The ebproof package
Formal proofs in the style of sequent calculus

Emmanuel Beffara*

Version 2.1.1 – Released 2021-01-28

Contents

1 Introduction

The ebproof package provides commands to typeset proof trees, in the style of sequent calculus and related systems:

$$\dfrac{\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \text{abs} \qquad \Gamma \vdash A}{\Gamma \vdash B}\ \text{app}$$

```
\begin{prooftree}
  \hypo{ \Gamma, A &\vdash B }
  \infer1[abs]{ \Gamma &\vdash A\to B }
  \hypo{ \Gamma \vdash A }
  \infer2[app]{ \Gamma \vdash B }
\end{prooftree}
```

The structure is very much inspired by the bussproofs package, in particular for the postfix notation. I actually wrote ebproof because there were some limitations in bussproofs that I did not know how to lift, and also because I did not like some choices in that package (and also because it was fun to write).

Any feedback is welcome, in the form of bug reports, feature requests or suggestions, through the web page of the project at https://framagit.org/manu/ebproof.

*E-mail: manu@beffara.org

2 Environments

prooftree
prooftree*

```
\begin{prooftree}[⟨options⟩]
  ⟨statements⟩
\end{prooftree}
```

The package provides the `prooftree` environment, in standard and starred variants. This typesets the proof tree described by the ⟨*statements*⟩, as described in section 3. The ⟨*options*⟩ provide default formatting options for the proof tree, available options are described in section 4.

Following the conventions of amsmath for alignment environments, the non-starred version produces a proof tree at the current position in the text flow (it can be used in math mode or text mode) while the starred version typesets the proof on a line of its own, like a displayed formula.

$$\dfrac{\dfrac{}{\vdash A}\quad\dfrac{\vdash B}{\vdash B, C}}{\vdash A\wedge B, C}\quad\rightsquigarrow\quad\dfrac{\dfrac{\dfrac{}{\vdash A}\quad\vdash B}{\vdash A\wedge B}}{\vdash A\wedge B, C}$$

```
\[
  \begin{prooftree}
    \infer0{ \vdash A }
    \hypo{ \vdash B } \infer1{ \vdash B, C }
    \infer2{ \vdash A\wedge B, C }
  \end{prooftree}
  \quad \rightsquigarrow \quad
  \begin{prooftree}
    \infer0{ \vdash A } \hypo{ \vdash B }
    \infer2{ \vdash A\wedge B }
    \infer1{ \vdash A\wedge B, C }
  \end{prooftree}
\]
```

3 Statements

Statements describe proofs in postfix notation: when typesetting a proof tree whose last rule has, say, two premisses, you will first write statements for the subtree of the first premiss, then statements for the subtree of the second premiss, then a statement like `\infer2{`⟨*conclusion*⟩`}` to build an inference with these two subtrees as premisses and the given text as conclusion.

Hence statements operate on a stack of proof trees. At the beginning of a `prooftree` environment, the stack is empty. At the end, it must contain exactly one tree, which is the one that will be printed.

Note that the commands defined in this section only exist right inside `prooftree` environments. If you have a macro with the same name as one of the statements, for instance `\hypo`, then this macro will keep its meaning outside `prooftree` environments as well as inside the arguments of a statement. If you really need to access the statements in another context, you can can always call them by prefixing their names with `ebproof`, for instance as `\ebproofhypo`.

3.1 Basic statements

The basic statements for building proofs are the following, where ⟨*options*⟩ stands for arbitrary options as described in section 4.

`\hypo` `\hypo[⟨options⟩]{⟨text⟩}`

The statement `\hypo` pushes a new proof tree consisting only in one conclusion line, with no premiss and no line above, in other words a tree with only a leaf (`\hypo` stands for *hypothesis*).

`\infer` `\infer[⟨options⟩]⟨arity⟩[⟨label⟩]{⟨text⟩}`

The statement `\infer` builds an inference step by taking some proof trees from the top of the stack, assembling them with a rule joining their conclusions and putting a new conclusion below. The ⟨*arity*⟩ is the number of sub-proofs, it may be any number including 0 (in this case there will be a line above the conclusion but no sub-proof). If ⟨*label*⟩ is present, it is used as the label on the right of the inference line; it is equivalent to using the `right label` option.

The ⟨*text*⟩ in these statements is the contents of the conclusion at the root of the tree that the statements create. It is typeset in math mode by default but any kind of formatting can be used instead, using the `template` option. The ⟨*label*⟩ text is formatted in horizontal text mode by default.

Each proof tree has a vertical axis, used for alignment of successive steps. The position of the axis is deduced from the text of the conclusion at the root of the tree: if ⟨*text*⟩ contains the alignment character `&` then the axis is set at that position, otherwise the axis is set at the center of the conclusion text. The `\infer` statement makes sure that the axis of the premiss is at the same position as the axis of the conclusion. If there are several premisses, it places the axis at the center between the left of the leftmost conclusion and the right of the rightmost conclusion:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash A, B, C}{A \vdash B, C}}{A, B \vdash C} \qquad D \vdash E}{A, B, D \vdash C, E}}{A, B \vdash C, D, E}}{A \vdash B, C, D, E}$$

```
\begin{prooftree}
  \hypo{ &\vdash A, B, C }
  \infer1{ A &\vdash B, C }
  \infer1{ A, B &\vdash C }
  \hypo{ D &\vdash E }
  \infer2{ A, B, D &\vdash C, E }
  \infer1{ A, B &\vdash C, D, E }
  \infer1{ A &\vdash B, C, D, E }
\end{prooftree}
```

`\ellipsis` `\ellipsis{⟨label⟩}{⟨text⟩}`

The statement `\ellipsis` typesets vertical dots, with a label on the right, and a new conclusion. No inference lines are inserted.

$$\begin{array}{l}\Gamma \vdash A \\ \quad\vdots\ \text{foo} \\ \Gamma \vdash A, B\end{array}$$

```
\begin{prooftree}
  \hypo{ \Gamma &\vdash A }
  \ellipsis{foo}{ \Gamma &\vdash A, B }
\end{prooftree}
```

3.2 Modifying proof trees

The following additional statements may be used to affect the format of the last proof tree on the stack.

`\rewrite` `\rewrite{⟨code⟩}`

The statement `\rewrite` is used to modify the proof of the stack while preserving its size and alignment. The ⟨*code*⟩ is typeset in horizontal mode, with the following control sequences defined:

- `\treebox` is a box register that contains the original material,
- `\treemark{⟨name⟩}` expands as the position of a given mark with respect to the left of the box.

A simple use of this statement is to change the color of a proof tree:

$$\dfrac{\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,\text{abs} \qquad \Gamma \vdash A}{\Gamma \vdash B}\,\text{app}$$

```
\begin{prooftree}
  \hypo{ \Gamma, A &\vdash B }
  \infer1[abs]{ \Gamma &\vdash A\to B }
  \rewrite{\color{red}\box\treebox}
  \hypo{ \Gamma \vdash A }
  \infer2[app]{ \Gamma \vdash B }
\end{prooftree}
```

Note the absence of spaces inside the call to `\rewrite`, because spaces would affect the position of the tree box. Note also that explicit use of `\treebox` is required to actually draw the subtree. Not using it will effectively not render the subtree, while still reserving its space in the enclosing tree:

$$\dfrac{\qquad \Gamma \vdash A}{\Gamma \vdash B}\,\text{app}$$

```
\begin{prooftree}
  \hypo{ \Gamma, A &\vdash B }
  \infer1[abs]{ \Gamma &\vdash A\to B }
  \rewrite{}
  \hypo{ \Gamma \vdash A }
  \infer2[app]{ \Gamma \vdash B }
\end{prooftree}
```

This kind of manipulation is useful for instance in conjunction with the beamer package to allow revealing subtrees of a proof tree progressively in successive slides of a given frame.

`\delims` `\delims{⟨left⟩}{⟨right⟩}`

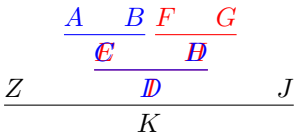
The statement `\delims` puts left and right delimiters around the whole sub-proof, without changing the alignment (the spacing is affected by the delimiters, however). The ⟨*left*⟩ text must contain an opening occurrence of `\left` and the ⟨*right*⟩ text must contain a matching occurrence of `\right`. For instance, `\delims{\left(}{\right)}` will put the sub-proof between parentheses.

$$\dfrac{A_1 \vee \cdots \vee A_n \qquad \left(\begin{array}{c}[A_i]\\ \vdots \\ B\end{array}\right)_{1\leq i\leq n}}{B}$$

```
\begin{prooftree}
  \hypo{ A_1 \vee \cdots \vee A_n }
  \hypo{ [A_i] }
  \ellipsis{}{ B }
  \delims{ \left( }{ \right)_{1\leq i\leq n} }
  \infer2{ B }
\end{prooftree}
```

`\overlay` `\overlay`

The statement `\overlay` combines the last two proofs on the stack into a single one, so that their conclusions are placed at the same point.



```
\begin{prooftree}
  \hypo{Z}
  \hypo{A} \hypo{B} \infer2{C} \hypo{D} \infer2{D}
    \rewrite{\color{blue}\box\treebox}
  \hypo{E} \hypo{F} \hypo{G} \infer2{H} \infer2{I}
    \rewrite{\color{red}\box\treebox}
  \overlay \hypo{J} \infer3{K}
\end{prooftree}
```

The primary use of this feature is for building animated presentations where a subtree in a proof has to be modified without affecting the general alignment of the surrounding proof. For instance, the example above could be used in Beamer to build successive slides in a given frame with two different subtrees:

```
\begin{prooftree}
  \hypo{Z}
  \hypo{A} \hypo{B} \infer2{C} \hypo{D} \infer2{D}
    \only<2>{\rewrite{}} % erases this version on slide 2
  \hypo{E} \hypo{F} \hypo{G} \infer2{H} \infer2{I}
    \only<1>{\rewrite{}} % erases this version on slide 1
  \overlay \hypo{J} \infer3{K}
\end{prooftree}
```

4 Options

The formatting of trees, conclusion texts and inference rules is affected by options, specified using the LaTeX3 key-value system. All options are in the `ebproof` module in the key tree. They can be set locally for a proof tree or for a single statement using optional arguments in the associated commands.

`\ebproofset`
`\set`

`\ebproofset{`⟨*options*⟩`}`

The statement `\ebproofset` is used to set some options. When used inside a `prooftree` environment, it can written `\set`. The options will apply in the current scope; using this in preamble will effectively set options globally. Specific options may also be specified for each proof tree and for each statement in a proof tree, using optional arguments.

4.1 General shape

The options in this section only make sense at the global level and at the proof level. Changing the proof style inside a `proof` environment has undefined behaviour.

`proof␣style` The option `proof style` sets the general shape for representing proofs. The following styles are provided:

upwards This is the default style. Proof trees grow upwards, with conclusions below and premisses above.

downwards Proof trees grow downwards, with conclusions above and premisses below.

$$\dfrac{\dfrac{\dfrac{}{\Gamma, A \vdash B}}{\Gamma \vdash A \to B}\,\text{abs} \qquad \Gamma \vdash A}{\Gamma \vdash B}\,\text{app}$$

```
\begin{prooftree}[proof style=downwards]
  \hypo{ \Gamma, A &\vdash B }
  \infer1[abs]{ \Gamma &\vdash A\to B }
  \hypo{ \Gamma \vdash A }
  \infer2[app]{ \Gamma \vdash B }
\end{prooftree}
```

In the optional argument of `prooftree` environments, proof styles can be specified directly, without prefixing the name by "`proof style=`". For instance, the first line of the example above could be written `\begin{prooftree}` equivalently.

`center` The option `center` toggles vertical centering of typeset proofs. If set to `true`, the tree produced by the `prooftree` environment will be vertically centered around the text line. If set to `false`, the base line of the tree will be the base line of the conclusion. The default value is `true`.

$$\dfrac{}{A \vdash A} \qquad \dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}$$

```
\begin{prooftree}[center=false]
  \infer0{ A \vdash A }
\end{prooftree}
\qquad
\begin{prooftree}[center=false]
  \hypo{ \Gamma, A \vdash B }
  \infer1{ \Gamma \vdash A \to B }
\end{prooftree}
```

4.2 Spacing

`separation` Horizontal separation between sub-proofs in an inference is defined by the option `separation`. The default value is `1.5em`.

$$\dfrac{\dfrac{A \quad B}{C} \qquad \dfrac{D \quad E \quad F}{G} \qquad H}{K}$$

```
\begin{prooftree}[separation=0.5em]
  \hypo{ A } \hypo{ B } \infer2{ C }
  \hypo{ D } \hypo{ E } \hypo{ F } \infer3{ G }
  \hypo{ H } \infer[separation=3em]3{ K }
\end{prooftree}
```

`rule␣margin` The spacing above and below inference lines is defined by the option `rule margin`. The default value is `0.7ex`.

$$\dfrac{\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,\text{abs} \qquad \Gamma \vdash A}{\Gamma \vdash B}\,\text{app}$$

```
\begin{prooftree}[rule margin=2ex]
  \hypo{ \Gamma, A &\vdash B }
  \infer1[abs]{ \Gamma &\vdash A\to B }
  \hypo{ \Gamma \vdash A }
  \infer2[app]{ \Gamma \vdash B }
\end{prooftree}
```

4.3 Shape of inference lines

`rule␣style`

The shape of inference lines is set by the option `rule style`. The following values are provided:

`simple`	a simple line (this is the default style)
`no rule`	no rule, only a single space of length `rule margin`
`double`	a double line
`dashed`	a single dashed line

The precise rendering is influenced by parameters specified below. Arbitrary new shapes can defined using the `\ebproofnewrulestyle` command described in section 4.5, using `rule code` option described below.

In the optional argument of the `\infer` statement, rule styles can be specified directly, without prefixing the style name by "`rule style=`". For instance, `\infer[dashed]` is equivalent to `\infer[rule style=dashed]`.

$$\dfrac{\Gamma \vdash A \to B \quad \Gamma \vdash !A \multimap B \qquad \dfrac{\dfrac{\Delta \vdash A}{\Delta \vdash !A} \qquad \dfrac{}{B \vdash B}}{\Delta, !A \multimap B \vdash B}}{\dfrac{\Gamma, \Delta \vdash B}{\Gamma \cup \Delta \vdash B}}$$

```
\begin{prooftree}
  \hypo{ \Gamma &\vdash A \to B }
  \infer[no rule]1{ \Gamma &\vdash {!A} \multimap B }
  \hypo{ \Delta &\vdash A }
  \infer[rule thickness=2pt]1{ \Delta &\vdash {!A} }
  \infer0{ B \vdash B }
  \infer[dashed]2{ \Delta, {!A}\multimap B \vdash B }
  \infer2{ \Gamma, \Delta &\vdash B }
  \infer[double]1{ \Gamma \cup \Delta &\vdash B }
\end{prooftree}
```

`rule␣thickness`
`rule␣separation`

The thickness of inference lines is defined by option `rule thickness`, it is `0.4pt` by default. The distance between the two lines in the `double` rule style is defined by the `rule separation` option. It is `2pt` by default.

`rule␣dash␣length`
`rule␣dash␣space`

For dashed rules, the length of dashes is defined by the option `rule dash length` and the space between dashes is defined by the option `rule dash space`. The default values are `0.2em` and `0.3em` respectively.

`rule␣code`

Arbitrary rule shapes can be optained using the `rule code` option. The argument is code is used to render the rule, it is executed in vertical mode in a `\vbox` whose `\hsize` is set to the width of the rule. Margins above and below are inserted automatically (they can be removed by setting `rule margin` to `0pt`).

$$\dfrac{\dfrac{\Gamma \vdash A}{\Gamma \vdash A, \ldots, A} \qquad \Delta, A, \ldots, A \vdash \Theta}{\Gamma, \Delta \vdash \Theta}$$

```
\begin{prooftree}[rule code={\hbox{\tikz
    \draw[decorate,decoration={snake,amplitude=.3ex}]
    (0,0) -- (\hsize,0);}}]
  \hypo{ \Gamma &\vdash A }
  \infer1{ \Gamma &\vdash A, \ldots, A }
  \hypo{ \Delta, A, \ldots, A \vdash \Theta }
  \infer2{ \Gamma, \Delta \vdash \Theta }
\end{prooftree}
```

Note that this example requires the tikz package, with the decorations.pathmorphing library for the `snake` decoration.

4.4 Format of conclusions and labels

`template`
`left␣template`
`right␣template`

The format of text in inferences is defined by templates. The option `template` is used for text with no alignment mark, the options `left template` and `right template` are used for the left and right side of the alignment mark when it is present. The value of these options is arbitrary TEX code, composed in horizontal mode. The macro `\inserttext` is used to insert the actual text passed to the `\hypo` and `\infer` statements. The default value for `template` is simply `$\inserttext$`, so that conclusions are set in math mode. The default values for `left template` and `right template` are similar, with spacing assuming that a relation symbol is put near the alignment mark, so that `\infer1{A &\vdash B}` is spaced correctly.

$$\dfrac{\dfrac{(\ \textbf{foo}\ ) \qquad \dfrac{(\ \textbf{bar}\ )}{(\ \textbf{baz}\ )}}{(\ \textbf{quux}\ )}$$

```
\begin{prooftree}[template=(\textbf\inserttext)]
  \hypo{ foo }
  \hypo{ bar }
  \infer1{ baz }
  \infer2{ quux }
\end{prooftree}
```

`left␣label`
`right␣label`

The text to use as the labels of the rules, on the left and on the right of the inference line, is defined by the options `left label` and `right label`. Using the second optional argument in `\infer` is equivalent to setting the `right label` option with the value of that argument.

$$\lambda\ \dfrac{\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \text{abs}}{@\ \dfrac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ \text{app}}$$

```
\begin{prooftree}
  \hypo{ \Gamma, A &\vdash B }
  \infer[left label=$\lambda$]1[abs]
    { \Gamma &\vdash A\to B }
  \hypo{ \Gamma \vdash A }
  \infer[left label=@]2[app]{ \Gamma \vdash B }
\end{prooftree}
```

`left␣label␣template`
`right␣label␣template`

Similarly to conclusions, labels are formatted according to templates. The code is arbitrary TEX code, composed in horizontal mode, where the macro `\inserttext` can be used to insert the actual label text. The default values are simply `\inserttext` so that labels are set in plain text mode.

`label␣separation`

The spacing between an inference line and its labels is defined by the option `label separation`, the default value is `0.5em`. The height of the horizontal axis used for aligning the labels with the rules is defined by the option `label axis`, the default value is `0.5ex`.

4.5 Style macros

The following commands allow for the definition of custom styles using the basic style options, in a way similar to PGF's "styles" and LaTeX3's "meta-keys". This allows setting a bunch of options with the same values in many proofs using a single definition.

`\ebproofnewstyle` `\ebproofnewstyle{`⟨*name*⟩`}{`⟨*options*⟩`}`

The statement `\ebproofnewstyle` defines a new style option with some ⟨*name*⟩ that sets a given set of ⟨*options*⟩.

For instance, the following code defines a new option `small` that sets various parameters so that proofs are rendered smaller.

$$\dfrac{\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \quad \Gamma \vdash A}{\Gamma \vdash B}$$

```
\ebproofnewstyle{small}{
  separation = 1em, rule margin = .5ex,
  template = \footnotesize$\inserttext$ }
\begin{prooftree}[small]
  \hypo{ \Gamma, A \vdash B }
  \infer1{ \Gamma \vdash A\to B }
  \hypo{ \Gamma \vdash A } \infer2{ \Gamma \vdash B }
\end{prooftree}
```

`\ebproofnewrulestyle` `\ebproofnewrulestyle{`⟨*name*⟩`}{`⟨*options*⟩`}`

The statement `\ebproofnewrulestyle` does the same for rule styles. The ⟨*options*⟩ part includes options used to set how to draw rules in the new style.

The option `rule code` is useful in this command as it allows to define arbitrary rule styles. For instance, the squiggly rule example above could be turned into a new rule style `zigzag` with the following code:

$$\dfrac{\dfrac{\Gamma \vdash A}{\Gamma \vdash A, \ldots, A} \qquad \Delta, A, \ldots, A \vdash \Theta}{\Gamma, \Delta \vdash \Theta}$$

```
\ebproofnewrulestyle{zigzag}{
  rule code = {\hbox{\tikz
    \draw[decorate,decoration={snake,amplitude=.3ex}]
    (0,0) -- (\hsize,0);}}}
\begin{prooftree}
  \hypo{ \Gamma &\vdash A }
  \infer1{ \Gamma &\vdash A, \ldots, A }
  \hypo{ \Delta, A, \ldots, A \vdash \Theta }
  \infer[zigzag]2{ \Gamma, \Delta \vdash \Theta }
\end{prooftree}
```

5 License

and version 1.3 or later is part of all distributions of LaTeX version 2005/12/01 or later.

This work has the LPPL maintenance status 'maintained'.

The Current Maintainer of this work is Emmanuel Beffara.

This work consists of the files `ebproof.sty` and `ebproof.tex`.

6 History

This section lists the principal evolutions of the package, in reverse chronological order.

Version 2.1.1 (2021-01-28) Bugfix release, no changes in the user interface.

- Fixes a deprecation issue with LaTeX3 release 2021-01-09 and various warnings that appear in LaTeX3 debugging mode.
- Fixes `proof style=downwards`.

Version 2.1 (2020-08-19) Mostly a bugfix release.

- Makes the `prooftree` environment robust to use in tabular contexts.
- Adds the `\overlay` statement.
- Fixes a compatibility issue with LaTeX release 2020-10-01.

Version 2.0 (2017-05-17) A complete rewrite of the code using the LaTeX3 programming environment. The incompatible changes from the user's point of view are the following:

- Proof statements are now writtten in lowercase (i.e. `\Infer` is now written `\infer` etc.) but the syntax is otherwise unchanged. The old uppercase commands still work but produce a deprecation warning, they will be removed in a future version.
- New styles are now defined using `\ebproofnewstyle` and `\ebproofnewrulestyle`. The previous method using PGF styles does not work anymore (because PGF is not used anymore).

The new commands and options are the following:

- The statement `\rewrite` generalizes `\Alter`,
- The option `label axis` controls vertical alignment of labels.

Version 1.1 (2015-03-13) A bugfix release. In `template` options, one now uses `\inserttext` instead of `#1` for the text arguments, which improves robustness.

Version 1.0 (2015-02-04) The first public release.

A Implementation

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}
\RequirePackage{expl3}
\RequirePackage{xparse}
\ProvidesExplPackage{ebproof}{2021/01/28}{2.1.1}{EB's proof trees}
⟨@@=ebproof⟩
```

A.1 Parameters

We first declare all options. For the meaning of options, see section 4.

```
\bool_new:N \l__ebproof_updown_bool
\keys_define:nn { ebproof } {
center .bool_set:N = \l__ebproof_center_bool,
proof~style .choice: ,
proof~style / upwards .code:n = \bool_set_false:N \l__ebproof_updown_bool,
proof~style / downwards .code:n = \bool_set_true:N \l__ebproof_updown_bool,
separation .dim_set:N = \l__ebproof_separation_dim,
rule~margin .dim_set:N = \l__ebproof_rule_margin_dim,
rule~thickness .dim_set:N = \l__ebproof_rule_thickness_dim,
rule~separation .dim_set:N = \l__ebproof_rule_separation_dim,
rule~dash~length .dim_set:N = \l__ebproof_rule_dash_length_dim,
rule~dash~space .dim_set:N = \l__ebproof_rule_dash_space_dim,
rule~code .tl_set:N = \l__ebproof_rule_code_tl,
rule~style .choice:,
template .tl_set:N = \l__ebproof_template_tl,
left~template .tl_set:N = \l__ebproof_left_template_tl,
right~template .tl_set:N = \l__ebproof_right_template_tl,
left~label .tl_set:N = \l__ebproof_left_label_tl,
right~label .tl_set:N = \l__ebproof_right_label_tl,
left~label~template .tl_set:N = \l__ebproof_left_label_template_tl,
right~label~template .tl_set:N = \l__ebproof_right_label_template_tl,
label~separation .dim_set:N = \l__ebproof_label_separation_dim,
label~axis .dim_set:N = \l__ebproof_label_axis_dim,
}
```

\ebproofnewrulestyle

We then define the document-level macro `\ebproofnewrulestyle` and use it to define the default styles. This simply consists in defining a meta-key.

```
\NewDocumentCommand \ebproofnewrulestyle { mm } {
  \keys_define:nn { ebproof } {
    rule~style / #1 .meta:nn = { ebproof } { #2 }
  }
}
```

(*End definition for* `\ebproofnewrulestyle`*. This function is documented on page 9.*)

The styles `simple`, `no rule` and `double` are defined in a straightforward way.

```
\ebproofnewrulestyle { simple } {
  rule~code = { \tex_hrule:D height \l__ebproof_rule_thickness_dim }
}
\ebproofnewrulestyle { no~rule } {
  rule~code =
}
\ebproofnewrulestyle { double } {
```

```
  rule~code = {
    \tex_hrule:D height \l__ebproof_rule_thickness_dim
    \skip_vertical:N \l__ebproof_rule_separation_dim
    \tex_hrule:D height \l__ebproof_rule_thickness_dim
  }
}
```

The `dashed` style uses leaders and filling for repeating a single dash. We use TEX primitives that have no LATEX3 counterpart for this.

```
\ebproofnewrulestyle { dashed } {
  rule~code = {
    \hbox_to_wd:nn { \tex_hsize:D } {
      \dim_set:Nn \l_tmpa_dim { \l__ebproof_rule_dash_space_dim / 2 }
      \skip_horizontal:n { -\l_tmpa_dim }
      \tex_cleaders:D \hbox:n {
        \skip_horizontal:N \l_tmpa_dim
        \tex_vrule:D
          height \l__ebproof_rule_thickness_dim
          width \l__ebproof_rule_dash_length_dim
        \skip_horizontal:N \l_tmpa_dim
      } \tex_hfill:D
      \skip_horizontal:n { -\l_tmpa_dim }
    }
  }
}
```

Now we can define the default values, including the default rule style.

```
\keys_set:nn { ebproof } {
  center = true,
  proof~style = upwards,
  separation = 1.5em,
  rule~margin = .7ex,
  rule~thickness = .4pt,
  rule~separation = 2pt,
  rule~dash~length = .2em,
  rule~dash~space = .3em,
  rule~style = simple,
  template = $\inserttext$,
  left~template = $\inserttext\mathrel{}$,
  right~template = $\mathrel{}\inserttext$,
  left~label = ,
  right~label = ,
  left~label~template = \inserttext,
  right~label~template = \inserttext,
  label~separation = 0.5em,
  label~axis = 0.5ex
}
```

`\ebproofnewstyle` Defining a style simply means defining a meta-key.

```
\NewDocumentCommand \ebproofnewstyle { mm } {
  \keys_define:nn { ebproof } { #1 .meta:n = { #2 } }
}
```

(*End definition for* `\ebproofnewstyle`. *This function is documented on page* 9.)

A.2 Proof boxes

TEX does not actually provide data structures, so we have to encode things. We provide an allocator for "registers" holding boxes with attributes. Such a register consists in a box register and a property list for marks, which maps mark names to values as explicit dimensions with units.

__ebproof_new:N

Using only public interfaces forces a convoluted approach to allocation: we use a global counter `\g_ebproof_register_int` to number registers, then each allocation creates registers named `\S_ebproof_K_N` where S is the scope of the register (local or global, deduced from the argument), K is the kind of component (box or marks) and N is the identifier of the register. The proof box register itself only contains the identifier used for indirection.

```
\int_new:N \g__ebproof_register_int
\cs_new:Nn \__ebproof_box:N {
  \str_item:nn { #1 } { 2 } __ebproof_ \tl_use:N #1 _box
}
\cs_new:Nn \__ebproof_marks:N {
  \str_item:nn { #1 } { 2 } __ebproof_ \tl_use:N #1 _prop
}
\cs_new:Nn \__ebproof_new:N {
  \tl_new:N #1
  \int_gincr:N \g__ebproof_register_int
  \str_if_eq:eeTF { \str_item:nn { #1 } { 2 } } { g }
    { \tl_gset:Nx #1 { \int_to_arabic:n { \g__ebproof_register_int } } }
    { \tl_set:Nx #1 { \int_to_arabic:n { \g__ebproof_register_int } } }
  \box_new:c { \__ebproof_box:N #1 }
  \prop_new:c { \__ebproof_marks:N #1 }
}
```

(*End definition for* `\__ebproof_new:N`.)

__ebproof_clear:N

The box is cleared by setting it to an empty hbox. Using `\box_clear:N` instead would not work because trying to push this box on the stack would not actually append any box.

```
\cs_new:Nn \__ebproof_clear:N {
  \hbox_set:cn { \__ebproof_box:N #1 } {}
  \prop_clear:c { \__ebproof_marks:N #1 }
  \__ebproof_set_mark:Nnn #1 { left } { 0pt }
  \__ebproof_set_mark:Nnn #1 { right } { 0pt }
  \__ebproof_set_mark:Nnn #1 { axis } { 0pt }
}
```

(*End definition for* `\__ebproof_clear:N`.)

A.2.1 Mark operations

__ebproof_set_mark:Nnn

Setting the value of a mark uses a temporary register to evaluate the dimension expression because values are stored textually in a property list.

```
\dim_new:N \l__ebproof_transit_dim
\cs_new:Nn \__ebproof_set_mark:Nnn {
  \dim_set:Nn \l__ebproof_transit_dim { #3 }
  \prop_put:cnV { \__ebproof_marks:N #1 } { #2 }
```

```
    \l__ebproof_transit_dim
}
```

(*End definition for* `\__ebproof_set_mark:Nnn`.)

`\__ebproof_mark:Nn` Getting the value of a mark simply consists in getting an item in a property list.

```
\cs_new:Nn \__ebproof_mark:Nn {
  \prop_item:cn { \__ebproof_marks:N #1 } { #2 }
}
```

(*End definition for* `\__ebproof_mark:Nn`.)

`\__ebproof_shift_x:Nn` This function shifts the marks by a specified amount, without modifying the box.

```
\cs_new:Nn \__ebproof_shift_x:Nn {
  \prop_map_inline:cn { \__ebproof_marks:N #1 } {
    \__ebproof_set_mark:Nnn #1 { ##1 } { ##2 + #2 }
  }
}
```

(*End definition for* `\__ebproof_shift_x:Nn`.)

`\__ebproof_enlarge_conclusion:NN` This function moves the left and right marks of the first tree so that they are at least as far from the axis as they are in the second tree. For instance we get the following:

L — A ——— R	box 1 before
L ——— A — R	box 2 before
L ——— A ——— R	box 1 after

The contents of the trees are unchanged.

```
\cs_new:Nn \__ebproof_enlarge_conclusion:NN {
  \dim_set:Nn \l_tmpa_dim { \__ebproof_mark:Nn #1 {axis}
    + \__ebproof_mark:Nn #2 {left} - \__ebproof_mark:Nn #2 {axis} }
  \dim_compare:nNnT { \l_tmpa_dim } < { \__ebproof_mark:Nn #1 {left} } {
    \__ebproof_set_mark:Nnn #1 {left} { \l_tmpa_dim } }
  \dim_set:Nn \l_tmpa_dim { \__ebproof_mark:Nn #1 {axis}
    + \__ebproof_mark:Nn #2 {right} - \__ebproof_mark:Nn #2 {axis} }
  \dim_compare:nNnT { \l_tmpa_dim } > { \__ebproof_mark:Nn #1 {right} } {
    \__ebproof_set_mark:Nnn #1 {right} { \l_tmpa_dim } }
}
```

(*End definition for* `\__ebproof_enlarge_conclusion:NN`.)

A.2.2 Building blocks

`\__ebproof_make_simple:Nn` Make a tree with explicit material in horizontal mode. Set the left and right marks to extremal positions and set the axis in the middle.

```
\cs_new:Nn \__ebproof_make_simple:Nn {
  \hbox_set:cn { \__ebproof_box:N #1 } { #2 }
  \__ebproof_set_mark:Nnn #1 { left } { 0pt }
  \__ebproof_set_mark:Nnn #1 { axis } { \box_wd:c { \__ebproof_box:N #1 } / 2 }
  \__ebproof_set_mark:Nnn #1 { right } { \box_wd:c { \__ebproof_box:N #1 } }
}
```

(*End definition for* `\__ebproof_make_simple:Nn`.)

__ebproof_make_split:Nnn Make a tree with explicit material in horizontal mode, split in two parts. Set the left and right marks to extremal positions and set the axis between the two parts.

```
\cs_new:Nn \__ebproof_make_split:Nnn {
  \__ebproof_set_mark:Nnn #1 { left } { 0pt }
  \hbox_set:cn { \__ebproof_box:N #1 } { #2 }
  \__ebproof_set_mark:Nnn #1 { axis } { \box_wd:c { \__ebproof_box:N #1 } }
  \hbox_set:cn { \__ebproof_box:N #1 } { \hbox_unpack:c { \__ebproof_box:N #1 } #3 }
  \__ebproof_set_mark:Nnn #1 { right } { \box_wd:c { \__ebproof_box:N #1 } }
}
```

(*End definition for* `\__ebproof_make_split:Nnn`.)

__ebproof_make_vertical:Nnnn Make a tree with explicit material in vertical mode, using an explicit width and axis.

```
\cs_new:Nn \__ebproof_make_vertical:Nnnn {
  \__ebproof_set_mark:Nnn #1 { left } { 0pt }
  \__ebproof_set_mark:Nnn #1 { axis } { #2 }
  \__ebproof_set_mark:Nnn #1 { right } { #3 }
  \vbox_set:cn { \__ebproof_box:N #1 } {
    \dim_set:Nn \tex_hsize:D { \__ebproof_mark:Nn #1 {right} }
    #4
  }
  \box_set_wd:cn { \__ebproof_box:N #1 } { \__ebproof_mark:Nn #1 {right} }
}
```

(*End definition for* `\__ebproof_make_vertical:Nnnn`.)

A.2.3 Assembling boxes

__ebproof_extend:Nnnnn Extend a tree box. The marks are shifted so that alignment is preserved. The arguments are dimensions for the left, top, right and bottom sides respectively.

```
\cs_new:Nn \__ebproof_extend:Nnnnn {
  \dim_compare:nNnF { #2 } = { 0pt } {
    \hbox_set:cn { \__ebproof_box:N #1 } {
      \skip_horizontal:n { #2 }
      \box_use:c { \__ebproof_box:N #1 }
    }
    \__ebproof_shift_x:Nn #1 { #2 }
  }
  \box_set_ht:Nn #1 { \box_ht:c { \__ebproof_box:N #1 } + #3 }
  \box_set_wd:Nn #1 { \box_wd:c { \__ebproof_box:N #1 } + #4 }
  \box_set_dp:Nn #1 { \box_dp:c { \__ebproof_box:N #1 } + #5 }
}
```

(*End definition for* `\__ebproof_extend:Nnnnn`.)

__ebproof_append_right:NnN Append the contents of the second tree to the first one on the right, with matching baselines. The marks of both trees are preserved. The middle argument specifies the space to insert between boxes.

```
\cs_new:Nn \__ebproof_append_right:NnN {
  \hbox_set:cn { \__ebproof_box:N #1 } {
    \box_use:c { \__ebproof_box:N #1 }
    \dim_compare:nNnF { #2 } = { 0pt } { \skip_horizontal:n { #2 } }
    \box_use:c { \__ebproof_box:N #3 }
  }
}
```

(*End definition for* `\__ebproof_append_right:NnN`.)

`\__ebproof_append_left:NnN` Append the contents of the second tree to the first one on the left, with matching baselines. The marks of the first tree are shifted accordingly. The middle argument specifies the space to insert between boxes.

```
\cs_new:Nn \__ebproof_append_left:NnN {
  \__ebproof_shift_x:Nn #1 { \box_wd:c { \__ebproof_box:N #3 } + #2 }
  \hbox_set:cn { \__ebproof_box:N #1 } {
    \box_use:c { \__ebproof_box:N #3 }
    \dim_compare:nNnF { #2 } = { 0pt } { \skip_horizontal:n { #2 } }
    \box_use:c { \__ebproof_box:N #1 }
  }
}
```

(*End definition for* `\__ebproof_append_left:NnN`.)

`\__ebproof_align:NN` Shift one of two trees to the right so that their axes match. The marks of the one that is shifted are updated accordingly.

```
\cs_new:Nn \__ebproof_align:NN {
  \dim_set:Nn \l_tmpa_dim
    { \__ebproof_mark:Nn #2 {axis} - \__ebproof_mark:Nn #1 {axis} }
  \dim_compare:nNnTF \l_tmpa_dim < { 0pt } {
    \__ebproof_extend:Nnnnn #2 { -\l_tmpa_dim } { 0pt } { 0pt } { 0pt }
  } {
    \__ebproof_extend:Nnnnn #1 { \l_tmpa_dim } { 0pt } { 0pt } { 0pt }
  }
}
```

(*End definition for* `\__ebproof_align:NN`.)

`\__ebproof_append_above:NN` Append the contents of the second tree above the first one, with matching axes. The marks of the first tree are preserved.

```
\cs_new:Nn \__ebproof_append_above:NN {
  \__ebproof_align:NN #1 #2
  \vbox_set:cn  { \__ebproof_box:N #1 } {
    \box_use:c { \__ebproof_box:N #2 }
    \tex_prevdepth:D -1000pt
    \box_use:c { \__ebproof_box:N #1 }
  }
}
```

(*End definition for* `\__ebproof_append_above:NN`.)

`\__ebproof_append_below:NN` Append the contents of the second tree below the first one, with matching axes. The marks of the first tree are preserved.

```
\cs_new:Nn \__ebproof_append_below:NN {
  \__ebproof_align:NN #1 #2
  \vbox_set_top:cn { \__ebproof_box:N #1 } {
    \box_use:c { \__ebproof_box:N #1 }
    \tex_prevdepth:D -1000pt
    \box_use:c { \__ebproof_box:N #2 }
  }
}
```

(*End definition for* `\__ebproof_append_below:NN`.)

__ebproof_overlay:NN Append the second tree as an overlay over the first one, so that the baselines and axes match. The bounding box of the result adjusts to contain both trees.

```
\cs_new:Nn \__ebproof_overlay:NN {
  \__ebproof_align:NN #1 #2
  \hbox_set:cn { \__ebproof_box:N #1 } {
    \hbox_overlap_right:n { \box_use:c { \__ebproof_box:N #1 } }
    \box_use:c { \__ebproof_box:N #2 }
    \dim_compare:nNnT
      { \box_wd:c { \__ebproof_box:N #2 } } < { \box_wd:c { \__ebproof_box:N #1 } }
      { \skip_horizontal:n
        { \box_wd:c { \__ebproof_box:N #1 } - \box_wd:c { \__ebproof_box:N #2 } } }
  }
}
```

(*End definition for* `\__ebproof_overlay:NN`.)

__ebproof_vcenter:N Shift the material in a tree vertically so that the height and depth are equal (like TEX's `\vcenter` but around the baseline).

```
\cs_new:Nn \__ebproof_vcenter:N {
  \dim_set:Nn \l_tmpa_dim
    { ( \box_ht:c { \__ebproof_box:N #1 } - \box_dp:c { \__ebproof_box:N #1 } ) / 2 }
  \box_set_eq:Nc \l_tmpa_box { \__ebproof_box:N #1 }
  \hbox_set:cn { \__ebproof_box:N #1 }
    { \box_move_down:nn { \l_tmpa_dim } { \box_use:N \l_tmpa_box } }
}
```

(*End definition for* `\__ebproof_vcenter:N`.)

A.3 Making inferences

The following commands use the parameters defined at the beginning of the package for actually building proof trees using the commands defined above.

__ebproof_append_vertical:NN Append the contents of the second tree above or below the first one, depending on current settings. Axes are aligned and the marks of the first tree are preserved.

```
\cs_new:Nn \__ebproof_append_vertical:NN {
  \bool_if:NTF \l__ebproof_updown_bool
    { \__ebproof_append_below:NN #1 #2 }
    { \__ebproof_append_above:NN #1 #2 }
}
```

(*End definition for* `\__ebproof_append_vertical:NN`.)

__ebproof_make_rule_for:NNN Make a box containing an inference rule with labels, using the current settings. The width and axis position are taken as those of the conclusion of another tree box. The third argument is used as a temporary register for building labels.

```
\cs_new:Nn \__ebproof_make_rule_for:NNN {
```

Build the rule.

```
  \__ebproof_make_vertical:Nnnn #1
    { \__ebproof_mark:Nn #2 {axis} - \__ebproof_mark:Nn #2 {left} }
    { \__ebproof_mark:Nn #2 {right} - \__ebproof_mark:Nn #2 {left} }
    {
      \skip_vertical:N \l__ebproof_rule_margin_dim
```

```
      \tl_if_empty:NF { \l__ebproof_rule_code_tl } {
        \tl_use:N \l__ebproof_rule_code_tl
        \skip_vertical:N \l__ebproof_rule_margin_dim
      }
    }
  \__ebproof_vcenter:N #1
```

Append the left label.

```
  \tl_if_blank:VF \l__ebproof_left_label_tl {
    \__ebproof_make_simple:Nn #3 {
      \box_move_down:nn { \l__ebproof_label_axis_dim } { \hbox:n {
        \cs_set_eq:NN \inserttext \l__ebproof_left_label_tl
        \tl_use:N \l__ebproof_left_label_template_tl
      } }
    }
    \box_set_ht:cn { \__ebproof_box:N #3 } { 0pt }
    \box_set_dp:cn { \__ebproof_box:N #3 } { 0pt }
    \__ebproof_append_left:NnN
      \l__ebproof_c_box \l__ebproof_label_separation_dim \l__ebproof_d_box
  }
```

Append the right label.

```
  \tl_if_blank:VF \l__ebproof_right_label_tl {
    \__ebproof_make_simple:Nn #3 {
      \box_move_down:nn { \l__ebproof_label_axis_dim } { \hbox:n {
        \cs_set_eq:NN \inserttext \l__ebproof_right_label_tl
        \tl_use:N \l__ebproof_right_label_template_tl
      } }
    }
    \box_set_ht:cn { \__ebproof_box:N #3 } { 0pt }
    \box_set_dp:cn { \__ebproof_box:N #3 } { 0pt }
    \__ebproof_append_right:NnN
      \l__ebproof_c_box \l__ebproof_label_separation_dim \l__ebproof_d_box
  }
}
```

(*End definition for* `\__ebproof_make_rule_for:NNN`.)

A.4 Stack-based interface

A.4.1 The stack

Logically, box structures are stored on a stack. However, TEX does not provide data structures for that and the grouping mechanism is not flexible enough, so we encode them using what we actually have. A stack for boxes is implemented using a global hbox `\g__ebproof_stack_box` that contains all the boxes successively. A sequence `\g__ebproof_stack_seq` is used to store the dimensions property lists textually. We maintain a counter `\g__ebproof_level_int` with the number of elements on the stack, for consistency checks.

```
\int_new:N \g__ebproof_level_int
\box_new:N \g__ebproof_stack_box
\seq_new:N \g__ebproof_stack_seq
```

__ebproof_clear_stack: Clear the stack.

```
\cs_new:Nn \__ebproof_clear_stack: {
  \int_gset:Nn \g__ebproof_level_int { 0 }
  \hbox_gset:Nn \g__ebproof_stack_box { }
  \seq_gclear:N \g__ebproof_stack_seq
}
```

(*End definition for* `\__ebproof_clear_stack:`.)

__ebproof_push:N Push the contents of a register on the stack.

```
\cs_new:Nn \__ebproof_push:N {
  \int_gincr:N \g__ebproof_level_int
  \hbox_gset:Nn \g__ebproof_stack_box
    { \hbox_unpack:N \g__ebproof_stack_box \box_use:c { \__ebproof_box:N #1 } }
  \seq_gput_left:Nv \g__ebproof_stack_seq
    { \__ebproof_marks:N #1 }
}
```

(*End definition for* `\__ebproof_push:N`.)

__ebproof_pop:N Pop the value from the top of the stack into a register.

```
\cs_new:Nn \__ebproof_pop:N {
  \int_compare:nNnTF { \g__ebproof_level_int } > { 0 } {
    \int_gdecr:N \g__ebproof_level_int
    \hbox_gset:Nn \g__ebproof_stack_box {
      \hbox_unpack:N \g__ebproof_stack_box
      \box_gset_to_last:N \g_tmpa_box
    }
    \box_set_eq_drop:cN { \__ebproof_box:N #1 } \g_tmpa_box
    \seq_gpop_left:NN \g__ebproof_stack_seq \l_tmpa_tl
    \tl_set_eq:cN { \__ebproof_marks:N #1 } \l_tmpa_tl
  } {
    \PackageError{ebproof}{Missing~premiss~in~a~proof~tree}{}
    \__ebproof_clear:N #1
  }
}
```

(*End definition for* `\__ebproof_pop:N`.)

A.4.2 Assembling trees

```
\__ebproof_new:N \l__ebproof_a_box
\__ebproof_new:N \l__ebproof_b_box
\__ebproof_new:N \l__ebproof_c_box
\__ebproof_new:N \l__ebproof_d_box
```

__ebproof_join_horizontal:n Join horizontally a number of elements at the top of the stack. If several trees are joined, use the left mark of the left tree, the right mark of the right tree and set the axis in the middle of these marks.

```
\cs_new:Nn \__ebproof_join_horizontal:n {
  \int_case:nnF { #1 } {
  { 0 } {
    \group_begin:
    \__ebproof_clear:N \l__ebproof_a_box
    \__ebproof_push:N \l__ebproof_a_box
```

```
    \group_end:
  }
  { 1 } { }
  } {
    \group_begin:
    \__ebproof_pop:N \l__ebproof_a_box
    \prg_replicate:nn { #1 - 1 } {
      \__ebproof_pop:N \l__ebproof_b_box
      \__ebproof_append_left:NnN
        \l__ebproof_a_box \l__ebproof_separation_dim \l__ebproof_b_box
    }
    \__ebproof_set_mark:Nnn \l__ebproof_a_box { left }
      { \__ebproof_mark:Nn \l__ebproof_b_box { left } }
    \__ebproof_set_mark:Nnn \l__ebproof_a_box { axis }
      { ( \__ebproof_mark:Nn \l__ebproof_a_box { left }
        + \__ebproof_mark:Nn \l__ebproof_a_box { right } ) / 2 }
    \__ebproof_push:N \l__ebproof_a_box
    \group_end:
  }
}
```

(*End definition for* `\__ebproof_join_horizontal:n`.)

`\__ebproof_join_vertical:` Join vertically the two elements at the top of the stack, with a horizontal rule of the appropriate size.

```
\cs_new:Nn \__ebproof_join_vertical: {
  \group_begin:
  \__ebproof_pop:N \l__ebproof_a_box
  \__ebproof_pop:N \l__ebproof_b_box
  \__ebproof_enlarge_conclusion:NN \l__ebproof_b_box \l__ebproof_a_box
  \__ebproof_make_rule_for:NNN \l__ebproof_c_box \l__ebproof_b_box
    \l__ebproof_d_box
  \__ebproof_append_vertical:NN \l__ebproof_a_box \l__ebproof_c_box
  \__ebproof_append_vertical:NN \l__ebproof_a_box \l__ebproof_b_box
  \__ebproof_push:N \l__ebproof_a_box
  \group_end:
}
```

(*End definition for* `\__ebproof_join_vertical:`.)

A.4.3 High-level commands

`\__ebproof_statement_parse:w` An auxiliary function for parsing the argument in `\__ebproof_push_statement:n`.

```
\cs_new:Npn \__ebproof_statement_parse:w #1 & #2 & #3 \q_stop {
  \tl_if_empty:nTF { #3 } {
    \__ebproof_make_simple:Nn \l__ebproof_a_box
      { \cs_set:Npn \inserttext { #1 } \tl_use:N \l__ebproof_template_tl }
  } {
    \__ebproof_make_split:Nnn \l__ebproof_a_box
      { \cs_set:Npn \inserttext { #1 } \tl_use:N \l__ebproof_left_template_tl }
      { \cs_set:Npn \inserttext { #2 } \tl_use:N \l__ebproof_right_template_tl }
  }
  \__ebproof_push:N \l__ebproof_a_box
}
```

(*End definition for* `\__ebproof_statement_parse:w`.)

`\__ebproof_push_statement:n` Push a box with default formatting, using explicit alignment if the code contains a `&` character

```
\cs_new:Nn \__ebproof_push_statement:n {
  \__ebproof_statement_parse:w #1 & & \q_stop
}
```

(*End definition for* `\__ebproof_push_statement:n`.)

A.5 Document interface

A.5.1 Functions to define statements

The `\g__ebproof_statements_seq` variable contains the list of all defined statements. For each statement `X`, there is a document command `\ebproofX` and the alias `\X` is defined when entering a `prooftree` environment.

```
\seq_new:N \g__ebproof_statements_seq
```

`\__ebproof_setup_statements:` Install the aliases for statements, saving the original value of the control sequences.

```
\cs_new:Nn \__ebproof_setup_statements: {
  \seq_map_inline:Nn \g__ebproof_statements_seq {
    \cs_set_eq:cc { ebproof_saved_ ##1 } { ##1 }
    \cs_set_eq:cc { ##1 } { ebproof ##1 }
  }
}
```

(*End definition for* `\__ebproof_setup_statements:`.)

`\__ebproof_restore_statements:` Restore the saved meanings of the control sequences. This is useful when interpreting user-provided code in statement arguments. The meanings are automatically restored when leaving a `prooftree` environment because of grouping.

```
\cs_new:Nn \__ebproof_restore_statements: {
  \seq_map_inline:Nn \g__ebproof_statements_seq {
    \cs_set_eq:cc { ##1 } { ebproof_saved_ ##1 }
  }
}
```

(*End definition for* `\__ebproof_restore_statements:`.)

`\__ebproof_new_statement:nnn` Define a new statement. The first argument is the name, the second one is an argument specifier as used by `xparse` and the third one is the body of the command.

```
\cs_new:Nn \__ebproof_new_statement:nnn {
  \exp_args:Nc \NewDocumentCommand { ebproof#1 }{ #2 } { #3 }
  \seq_gput_right:Nn \g__ebproof_statements_seq { #1 }
}
```

(*End definition for* `\__ebproof_new_statement:nnn`.)

__ebproof_new_deprecated_statement:nnnn

Define a deprecated statement. The syntax is the same as above except that an extra argument in third position indicates what should be used instead. The effect is the same except that a warning message is issued the first time the statement is used.

```
\cs_new:Nn \__ebproof_new_deprecated_statement:nnnn {
  \cs_new:cpn { ebproof_#1_warning: } {
    \PackageWarning { ebproof } { \token_to_str:c{#1}~is~deprecated,~#3 }
    \cs_gset:cn { ebproof_#1_warning: } { }
  }
  \__ebproof_new_statement:nnn { #1 } { #2 }
    { \use:c { ebproof_#1_warning: } #4 }
}
```

(*End definition for* `\__ebproof_new_deprecated_statement:nnnn`.)

A.5.2 Basic commands

\ebproofset
\set

This is a simple wrapper around `\keys_set:nn`.

```
\__ebproof_new_statement:nnn { set } { m } {
  \keys_set:nn { ebproof } { #1 }
}
```

(*End definition for* `\ebproofset` *and* `\set`. *These functions are documented on page 5.*)

\hypo

This is mostly a wrapper around `\ebproof_push_statement:n`, with material to handle options and the statements macros.

```
\__ebproof_new_statement:nnn { hypo } { O{} m } {
  \group_begin:
  \__ebproof_restore_statements:
  \keys_set:nn { ebproof } { #1 }
  \__ebproof_push_statement:n { #2 }
  \group_end:
}
```

(*End definition for* `\hypo`. *This function is documented on page 3.*)

\infer

This is a bit more involved than `\hypo` because we have to handle rule style options and joining.

```
\__ebproof_new_statement:nnn { infer } { O{} m O{} m } {
  \group_begin:
  \__ebproof_restore_statements:
  \keys_set_known:nnN { ebproof / rule~style } { #1 } \l_tmpa_tl
  \keys_set:nV { ebproof } \l_tmpa_tl
  \tl_set:Nn \l__ebproof_right_label_tl { #3 }
  \__ebproof_join_horizontal:n { #2 }
  \__ebproof_push_statement:n { #4 }
  \__ebproof_join_vertical:
  \group_end:
}
```

(*End definition for* `\infer`. *This function is documented on page 3.*)

\ellipsis An ellipsis is made by hand using vertical leaders to render the dots after rendering the label.

```
\__ebproof_new_statement:nnn { ellipsis } { m m } {
  \group_begin:
  \__ebproof_restore_statements:
  \tl_clear:N \l__ebproof_rule_code_tl
  \__ebproof_make_split:Nnn \l__ebproof_a_box { } {
    \vbox_set:Nn \l_tmpa_box {
      \skip_vertical:n { 1.2ex }
      \hbox:n { \tex_ignorespaces:D #1 }
      \skip_vertical:n { 1.2ex }
    }
    \vbox_to_ht:nn { \box_ht:N \l_tmpa_box } {
      \tex_xleaders:D \vbox_to_ht:nn { .8ex }
        { \tex_vss:D \hbox:n { . } \tex_vss:D }
      \tex_vfill:D
    }
    \hbox_overlap_right:n { ~ \box_use:N \l_tmpa_box }
  }
  \__ebproof_push:N \l__ebproof_a_box
  \__ebproof_join_vertical:
  \__ebproof_push_statement:n {#2}
  \__ebproof_join_vertical:
  \group_end:
}
```

(*End definition for* `\ellipsis`*. This function is documented on page 3.*)

A.5.3 Modifying trees

\rewrite Rewrite the box at the top of the stack while preserving its dimensions an marks. The code is typeset in horizontal mode, with control sequences to access the original box and its marks.

```
\__ebproof_new_statement:nnn { rewrite } { m } {
  \group_begin:
  \__ebproof_restore_statements:
  \__ebproof_pop:N \l__ebproof_a_box
  \box_set_eq:Nc \l_tmpa_box { \__ebproof_box:N \l__ebproof_a_box }
  \hbox_set:Nn \l_tmpb_box {
    \cs_set_eq:NN \treebox \l_tmpa_box
    \cs_set:Npn \treemark { \__ebproof_mark:Nn \l__ebproof_a_box }
    { #1 }
  }
  \box_set_wd:Nn \l_tmpb_box { \box_wd:c { \__ebproof_box:N \l__ebproof_a_box } }
  \box_set_ht:Nn \l_tmpb_box { \box_ht:c { \__ebproof_box:N \l__ebproof_a_box } }
  \box_set_dp:Nn \l_tmpb_box { \box_dp:c { \__ebproof_box:N \l__ebproof_a_box } }
  \box_set_eq:cN { \__ebproof_box:N \l__ebproof_a_box } \l_tmpb_box
  \__ebproof_push:N \l__ebproof_a_box
  \group_end:
}
```

(*End definition for* `\rewrite`*. This function is documented on page 4.*)

\delims Insert \left and \right delimiters without changing the alignment.

```
\__ebproof_new_statement:nnn { delims } { m m } {
  \group_begin:
  \__ebproof_restore_statements:
  \__ebproof_pop:N \l__ebproof_a_box
  \hbox_set:Nn \l_tmpa_box
    { $ \tex_vcenter:D { \box_use:c { \__ebproof_box:N \l__ebproof_a_box } } $ }
  \dim_set:Nn \l_tmpa_dim
    { \box_ht:N \l_tmpa_box - \box_ht:c { \__ebproof_box:N \l__ebproof_a_box } }
  \hbox_set:cn { \__ebproof_box:N \l__ebproof_a_box } {
    $ #1 \tex_vrule:D
      height \box_ht:N \l_tmpa_box depth \box_dp:N \l_tmpa_box width 0pt
    \tex_right:D . $
  }
  \__ebproof_shift_x:Nn \l__ebproof_a_box
    { \box_wd:c { \__ebproof_box:N \l__ebproof_a_box } }
  \hbox_set:cn { \__ebproof_box:N \l__ebproof_a_box } {
    \hbox_unpack:c { \__ebproof_box:N \l__ebproof_a_box }
    $ \tex_left:D . \box_use:N \l_tmpa_box #2 $
  }
  \hbox_set:cn { \__ebproof_box:N \l__ebproof_a_box }
    { \box_move_down:nn { \l_tmpa_dim }
        { \box_use:c { \__ebproof_box:N \l__ebproof_a_box } } }
  \__ebproof_push:N \l__ebproof_a_box
  \group_end:
}
```

(*End definition for* \delims*. This function is documented on page 4.*)

\overlay Pop two trees and append the second tree as an overlay over the first one, so that the baselines and axes match. The bounding box of the result adjusts to contain both trees.

```
\__ebproof_new_statement:nnn { overlay } { } {
  \group_begin:
  \__ebproof_pop:N \l__ebproof_a_box
  \__ebproof_pop:N \l__ebproof_b_box
  \__ebproof_overlay:NN \l__ebproof_a_box \l__ebproof_b_box
  \__ebproof_push:N \l__ebproof_a_box
  \group_end:
}
```

(*End definition for* \overlay*. This function is documented on page 4.*)

A.5.4 Deprecated statements

These statements were defined in versions 1.x of the package, they are preserved for temporary upwards compatibility and will be removed in a future version.

```
\__ebproof_new_deprecated_statement:nnnn { Alter } { m }
  { use~\token_to_str:c{rewrite}~instead } { \ebproofrewrite{ #1 \box\treebox } }
\__ebproof_new_deprecated_statement:nnnn { Delims } { }
  { use~\token_to_str:c{delims}~instead } { \ebproofdelims }
\__ebproof_new_deprecated_statement:nnnn { Ellipsis } { }
  { use~\token_to_str:c{ellipsis}~instead } { \ebproofellipsis }
\__ebproof_new_deprecated_statement:nnnn { Hypo } { }
  { use~\token_to_str:c{hypo}~instead } { \ebproofhypo }
```

```
\__ebproof_new_deprecated_statement:nnnn { Infer } { }
  { use~\token_to_str:c{infer}~instead } { \ebproofinfer }
```

A.5.5 Environment interface

The stack is initialised globally. The `prooftree` environment does not clear the stack, instead it saves the initial level in order to check that statements are properly balanced. This allows for nested uses of the environment, if it ever happens to be useful.

```
\__ebproof_clear_stack:
\tl_new:N \l__ebproof_start_level_tl
```

prooftree
prooftree*

The `prooftree` environment.

```
\NewDocumentEnvironment { prooftree } { s O{} } {
  \group_align_safe_begin:
  \keys_set_known:nnN { ebproof / proof~style } { #2 } \l_tmpa_tl
  \keys_set:nV { ebproof } \l_tmpa_tl
  \tl_set:Nx \l__ebproof_start_level_tl { \int_use:N \g__ebproof_level_int }
  \vbox_set:Nw \l_tmpa_box
  \__ebproof_setup_statements:
} {
  \vbox_set_end:
  \__ebproof_pop:N \l__ebproof_a_box
  \int_compare:nNnF { \g__ebproof_level_int } = { \tl_use:N \l__ebproof_start_level_tl } {
    \PackageError{ebproof}{Malformed~proof~tree}{
      Some~hypotheses~were~declared~but~not~used~in~this~tree.}
  }
  \IfBooleanTF { #1 } {
    \[ \box_use:c { \__ebproof_box:N \l__ebproof_a_box } \]
    \ignorespacesafterend
  } {
    \hbox_unpack:N \c_empty_box
    \bool_if:NTF \l__ebproof_center_bool {
      \hbox:n { $ \tex_vcenter:D { \box_use:c { \__ebproof_box:N \l__ebproof_a_box } } $ }
    } {
      \box_use:c { \__ebproof_box:N \l__ebproof_a_box }
    }
  }
  \group_align_safe_end:
}
```

A trick for the starred version:

```
\cs_new:cpn { prooftree* } { \prooftree* }
\cs_new:cpn { endprooftree* } { \endprooftree }
```

(*End definition for* `prooftree` *and* `prooftree*`. *These functions are documented on page* 2.)

⟨/package⟩