Package 'theoryforge'

September 1, 2026

Title Systematic Theory Development

Version 0.6.0

Description Provides a rigorous, 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 lexical redundancy, and emit byte-identical diagram intermediate representations. The package is the feature-parity twin of a 'Python' package of the same name, with behaviour pinned by a shared specification so the two implementations produce identical verdicts.

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

Suggests testthat (>= 3.0.0), knitr, rmarkdown, httr, curl, dagitty, ggm, DiagrammeR, DiagrammeRsvg, htmltools, 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 CRAN

Date/Publication 2026-09-01 08:30:02 UTC

Contents

`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 reason the assumption was added.
`protects`	Optional character vector of prediction ids it protects.

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

Add a prediction to a theory (BUILDING mode)

Usage

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

Arguments

`theory`	A theory object (named list).
`id`, `statement`, `type`	Prediction fields.
`derives_from`, `diagnostic_vs`	Optional character vectors.

Value

The (mutated) theory object.

Examples

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

`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).
`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` against its `prior` version and returns a Lakatosian verdict.

Usage

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

Arguments

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

Value

A named list with `verdict` (one of `"progressive"`, `"degenerating"`, `"neutral"`) and the ascending-sorted character vectors `new_predictions`, `corroborated_new`, `ad_hoc_assumptions`.

References

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

Meehl, P. E. (1990). Appraising and amending theories. *Psychological Inquiry*, 1(2), 108-141. doi:10.1207/s15327965pli0102_1

Examples

```
prior <- tf_theory("demo-1", "A demonstration theory") |>
  tf_add_prediction("h1", "Effect is positive.", "directional")
new <- prior |>
  tf_add_prediction("h2", "Effect is exactly 0.30.", "point")
new$test_outcomes <- list(list(prediction_id = "h2", passed = TRUE))
tf_appraise_amendment(new, prior)
```

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), e.g. from tf_read().

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, gate, n_blockers_failed, and items (a list of per-item lists).

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 a **lavaan** model string: constructs with measurement indicators become a latent measurement model (=~), and propositions become structural paths (~), covariances (~~), or moderation comment lines. The output is deterministic.

Usage

```
tf_compile_sem(theory)
```

Arguments

theory	A theory object (named list), e.g. from tf_read().

Value

The lavaan model syntax 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.",
                   measurement = c("heart-rate variability",
                                   "self-reported arousal")) |>
  tf_add_construct("c_threat", "Perceived threat", "Appraised danger.",
                   measurement = c("threat appraisal questionnaire")) |>
  tf_add_proposition("p1", "c_arousal", "c_threat", "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 two digraphs, dagitty syntax for the causal DAG).

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" (construct scope overlap, as an SVG), "rigour" (the checklist as a status grid, as an SVG), or "severity" (per-prediction severity bars, as an SVG).
engine	Rendering engine label, accepted but unused (default "graphviz").

Value

A single string ending in a newline. Graphviz DOT for the digraphs, dagitty syntax for the causal DAG, and SVG for the Venn.

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 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), e.g. from `tf_read()`.

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), e.g. from `tf_read()`.
`embedder`	A function mapping a definition string to a numeric vector.
`threshold`	Cosine threshold for the `"review"` flag; defaults to the checklist's `redundancy_similarity_max`.

Value

A data frame with columns `a`, `b`, `cosine`, `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 one literature corpus, `panic-corpus.yaml`, which `tf_read_corpus()` reads.

Usage

```
tf_example_names()
```

Value

A sorted character vector of the `.yaml` file names. The Python twin's `example_names()` applies the same extension filter, so the two list the same set.

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, e.g. `"panic-network.theory.yaml"`. See `tf_example_names()`.

Value

The path to the installed file.

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: it depends on a live external service whose results change over time, so it sits outside the package's deterministic core. Each work is mapped to `{id, title, year, keywords, references}` (keywords falls back to the top concepts when no keywords are present).

Usage

```
tf_fetch_corpus(query, per_page = 25, mailto = NULL)
```

Arguments

`query`	Free-text search query.
`per_page`	Number of works to request (default 25); OpenAlex accepts 1 to 200.
`mailto`	Optional contact email for the OpenAlex "polite pool".

Value

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

Examples

```
## Not run:
corpus <- tf_fetch_corpus("panic disorder interoception", mailto = "me@example.org")

## End(Not run)
```

tf_implications *Derive a theory's implied conditional independencies*

Description

Reads the propositions whose relation is causal (`"causes"`, `"increases"`, `"decreases"`) as directed edges over the constructs they connect, checks that the resulting graph is acyclic, and returns its basis set: for every pair of non-adjacent constructs, the claim that the two are independent given the parents of both. A basis set implies every other conditional independence the graph entails, so it is the shortest complete statement of what the theory forbids in data, and each entry is something a study could find and refute.

Usage

```
tf_implications(theory)
```

Arguments

`theory`	A theory object (named list), e.g. from `tf_read()`.

Details

Constructs that no causal proposition connects are left out, because silence about a construct is not a claim that it is independent of anything. A theory with no causal propositions therefore comes back with an empty basis set and no error.

Value

A named list `list(theory_id, acyclic, constructs, n_edges, implications, n_implications)`. `constructs` holds, in file order, the constructs a causal proposition connects. `acyclic` is always `TRUE` in a returned record, since a cyclic graph is refused; it is carried so that a serialised record states the verdict rather than leaving a reader to infer that the check ran. Each entry of `implications` is a list `list(a, b, given, statement)`, where `statement` renders the claim as `a _||_ b | z1, z2`. Pairs come in construct file order, as do the members of `given`.

Refusals

The function stops in three cases. Two constructs sharing an id would give one node two sets of parents, and a causal proposition naming an undeclared construct would shrink the graph and so imply independencies the theory never claimed. A cyclic causal graph has no basis set at all, and the message names a cycle that was found. The Python twin raises `ValueError` on the same three, with the same message text.

References

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

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

See Also

`tf_diagram()` with `type = "causal_dag"`, which exports the same subgraph 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
```

`tf_jaccard` *Jaccard similarity of two token sets*

Description

Returns 0.0 if both sets are empty, 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)
```

Arguments

`theory`	A theory object (named list), e.g. from `tf_read()`.
`corpus`	A corpus object (named list), e.g. from `tf_read_corpus()`.
`min_link`	Minimum co-occurrence count passed to `tf_litmap()` (default 2).

Value

A named list with elements `theory_id`, `themes` (each {`id`, `keywords`, `alternatives`, `focal`, `status`}), `under_theorised_fronts`, and `redundancy_risk`.

Examples

```
theory <- tf_theory("demo-1", "Arousal and threat") |>
  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"))
  )
)
tf_landscape(theory, corpus)
```

---

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

Description

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

Usage

```
tf_litmap(corpus, min_link = 2)
```

Arguments

`corpus`	A corpus object (named list), e.g. from `tf_read_corpus()`.
`min_link`	Minimum co-occurrence count for an edge to be kept (default 2).

Value

A named list with elements `n_records`, `keywords`, `keyword_cooccurrence`, `themes`, and `co_citation`.

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"))
  )
)
tf_litmap(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")
```

Arguments

obj	A tf_litmap() result (for "keyword_cooccurrence" / "co_citation") or a tf_landscape() result (for "theme_landscape").
type	One of "keyword_cooccurrence" (default), "co_citation", or "theme_landscape".

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"))
  )
)
cat(tf_lit_diagram(tf_litmap(corpus), "keyword_cooccurrence"))
```

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 (lowercased, with any doi.org/dx.doi.org URL prefix stripped), so a fresh literature search, for example via OpenAlex, Scopus, or any other source, can be checked against what the theory already engages with. Returns the qualifying DOIs in their original form, deduplicated and sorted by normalised form. Deterministic and takes no network dependency: 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), e.g. 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. The live path is network- and credential-dependent.

Usage

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

Arguments

theory	A theory object (named list), e.g. 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.

filename	Destination filename; defaults to <id>.dossier.md.
dry_run	When TRUE (default), return the planned request without sending it.
base_url	OSF storage base URL; override to target a non-default host.

Value

When dry_run = TRUE, a list list(dry_run = TRUE, request = list(method, url, filename, content_bytes), note). When dry_run = FALSE, a list describing the completed upload.

Examples

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

---

tf_preregister *Render a preregistration document*

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), e.g. from tf_read().
path	Optional destination path; when given, the markdown is written with LF line endings.

Value

The preregistration markdown as a single string.

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.

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 format is chosen by the file extension (.json -> JSON, otherwise YAML).

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

Computes Jaccard similarity for every unordered pair of construct definitions. Returns a data frame with one row per pair, sorted by descending similarity then `(a, b)` ascending. The `flag` column is `"review"` when similarity meets or exceeds the configured `redundancy_similarity_max` threshold, otherwise `"ok"`.

Usage

```
tf_redundancy_check(theory)
```

Arguments

`theory`	A theory object (named list).

Value

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

References

Le, H., Schmidt, F. L., Harter, J. K., & Lauver, K. J. (2010). The problem of empirical redundancy of constructs. *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. *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), an htmlwidget for the viewer and for 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.

Value

An htmlwidget when `as = "widget"`; a single SVG string when `as = "svg"`.

See Also

`tf_diagram()` for the intermediate representation itself, which needs no optional packages and stays byte-identical across the R and Python implementations.

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.

Usage

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

Arguments

`theory`	A theory object (named list), e.g. from `tf_read()`.
`path`	Destination path; any extension is replaced with `.qmd`.
`title`	Optional report title; defaults to "`theoryforge report: <title-or-id>`". Double quotes are escaped.
`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.

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 (e.g. `"ode"`).
`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 risk and computed severity*

Description

Computes, for each prediction (in file order), the riskiness of the claim form and the discounted/bonus-adjusted severity.

Usage

```
tf_severity(theory)
```

Arguments

`theory`	A theory object (named list), e.g. from `tf_read()`.

Value

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

References

Mayo, D. G. (2018). *Statistical inference as severe testing*. Cambridge University Press. doi:10.1017/9781107286184

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

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 directed proposition as a signed linear coupling term, then integrates `dX/dt = A X - damping * X` with fixed-step (Euler) updates. The result is fully deterministic. Construct ids must be unique; duplicates are refused rather than resolved to an arbitrary state slot.

Usage

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

Arguments

`theory`	A theory object (named list), e.g. from `tf_read()`.
`steps`	Number of Euler steps (default `10`).
`dt`	Integration step size (default `0.1`).
`k`	Coupling gain applied to each signed edge (default `1.0`).
`damping`	Per-state linear decay (default `0.5`).
`init`	Initial value for every state (default `1.0`).

Value

A named list `list(states, dt, steps, k, damping, init, trajectory)`, where `states` are the construct ids in file order and `trajectory` is a list of `steps + 1` numeric vectors (row 0 = initial state), every value rounded to 6 decimals. All five knobs are echoed back, because the trajectory cannot be reproduced without them.

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)
sim$states
sim$trajectory[[1]] # the common initial state
sim$trajectory[[length(sim$trajectory)]] # after five Euler steps
```

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 (default `"building"`).
`theory_form`	Theory form (default `"network"`).

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

Lowercases, replaces every run of non-`[a-z0-9]` characters with a single space, splits, drops tokens shorter than 3 characters and the canonical stopwords, then returns the unique set.

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")
```

`tf_validate` *Validate a theory object*

Description

Built-in validation. The default (`full = FALSE`) checks required fields and enum membership. With `full = TRUE` it additionally checks referential integrity: that every id is unique within its collection and that every cross-reference (proposition endpoints, prediction derivations and diagnostics, and assumption, evidence and test-outcome targets) points to a declared id, and that every prediction `severity` is a number within [0, 1]. The `full` checks are deterministic.

Usage

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

Arguments

`theory`	A theory object (named list), e.g. from `tf_read()`.
`full`	When `TRUE`, also run the referential-integrity checks.

Value

`TRUE` (invisibly) on success; otherwise stops with a message listing every problem found.

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 and cross-references

# 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 format is chosen by the file extension (`.json` -> JSON, otherwise YAML). Files are written with LF line endings.

Usage

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

Arguments

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

Value

The `path` (invisibly).

Examples

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

Index