The leanify package

A LaTeX article checked by Lean

Marcus Rockel
https://github.com/Corrram/leanify

Version 0.9, 2026-11-23

Abstract

Leanify compiles an ordinary mathematical article, written in LaTeX, to the Lean proof assistant. Every definition and claim of the article becomes a Lean statement checked by the kernel, and the proofs are checked sentence by sentence against the results they cite. There is no second, hand-written formal version that could drift away from the paper: one source is typeset as the article you submit and compiled by Lean. This document describes the LaTeX package leanify, the typesetting side of Leanify. The Lean side, the grammar of the articles it reads and a tutorial are documented at https://corrram.github.io/leanify/.

1 What the package does

Leanify reads the article source itself: its sentences, formulas and the macros of its preamble. The package does not change what the article says, and Leanify reads no markup from it. The package

- loads the packages for the notation the Leanify grammar reads: `\coloneqq` from mathtools, `\mathbb` and `\le` from amssymb;
- checks the article while it compiles, when it can (Section 3);
- with the option `status`, marks every labelled result in the margin with its verification status (Section 5);
- records the Lean module that certifies the article (`\leanifymodule`).

Without options and without shell escape the typeset article is unchanged, so an article that loads leanify compiles anywhere, for instance on arXiv or a journal's server.

For example, Leanify compiles the sentence

```
Define $f:(0,\infty)\to\mathbb{R}$ by $f(x)\coloneqq x+\frac1x$.
```

to the Lean definition `f := fun x => x + 1 / x`, with the obligation that it is well defined, and the theorem

```
\begin{theorem}\label{thm:bound}
For every $x\in(0,\infty)$, $f(x)\ge2$.
\end{theorem}
```

to the Lean statement $\forall\,(x : \mathbb{R}),\ 0 < x \to 2 \le f\,x$, derived in Lean from exactly the sentences of its proof and the results they cite.

2 Usage

```
\usepackage{leanify}              % check as you compile
\usepackage[status]{leanify}      % and mark results in the margin
\usepackage[certify]{leanify}     % and build the certificate first
\leanifymodule{Note.Certified}    % optional: the certifying module
\leanifyreport                    % optional: a one-line summary
\leanifyreadings                  % optional: the formal readings
```

The options are:

Option	Effect
`status`	Marks every labelled result in the margin with its verification status, read from `\jobname.leanify.tex`.
`check=false`	No quick check during the compile.
`certify`	Before the marks are read, updates the pin of the article and builds its certificate with `lake build`. The first time, when the article has no Lean files yet, writes them. Implies `status`.
`vocab=FIELDS`	The fields of the article's vocabulary, for instance `vocab=unitsquare` (probability on the unit square, with $[0,1]$ as a type of its own) or `vocab={numbers,hyperbolic}`. Other fields: `arithmetic`, `combinatorics`, `polynomials`, `graphs`, `permutations`, `matrices`, `special`, `topology`, `analysis`, `vectors`, `geometry`, `congruences`, `groups`, `trigonometry`, `complex`, `torus`, `distributions`, `hilbert`, `markov`. The core vocabulary (real numbers, integers, their signs) is always there.
`lean=FILE`	The compiled Lean file of the article, if it is not `Compiled.lean` next to it or in the folder that `leanify init` made.
`exe=COMMAND`	The checker to run, instead of the project's or of `leanify` on the `PATH`: a command or a path.
`project=false`	An article that lies in no Lean project gets none. Without this option the checker makes one, once, in the folder `lean` next to the article (Section 3).
`notions=NOUNS`	The nouns that the article uses without defining them, for instance `notions={graph, finite graph, vertex (vertices)}` (Section 4). Nothing is printed for them.
`quantities=WORDS`	Words that name a real number in the names of quantities that the article does not define, for instance `quantities={degree}`.

The commands are:

`\leanifymodule{`*`module`*`}` The Lean module whose `#leanify_certify` certifies the article. By default it is `NS.Certified`, for the namespace `NS` that the compiled Lean file compiles.

`\leanifyreport` Prints a one-line summary of the last build where it appears, for instance: *12 of 12 claims verified (3 derived in the text from their citations); 85 of 90 sentences compiled; 7 definitions well defined.*

`\leanifyreadings` Prints the formal reading of every definition and of every statement of the article, with its line, the version of Leanify, the vocabulary and the conventions of the grammar: what Lean was asked to prove. In an appendix, it lets

a reader of the PDF compare each statement with its reading. It prints from `\jobname.leanify.tex`, like the margin marks.

3 Checking as you compile

When the compiler may run programs (shell escape: `pdflatex -shell-escape`, `latexmk -shell-escape`, or the setting of the editor), every compile runs one command, `leanify tex`, on the article. If the article lies in a Lean project (a `lakefile.toml` or `lakefile.lean` in its folder or up to three folders above), the checker of that project runs. Otherwise the executable `leanify` on the `PATH` runs, which needs neither Lean nor mathlib (the releases of Leanify have it for Linux, macOS and Windows). It writes its findings as records, `\jobname.leanify.out`, and they become LaTeX warnings at their lines of the article,

```
Package leanify Warning: g is not defined: sentence not checked on input line
(leanify)                70.
```

so editors that read the LaTeX log list them with the other warnings. The check reads the article with the files it inputs, and a finding in such a file names its file and line. If some proofs have sentences that Leanify does not read, one warning says how many proofs are read completely: only their results are derived from the printed proofs. With `certify`, the errors of the Lean build and the refuted claims are warnings too. The full output of Leanify is in `\jobname.leanify.log`. `lake` must be on the `PATH` of the compiler; the first check also builds the checker, which takes a few minutes once. Later compiles run the checker directly, without `lake`, and take a fraction of a second. The records of an unchanged article are the same on every run, so the later passes of `latexmk` see no change.

The Lean project of the article. An article that lies in no Lean project gets one the first time it is checked: the folder `lean` next to the article, with a `lakefile.toml` that requires the Leanify of the checker, a `lean-toolchain` with its Lean version, and a `.gitignore` for what Lake builds. Nothing is fetched or built for it, and nothing else is written to the folder of the article. The Lean files of the article go there when its certificate is first built, by the option `certify` or by `leanify certify`, from the article as it is then. The option `project=false` leaves the article without a project.

The package is never silent about an article that it did not check. Without shell escape nothing runs, and a warning says so. With shell escape on a machine where the checker does not run, a warning says what to install. If the article has changed since the status file was written and Leanify did not run, a warning says that the status is that of an earlier version. The option `check=false` is for an article that need not be checked (a submission): no warning then.

4 Notions that the article does not define

An article on graphs does not define a graph. If the vocabulary has no field for its subject, the article names the nouns that it uses without a definition:

```
\usepackage[notions={graph, finite graph, vertex (vertices)},
            quantities={degree}]{leanify}
```

A noun is then a kind of object of which nothing is known, and an entry that ends with a noun (`finite graph`) is an adjective of it. An irregular plural is given in parentheses. A sentence such as *Let $\chi(G)$ denote the chromatic number of a graph G.* introduces a symbol

for a quantity that the article does not define. What Lean proves about these notions is proved whatever they mean.

A result whose heading cites its source and that the article does not prove, as in `\begin{lemma}[\cite{diestel}]`, is *quoted from the literature*. The article assumes it: it is an axiom of the certificate, and every verified claim records the quoted results on which it rests. The summary of `\leanifyreport` names what is not defined in the article and how many statements are quoted. A kind of object is assumed to have an element, so a sentence that says only that such an object exists is not read as a claim.

5 Verification status in the margin

With `status`, every labelled result whose status is known gets one of these marks in the margin:

Mark	Meaning
✓ checked	Every sentence compiled, and the claim is verified in Lean.
✓ derived	Every sentence compiled, and the proof in the text derives the claim from its citations, each step by the standard tactic, under hypotheses that are shown to hold together.
✓ argued	As derived, with a reservation that the report names: a step of the proof needed a longer Lean argument than the standard tactic (the printed proof leaves out more than a routine step there), or the hypotheses of a claim are not shown to hold together (it may be true for an empty reason).
○ quoted	The result cites its source and has no proof in the article. It is assumed, and the certificate names it among its axioms.
○ partly checked	The result is verified in part.
○ compiled	The statement compiled, but is not verified yet.
× false	The claim is refuted in Lean.

The marks are read from `\jobname.leanify.tex`, which the Lean command `#leanify_certify` writes next to the article. If the article is newer than that file, a warning says that the marks may be out of date. Leave `status` out for the version you submit.

The summary of `\leanifyreport` also says how many steps the standard tactic closes, and of the sets of hypotheses under which claims are made, how many are shown to hold together. A verified claim that follows a sentence of its result or proof which is not read is counted apart, as *verified as read*: its reading has the hypotheses of the sentences that are read, and may lack one or keep one that the text has ended.

6 Requirements

The package needs LaTeX 2022-06-01 or newer, mathtools and amssymb. Checking and the status marks need the Lean toolchain and the Leanify Lean package in the article's Lake project; typesetting does not. Installation and a step-by-step tutorial are at https://corrram.github.io/leanify/.

7 Licence

Copyright © 2026 Marcus Rockel. The package is released under the Apache License, Version 2.0.