```
theory <- tf_read(system.file("fixtures", "panic-network.theory.yaml",
                              package = "theoryforge"))
isTRUE(tf_validate(theory))              # required fields and enums
isTRUE(tf_validate(theory, full = TRUE)) # also ids, cross-references and the rest of the schema

# The failure path is the more informative one. Point a prediction at a
# proposition that was never declared.
broken <- theory
broken$predictions[[1]]$derives_from <- "p_missing"
tryCatch(tf_validate(broken, full = TRUE), error = conditionMessage)
```

`tf_write` *Write a theory object to YAML or JSON*

Description

Serialises a theory object to disk. The file extension sets the format: JSON for `.json` and YAML otherwise. Files are UTF-8 with LF line endings.

Usage

```
tf_write(theory, path)
```

Arguments

`theory`	A theory object (named list).
`path`	Destination path.

Details

Numbers keep 15 significant digits in both formats, logicals are written as `true` and `false`, and a missing value (`NA`) is written as null, which reads back as `NULL` in R and `None` in Python. Every field the schema types as an array of strings (`derives_from`, `diagnostic_vs`, `protects`, `measurement`, `boundary_conditions`, `key_constructs`) is written as an array, even when the theory holds it as a single string. A theory that validates against the package's own schema therefore still validates once written, and it reads back the same in R and in Python.

Value

The path (invisibly).

Examples

```
theory <- tf_theory("demo-1", "A demonstration theory")
tf_write(theory, tempfile(fileext = ".yaml"))
```

Index

tf_add_alternative, 3
tf_add_assumption, 3
tf_add_construct, 4
tf_add_prediction, 5
tf_add_proposition, 6
tf_appraise_amendment, 7
tf_appraise_amendment(), *3*
tf_check, 8
tf_check(), *12, 13*, *27*, *30*, *34, 35*, *40*
tf_compile_sem, 9
tf_compile_sem(), *18*
tf_diagram, 11
tf_diagram(), *19*, *33, 34*
tf_dossier, 13
tf_dossier(), *5*, *28*
tf_embedding_redundancy, 14
tf_example_names, 15
tf_example_names(), *15*
tf_example_path, 15
tf_fetch_corpus, 16
tf_implications, 17
tf_implications(), *10, 11*, *30*
tf_jaccard, 20
tf_landscape, 21
tf_landscape(), *23*, *32*
tf_lit_diagram, 22
tf_lit_diagram(), *24*, *34*
tf_litmap, 23
tf_litmap(), *17*, *21–23*, *32*
tf_new_evidence_dois, 26
tf_new_evidence_dois(), *24*
tf_osf_push, 27
tf_post_data_severity, 28
tf_post_data_severity(), *13*
tf_preregister, 29
tf_preregister(), *5*
tf_read, 31
tf_read(), *8*, *10*, *13, 14*, *18*, *21*, *26–28*, *30*, *32*, *35*, *37, 38*, *41*
tf_read_corpus, 32
tf_read_corpus(), *15*, *21*, *24*
tf_redundancy_check, 32
tf_redundancy_check(), *14*
tf_render_diagram, 33
tf_render_report, 34
tf_report, 35
tf_set_formal_model, 36
tf_severity, 37
tf_severity(), *5*, *12*, *28*
tf_simulate, 38
tf_theory, 39
tf_tokens, 40
tf_validate, 41
tf_validate(), *9*, *29*, *31*, *40*
tf_write, 42
tf_write(), *41*