The proofgraph package

Pierre Senellart
pierre@senellart.com

2026/06/02 v1.0.0

Abstract

The proofgraph package automatically produces a graph of the dependencies between the results (theorems, lemmas, propositions) of a mathematical article. It infers an edge from one result to another whenever the proof of the former refers to the latter through an ordinary cross-reference, so that most dependencies need no manual annotation; commands are provided for those a visible reference does not capture. The graph is written as a Graphviz `.dot` file, which can be rendered externally or, optionally, embedded into the document automatically.

1 Introduction

When writing or reading a mathematical article, it is often useful to visualise how the results depend on one another: which lemma is used in the proof of which theorem, and so on. proofgraph builds such a dependency graph automatically. Each theorem-like environment becomes a node; each cross-reference (`\ref`, `\cref`…) appearing *inside a proof* becomes an edge from the proved result to the referred-to result. Usually no manual annotation is required, and commands are provided for the dependencies a visible reference does not capture.

The package is best explained by a small example of its use. This manual is itself a proofgraph document, so the results stated below are typeset with the package enabled, and the graph shown at the end of this section is the one they produce.

Load proofgraph in the preamble, before the `\newtheorem` commands that declare your theorem-like environments (so it sees them as they are created). Here, as in this manual, we pass three options: `autorun` renders the graph with Graphviz and embeds it automatically (it needs shell-escape); `cite` captures the `\cite`s made inside proofs as dependencies on external work; and `citelabel=tag` labels those citation nodes by their printed tag:

```
\usepackage[autorun,cite,citelabel=tag]{proofgraph}
\newtheorem{theorem}{Theorem}
\newtheorem{lemma}{Lemma}
\newtheorem{corollary}{Corollary}
```

Then state and prove, as usual, a few results that build on one another:

```
\begin{lemma}\label{lem:even}
  The sum of two even integers is even.
\end{lemma}
```

```
\begin{lemma}\label{lem:odd}
  The sum of two odd integers is even.
\end{lemma}
\begin{theorem}\label{thm:parity}
  The sum of two integers of the same parity is even.
\end{theorem}
\begin{proof}
  By Lemmas~\ref{lem:even} and~\ref{lem:odd}; see~\cite{euclid}.
\end{proof}
\begin{corollary}\label{cor:double}
  For every integer $n$, the number $2n$ is even.
\end{corollary}
\begin{proof}
  Apply Theorem~\ref{thm:parity} to $n$ and $n$.
\end{proof}
```

We obtain the following results, exactly as without the package:

Lemma 1. *The sum of two even integers is even.*

Lemma 2. *The sum of two odd integers is even.*

Theorem 1. *The sum of two integers of the same parity is even.*

Proof. By Lemmas 1 and 2; see [1]. □

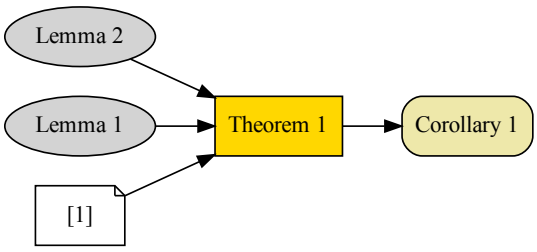
Corollary 1. *For every integer n, the number $2n$ is even.*

Proof. Apply Theorem 1 to n and n. □

In addition, proofgraph writes out the dependency graph. With the `autorun` option it is rendered with Graphviz and included wherever you call `\proofgraph`:

```
\proofgraph[width=0.7\textwidth]
```

which here produces:



Each cross-reference made inside a proof becomes an arrow *into* the result that proof establishes: the two lemmas point to Theorem 1, which in turn points to Corollary 1. The `\cite` in the proof of Theorem 1 is captured in the same way, thanks to the `cite` option: the external work appears as a distinct node (drawn note-shaped, and labelled here by its printed citation tag, "`[1]`", because of `citelabel=tag`) with an arrow into the theorem it helps prove. Moreover, since

this manual loads hyperref and TikZ, the graph's nodes are *clickable*: try clicking one to jump to the result it represents (see `\proofgraph` and the `hyperlinks` option).

A handful of commands tune such a graph: `\proofgraphstyle` sets the appearance of each kind of node (the colours here come from it); `\uses` and `\proofgraphedge` record a dependency that no visible `\ref` expresses; `\proofof` attaches a proof to its result; `\proofgraphuntrack` keeps an environment out of the graph; and `\proofgraphignore` and `\proofgraphexclude` drop an unwanted edge or node. The next section puts several of these to work on a real article; Section 3 then documents them, and the package options, in full.

2 A real-world example

The introductory example is deliberately tiny. To show what proofgraph produces on a genuine article, the graph in Figure 1 is the one obtained from *Connecting Knowledge Compilation Classes and Width Parameters* [2]. It has 62 numbered results, drawn from five environments (`result`, `theorem`, `proposition`, `corollary` and `lemma`), depends on 19 external works cited inside proofs, and contains 106 edges in all.

The graph was produced from the paper essentially as published. Beyond loading the package, the only additions were the following.

- The `cite` option was switched on, and citation nodes labelled by their printed tag, by loading the package as `\usepackage[cite=true,citelabel=tag]{proofgraph}`. The `\cite`s made inside proofs then appear as the note-shaped nodes ("`[17]`", "`Theorem 4.10, [46]`"…) along the left of the graph; a work cited for two different results, or for two of its own theorems, gives one node per `\cite` note, so the 19 works appear as 32 nodes.

- One environment was kept out of the graph with a single `\proofgraphuntrack{observation}` (a lone preliminary observation that nothing else uses).

- The five result kinds were styled, with decreasing prominence, by five `\proofgraphstyle` declarations:

```
\proofgraphstyle{result}{shape=box,style="filled,bold",%
                         fillcolor=orange,peripheries=2}
\proofgraphstyle{theorem}{shape=box,style=filled,%
                          fillcolor=gold,penwidth=2}
\proofgraphstyle{corollary}{shape=box,style="rounded,filled",%
                            fillcolor=palegoldenrod}
\proofgraphstyle{proposition}{shape=box,style="rounded,filled",%
                              fillcolor=lightblue}
\proofgraphstyle{lemma}{shape=ellipse,style=filled,%
                        fillcolor=lightgray}
```

- Eleven `\proofgraphedge` commands, recording 27 edges in total, were added for the dependencies that no `\ref` inside a `proof` expresses: eight "obvious" corollaries stated without a `proof` environment

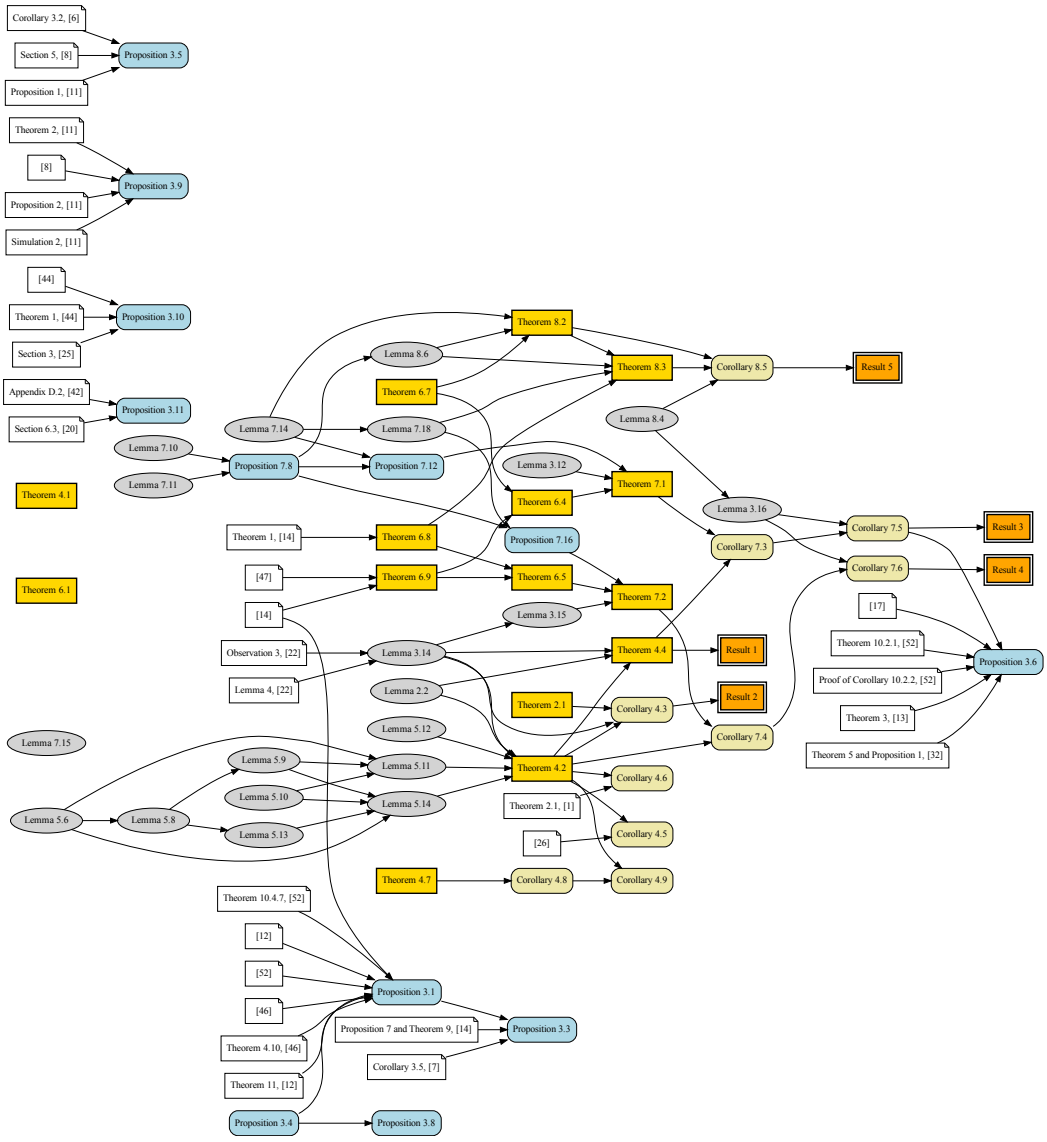


Figure 1: The graph proofgraph produces for [2]. The colours and shapes make the structure legible, with the orange double-bordered introduction `results` on the right, fed by the gold theorems, the blue and pale-gold propositions and corollaries, and the grey lemmas.

(`\proofgraphedge{cor:upper_bound}{thm:upper_bound,...}`), and three theorems whose proof is developed across a whole section rather than inside a single `proof`.

Nothing else was annotated: every other edge, and the numbering of all 62 nodes, was inferred automatically from the ordinary cross-references the proofs already contained. In particular, no `\uses`, `\proofof`, `\proofgraphignore` or `\proofgraphexclude` was needed. The five `result` environments that restate the headline results in the introduction (`\begin{result}[(Corollary~\ref{cor:obdd_omega})]`...) are linked to the body results they preview through the `\ref` in their optional title, captured automatically (see the note on reference-carrying titles in Section 3). Two compilation runs (plus BibTeX, needed once for the `tag` labels) suffice; the `.dot` file is then rendered with Graphviz, `dot -Tpdf main.dot -o main.pdf`, as with any other document.

3 Usage

3.1 Default behaviour

Load proofgraph in the preamble. It only needs to come *after* whatever provides your theorem environments, that is, after `\usepackage{amsthm}` if you load it yourself (some classes, such as lipics or acmart, provide theorems for you, in which case there is nothing to load first), and *before* the `\newtheorem` commands that declare your theorem-like environments, so that proofgraph sees them as they are created:

```
\usepackage{amsthm}      % or nothing, if your class provides theorems
\usepackage{proofgraph}
\newtheorem{theorem}{Theorem}
\newtheorem{lemma}{Lemma}
```

Then write your document as usual. After two compilation runs, a file `\jobname.dot` is produced. Render it with Graphviz:

```
dot -Tpdf myfile.dot -o mygraph.pdf
```

Every *numbered* result of a theorem-like environment you declare becomes a node, and every cross-reference (`\ref`, `\cref`...) inside a proof becomes an edge into the result that proof establishes, with no annotation. Nodes are labelled by the result's formatted name and number ("Theorem 1"), not the environment name. Unnumbered (starred) results are skipped, as they are usually expository or repeated statements; `\proofgraphtrack` draws a chosen one anyway. A cross-reference in the *optional title* of a result is captured the same way: `\begin{theorem}[(generalises~\ref{thm:x})]` in a result labelled ⟨*A*⟩ adds an edge from ⟨*A*⟩ to `thm:x`, handy for restatements (e.g., an introduction that previews results with `\begin{result}[(Corollary~\ref{cor:y})]`).

3.2 Recording dependencies by hand

`\uses{`⟨***label***⟩`}` Inside a proof, records a dependency on the result labelled ⟨*label*⟩ even when no visible reference is made. A comma-separated list is accepted, as in `\uses{lem:a,lem:b}`.

`\proofgraphedge{`⟨***A***⟩`}{`⟨***labels***⟩`}` Adds edges recording that result ⟨*A*⟩ depends on the result(s) ⟨*labels*⟩ (a comma-separated list), like a `\uses` but stated outside any proof. Handy for an "obvious" corollary that relies on earlier results without a `proof` environment, as in `\proofgraphedge{cor:x}{thm:y,lem:z}`. It may appear anywhere; all labels are resolved when the graph is written.

`\proofof{`⟨***label***⟩`}` By default a proof is attached to the most recently started result. A detached proof whose optional title names its result is attached automatically: `\begin{proof}` with `[of Theorem~\ref{thm:x}]` attaches the proof to `thm:x` (for the `amsthm` `proof` environment, for `proof` defined as a theorem in the Springer classes, and for deferred proofs as with apxproof). When none of this applies, `\proofof` inside the proof pins it to the result labelled ⟨*label*⟩.

3.3 Choosing what appears in the graph

Result environments are detected automatically; these commands adjust which results, and which edges, are kept.

`\proofgraphtrack{`⟨***names***⟩`}` Forces the listed environment types to be tracked, overriding the default exclusions, which are `definition`, `example`, `remark` and `note`. Use it in the preamble.

`\proofgraphuntrack{`⟨***names***⟩`}` Excludes further environment types, as in `\proofgraphuntrack{observation}`. Use it in the preamble.

`\proofgraphignore{`⟨***from***⟩`}{`⟨***to***⟩`}` Suppresses the edge from ⟨*from*⟩ to ⟨*to*⟩ (e.g., an unwanted forward citation).

`\proofgraphexclude{`⟨***label***⟩`}` Removes the result labelled ⟨*label*⟩, together with all its incident edges, from the graph.

3.4 Styling the graph

The attributes are passed verbatim to Graphviz; see its attribute reference, https://graphviz.org/doc/info/attrs.html, for the node attributes available (and https://graphviz.org/doc/info/shapes.html for the shapes).

`\proofgraphstyle{`⟨***env***⟩`}{`⟨***attrs***⟩`}` Sets Graphviz node attributes for all results of the environment ⟨*env*⟩, e.g., `shape=ellipse` or `style=filled,fillcolor=gold`.

`\proofgraphstylecite{`⟨***attrs***⟩`}` Sets the Graphviz attributes of the external citation nodes added by the `cite` option (default `shape=note`), as in `\proofgraphstylecite{shape=box,style=dashed,color=gray}`.

3.5 Embedding the rendered graph

`\proofgraph[`⟨***options***⟩`]` Includes the rendered graph image at this point, passing the optional ⟨*options*⟩ to the underlying `\includegraphics` (e.g., `\proofgraph[width=\linewidth]`). This is the only feature that needs

graphicx, which the package therefore does not load on its own: a document that uses `\proofgraph` must load graphicx itself (loading tikz for the `hyperlinks` overlay also suffices). With the `autorun` option, Graphviz is run at the end of every compilation (shell-escape required) to produce that image, so `\proofgraph` embeds the graph from the previous run and is up to date after two runs. Without `autorun`, no Graphviz is run: `\proofgraph` embeds a pre-existing `\jobname-proofgraph.`⟨*format*⟩ file if you have rendered one under that name, and otherwise warns. When hyperref and TikZ are both loaded (with `hyperlinks` on and the `-Tplain` file from `autorun` present), every node becomes an internal hyperlink to the result it represents: Graphviz writes the node positions to that `-Tplain` file and `\proofgraph` overlays a transparent `\hyperref` area on each node (needed because the link annotations Graphviz puts in the `.pdf` are discarded by `\includegraphics`).

3.6 Options

`selfloops` Either `remove` (default), dropping edges from a result to itself, or `keep`, retaining them.

`autorun` Boolean (default false). Runs Graphviz automatically at the end of the run; requires compilation with shell-escape enabled.

`engine` Graphviz layout engine (default `dot`); any Graphviz engine works, such as `neato`, `fdp`, `sfdp`, `circo` or `twopi` (see https://graphviz.org/docs/layouts/).

`format` Output format for `autorun` and `\proofgraph` (default `pdf`); any Graphviz output format works, such as `png`, `svg` or `ps` (see https://graphviz.org/docs/outputs/).

`hyperlinks` Boolean (default true). Makes the nodes of an embedded graph clickable when hyperref and TikZ are available (see `\proofgraph`); set it false to embed the graph as a plain image.

`direction` Either `usedby` (default), orienting an edge from the result that is used to the result that uses it (so an arrow `a->b` reads "`a` is used by `b`"), or `uses`, the reverse.

`rankdir` Graphviz graph direction (default `LR`).

`cite` Boolean (default false). Also captures `\cite` inside proofs as dependencies on external work, drawn as distinct nodes (Section 2). A `\cite` note is kept: `\cite[Thm~3]{key}` labels the node "Thm 3, . . . ".

`citelabel` Either `key` (default), labelling citation nodes by their BibTeX key, or `tag`, using the printed citation tag instead ("[1]", "[Knu74]", "[Abiteboul et al. 1995]"...), recovered from the `.aux` (so it needs two runs). It supports the standard, hyperref-wrapped and natbib (numeric and author–year) bibliographies; if no clean tag can be recovered it keeps the key.

`file` Base name of the output file (default `\jobname`).

3.7 Use with apxproof

apxproof moves proofs to the appendix and typesets them there, so the references a proof makes are only executed at that later point. To attribute them to the right result, proofgraph relies on the hook `\apxproofhook`, which apxproof fires inside each deferred proof. This hook is available in apxproof **v1.4.1** and later; load the two packages in either order and no further setup is needed.

With an *earlier* apxproof the document still compiles cleanly and every result still becomes a node, but *no proof dependencies are captured for the proofs deferred to the appendix*, because proofgraph cannot tell which appendix proof belongs to which result (dependencies from any proofs left in the main text are still captured as usual). In that case, add the deferred ones manually with `\uses` (and `\proofof` to pin a proof to its result), or upgrade apxproof.

4 Limitations

- Environments declared through `\newtheorem` (or `\spnewtheorem`) *after* the package is loaded are tracked automatically. Environments predefined by the document class *before* that (as in lipics, acmart or the Springer classes) are also detected, provided their name is one of the common result names proofgraph probes at `\begin{document}` (`theorem`, `lemma`, `proposition`, `corollary`…); an environment with an unusual name predefined before the package is loaded must be added with `\proofgraphtrack`.
- Results must be labelled for references to them to be resolved.
- Two compilation runs are required, as for any cross-reference.

References

[1] Euclid. *Elements*, Book IX. c. 300 BCE.

[2] A. Amarilli, F. Capelli, M. Monet, and P. Senellart. Connecting knowledge compilation classes and width parameters. *Theory of Computing Systems*, **64**(5):861–914, 2020. doi:10.1007/s00224-019-09930-2.

5 Implementation

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}[2005/12/01]
\ProvidesPackage{proofgraph}
  [2026/06/02 v1.0.0 Dependency graph of results]
```

5.1 Dependencies

amsthm (loaded for its heading hooks) defines the `proof` environment with `\newenvironment`, which aborts with "Command `\proof` already defined" under a class that already provides one (e.g. the Springer `svjour3` `\proof`). When we are about to load amsthm ourselves (it is not already loaded), we save the class's `proof`/`\endproof` and free them so amsthm loads cleanly, then restore the class's versions afterwards, so its proof styling is kept (and proofgraph still hooks that `proof` environment). If amsthm is already loaded, `\RequirePackage` is a no-op and `\proof` is left untouched.

```
\@ifpackageloaded{amsthm}{}{%
  \@ifundefined{proof}{}{%
    \let\pgph@class@proof\proof \let\pgph@class@endproof\endproof}%
  \let\proof\relax \let\endproof\relax}
\RequirePackage{amsthm}
\@ifundefined{pgph@class@proof}{}{%
  \let\proof\pgph@class@proof \let\endproof\pgph@class@endproof}
\RequirePackage{etoolbox}
\RequirePackage{xstring}
\RequirePackage{kvoptions}
```

graphicx is needed only to embed a rendered graph with `\proofgraph`; the core `.dot`-writing machinery does not use it. Rather than burden every document with it, we leave it unloaded and check for `\includegraphics` at the point of use (see `\proofgraph`). When the `hyperlinks` overlay is active tikz is loaded, which pulls graphicx in anyway.

`\pgph@trimsp` An expandable space-trimmer (labels inside a `\cite`/`\cref`/`\uses` list arrive with the space after each comma), built on the `expl3` primitive so it is robust to leading spaces and braces.

```
\ExplSyntaxOn
\cs_new:Npn \pgph@trimsp #1 { \tl_trim_spaces:n {#1} }
\ExplSyntaxOff
```

5.2 Options

```
\SetupKeyvalOptions{family=pgph,prefix=pgph@}
\DeclareStringOption[remove]{selfloops}
\DeclareBoolOption{autorun}
\DeclareStringOption[dot]{engine}
\DeclareStringOption[pdf]{format}
\DeclareStringOption[LR]{rankdir}
\DeclareStringOption[usedby]{direction}
\DeclareBoolOption{cite}
\DeclareStringOption[key]{citelabel}
\DeclareBoolOption[true]{hyperlinks}
\DeclareStringOption{file}
```

```
\ProcessKeyvalOptions*
\ifx\pgph@file\@empty\def\pgph@file{\jobname}\fi
```

5.3 Literal braces

We need catcode-12 braces to write the Graphviz `digraph { }` block.

```
\begingroup
  \catcode`\[=1 \catcode`\]=2
  \catcode`\{=12 \catcode`\}=12
  \gdef\pgph@ob[{]%
  \gdef\pgph@cb[}]%
\endgroup
```

5.4 Data structures

A global counter gives each result instance a unique numeric id. Nodes and edges are accumulated in list macros and emitted at end of document, so that editing commands (ignore, exclude, self-loops) can be applied to the complete graph regardless of where they appear.

```
\newcount\pgph@idcnt
\newcount\pgph@citeslotcnt
\def\pgph@nodes{}
\def\pgph@edges{}
\def\pgph@ignores{}
\def\pgph@excludes{}
\let\pgph@curresult\@empty
\let\pgph@curproof\@empty
```

`\pgph@auxmap` The label-to-id map. `\label` inside a result records, both in memory and in the `.aux` file, that the user label maps to the current node id, so references resolve independently of hyperref.

```
\newcommand\pgph@auxmap[2]{\global\@namedef{pgph@map@#1}{#2}%
  \ifcsname pgph@rmap@#2\endcsname\else
    \global\@namedef{pgph@rmap@#2}{#1}\fi}
\newcommand\pgph@setmap[2]{%
  \edef\pgph@tmpid{#2}%
  \global\expandafter\edef\csname pgph@map@#1\endcsname{\pgph@tmpid}%
  \ifcsname pgph@rmap@\pgph@tmpid\endcsname\else
    \global\expandafter\def\csname pgph@rmap@\pgph@tmpid\endcsname{#1}%
  \fi
  \protected@write\@auxout{}{\string\pgph@auxmap{#1}{\pgph@tmpid}}%
}
```

5.5 Registering result environments

`\pgph@register` A result environment is tracked by installing a begin-hook that records a node. In every declaration command (`\newtheorem`, `\spnewtheorem`) the environment *name* is the first mandatory argument, so registration does not need to parse the remaining (varied) arguments. Each name met is queued as a *candidate* (deduplicated); the begin-hooks themselves are installed only at `\begin{document}`, once the inclusion and exclusion lists are settled, so the customisation commands below may appear anywhere in the preamble.

```
\let\pgph@candidates\@empty
\newcommand\pgph@register[1]{%
  \edef\pgph@thisname{#1}%
  \IfBeginWith\pgph@thisname{axp@}{}{%
    \ifcsname pgph@cand@\pgph@thisname\endcsname\else
      \global\@namedef{pgph@cand@\pgph@thisname}{}%
      \ifx\pgph@candidates\@empty
        \global\let\pgph@candidates\pgph@thisname
      \else
        \xdef\pgph@candidates{\pgph@candidates,\pgph@thisname}%
      \fi
    \fi
  }%
}
```

We wrap `\newtheorem` (and, for Springer classes, `\spnewtheorem`) to register each newly declared environment, replaying the original command so its own argument parsing is untouched. The starred (unnumbered) forms are *not* auto-registered: a results graph wants numbered statements, and these forms are commonly used for unnumbered repeats of a result (e.g., the appendix copies produced by apxproof and myproofs), which would otherwise appear as spurious unnumbered duplicate nodes. Use `\proofgraphtrack` to track an unnumbered environment deliberately.

```
\let\pgph@orig@newtheorem\newtheorem
\renewcommand\newtheorem{\@ifstar{\pgph@nt@star}{\pgph@nt@plain}}
\newcommand\pgph@nt@plain[1]{%
  \pgph@register{#1}\pgph@orig@newtheorem{#1}}
\newcommand\pgph@nt@star[1]{\pgph@orig@newtheorem*{#1}}
\ifdefined\spnewtheorem
  \let\pgph@orig@spnewtheorem\spnewtheorem
  \renewcommand\spnewtheorem{%
    \@ifstar{\pgph@spnt@star}{\pgph@spnt@plain}}
  \newcommand\pgph@spnt@plain[1]{%
    \pgph@register{#1}\pgph@orig@spnewtheorem{#1}}
  \newcommand\pgph@spnt@star[1]{\pgph@orig@spnewtheorem*{#1}}
\fi
```

`\proofgraphtrack`
`\proofgraphuntrack`

Result environments are detected automatically, but the analysis can be tuned. `\proofgraphtrack{`⟨*names*⟩`}` forces the listed environments to be tracked, overriding any exclusion; `\proofgraphuntrack{`⟨*names*⟩`}` excludes the listed environment *types*. Expository environments are excluded by default: `definition`, `example`, `remark`, `note`. Both commands belong in the preamble. `\pgph@defaultenvs` is the list of class-predefined names probed at `\begin{document}`; `\pgph@defaultexclude` seeds the exclusion set.

```
\newcommand\pgph@defaultenvs{theorem,lemma,proposition,corollary,%
  definition,conjecture,claim,fact,remark,example,observation,%
  notation,problem,question,property,assumption,hypothesis,principle}
\newcommand\pgph@defaultexclude{definition,example,remark,note}
\newcommand\proofgraphtrack[1]{%
  \@for\pgph@one:=#1\do{%
    \csundef{pgph@xtype@\pgph@one}%
    \global\@namedef{pgph@force@\pgph@one}{}%
```

```
    \expandafter\pgph@register\expandafter{\pgph@one}%
  }%
}
\newcommand\proofgraphuntrack[1]{%
  \@for\pgph@one:=#1\do{%
    \csundef{pgph@force@\pgph@one}%
    \global\@namedef{pgph@xtype@\pgph@one}{}%
  }%
}
\expandafter\proofgraphuntrack\expandafter{\pgph@defaultexclude}
```

Environments predefined by the document class (acmart, lipics, Springer classes) are declared before the package is loaded, so the wrappers above never see them. At `\begin{document}` we add every common result name that exists to the candidate list, then install a begin-hook for each candidate that is forced or not excluded.

```
\newcommand\pgph@dohook[1]{%
  \ifcsname pgph@hooked@#1\endcsname\else
    \global\@namedef{pgph@hooked@#1}{}%
    \AtBeginEnvironment{#1}{\pgph@beginresult{#1}}%
  \fi
}
\newcommand\pgph@maybehook[1]{%
  \ifcsname pgph@force@#1\endcsname
    \pgph@dohook{#1}%
  \else
    \ifcsname pgph@xtype@#1\endcsname\else\pgph@dohook{#1}\fi
  \fi
}
\newcommand\pgph@resolveregistration[1]{%
  \@for\pgph@one:=#1\do{%
    \ifcsname\pgph@one\endcsname%
      \expandafter\pgph@register\expandafter{\pgph@one}\fi}%
  \ifx\pgph@candidates\@empty\else
    \@for\pgph@one:=\pgph@candidates\do{%
      \expandafter\pgph@maybehook\expandafter{\pgph@one}}%
  \fi
}
\AtBeginDocument{%
  \expandafter\pgph@resolveregistration\expandafter{\pgph@defaultenvs}}
```

`\pgph@beginresult` `\pgph@hooknames` `\pgph@recordresult`

Start a result: allocate an id and record the node. The node is keyed and styled by the environment name (third argument, also a fallback label), but displayed by the result's *formatted* name ("Theorem", "Lemma"), captured from the heading. We get that name by locally wrapping the heading macros, whose first two arguments are the formatted name and the number. Which macro carries the *optional note* (where a restatement's cross-reference lives) depends on the setup: amsthm, which proofgraph always loads, routes every heading through `\@begintheorem`, defined as `#1#2[#3]` with the note as the bracketed `#3` (and leaves `\@opargbegintheorem \relax`); the LaTeX kernel instead uses a two-argument `\@begintheorem` with a separate three-argument `\@opargbegintheorem` for the note; the Springer classes use `\@spbegintheorem`/`\@spopargbegintheorem`. We detect the amsthm form (`\@opargbegintheorem` is `\relax`) and read the bracketed note there, otherwise

we hook `\@opargbegintheorem`; either way the note goes to `\pgph@scanrefs`. These redefinitions are confined to the current environment group, so they revert at `\end`. We also locally redefine `\label` to populate the map.

Nodes are accumulated in *reverse* order of appearance (`\pgph@prependnode`). With `rankdir=LR`, Graphviz stacks disconnected components vertically in the order they are read, from the bottom up; emitting the results last-to-first therefore lays them out top-to-bottom in document order (first result on top), while connected results stay grouped. This is a layout tendency, not a guarantee, but needs no `pack` trickery.

```
\newcommand\pgph@prependnode[1]{%
  \xdef\pgph@nodes{\unexpanded{#1}\unexpanded\expandafter{\pgph@nodes}}%
}
\newcommand\pgph@beginresult[1]{%
  \global\advance\pgph@idcnt\@ne
  \xdef\pgph@curresult{\the\pgph@idcnt}%
  \protected@edef\pgph@tmp{%
    \noexpand\pgph@node{\pgph@curresult}{#1}{#1}}%
  \expandafter\pgph@prependnode\expandafter{\pgph@tmp}%
  \pgph@hooknames
  \ifx\label\pgph@maplabel\else\let\pgph@savedlabel\label\fi
  \let\label\pgph@maplabel
}
\newcommand\pgph@hooknames{%
  \ifdefined\@begintheorem
    \let\pgph@orig@bt\@begintheorem
    \ifx\@opargbegintheorem\relax
      \def\@begintheorem##1##2[##3]{%
        \pgph@recordresult{##1}{##2}\pgph@scanrefs{##3}%
        \pgph@orig@bt{##1}{##2}[##3]}%
    \else
      \def\@begintheorem##1##2{%
        \pgph@recordresult{##1}{##2}\pgph@orig@bt{##1}{##2}}%
    \fi
  \fi
  \ifdefined\@opargbegintheorem\ifx\@opargbegintheorem\relax\else
    \let\pgph@orig@obt\@opargbegintheorem
    \def\@opargbegintheorem##1##2##3{%
      \pgph@recordresult{##1}{##2}\pgph@scanrefs{##3}%
      \pgph@orig@obt{##1}{##2}{##3}}%
  \fi\fi
  \ifdefined\@spbegintheorem
    \let\pgph@orig@spbt\@spbegintheorem
    \def\@spbegintheorem##1##2##3##4{%
      \pgph@recordresult{##1}{##2}\pgph@orig@spbt{##1}{##2}{##3}{##4}}%
  \fi
  \ifdefined\@spopargbegintheorem
    \let\pgph@orig@spobt\@spopargbegintheorem
    \def\@spopargbegintheorem##1##2##3##4##5{%
      \pgph@recordresult{##1}{##2}\pgph@scanrefs{##3}%
      \pgph@orig@spobt{##1}{##2}{##3}{##4}{##5}}%
  \fi
}
```

`\pgph@scanrefs` A result whose optional title carries a cross-reference, e.g., a `result` whose title is `[(Corollary~\ref{cor:x})]`, is understood to restate or build on that result: we record an edge from the current result to each label referenced in the title. We capture these by typesetting the title once into a discarded box with the reference-capturing commands installed and `\pgph@curproof` pointing at the current result; `\pgph@curproof` is saved and restored, so this affects nothing outside the title.

```
\newcommand\pgph@scanrefs[1]{%
  \ifx\pgph@curresult\@empty\else
    \begingroup
      \global\let\pgph@savedcurproof\pgph@curproof
      \global\let\pgph@curproof\pgph@curresult
      \pgph@install{ref}\pgph@install{cref}\pgph@install{Cref}%
      \pgph@install{autoref}\pgph@install{eqref}\pgph@install{vref}%
      \setbox\z@\hbox{#1}%
      \global\let\pgph@curproof\pgph@savedcurproof
    \endgroup
  \fi
}
```

The formatted name is the first argument of every heading macro (`\@begintheorem`/`\@opargbegintheorem` for `amsthm`/`\newtheorem`, `\@spbegintheorem`/`\@spopargbegintheorem` for Springer's `\spnewtheorem`); the second argument is the number. We use that second argument only to tell *whether* the result is numbered (it is empty for the starred, unnumbered forms): an unnumbered result records no number, so it is dropped at emission unless tracked explicitly. The number *value*, when present, is read from `\@currentlabel` rather than from that argument, which some classes pollute (e.g., lipics injects `\advance\par@deathcycles\@ne`). `\@currentlabel`, set by the result's `\refstepcounter` just before the heading, is the clean number and needs no `\label`. Values are reduced to plain strings now, with fragile wrappers neutralized (notably apxproof's forward-link). The first heading wins (guarded on the name), ignoring nested headings. The emptiness of the number argument is tested without expanding it, so a polluted (but non-empty) number does not trip the test.

```
\newcommand\pgph@recordresult[2]{%
  \ifcsname pgph@name@\pgph@curresult\endcsname\else
    \begingroup
      \let\protect\@empty
      \def\axp@forward@link##1##2{##2}%
      \edef\pgph@tmpname{#1}%
      \global\expandafter\let\csname pgph@name@\pgph@curresult\endcsname
                                \pgph@tmpname
      \global\@namedef{pgph@seen@\pgph@curresult}{}%
      \def\pgph@tmphn{#2}%
      \ifx\pgph@tmphn\@empty\else
        \edef\pgph@tmpnum{\@currentlabel}%
      \global\expandafter\let\csname pgph@num@\pgph@curresult%
        \endcsname\pgph@tmpnum
      \fi
    \endgroup
  \fi
}
```

`\pgph@maplabel` records the label-to-node mapping. As a fallback for classes whose headings pass through none of the hooked heading macros (so no heading was `seen`), it also captures the number from `\@currentlabel`. We do *not* do this once a heading was seen: there, an absent number means the result is genuinely unnumbered (a starred form), and `\@currentlabel` could be a stale value from an earlier result. The value is expanded to a plain string *now*, inside a group where fragile wrappers are neutralized, so that nothing dangerous survives to the emission pass: in particular, under apxproof forward-linking, `\@currentlabel` holds `\axp@forward@link{`⟨*target*⟩`}{`⟨*number*⟩`}`, whose hyperlink machinery must never be expanded in a non-typesetting context (it would run away). Reducing it to its number argument keeps it harmless.

`\pgph@maplabel`

```
\newcommand\pgph@maplabel[1]{%
  \pgph@setmap{#1}{\pgph@curresult}%
  \ifcsname pgph@num@\pgph@curresult\endcsname\else
   \ifcsname pgph@seen@\pgph@curresult\endcsname\else
    \begingroup
      \let\protect\@empty
      \def\axp@forward@link##1##2{##2}%
      \edef\pgph@tmpnum{\@currentlabel}%
      \global\expandafter\let\csname pgph@num@\pgph@curresult%
        \endcsname\pgph@tmpnum
    \endgroup
   \fi
  \fi
  \pgph@savedlabel{#1}%
}
```

5.6 Hooking proofs

Inside a proof we redefine the reference commands to capture edges from the proved result.

```
\newcommand\pgph@addedge[2]{%
  \protected@edef\pgph@tmpedge{\noexpand\pgph@edge{#1}{#2}}%
  \expandafter\g@addto@macro\expandafter\pgph@edges%
    \expandafter{\pgph@tmpedge}%
}
\newcommand\pgph@recordref[1]{%
  \ifx\pgph@curproof\@empty\else
    \@for\pgph@one:=#1\do{%
      \edef\pgph@k{\expandafter\pgph@trimsp\expandafter{\pgph@one}}%
      \pgph@addedge{\pgph@curproof}{\pgph@k}}%
  \fi
}
```

`\pgph@beginproof` Attach the proof to the current result and install the capturing versions of the reference commands (only those that exist).

The capturing commands are `\protected` so that, when a reference appears in a moving argument inside a proof (e.g., a `\caption`), they do not expand into the `.aux`/`.lof` and leak internal tokens.

```
\protected\def\pgph@cap@ref#1{\pgph@recordref{#1}\pgph@saved@ref{#1}}
```

```
\protected\def\pgph@cap@cref#1{\pgph@recordref{#1}\pgph@saved@cref{#1}}
\protected\def\pgph@cap@Cref#1{\pgph@recordref{#1}\pgph@saved@Cref{#1}}
\protected\def\pgph@cap@autoref#1{%
  \pgph@recordref{#1}\pgph@saved@autoref{#1}}
\protected\def\pgph@cap@eqref#1{%
  \pgph@recordref{#1}\pgph@saved@eqref{#1}}
\protected\def\pgph@cap@vref#1{\pgph@recordref{#1}\pgph@saved@vref{#1}}
\newcommand\pgph@install[1]{%
  \ifcsname #1\endcsname
    \expandafter\ifx\csname #1%
      \expandafter\endcsname\csname pgph@cap@#1\endcsname
    \else
      \expandafter\let\csname pgph@saved@#1%
        \expandafter\endcsname\csname #1\endcsname
      \expandafter\let\csname #1%
        \expandafter\endcsname\csname pgph@cap@#1\endcsname
    \fi
  \fi
}
\newcommand\pgph@beginproof{%
  \let\pgph@curproof\pgph@curresult
  \pgph@install{ref}\pgph@install{cref}\pgph@install{Cref}%
  \pgph@install{autoref}\pgph@install{eqref}\pgph@install{vref}%
  \ifpgph@cite\pgph@installcite\fi
  \pgph@hookproofhead
}
```

When the `proof` environment is itself a theorem-like environment (as with Springer's `\spnewtheorem*{proof}`), its optional title is set by `\@opargbegintheorem`/`\@spopargbegintheorem`. We wrap these for the duration of the proof so that a title such as `[of Theorem~\ref{thm:x}]` sets `\proofof{thm:x}`, attributing the proof to its result.

```
\newcommand\pgph@hookproofhead{%
  \ifdefined\@opargbegintheorem
    \let\pgph@orig@obtp\@opargbegintheorem
    \def\@opargbegintheorem##1##2##3{%
      \pgph@scanproofarg{##3}\pgph@orig@obtp{##1}{##2}{##3}}%
  \fi
  \ifdefined\@spopargbegintheorem
    \let\pgph@orig@spobtp\@spopargbegintheorem
    \def\@spopargbegintheorem##1##2##3##4##5{%
      \pgph@scanproofarg{##3}%
      \pgph@orig@spobtp{##1}{##2}{##3}{##4}{##5}}%
  \fi
  \ifdefined\@Opargbegintheorem
    \let\pgph@orig@Obtp\@Opargbegintheorem
    \def\@Opargbegintheorem##1##2##3##4{%
      \pgph@scanproofarg{##2}\pgph@orig@Obtp{##1}{##2}{##3}{##4}}%
  \fi
}
```

`\pgph@scanproofarg` `\pgph@wrapproof`

A proof whose optional title names its result (a detached proof) has its result given by that title, e.g., `thm:x` for an optional argument `[of Theorem~\ref{thm:x}]`, rather than by the most recently opened result. We capture this by wrapping the

`proof` environment so that, when an optional title is given, the cross-reference it contains is turned into a `\proofof` (the title is typeset once into a discarded box with `\ref` and friends redefined). Without an optional title the original environment is used unchanged.

```
\newcommand\pgph@scanproofarg[1]{%
  \begingroup
    \let\protect\@empty
    \def\ref##1{\proofof{##1}}\def\cref##1{\proofof{##1}}%
    \def\Cref##1{\proofof{##1}}\def\autoref##1{\proofof{##1}}%
    \setbox\z@\hbox{#1}%
  \endgroup
}
```

To capture the optional title of a detached `amsthm` proof (an optional `[of Theorem~\ref{thm:x}]`) we redefine the outer `\proof` to read the optional argument and delegate to the original inner macro (saved as `\pgph@proofinner`), so no recursion occurs. We must do this *only* for a genuine optional-argument environment: a package that captures the proof body verbatim (apxproof, myproofs) redefines `proof` with no optional argument, and wrapping it would break that capture. We recognise the optional-argument form by the `\@protected@testopt` in its definition. When `proof` is instead a theorem-like environment (Springer `\spnewtheorem*{proof}`) its title is already handled by `\pgph@hookproofhead`, so here we just install the begin-hook.

```
\newcommand\pgph@wrapproof{%
  \ifdefined\proof
    \edef\pgph@proofmeaning{\meaning\proof}%
    \edef\pgph@testoptstr{\detokenize{\@protected@testopt}}%
    \IfSubStr\pgph@proofmeaning\pgph@testoptstr
      {\pgph@redefproof}%
      {\AtBeginEnvironment{proof}{\pgph@beginproof}}%
  \fi
}
\newcommand\pgph@redefproof{%
  \expandafter\let\expandafter\pgph@proofinner%
    \csname\string\proof\endcsname
  \def\proof{\pgph@beginproof
    \@ifnextchar[{\pgph@proofopt}{\pgph@proofinner[\proofname]}}%
  \def\pgph@proofopt[##1]{\pgph@scanproofarg{##1}\pgph@proofinner[##1]}%
}
```

Packages such as apxproof capture the body of a `proof` verbatim (to defer it to the appendix) and replay it there. Patching the `proof` environment then corrupts that capture, and the references are only executed at replay time anyway. So we wrap `proof` only when no such package is active; otherwise the deferred-proof capture (below) takes over. This happens at `\begin{document}`, once we know what is loaded.

```
\newif\ifpgph@apxproof
\AtBeginDocument{%
  \@ifpackageloaded{apxproof}%
    {\pgph@apxprooftrue\pgph@apxinit}%
    {\pgph@wrapproof}%
}
```

\pgph@apxinit
\pgph@apxproofcapture

Under apxproof, deferred proofs are typeset in the appendix, where their references finally execute. apxproof fires \apxproofhook inside each such proof, passing the axp@r⟨*n*⟩ label of the proved theorem, which (because apxproof also attaches that label to the theorem in the main text) is already in our map. We resolve it to the source node and install the reference-capturing commands for the duration of the proof.

```
\newcommand\pgph@apxinit{%
  \def\apxproofhook##1{\pgph@apxproofcapture{##1}}}
\newcommand\pgph@apxproofcapture[1]{%
  \edef\pgph@curproof{\pgph@resolve{#1}}%
  \ifx\pgph@curproof\@empty\else
    \pgph@install{ref}\pgph@install{cref}\pgph@install{Cref}%
    \pgph@install{autoref}\pgph@install{eqref}\pgph@install{vref}%
    \ifpgph@cite\pgph@installcite\fi
  \fi
}
```

\pgph@hookcite

Optional \cite capture: external references become cite:-prefixed target labels, drawn later as distinct nodes. The node identity is the pair (key, note): a \cite[⟨*note*⟩]{⟨*key*⟩} inside a proof is distinguished from a \cite of the same key with a *different* note, so that citing two different results of the same work (say two theorems of one book) yields two nodes rather than one mislabelled with whichever note was seen first. Each pair is assigned a numeric *slot* (\pgph@citeslot); the edge target is cite:⟨*slot*⟩, and the slot remembers the key (for the tag and the bibliography hyperlink) and the note (for the label). The note is reduced to a plain string at capture time (fragile wrappers neutralized, ~ normalised to a space), which also serves as the signature so [Thm 1] and [Thm~1] coincide. We peek for the optional argument with \@ifnextchar rather than declaring one, so that a note-less \cite{⟨*key*⟩} is passed on *as written*: re-emitting it as \cite[]{⟨*key*⟩} would make the typeset citation grow a spurious empty note ("[1,]").

```
\newcommand\pgph@gxdefcs[2]{%
  \global\expandafter\edef\csname#1\endcsname{#2}}
\newcommand\pgph@cap@cite{%
  \@ifnextchar[\pgph@cap@cite@opt\pgph@cap@cite@noopt}
\def\pgph@cap@cite@opt[#1]#2{%
  \pgph@cap@citerec{#1}{#2}\pgph@saved@cite[#1]{#2}}
\def\pgph@cap@cite@noopt#1{\pgph@cap@citerec{}{#1}\pgph@saved@cite{#1}}
\newcommand\pgph@cap@citerec[2]{%
  \ifx\pgph@curproof\@empty\else
    \@for\pgph@one:=#2\do{%
      \edef\pgph@k{\expandafter\pgph@trimsp\expandafter{\pgph@one}}%
      \pgph@citeslot{\pgph@k}{#1}%
      \pgph@addedge{\pgph@curproof}{cite:\pgph@curslot}}%
  \fi
}
\newcommand\pgph@citeslot[2]{%
  \begingroup
    \pgph@citesanitize
    \xdef\pgph@gnotestr{#2}%
  \endgroup
  \edef\pgph@sig{#1@@\detokenize\expandafter{\pgph@gnotestr}}%
  \ifcsname pgph@cslot@\pgph@sig\endcsname
```

```
    \edef\pgph@curslot{\csname pgph@cslot@\pgph@sig\endcsname}%
  \else
    \global\advance\pgph@citeslotcnt\@ne
    \edef\pgph@curslot{\the\pgph@citeslotcnt}%
    \pgph@gxdefcs{pgph@cslot@\pgph@sig}{\pgph@curslot}%
    \pgph@gxdefcs{pgph@cskey@\pgph@curslot}{#1}%
    \ifx\pgph@gnotestr\@empty\else
      \pgph@gxdefcs{pgph@citenote@\pgph@curslot}{\pgph@gnotestr}%
    \fi
  \fi
}
\newcommand\pgph@installcite{%
  \ifcsname cite\endcsname
    \let\pgph@saved@cite\cite
    \let\cite\pgph@cap@cite
  \fi
}
```

5.7 User commands

```
\newcommand\uses[1]{\pgph@recordref{#1}}
\newcommand\proofof[1]{%
  \ifcsname pgph@map@#1\endcsname
    \xdef\pgph@curproof{\csname pgph@map@#1\endcsname}%
  \fi
}
\newcommand\proofgraphedge[2]{%
  \g@addto@macro\pgph@edges{\pgph@labeledge{#1}{#2}}%
}
\newcommand\pgph@labeledge[2]{%
  \edef\pgph@ef{\pgph@resolve{#1}}%
  \ifx\pgph@ef\@empty\else
    \@for\pgph@one:=#2\do{%
      \edef\pgph@k{\expandafter\pgph@trimsp\expandafter{\pgph@one}}%
      \pgph@plainedge{\pgph@ef}{\pgph@k}}%
  \fi
}
\newcommand\proofgraphignore[2]{%
  \g@addto@macro\pgph@ignores{\pgph@ignorerule{#1}{#2}}%
}
\newcommand\proofgraphexclude[1]{%
  \g@addto@macro\pgph@excludes{\pgph@excluderule{#1}}%
}
\newcommand\proofgraphstyle[2]{\global\@namedef{pgph@style@#1}{#2}}
```

External citation nodes (option `cite`) share one style, held in `\pgph@citestyle` and overridable with `\proofgraphstylecite`; it defaults to `shape=note`.

```
\newcommand\pgph@citestyle{shape=note}
\newcommand\proofgraphstylecite[1]{\renewcommand\pgph@citestyle{#1}}
```

5.8 Resolution helpers

These run at end of document, after the `.aux` has populated the map.

```
\newcommand\pgph@resolve[1]{%
  \ifcsname pgph@map@#1\endcsname\csname pgph@map@#1\endcsname\fi
```

```
}
\newcommand\pgph@ignorerule[2]{%
  \edef\pgph@f{\pgph@resolve{#1}}\edef\pgph@t{\pgph@resolve{#2}}%
  \ifx\pgph@f\@empty\else\ifx\pgph@t\@empty\else
    \@namedef{pgph@ig@\pgph@f @\pgph@t}{}%
  \fi\fi
}
\newcommand\pgph@excluderule[1]{%
  \edef\pgph@x{\pgph@resolve{#1}}%
  \ifx\pgph@x\@empty\else\@namedef{pgph@ex@\pgph@x}{}\fi
}
```

5.9 Emission

`\pgph@dotesc` Escape the two characters that are special in a Graphviz double-quoted string, so a theorem name, number or citation note containing them does not corrupt the `.dot`: a backslash is doubled and a double quote is backslash-escaped (in that order). `\pgph@bschar` and `\pgph@dqchar` hold the catcode-12 backslash and double quote to search for; xstring's expansion mode is saved and restored around the two substitutions.

```
\begingroup
  \catcode`\|=0 \catcode`\"=12 \catcode`\\=12
  |gdef|pgph@bschar{\}%
  |gdef|pgph@dqchar{"}%
|endgroup
\newcommand\pgph@dotesc[1]{%
  \saveexpandmode\expandarg
  \StrSubstitute{#1}{\pgph@bschar}{\pgph@bschar\pgph@bschar}[#1]%
  \StrSubstitute{#1}{\pgph@dqchar}{\pgph@bschar\pgph@dqchar}[#1]%
  \restoreexpandmode}
```

`\pgph@node` Emit one node line, honouring exclusions and per-environment styling. External (`cite:`) targets are emitted separately as a pass over edges. An *unnumbered* result (no number captured) is omitted: a results graph is about numbered statements, and unnumbered theorem-like environments are typically expository or repeated copies (e.g., a class-predefined `\spnewtheorem*{claim}`). `\proofgraphtrack{`⟨*env*⟩`}` overrides this, so a deliberately unnumbered environment is still drawn.

```
\newcommand\pgph@node[3]{%
  \ifcsname pgph@ex@#1\endcsname\else
    \edef\pgph@nm{\ifcsname pgph@num@#1\endcsname%
      \csname pgph@num@#1\endcsname\fi}%
    \ifx\pgph@nm\@empty
      \ifcsname pgph@force@#2\endcsname
        \pgph@emitnode{#1}{#2}{#3}%
      \else
        \@namedef{pgph@ex@#1}{}%
      \fi
    \else
      \pgph@emitnode{#1}{#2}{#3}%
    \fi
  \fi
}
```

```
\newcommand\pgph@emitnode[3]{%
  \edef\pgph@s{\ifcsname pgph@style@#2\endcsname
    , \csname pgph@style@#2\endcsname\fi}%
  \edef\pgph@dn{\ifcsname pgph@name@#1\endcsname
    \csname pgph@name@#1\endcsname\else#3\fi}%
  \edef\pgph@lbl{\pgph@dn\space\pgph@nm}%
  \pgph@dotesc\pgph@lbl
  \immediate\write\pgph@out{%
    \space\space"n#1" [label="\pgph@lbl"\pgph@s];}%
  \ifcsname pgph@rmap@#1\endcsname
    \immediate\write\pgph@mapout{%
      \string\pgphnodemap{n#1}{\csname pgph@rmap@#1\endcsname}}%
  \fi
}
```

`\pgph@edge` Emit one edge, resolving the target label to an id and applying the self-loop, ignore and exclude filters. Targets beginning with `cite:` denote external nodes.

```
\newcommand\pgph@edge[2]{%
  \IfBeginWith{#2}{cite:}%
    {\pgph@citeedge{#1}{#2}}%
    {\pgph@plainedge{#1}{#2}}%
}
\newcommand\pgph@plainedge[2]{%
  \edef\pgph@t{\pgph@resolve{#2}}%
  \ifx\pgph@t\@empty\else
    \ifcsname pgph@ex@#1\endcsname\else
    \ifcsname pgph@ex@\pgph@t\endcsname\else
    \ifcsname pgph@ig@#1@\pgph@t\endcsname\else
      \ifboolexpr{ test {\ifnumequal{#1}{\pgph@t}}
                   and test {\ifdefstring{\pgph@selfloops}{remove}} }%
        {}{\pgph@emitedge{n#1}{n\pgph@t}}%
    \fi\fi\fi
  \fi
}
```

`\pgph@emitedge` `\pgph@writeedge` The relation recorded is always "the result whose proof we are in (the first argument) uses the target (the second)". The `direction` option chooses the arrow orientation: `direction=usedby` (default) draws *target to result*, so an arrow `a->b` reads "`a` is used by `b`" (i.e. `b` relies on `a`); `direction=uses` draws *result to target* ("`a` uses `b`"). The filters above work on the recorded relation, so they are unaffected by the chosen orientation. `\pgph@writeedge` deduplicates.

```
\newcommand\pgph@emitedge[2]{%
  \ifdefstring{\pgph@direction}{uses}%
    {\pgph@writeedge{#1}{#2}}%
    {\pgph@writeedge{#2}{#1}}%
}
\newcommand\pgph@writeedge[2]{%
  \ifcsname pgph@drawn@#1@#2\endcsname\else
    \@namedef{pgph@drawn@#1@#2}{}%
    \immediate\write\pgph@out{\space\space"#1" -> "#2";}%
  \fi
}
```

`\pgph@citeedge` `\pgph@setcitelabel` A `cite:`⟨*slot*⟩ target: recover the slot's BibTeX key, declare the external node once

per slot, then draw the edge. The node *label* is built by `\pgph@setcitelabel`: it is the key by default, or, with `citelabel=tag`, the printed citation tag (the "[42]"/"[Knu74]" text) recovered from the `\b@`⟨*key*⟩ macro that `\bibcite` writes to the `.aux` (so two runs are needed, and it falls back to the key when no tag is known, e.g., on the first run or with citation packages that do not populate `\b@`⟨*key*⟩). The slot's `\cite` note, when present, is prepended as "⟨*note*⟩, ...". The node-to-label map records the *key*, so all slots of one work hyperlink to the same bibliography entry.

```
\newcommand\pgph@citeedge[2]{%
  \ifcsname pgph@ex@#1\endcsname\else
    \StrBehind{#2}{cite:}[\pgph@slot]%
    \edef\pgph@key{\csname pgph@cskey@\pgph@slot\endcsname}%
    \ifcsname pgph@extnode@\pgph@slot\endcsname\else
      \@namedef{pgph@extnode@\pgph@slot}{}%
      \expandafter\pgph@setcitelabel\expandafter{\pgph@slot}%
      \pgph@dotesc\pgph@clabel
      \immediate\write\pgph@out{%
        \space\space"c@\pgph@slot"
        [label="\pgph@clabel",\pgph@citestyle];}%
      \immediate\write\pgph@mapout{%
        \string\pgphnodemapcite{c@\pgph@slot}{\pgph@key}}%
    \fi
    \pgph@emitedge{n#1}{c@\pgph@slot}%
  \fi
}
\edef\pgph@obrace{\expandafter\@gobble\string\{}
\newif\ifpgph@natnum
\newcommand\pgph@nil{}
```

`\pgph@citesanitize` neutralizes the wrappers and spacing cruft that citation labels carry, so that an `\edef` of `\b@`⟨*key*⟩ yields plain text: hyperref's `\hyper@@link` (keep its printed last argument) and the boxing/penalty/`\ignorespaces` noise that natbib bakes into author lists.

```
\newcommand\pgph@citesanitize{%
  \let\protect\@empty
  \def\hyper@@link[##1]##2##3##4{##4}%
  \let\unskip\@empty \let\ignorespaces\@empty \let\unhbox\@empty
  \let\penalty\@gobble \let\@M\@empty \let\voidb@x\@empty
  \let\nobreakspace\space \let~\space
  \def\hbox##1{##1}\def\mbox##1{##1}%
}
```

A natbib `\b@`⟨*key*⟩ is `{`⟨*num*⟩`}{`⟨*year*⟩`}{ `⟨*short*⟩`}{`⟨*long*⟩`}`. In numbers mode the tag is the number; in author–year mode it is the short author list and the year.

```
\def\pgph@natpick#1#2#3#4\pgph@nil{%
  \ifpgph@natnum\def\pgph@x{#1}\else\def\pgph@x{#3 #2}\fi}
\newcommand\pgph@natdo{\expandafter\pgph@natpick\pgph@x{}{}{}\pgph@nil}
\newcommand\pgph@setcitelabel[1]{%
  \edef\pgph@ckey{\csname pgph@cskey@#1\endcsname}%
  \edef\pgph@cbase{\pgph@ckey}%
  \ifdefstring{\pgph@citelabel}{tag}{%
    \ifcsname b@\pgph@ckey\endcsname
      \global\let\pgph@cbaseg\relax
      \begingroup
```

```
        \pgph@citesanitize
        \edef\pgph@x{\csname b@\pgph@ckey\endcsname}%
        \edef\pgph@dx{\detokenize\expandafter{\pgph@x}}%
        \IfBeginWith\pgph@dx\pgph@obrace{%
          \pgph@natnumfalse
          \@ifundefined{ifNAT@numbers}{}{%
            \expandafter\ifx\csname ifNAT@numbers\endcsname\iftrue
              \pgph@natnumtrue\fi}%
          \pgph@natdo
          \edef\pgph@dx{\detokenize\expandafter{\pgph@x}}%
        }{}%
        \ifx\pgph@x\@empty\else
          \IfSubStr\pgph@dx\@backslashchar{}{%
            \global\let\pgph@cbaseg\pgph@x}%
        \fi
      \endgroup
      \ifx\pgph@cbaseg\relax\else\edef\pgph@cbase{[\pgph@cbaseg]}\fi
    \fi
  }{}%
  \edef\pgph@clabel{%
    \ifcsname pgph@citenote@#1\endcsname
      \csname pgph@citenote@#1\endcsname, \fi
    \pgph@cbase}%
}
```

`\pgph@emit` Open the file, resolve editing rules, write nodes then edges, close, and optionally run Graphviz.

```
\newwrite\pgph@out
\newwrite\pgph@mapout
\newcommand\pgph@emit{%
  \immediate\openout\pgph@out=\pgph@file.dot\relax
  \immediate\openout\pgph@mapout=\pgph@file-proofgraph.map\relax
  \immediate\write\pgph@out{digraph proofgraph \pgph@ob}%
  \immediate\write\pgph@out{\space\space rankdir="\pgph@rankdir";}%
  \begingroup
    \pgph@ignores
    \pgph@excludes
    \pgph@nodes
    \pgph@edges
  \endgroup
  \immediate\write\pgph@out{\pgph@cb}%
  \immediate\closeout\pgph@out
  \immediate\closeout\pgph@mapout
  \ifpgph@autorun
    \immediate\write18{\pgph@engine\space -T\pgph@format\space
      \pgph@file.dot -o \pgph@file-proofgraph.\pgph@format}%
    \immediate\write18{\pgph@engine\space -Tplain\space
      \pgph@file.dot -o \pgph@file-proofgraph.plain}%
  \fi
}
```

The emission is registered at `\begin{document}` rather than at load time so that it is added to the end-document hook *after* packages loaded later (notably apxproof, whose appendix, with the deferred proofs and their references, is built at end

of document). This way the graph is written only once that material has been typeset, and the emission does not disturb the verbatim replay of deferred proofs.

```
\AtBeginDocument{\AtEndDocument{\pgph@emit}}
```

If TikZ is available, load its `calc` library now (at `\begin{document}`, where category codes are normal) so that the hyperlink overlay in `\proofgraph` can use it without reading a file mid-document.

```
\AtBeginDocument{\@ifpackageloaded{tikz}{\usetikzlibrary{calc}}{}}
```

`\pgphnodemap` `\pgphnodemapcite` One entry of the node-to-target map written by `\pgph@emit`: it records, for a Graphviz node identifier, what the node should link to. `\pgphnodemap` records the user *label* of the result a node represents; `\pgphnodemapcite` records the BibTeX *key* of an external citation node, so that (when the cited work is in the same document) the node can link to its bibliography entry, hyperref's `cite.`⟨*key*⟩ destination. The map is read back when the graph is embedded, to turn each node into a hyperlink.

```
\newcommand\pgphnodemap[2]{\global\@namedef{pgph@lbl@#1}{#2}}
\newcommand\pgphnodemapcite[2]{\global\@namedef{pgph@clbl@#1}{#2}}
```

`\proofgraph` Include the rendered graph (from a previous run) in `autorun` mode. When hyperref and TikZ are both available (and `hyperlinks` is not switched off), each node is made into an internal hyperlink to what it represents: a result node links to the result (its `\label`), and an external citation node (from the `cite` option) links to the cited work's bibliography entry, hyperref's `cite.`⟨*key*⟩ destination, when that work is cited in the same document. Graphviz produces a `.pdf` whose link annotations would be lost by `\includegraphics` (they sit inside a form XOBJECT), so instead we overlay transparent link areas at the node positions, which we read from the `-Tplain` output. Without those packages we fall back to a plain inclusion.

```
\newif\ifpgph@canlink
\newif\ifpgph@link
\newsavebox\pgph@imgbox
\newsavebox\pgph@natbox
\newread\pgph@plainin
\def\pgph@stop{}
\newcommand\proofgraph[1][]{%
  \@ifundefined{includegraphics}%
    {\PackageError{proofgraph}{\string\proofgraph\space requires the
       graphicx package}{Add \string\usepackage{graphicx} to your preamble
       to embed the rendered graph.}}%
    {\IfFileExists{\pgph@file-proofgraph.\pgph@format}%
      {\pgph@includegraph{#1}}%
      {\PackageWarning{proofgraph}{Rendered graph
         \pgph@file-proofgraph.\pgph@format\space not found; compile with
         shell-escape and the autorun option, then re-run}}}%
}
\newcommand\pgph@includegraph[1]{%
  \pgph@canlinkfalse
  \ifpgph@hyperlinks
    \@ifpackageloaded{hyperref}%
      {\@ifpackageloaded{tikz}%
         {\IfFileExists{\pgph@file-proofgraph.plain}%
            {\pgph@canlinktrue}{}}{}}{}%
  \fi
```

```
  \ifpgph@canlink
    \pgph@graphlinked{#1}%
  \else
    \includegraphics[#1]{\pgph@file-proofgraph.\pgph@format}%
  \fi
}
```

The linked version. We read the node-to-target map under safe category codes (a label may contain `_` or, under babel, an active `:`), measure the rendered image, place it in a TikZ node, then read the `-Tplain` file and drop one transparent link box on each mapped node. Graphviz draws the graph inset by a margin, so the image is larger than the layout bounding box the `-Tplain` coordinates live in; we recover that margin by also measuring the image at its *natural* size and comparing it with the `-Tplain` graph size (the `graph` line gives it in inches, hence the factor 72). Node positions are then expressed as fractions of the natural image, so they line up with the drawn nodes whatever size the image is finally scaled to.

```
\newcommand\pgph@graphlinked[1]{%
  \begingroup
    \catcode`\:=12 \catcode`\_=12 \catcode`\;=12 \catcode`\!=12
    \catcode`\?=12 \catcode`\&=12 \catcode`\^=12 \catcode`\~=12
    \InputIfFileExists{\pgph@file-proofgraph.map}{}{}%
  \endgroup
  \sbox\pgph@imgbox{%
    \includegraphics[#1]{\pgph@file-proofgraph.\pgph@format}}%
  \sbox\pgph@natbox{%
    \includegraphics{\pgph@file-proofgraph.\pgph@format}}%
  \edef\pgph@imgw{\the\wd\pgph@imgbox}%
  \edef\pgph@imgh{\the\dimexpr\ht\pgph@imgbox+\dp\pgph@imgbox\relax}%
  \edef\pgph@natw{\the\wd\pgph@natbox}%
  \edef\pgph@nath{\the\dimexpr\ht\pgph@natbox+\dp\pgph@natbox\relax}%
  \begin{tikzpicture}
    \node[inner sep=\z@,outer sep=\z@](pgph@P){\usebox\pgph@imgbox};
    \openin\pgph@plainin=\pgph@file-proofgraph.plain\relax
    \pgph@loopplain
    \closein\pgph@plainin
  \end{tikzpicture}%
}
\newcommand\pgph@loopplain{%
  \unless\ifeof\pgph@plainin
    \read\pgph@plainin to \pgph@line
    \pgph@procline
    \expandafter\pgph@loopplain
  \fi
}
\newcommand\pgph@procline{%
  \IfBeginWith{\pgph@line}{graph }%
    {\expandafter\pgph@dograph\pgph@line\pgph@stop}%
    {\IfBeginWith{\pgph@line}{node }%
       {\expandafter\pgph@donode\pgph@line\pgph@stop}{}}%
}
\def\pgph@dograph graph #1 #2 #3 #4\pgph@stop{%
  \def\pgph@gw{#2}\def\pgph@gh{#3}}
\def\pgph@donode node #1 #2 #3 #4 #5 #6\pgph@stop{%
  \StrDel{#1}{"}[\pgph@nid]%
```

```
  \pgph@linkfalse
  \ifcsname pgph@lbl@\pgph@nid\endcsname
    \edef\pgph@tgt{\csname pgph@lbl@\pgph@nid\endcsname}%
    \edef\pgph@thislink{\noexpand\hyperref[\pgph@tgt]}%
    \pgph@linktrue
  \else
    \ifcsname pgph@clbl@\pgph@nid\endcsname
      \edef\pgph@tgt{\csname pgph@clbl@\pgph@nid\endcsname}%
      \edef\pgph@thislink{\noexpand\hyperlink{cite.\pgph@tgt}}%
      \pgph@linktrue
    \fi
  \fi
  \ifpgph@link
    % Per-side Graphviz margin: half the natural-minus-layout size.
    \pgfmathsetmacro\pgph@mx{(\pgph@natw-\pgph@gw*72)/2}%
    \pgfmathsetmacro\pgph@my{(\pgph@nath-\pgph@gh*72)/2}%
    % Node centre as a fraction of the natural image.
    \pgfmathsetmacro\pgph@fx{(\pgph@mx+#2*72)/\pgph@natw}%
    \pgfmathsetmacro\pgph@fy{(\pgph@my+#3*72)/\pgph@nath}%
    \pgfmathsetlengthmacro\pgph@bw{#4*72/\pgph@natw*\pgph@imgw*0.92}%
    \pgfmathsetlengthmacro\pgph@bh{#5*72/\pgph@nath*\pgph@imgh*0.9}%
    \node[anchor=center]
      at ($($(pgph@P.south west)!\pgph@fx!(pgph@P.south east)$)%
      !\pgph@fy!($(pgph@P.north west)!\pgph@fx!(pgph@P.north east)$)$)
      {\pgph@thislink{\phantom{\rule{\pgph@bw}{\pgph@bh}}}};%
  \fi
}
```

⟨/package⟩

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Change History