prooftrees

Version v0.9.3 (SVN Rev: 11666)

Clea F. Rees*

2026/02/21

Abstract

prooftrees is a LaTeX 2_ε package, based on forest, designed to support the typesetting of logical tableaux — 'proof trees' or 'truth trees' — in styles sometimes used in teaching introductory logic courses, especially those aimed at students without a strong background in mathematics. One textbook which uses proofs of this kind is Hodges (1991). Like forest, prooftrees supports memoize out-of-the-box. prooftrees uses forest-ext to support tagged PDFs out-of-the-box.

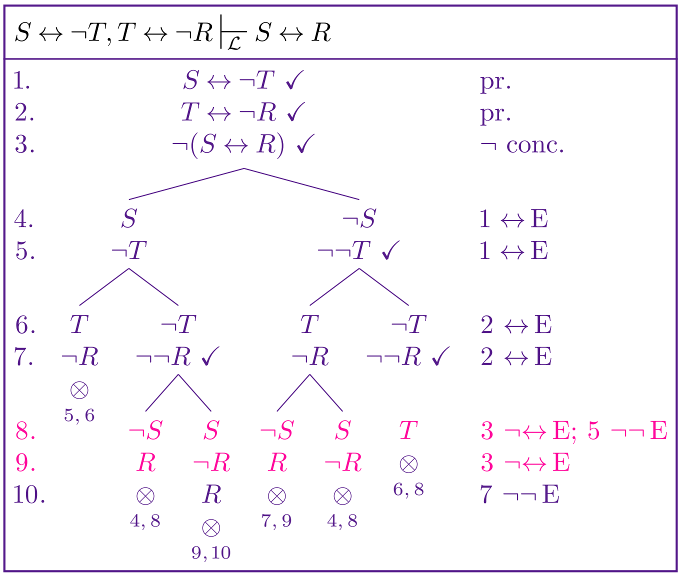
Note that this package requires version 2.1 (2016/12/04) of forest (Živanović 2016). It will not work with versions prior to 2.1.

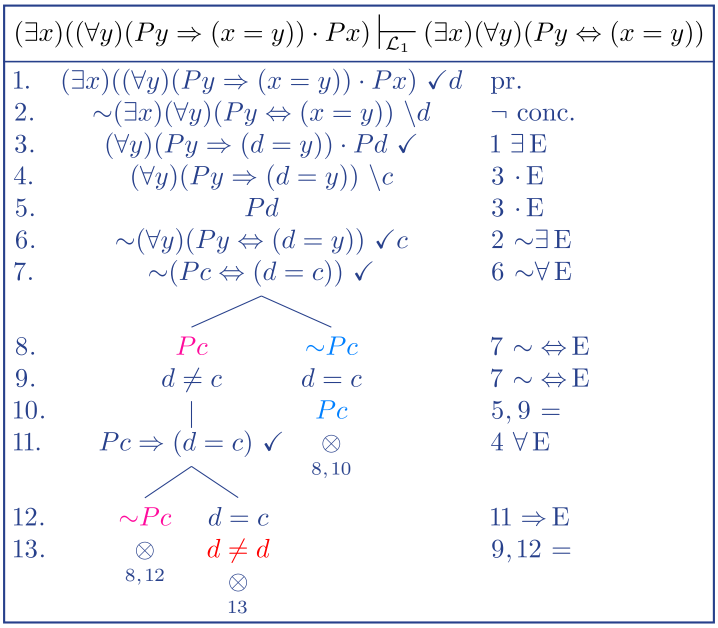
Versions 0.9.2 and later require forest-ext (Rees 2026).

I would like to thank Živanović both for developing forest and for considerable patience in answering my questions, addressing my confusions and correcting my mistakes. The many remaining errors are, of course, entirely my own. This package's deficiencies would be considerably greater and more numerous were it not for his assistance.

---

*Bug tracker: `codeberg.org/cfr/prooftrees/issues` | Code: `codeberg.org/cfr/prooftrees` | Mirror: `github.com/cfr42/prooftrees`


$S \leftrightarrow \neg T, T \leftrightarrow \neg R \vdash_{\mathcal{L}} S \leftrightarrow R$
1. $S \leftrightarrow \neg T$ ✓ pr.
2. $T \leftrightarrow \neg R$ ✓ pr.
3. $\neg(S \leftrightarrow R)$ ✓ ¬ conc.
4. S $\neg S$ 1 ↔E
5. $\neg T$ $\neg\neg T$ ✓ 1 ↔E
6. T $\neg T$ T $\neg T$ 2 ↔E
7. $\neg R$ $\neg\neg R$ ✓ $\neg R$ $\neg\neg R$ ✓ 2 ↔E
⊗ 5, 6
8. $\neg S$ S $\neg S$ S T 3 ¬↔E; 5 ¬¬E
9. R $\neg R$ R $\neg R$ ⊗ 3 ¬↔E
10. ⊗ R ⊗ ⊗ 6, 8 7 ¬¬E
4, 8 ⊗ 7, 9 4, 8
9, 10


$(\exists x)((\forall y)(Py \Rightarrow (x = y)) \cdot Px) \vdash_{\mathcal{L}_1} (\exists x)(\forall y)(Py \Leftrightarrow (x = y))$
1. $(\exists x)((\forall y)(Py \Rightarrow (x = y)) \cdot Px)$ ✓d pr.
2. $\sim(\exists x)(\forall y)(Py \Leftrightarrow (x = y))$ \ d ¬ conc.
3. $(\forall y)(Py \Rightarrow (d = y)) \cdot Pd$ ✓ 1 ∃E
4. $(\forall y)(Py \Rightarrow (d = y))$ \ c 3 ·E
5. Pd 3 ·E
6. $\sim(\forall y)(Py \Leftrightarrow (d = y))$ ✓c 2 ∼∃E
7. $\sim(Pc \Leftrightarrow (d = c))$ ✓ 6 ∼∀E
8. Pc $\sim Pc$ 7 ∼⇔E
9. $d \neq c$ $d = c$ 7 ∼⇔E
10. | Pc 5, 9 =
11. $Pc \Rightarrow (d = c)$ ✓ ⊗ 4 ∀E
8, 10
12. $\sim Pc$ $d = c$ 11 ⇒E
13. ⊗ $d \neq d$ 9, 12 =
8, 12 ⊗
13

Contents

1 Raison d'être

Suppose that we wish to typeset a typical tableau demonstrating the following entailment

$$\{P \vee (Q \vee \neg R), P \rightarrow \neg R, Q \rightarrow \neg R\} \vdash \neg R$$

We start by typesetting the tree using forest's default settings (box 1) and find our solution has several advantages: the proof is specified concisely and the code reflects the structure of the tree. It is relatively straightforward to specify a proof using forest's bracket notation, and the spacing of nodes and branches is automatically calculated.

Despite this, the results are not quite what we might have hoped for in a tableau. The assumptions should certainly be grouped more closely together and no edges (lines) should be drawn between them because these are not steps in the proof — they do not represent inferences. Preferably, edges should start from a common point in the case of branching inferences, rather than there being a gap.

Moreover, tableaux are often compacted so that *non-branching* inferences are grouped together, like assumptions, without explicitly drawn edges. Although explicit edges to represent non-branching inferences are useful when introducing students to tableaux, more complex proofs grow unwieldy and the more compact presentation becomes essential.

Furthermore, it is useful to have the option of *annotating* tableaux by numbering the lines of the proof on the left and entering the justification for each line on the right.

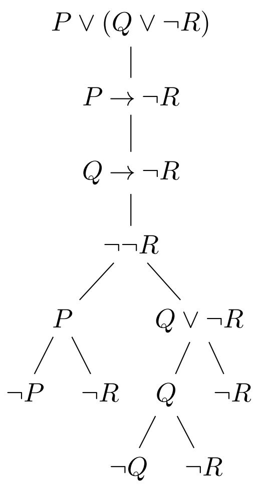
forest is a powerful and flexible package capable of all this and, indeed, a good deal more. It is not enormously difficult to customise particular trees to meet most of our desiderata. However, it is difficult to get things perfectly aligned even in simple cases, requires the insertion of 'phantom' nodes and management of several sub-trees in parallel (one for line numbers, one for the proof and one for the justifications). The process requires a good deal of manual intervention, trial-and-error and hard-coding of things it would be better to have LaTeX 2$_{\varepsilon}$ manage for us, such as keeping count of lines and line references.

prooftrees aims to make it as easy to specify tableaux as it was to specify our initial tree using forest's default settings. The package supports a small number of options which can be configured to customise the output. The code for a prooftrees tableau is shown in box 2, together with the output obtained using the default settings.

More extensive configuration can be achieved by utilising forest (Živanović 2016) and/or TikZ (Tantau 2015) directly. A sample of supported tableau styles are shown in box 3. The package is ***not*** intended for the typesetting of tableaux which differ significantly in structure.

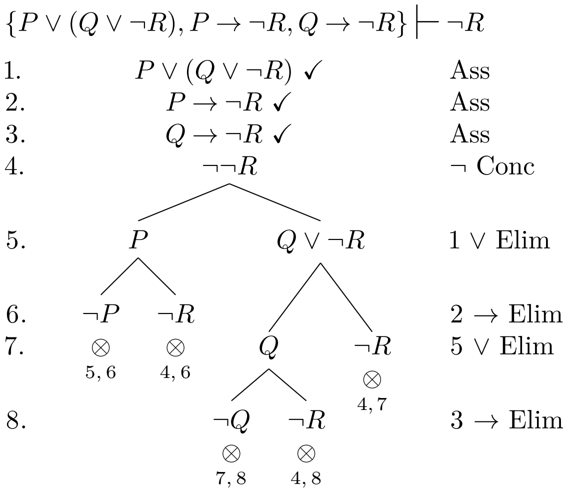
1 forest: default settings

```
\begin{forest}
  [$P \vee (Q \vee \lnot R)$
    [$P \lif \lnot R$
      [$Q \lif \lnot R$
        [$\lnot\lnot R$
          [$P$
            [$\lnot P$]
            [$\lnot R$]
          ]
          [$Q \vee \lnot R$
            [$Q$
              [$\lnot Q$]
              [$\lnot R$]
            ]
            [$\lnot R$]
          ]
        ]
      ]
    ]
  ]
\end{forest}
```



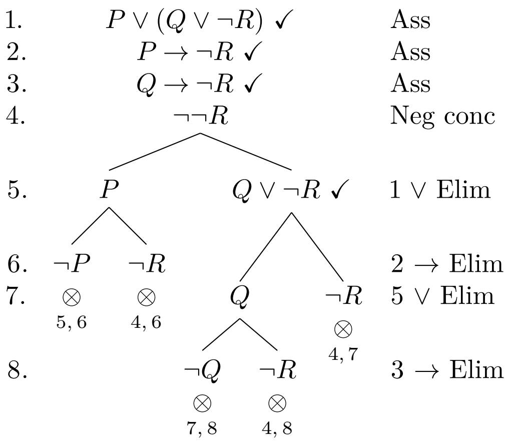
2 prooftrees: default settings

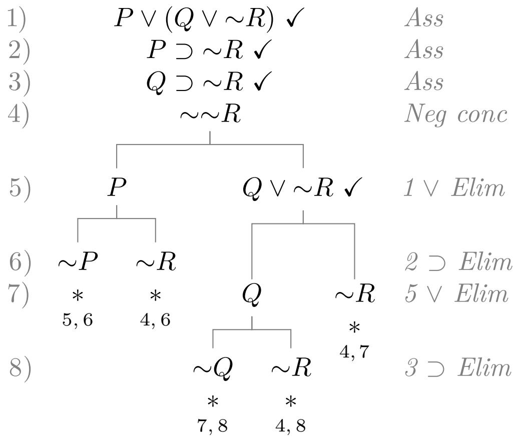
```
\begin{tableau}
  {
    to prove={\{P \vee (Q \vee \lnot R), P \lif
 \lnot R, Q \lif \lnot R\} \sststile{}{} \lnot
 R}
  }
  [P \vee (Q \vee \lnot R), just=Ass, checked
    [P \lif \lnot R, just=Ass, checked
      [Q \lif \lnot R, just=Ass, checked,
name=last premise
        [\lnot\lnot R, just={$\lnot$ Conc},
name=not conc
          [P, just={$\vee$ Elim:!uuuu}
            [\lnot P, close={:!u,!c}]
            [\lnot R, close={:not conc,!c},
just={$\lif$ Elim:!uuuu}]]
          [Q \vee \lnot R
            [Q, move by=1
              [\lnot Q, close={:!u,!c}]
              [\lnot R, close={:not conc,!c},
just={$\lif$ Elim:last premise}]]
            [\lnot R, close={:not conc,!c},
move by=1, just={$\vee$ Elim:!u}]]]]]]
\end{tableau}
```

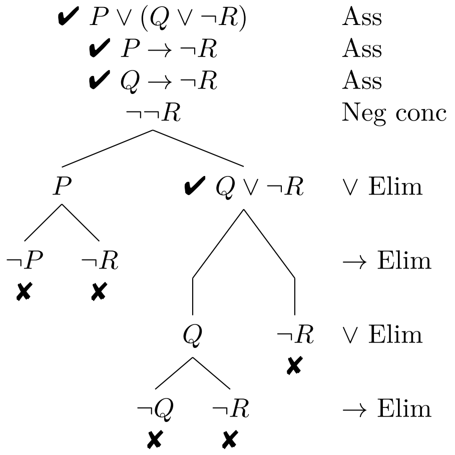

$\{P \vee (Q \vee \neg R), P \rightarrow \neg R, Q \rightarrow \neg R\} \mid\!\!- \neg R$
1. $P \vee (Q \vee \neg R)$ ✓ Ass
2. $P \rightarrow \neg R$ ✓ Ass
3. $Q \rightarrow \neg R$ ✓ Ass
4. $\neg\neg R$ ¬ Conc
5. P $Q \vee \neg R$ 1 ∨ Elim
6. $\neg P$ $\neg R$ 2 → Elim
7. Q $\neg R$ 5 ∨ Elim
⊗ 5, 6
⊗ 4, 6
⊗ 4, 7
8. $\neg Q$ $\neg R$ 3 → Elim
⊗ 7, 8
⊗ 4, 8

3 prooftrees: sample output

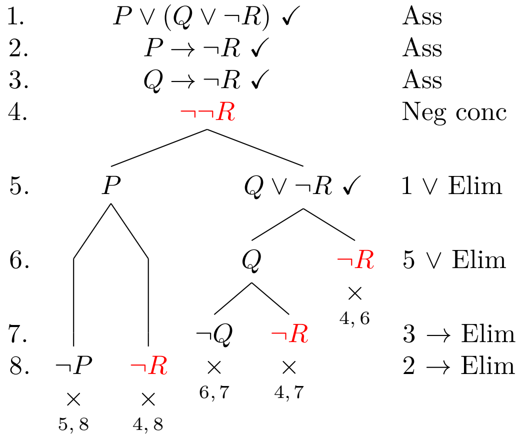
$\{P \vee (Q \vee \neg R), P \rightarrow \neg R, Q \rightarrow \neg R\} \vdash \neg R$



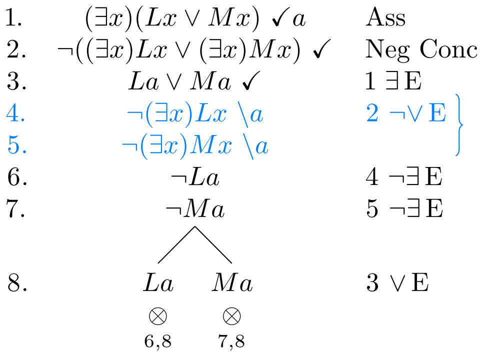


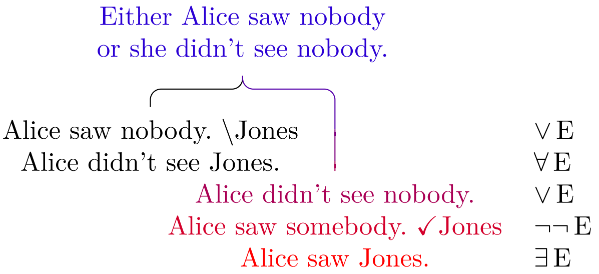


$\{P \vee (Q \vee \neg R), P \rightarrow \neg R, Q \rightarrow \neg R\} \therefore \neg R$



$(\exists x)(Lx \vee Mx) \vdash (\exists x)Lx \vee (\exists x)Mx$



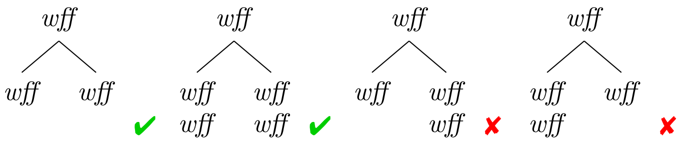


Either Alice saw nobody
or she didn't see nobody.
Alice saw nobody. \Jones ∨E
Alice didn't see Jones. ∀E
Alice didn't see nobody. ∨E
Alice saw somebody. ✓Jones ¬¬E
Alice saw Jones. ∃E

2 Assumptions & Limitations

prooftrees makes certain assumptions about the nature of the proof system, $\mathcal{L}$, on which proofs are based.

- All derivation rules yield equal numbers of *wff*s on all branches.



  If $\mathcal{L}$ fails to satisfy this condition, prooftrees is likely to violate the requirements of affected derivation rules by splitting branches 'mid-inference'.

- No derivation rule yields *wff*s on more than two branches.
- All derivation rules proceed in a downwards direction at an angle of -90° i.e. from north to south.
- Any justifications are set on the far right of the tableau.
- Any line numbers are set on the far left of the tableau.
- Justifications can refer only to earlier lines in the proof. prooftrees can typeset proofs if $\mathcal{L}$ violates this condition, but the cross-referencing system explained in section 7.2 cannot be used for affected justifications.

prooftrees does not support the automatic breaking of tableaux across pages[1]. Tableaux can be manually broken by using `line no shift` with an appropriate value for parts after the first (section 7.1). However, horizontal alignment across page breaks will not be consistent in this case.

In addition, prooftrees almost certainly relies on additional assumptions not articulated above and certainly depends on a feature of forest which its author classifies as experimental (`do dynamics`).

3 Typesetting a Tableau

After loading prooftrees in the document preamble:

```
% in document's preamble
\usepackage{prooftrees}
```

the `prooftree` environment is available for typesetting tableaux. This takes an argument used to specify a ⟨*tree preamble*⟩, with the body of the environment consisting of a ⟨*tree specification*⟩ in forest's notation. The ⟨*tree preamble*⟩ can be as simple as an empty argument — {} — or much more complex.

Customisation options and further details concerning loading and invocation are explained in section 4, section 5, section 6, section 7 and section 8. In this section, we begin by looking at a simple example using the default settings.

Suppose that we wish to typeset the tableau for

$$(\exists x)((\forall y)(Py \rightarrow x = y) \land Px) \vdash (\exists x)(\forall y)(Py \leftrightarrow x = y)$$

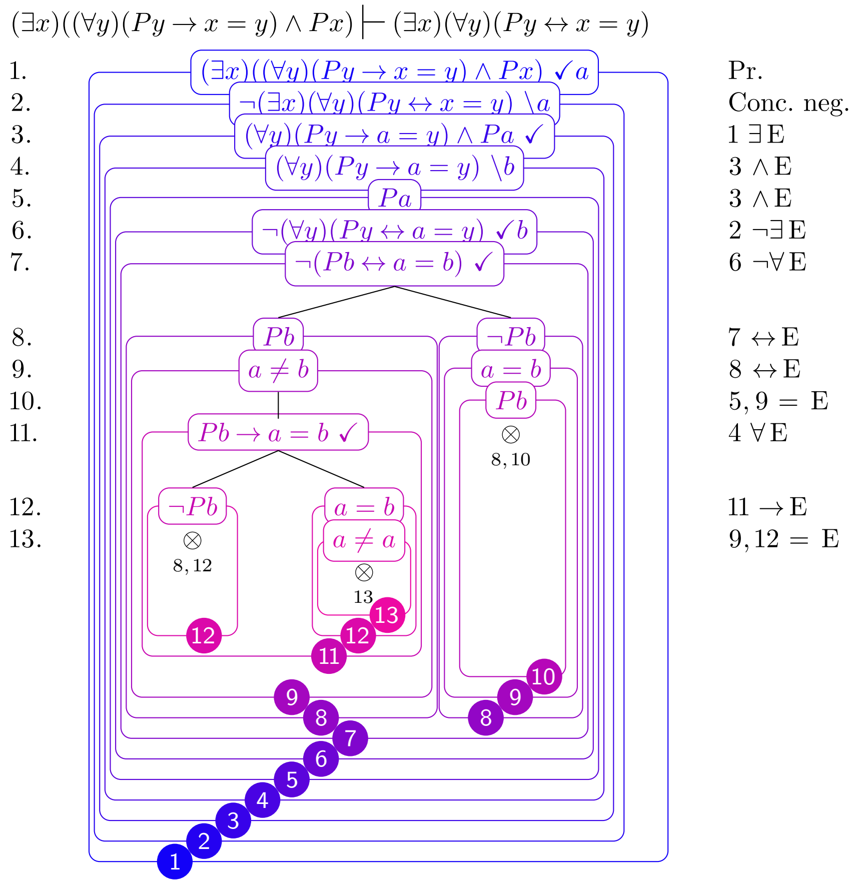
and we would like to typeset the entailment established by our proof at the top of the tree. Then we should begin like this:

```
\begin{tableau}
  {
    to prove={(\exists x)((\forall y)(Py \lif x = y) \land Px) \sststile{}{} (\exists x)(\forall y)(
Py \liff x = y)}
  }
\end{tableau}
```

---

[1] It is possible to persuade prooftrees to do this automatically or semi-automatically. However, the code is not in a state I would wish to inflict on an unsuspecting public. The perilously inquisitive may search TeX Stack Exchange at their own risk.

4 Nested structure of tableau


(∃x)((∀y)(Py → x = y) ∧ Px) ⊢ (∃x)(∀y)(Py ↔ x = y)
1. (∃x)((∀y)(Py → x = y) ∧ Px) ✓a Pr.
2. ¬(∃x)(∀y)(Py ↔ x = y) \a Conc. neg.
3. (∀y)(Py → a = y) ∧ Pa ✓ 1 ∃E
4. (∀y)(Py → a = y) \b 3 ∧E
5. Pa 3 ∧E
6. ¬(∀y)(Py ↔ a = y) ✓b 2 ¬∃E
7. ¬(Pb ↔ a = b) ✓ 6 ¬∀E
8. Pb ¬Pb 7 ↔E
9. a ≠ b a = b 8 ↔E
10. Pb 5, 9 = E
11. Pb → a = b ✓ 4 ∀E
12. ¬Pb a = b 11 →E
13. a ≠ a 9, 12 = E
⊗ 8, 12
⊗ 13
⊗ 8, 10
1 2 3 4 5 6 7 8 9 10 11 12 13

That is all the preamble we want, so we move onto consider the ⟨*tree specification*⟩. forest uses square brackets to specify trees' structures. To typeset a proof, think of it as consisting of nested trees, trunks upwards, and work from the outside in and the trunks down (box 4).

Starting with the outermost tree ① and the topmost trunk, we replace the ▢ with square brackets and enter the first *wff* inside, adding `just=Pr.` for the justification on the right and `checked=a` so that the line will be marked as discharged with a substituted for x. We also use forest's `name` to label the line for ease of reference later. (Technically, it is the node rather than the line which is named, but, for our purposes, this doesn't matter. forest will create a name if we don't specify one, but it will not necessarily be one we would have chosen for ease of use!)

```
\begin{tableau}
  {
    to prove={(\exists x)((\forall y)(Py \lif x = y) \land Px) \sststile{}{} (\exists x)(\forall y)(
Py \liff x = y)}
  }
  [{(\exists x)((\forall y)(Py \lif x = y) \land Px)}, checked=a, just=Pr., name=pr
  ]
\end{tableau}
```

We can refer to this line later as `pr`.

We then consider the next tree ② . Its ▢ goes inside that for ① , so the square brackets containing the next *wff* go inside those we used for ① . Again, we add the justification with `just`, but we use `subs=a` rather than `checked=a` as we want to mark substitution of a for x without discharging the line. Again, we use `name` so that we can refer to the line later as `neg conc`.

```
\begin{tableau}
  {
    to prove={(\exists x)((\forall y)(Py \lif x = y) \land Px) \sststile{}{} (\exists x)(\forall y)(
Py \liff x = y)}
  }
  [{(\exists x)((\forall y)(Py \lif x = y) \land Px)}, checked=a, just=Pr., name=pr
    [{\lnot (\exists x)(\forall y)(Py \liff x = y)}, subs=a, just=Conc.~neg., name=neg conc
    ]
  ]
\end{tableau}
```

Turning to tree ③, we again note that its ▭ is nested within the previous two, so the square brackets for its *wff* need to be nested within those for the previous *wff*s. This time, we want to mark the line as discharged without substitution, so we simply use `checked` without a value. Since the justification for this line includes mathematics, we need to ensure that the relevant part of the justification is surrounded by `$...$` or `\(...\)`. This justification also refers to an earlier line in the proof. We could write this as `just=1 $\exists\elim$`, but instead we use the name we assigned earlier with the referencing feature provided by prooftrees. To do this, we put the reference, `pr` *after* the rest of the justification, separating the two parts by a colon i.e. `$\exists\elim$:pr` and allow prooftrees to figure out the correct number.

```
\begin{tableau}
  {
    to prove={(\exists x)((\forall y)(Py \lif x = y) \land Px) \sststile{}{} (\exists x)(\forall y)(
Py \liff x = y)}
  }
  [{(\exists x)((\forall y)(Py \lif x = y) \land Px)}, checked=a, just=Pr., name=pr
    [{\lnot (\exists x)(\forall y)(Py \liff x = y)}, subs=a, just=Conc.~neg., name=neg conc
      [{(\forall y)(Py \lif a = y) \land Pa}, checked, just=$\exists\elim$:pr
      ]
    ]
  ]
\end{tableau}
```

Continuing in the same way, we surround each of the *wff*s for ④, ⑤, ⑥ and ⑦ within square brackets nested within those surrounding the previous *wff* since each of the trees is nested within the previous one. Where necessary, we use `name` to label lines we wish to refer to later, but we also use forest's *relative* naming system when this seems easier. For example, in the next line we add, we specify the justification as `just=$\land\elim$:!u`. `!` tells forest that the reference specifies a relationship between the current line and the referenced one, rather than referring to the other line by name. `!u` refers to the current line's parent line — in this case, `{(\forall y)(Py \lif a = y) \land Pa}, checked, just=$\exists\elim$:pr`. `!uu` refers to the current line's parent line's parent line and so on.

```
\begin{tableau}
  {
    to prove={(\exists x)((\forall y)(Py \lif x = y) \land Px) \sststile{}{} (\exists x)(\forall y)(
Py \liff x = y)}
  }
  [{(\exists x)((\forall y)(Py \lif x = y) \land Px)}, checked=a, just=Pr., name=pr
    [{\lnot (\exists x)(\forall y)(Py \liff x = y)}, subs=a, just=Conc.~neg., name=neg conc
      [{(\forall y)(Py \lif a = y) \land Pa}, checked, just=$\exists\elim$:pr
        [{(\forall y)(Py \lif a = y)}, subs=b, just=$\land\elim$:!u, name=mark
          [Pa, just=$\land\elim$:!uu, name=simple
            [{\lnot (\forall y)(Py \liff a = y)}, checked=b, just=$\lnot\exists\elim$:neg conc
              [{\lnot (Pb \liff a = b)}, checked, just=$\lnot\forall\elim$:!u
              ]
            ]
          ]
        ]
```

```
   ]
  ]
 ]
\end{tableau}
```

Reaching ⑧ , things get a little more complex since we now have not one, but *two* ▭ nested within ⑦ . This means that we need *two* sets of square brackets for ⑧ — one for each of its two trees. Again, both of these should be nested within the square brackets for ⑦ but neither should be nested within the other because the trees for the two branches at ⑧ are distinct.

```
\begin{tableau}
 {
  to prove={(\exists x)((\forall y)(Py \lif x = y) \land Px) \sststile{}{} (\exists x)(\forall y)(
Py \liff x = y)}
 }
 [{(\exists x)((\forall y)(Py \lif x = y) \land Px)}, checked=a, just=Pr., name=pr
  [{\lnot (\exists x)(\forall y)(Py \liff x = y)}, subs=a, just=Conc.~neg., name=neg conc
   [{(\forall y)(Py \lif a = y) \land Pa}, checked, just=$\exists\elim$:pr
    [{(\forall y)(Py \lif a = y)}, subs=b, just=$\land\elim$:!u, name=mark
     [Pa, just=$\land\elim$:!uu, name=simple
      [{\lnot (\forall y)(Py \liff a = y)}, checked=b, just=$\lnot\exists\elim$:neg conc
       [{\lnot (Pb \liff a = b)}, checked, just=$\lnot\forall\elim$:!u
        [Pb, just=$\liff\elim$:!u, name=to Pb or not to Pb
        ]
        [\lnot Pb
        ]
       ]
      ]
     ]
    ]
   ]
  ]
 ]
\end{tableau}
```

At this point, we need to work separately or in parallel on each of our two branches since each constitutes its own tree. Turning to trees ⑨ , each needs to be nested within the relevant tree ⑧ , since each ▭ is nested within the applicable branch's tree. Hence, we nest square brackets for each of the *wffs* at ⑨ within the previous set.

```
\begin{tableau}
 {
  to prove={(\exists x)((\forall y)(Py \lif x = y) \land Px) \sststile{}{} (\exists x)(\forall y)(
Py \liff x = y)}
 }
 [{(\exists x)((\forall y)(Py \lif x = y) \land Px)}, checked=a, just=Pr., name=pr
  [{\lnot (\exists x)(\forall y)(Py \liff x = y)}, subs=a, just=Conc.~neg., name=neg conc
   [{(\forall y)(Py \lif a = y) \land Pa}, checked, just=$\exists\elim$:pr
    [{(\forall y)(Py \lif a = y)}, subs=b, just=$\land\elim$:!u, name=mark
     [Pa, just=$\land\elim$:!uu, name=simple
      [{\lnot (\forall y)(Py \liff a = y)}, checked=b, just=$\lnot\exists\elim$:neg conc
       [{\lnot (Pb \liff a = b)}, checked, just=$\lnot\forall\elim$:!u
        [Pb, just=$\liff\elim$:!u, name=to Pb or not to Pb
         [a \neq b, just=$\liff\elim$:!u
         ]
        ]
        [\lnot Pb
         [{a = b}
         ]
```

```
                ]
              ]
            ]
          ]
        ]
      ]
    ]
  ]
\end{tableau}
```

We only have one tree ⑩ as there is no corresponding tree in the left-hand branch. This isn't a problem: we just need to ensure that we nest it within the appropriate tree ⑨. There are two additional complications here. The first is that the justification contains a comma, so we need to surround the argument we give `just` with curly brackets. That is, we must write `just={5,9 $=\elim$}` or `just={$=\elim$:{simple,!u}}`. The second is that we wish to close this branch with an indication of the line numbers containing inconsistent *wffs*. We can use `close={8,10}` for this or we can use the same referencing system we used to reference lines when specifying justifications and write `close={:to Pb or not to Pb,!c}`. In either case, we again surrounding the argument with curly brackets to protect the comma. `!c` refers to the current line — something useful in many close annotations, but not helpful in specifying non-circular justifications.

```
\begin{tableau}
  {
    to prove={(\exists x)((\forall y)(Py \lif x = y) \land Px) \sststile{}{} (\exists x)(\forall y)(
Py \liff x = y)}
  }
  [{(\exists x)((\forall y)(Py \lif x = y) \land Px)}, checked=a, just=Pr., name=pr
    [{\lnot (\exists x)(\forall y)(Py \liff x = y)}, subs=a, just=Conc.~neg., name=neg conc
      [{(\forall y)(Py \lif a = y) \land Pa}, checked, just=$\exists\elim$:pr
        [{(\forall y)(Py \lif a = y)}, subs=b, just=$\land\elim$:!u, name=mark
          [Pa, just=$\land\elim$:!uu, name=simple
            [{\lnot (\forall y)(Py \liff a = y)}, checked=b, just=$\lnot\exists\elim$:neg conc
              [{\lnot (Pb \liff a = b)}, checked, just=$\lnot\forall\elim$:!u
                [Pb, just=$\liff\elim$:!u, name=to Pb or not to Pb
                 [a \neq b, just=$\liff\elim$:!u
                 ]
                ]
                [\lnot Pb
                 [{a = b}
                    [Pb, just={$=\elim$:{simple,!u}}, close={:to Pb or not to Pb,!c}
                    ]
                 ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
  ]
\end{tableau}
```

This completes the main right-hand branch of the tree and we can focus solely on the remaining left-hand one. Tree ⑪ is straightforward — we just need to nest it within the left-hand tree ⑨.

```
\begin{tableau}
  {
    to prove={(\exists x)((\forall y)(Py \lif x = y) \land Px) \sststile{}{} (\exists x)(\forall y)(
Py \liff x = y)}
  }
  [{(\exists x)((\forall y)(Py \lif x = y) \land Px)}, checked=a, just=Pr., name=pr
```

```
    [{\lnot (\exists x)(\forall y)(Py \liff x = y)}, subs=a, just=Conc.~neg., name=neg conc
      [{(\forall y)(Py \lif a = y) \land Pa}, checked, just=$\exists\elim$:pr
        [{(\forall y)(Py \lif a = y)}, subs=b, just=$\land\elim$:!u, name=mark
          [Pa, just=$\land\elim$:!uu, name=simple
            [{\lnot (\forall y)(Py \liff a = y)}, checked=b, just=$\lnot\exists\elim$:neg conc
              [{\lnot (Pb \liff a = b)}, checked, just=$\lnot\forall\elim$:!u
                [Pb, just=$\liff\elim$:!u, name=to Pb or not to Pb
                 [a \neq b, just=$\liff\elim$:!u
                  [{Pb \lif a = b}, checked, just=$\forall\elim$:mark%, move by=1
                  ]
                ]
                ]
                [\lnot Pb
                 [{a = b}
                    [Pb, just={$=\elim$:{simple,!u}}, close={:to Pb or not to Pb,!c}
                    ]
                 ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
  ]
\end{tableau}
```

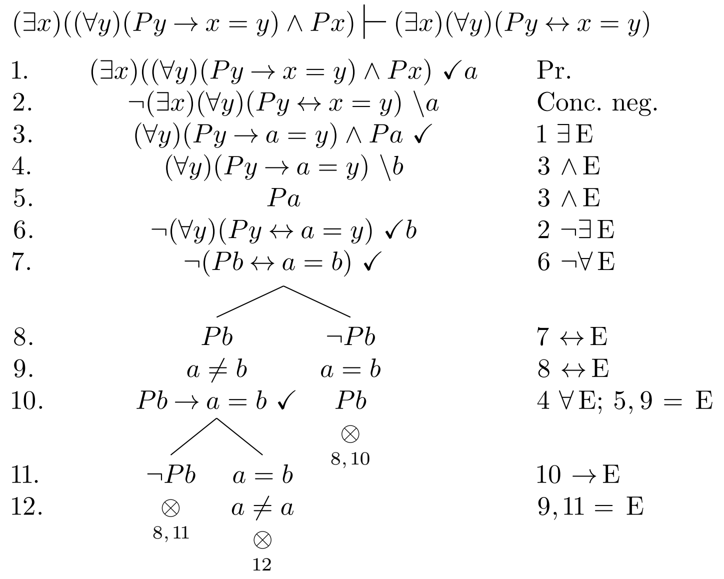
At this point, the main left-hand branch itself branches, so we have two trees ⑫. Treating this in the same way as the earlier branch at ⑧, we use two sets of square brackets nested within those for tree ⑫, but with neither nested within the other. Since we also want to mark the leftmost branch as closed, we add `close={:to Pb or not to Pb,!c}` in the same way as before.

```
\begin{tableau}
  {
    to prove={(\exists x)((\forall y)(Py \lif x = y) \land Px) \sststile{}{} (\exists x)(\forall y)(
Py \liff x = y)}
  }
  [{(\exists x)((\forall y)(Py \lif x = y) \land Px)}, checked=a, just=Pr., name=pr
    [{\lnot (\exists x)(\forall y)(Py \liff x = y)}, subs=a, just=Conc.~neg., name=neg conc
      [{(\forall y)(Py \lif a = y) \land Pa}, checked, just=$\exists\elim$:pr
        [{(\forall y)(Py \lif a = y)}, subs=b, just=$\land\elim$:!u, name=mark
          [Pa, just=$\land\elim$:!uu, name=simple
            [{\lnot (\forall y)(Py \liff a = y)}, checked=b, just=$\lnot\exists\elim$:neg conc
              [{\lnot (Pb \liff a = b)}, checked, just=$\lnot\forall\elim$:!u
                [Pb, just=$\liff\elim$:!u, name=to Pb or not to Pb
                 [a \neq b, just=$\liff\elim$:!u
                  [{Pb \lif a = b}, checked, just=4 $\forall\elim$
                      [\lnot Pb, close={:to Pb or not to Pb,!c}, just=$\lif\elim$:!u
                      ]
                      [{a = b}
                      ]
                  ]
                ]
                ]
                [\lnot Pb
                 [{a = b}
                    [Pb, just={$=\elim$:{simple,!u}}, close={:to Pb or not to Pb,!c}
                    ]
                 ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
  ]
\end{tableau}
```

We complete our initial specification by nesting ⑬ within the appropriate tree ⑫, again marking closure appropriately.

```
\begin{tableau}
  {
    to prove={(\exists x)((\forall y)(Py \lif x = y) \land Px) \sststile{}{} (\exists x)(\forall y)(
Py \liff x = y)}
  }
  [{(\exists x)((\forall y)(Py \lif x = y) \land Px)}, checked=a, just=Pr., name=pr
    [{\lnot (\exists x)(\forall y)(Py \liff x = y)}, subs=a, just=Conc.~neg., name=neg conc
      [{(\forall y)(Py \lif a = y) \land Pa}, checked, just=$\exists\elim$:pr
        [{(\forall y)(Py \lif a = y)}, subs=b, just=$\land\elim$:!u, name=mark
          [Pa, just=$\land\elim$:!uu, name=simple
            [{\lnot (\forall y)(Py \liff a = y)}, checked=b, just=$\lnot\exists\elim$:neg conc
              [{\lnot (Pb \liff a = b)}, checked, just=$\lnot\forall\elim$:!u
                [Pb, just=$\liff\elim$:!u, name=to Pb or not to Pb
                 [a \neq b, just=$\liff\elim$:!u
                  [{Pb \lif a = b}, checked, just=4 $\forall\elim$
                      [\lnot Pb, close={:to Pb or not to Pb,!c}, just=$\lif\elim$:!u
                      ]
                      [{a = b}
                        [a \neq a, close={:!c}, just={$=\elim$:{!uuu,!u}}
                        ]
                      ]
                  ]
                ]
                ]
                [\lnot Pb
                 [{a = b}
                    [Pb, just={$=\elim$:{simple,!u}}, close={:to Pb or not to Pb,!c}
                    ]
                 ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
  ]
\end{tableau}
```

Compiling our code, we find that the line numbering is not quite right:


$(\exists x)((\forall y)(Py \rightarrow x = y) \land Px) \vdash (\exists x)(\forall y)(Py \leftrightarrow x = y)$
1. $(\exists x)((\forall y)(Py \rightarrow x = y) \land Px)$ ✓a Pr.
2. $\neg(\exists x)(\forall y)(Py \leftrightarrow x = y)$ \$a$ Conc. neg.
3. $(\forall y)(Py \rightarrow a = y) \land Pa$ ✓ 1 $\exists$E
4. $(\forall y)(Py \rightarrow a = y)$ \$b$ 3 $\land$E
5. Pa 3 $\land$E
6. $\neg(\forall y)(Py \leftrightarrow a = y)$ ✓b 2 $\neg\exists$E
7. $\neg(Pb \leftrightarrow a = b)$ ✓ 6 $\neg\forall$E
8. Pb $\neg Pb$ 7 $\leftrightarrow$E
9. $a \neq b$ $a = b$ 8 $\leftrightarrow$E
10. $Pb \rightarrow a = b$ ✓ Pb 4 $\forall$E; 5, 9 = E
⊗ 8, 10
11. $\neg Pb$ $a = b$ 10 $\rightarrow$E
12. ⊗ 8, 11 $a \neq a$ 9, 11 = E
⊗ 12

prooftrees warns us about this:

```
Package prooftrees Warning: Merging conflicting justifications for line 10! Please examine the output
 carefully and use "move by" to move lines later in the proof if required. Details of how to do this
are included in the documentation.
```

We would like line 10 in the left-hand branch to be moved down by one line, so we add `move by=1` to the relevant line of our proof. That is, we replace the line

```
                  [{Pb \lif a = b}, checked, just=4 $\forall\elim$
```

by

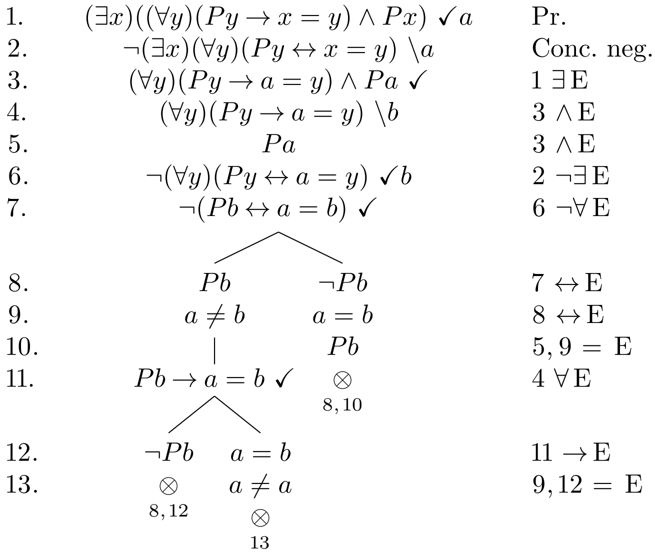
```
                  [{Pb \lif a = b}, checked, just=$\forall\elim$:mark, move by=1
```

giving us the following code:

```
\begin{tableau}
  {
    to prove={(\exists x)((\forall y)(Py \lif x = y) \land Px) \sststile{}{} (\exists x)(\forall y)(
Py \liff x = y)}
  }
  [{(\exists x)((\forall y)(Py \lif x = y) \land Px)}, checked=a, just=Pr., name=pr
    [{\lnot (\exists x)(\forall y)(Py \liff x = y)}, subs=a, just=Conc.~neg., name=neg conc
      [{(\forall y)(Py \lif a = y) \land Pa}, checked, just=$\exists\elim$:pr
        [{(\forall y)(Py \lif a = y)}, subs=b, just=$\land\elim$:!u, name=mark
          [Pa, just=$\land\elim$:!uu, name=simple
            [{\lnot (\forall y)(Py \liff a = y)}, checked=b, just=$\lnot\exists\elim$:neg conc
              [{\lnot (Pb \liff a = b)}, checked, just=$\lnot\forall\elim$:!u
                [Pb, just=$\liff\elim$:!u, name=to Pb or not to Pb
                 [a \neq b, just=$\liff\elim$:!u
                  [{Pb \lif a = b}, checked, just=$\forall\elim$:mark, move by=1
                      [\lnot Pb, close={:to Pb or not to Pb,!c}, just=$\lif\elim$:!u
                      ]
                      [{a = b}
                        [a \neq a, close={:!c}, just={$=\elim$:{!uuu,!u}}
                        ]
                      ]
                  ]
                ]
                ]
                [\lnot Pb
                 [{a = b}
                    [Pb, just={$=\elim$:{simple,!u}}, close={:to Pb or not to Pb,!c}
                    ]
                 ]
                ]
              ]
            ]
          ]
        ]
      ]
    ]
  ]
\end{tableau}
```

which produces our desired result:

$$(\exists x)((\forall y)(Py \to x = y) \wedge Px) \vdash (\exists x)(\forall y)(Py \leftrightarrow x = y)$$


1. $(\exists x)((\forall y)(Py \to x = y) \wedge Px)$ ✓a — Pr.
2. $\neg(\exists x)(\forall y)(Py \leftrightarrow x = y)$ \$a$ — Conc. neg.
3. $(\forall y)(Py \to a = y) \wedge Pa$ ✓ — 1 $\exists$E
4. $(\forall y)(Py \to a = y)$ \$b$ — 3 $\wedge$E
5. Pa — 3 $\wedge$E
6. $\neg(\forall y)(Py \leftrightarrow a = y)$ ✓b — 2 $\neg\exists$E
7. $\neg(Pb \leftrightarrow a = b)$ ✓ — 6 $\neg\forall$E
8. Pb | $\neg Pb$ — 7 $\leftrightarrow$E
9. $a \neq b$ | $a = b$ — 8 $\leftrightarrow$E
10. | Pb — 5, 9 = E
11. $Pb \to a = b$ ✓ | $\otimes$ 8, 10 — 4 $\forall$E
12. $\neg Pb$ | $a = b$ — 11 $\to$E
13. $\otimes$ 8, 12 | $a \neq a$ $\otimes$ 13 — 9, 12 = E

4 Loading the Package

To load the package simply add the following to your document's preamble.

```
\usepackage{prooftrees}
```

prooftrees will load forest automatically.

The only option currently supported is `tableaux`. If this option is specified, the `prooftree` environment will be called `tableau` instead.

Example: `\usepackage[tableaux]prooftrees`

would cause the `tableau` environment to be defined *rather than* `prooftree`.

Any other options given will be passed to forest.

Example: `\usepackage[debug]prooftrees`

would enable forest's debugging.

If one or more of forest's libraries are to be loaded, it is recommended that these be loaded separately and their defaults applied, if applicable, within a local TeX group so that they do not interfere with prooftrees's environment.

5 Invocation

`prooftree`
environment

`\begin{prooftree}{`⟨*tree preamble*⟩`}`⟨*tree specification*⟩`\end{prooftree}`

The ⟨*tree preamble*⟩ is used to specify any non-default options which should be applied to the tree. It may contain any code valid in the preamble of a regular forest tree, in addition to setting prooftree options. The preamble may be empty, but the argument is *required*[2]. The ⟨*tree specification*⟩ specifies the tree in the bracket notation parsed by forest.

Users of forest should note that the environments prooftree and forest differ in important ways.

- *`prooftree`'s argument is* mandatory.
- *The tree's preamble* cannot *be given in the body of the environment.*
- *`\end{prooftree}`* must *follow the* ⟨*tree specification*⟩ immediately.

`tableau`
environment

`\begin{tableau}{`⟨*tree preamble*⟩`}`⟨*tree specification*⟩`\end{tableau}`

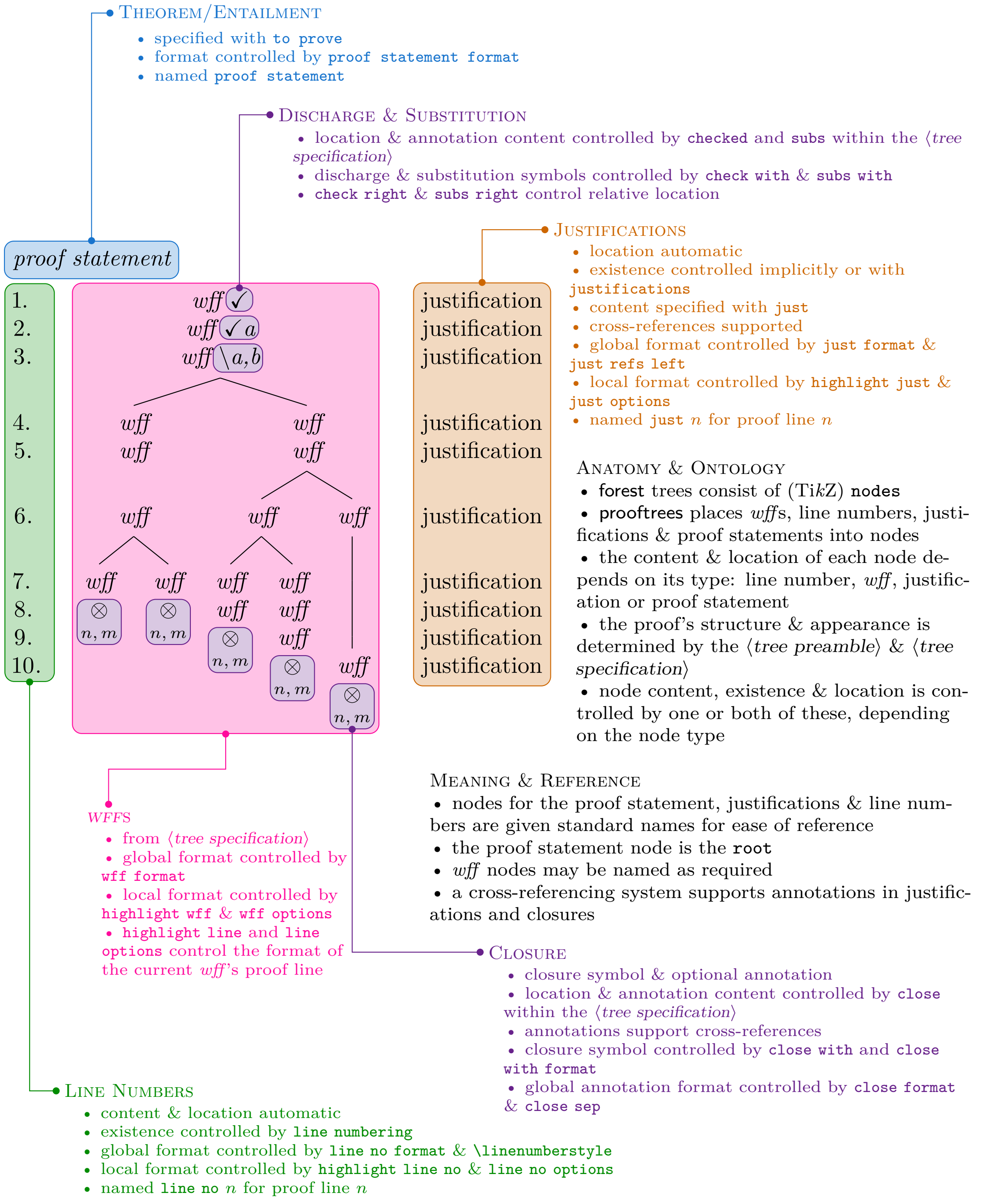
A substitute for `prooftee`, defined *instead* of `prooftree` if the package option `tableaux` is specified or a `\prooftree` macro is already defined when prooftrees is loaded. See section 4 for details and section 14 for this option's raison d'être.

6 Tableau Anatomy

The following diagram provides an overview of the configuration and anatomy of a prooftrees proof tree. Detailed documentation is provided in section 7 and section 8.

---

[2] Failure to specify a required argument does not always yield a compilation error in the case of environments. However, failure to specify required arguments to environments often fails to achieve the best consequences, even when it does not result in compilation failures, and will, therefore, be avoided by the prudent.



Theorem/Entailment
- specified with `to prove`
- format controlled by `proof statement format`
- named `proof statement`

Discharge & Substitution
- location & annotation content controlled by `checked` and `subs` within the ⟨*tree specification*⟩
- discharge & substitution symbols controlled by `check with` & `subs with`
- `check right` & `subs right` control relative location

Justifications
- location automatic
- existence controlled implicitly or with `justifications`
- content specified with `just`
- cross-references supported
- global format controlled by `just format` & `just refs left`
- local format controlled by `highlight just` & `just options`
- named `just` n for proof line n

Anatomy & Ontology
- forest trees consist of (Ti*k*Z) `nodes`
- prooftrees places *wff*s, line numbers, justifications & proof statements into nodes
- the content & location of each node depends on its type: line number, *wff*, justification or proof statement
- the proof's structure & appearance is determined by the ⟨*tree preamble*⟩ & ⟨*tree specification*⟩
- node content, existence & location is controlled by one or both of these, depending on the node type

Meaning & Reference
- nodes for the proof statement, justifications & line numbers are given standard names for ease of reference
- the proof statement node is the `root`
- *wff* nodes may be named as required
- a cross-referencing system supports annotations in justifications and closures

WFFs
- from ⟨*tree specification*⟩
- global format controlled by `wff format`
- local format controlled by `highlight wff` & `wff options`
- `highlight line` and `line options` control the format of the current *wff*'s proof line

Closure
- closure symbol & optional annotation
- location & annotation content controlled by `close` within the ⟨*tree specification*⟩
- annotations support cross-references
- closure symbol controlled by `close with` and `close with format`
- global annotation format controlled by `close format` & `close sep`

Line Numbers
- content & location automatic
- existence controlled by `line numbering`
- global format controlled by `line no format` & `\linenumberstyle`
- local format controlled by `highlight line no` & `line no options`
- named `line no` n for proof line n

7 Options

Most configuration uses the standard key/value interface provided by Ti*k*Z and extended by forest. These are divided into those which determine the overall appearance of the proof as a whole and those with more local effects. See section 10 for advanced customisation.

7.1 Global Options

The following options affect the global style of the tree and should typically be set in the tree's preamble if non-default values are desired. The default values for the document can be set outside the `prooftree` environment using `\forestset{`⟨*settings*⟩`}`. If *only* tableaux will be typeset, a default style can be configured using forest's `default preamble`.

`auto move`
`not auto move`
Forest boolean register

`= true|false`

Default: `true`

Determines whether prooftrees will move lines automatically, where possible, to avoid combining different justifications when different branches are treated differently. The default is to avoid conflicts automatically where possible. Turning this off permits finer-grained control of what gets moved using `move by`. The following are equivalent to the default setting:

```
auto move
auto move=true
```

Either of the following will turn auto move off:

```
not auto move
auto move=false
```

`line numbering`
`not line numbering`
Forest boolean register

`= true|false`

Default: `true`

This determines whether lines should be numbered. The default is to number lines. The following are equivalent to the default setting:

```
line numbering
line numbering=true
```

Either of the following will turn line numbering off:

```
not line numbering
line numbering=false
```

`justifications`
`not justifications`
Forest boolean register

`= true|false`

This determines whether justifications for lines of the proof should be typeset to the right of the tree. It is rarely necessary to set this option explicitly as it will be automatically enabled if required. The only exception concerns a proof for which a line should be moved but no justifications are specified. In this case either of the following should be used to activate the option:

```
justifications
justifications=true
```

This is not necessary if `just` is used for any line of the proof.

`single branches`
`not single branches`
Forest boolean register

`= true|false`

Default: `false`

This determines whether inference steps which do not result in at least two branches should draw and explicit branch. The default is to not draw single branches explicitly. The following are equivalent to the default setting:

```
not single branches
single branches=false
```

Either of the following will turn line numbering off:

```
single branches
single branches=true
```

7.1.1 Dimensions

`line no width` = ⟨*dimension*⟩
Forest dimension register

The maximum width of line numbers. By default, this is set to the width of the formatted line number `99`.

Example: `line no width=20pt`

`just sep` = ⟨*dimension*⟩
Forest dimension register

Default: `1.5em`

Amount by which to shift justifications away from the tree. A larger value will shift the justifications further to the right, increasing their distance from the tree, while a smaller one will decrease this distance. Note that a negative value ought never be given. Although this will not cause an error, it may result in strange things happening. If you wish to decrease the distance between the tree and the justifications further, please set `just sep` to zero and use the options provided by forest and/or Ti*k*Z to make further negative adjustments.

Example: `just sep=.5em`

`line no sep` = ⟨*dimension*⟩
Forest dimension register

Default: `1.5em`

Amount by which to shift line numbers away from the tree. A larger value will shift the line numbers further to the left, increasing their distance from the tree, while a smaller one will decrease this distance. Note that a negative value ought never be given. Although this will not cause an error, it may result in strange things happening. If you wish to decrease the distance between the tree and the line numbers further, please set `line no sep` to zero and use the options provided by forest and/or Ti*k*Z to make further negative adjustments.

Example: `line no sep=5pt`

`close sep` = ⟨*dimension*⟩
Forest dimension register

Default: `.75\baselineskip`

Distance between the symbol marking branch closure and any following annotation. If the format of such annotations is changed with `close format`, this dimension may require adjustment.

Example: `close sep=\baselineskip`

`proof tree inner proof width` = ⟨*dimension*⟩
Forest dimension register

Default: `0pt`

`proof tree inner proof midpoint` = ⟨*dimension*⟩
Forest dimension register

Default: `0pt`

7.1.2 Line Numbers

`line no shift` = ⟨*integer*⟩
Forest count register

Default: `0`

This value increments or decrements the number used for the first line of the proof. By default, line numbering starts at `1`.

Example: `line no shift=3`

would begin numbering the lines at 4.

`zero start`
Forest style

Start line numbering from 0 rather than 1. The following are equivalent:

```
zero start
line no shift=-1
```

7.1.3 Proof Statement

`to prove`
Forest style

= ⟨*wff*⟩

Statement of theorem or entailment to be typeset above the proof. In many cases, it will be necessary to enclose the statement in curly brackets.

Example: `to prove={\sststile{}{} P \lif P}`

By default, the content is expected to be suitable for typesetting in maths mode and should *not*, therefore, be enclosed by dollar signs or equivalent.

7.1.4 Format

`check with`
Forest toks register

= ⟨*symbol*⟩

Default: `\ensuremath{\checkmark}` (✓)

Symbol with which to mark discharged lines.

Example: `check with={\text{\ding{52}}}`

Within the tree, `checked` is used to identify discharged lines.

`check right`
`not check right`
Forest boolean register

= `true|false`

Default: `true`

Determines whether the symbol indicating that a line is discharged should be placed to the right of the *wff*. The alternative is, unsurprisingly, to place it to the left of the *wff*. The following are equivalent to the default setting:

```
check right
check right=true
```

`check left`
Forest style

Set `check right=false`. The following are equivalent ways to place the markers to the left:

```
check right=false
not check right
check left
```

`close with`
Forest toks register

= ⟨*symbol*⟩

Default: `\ensuremath{\otimes}` (⊗)

Symbol with which to close branches.

Example: `close with={\ensuremath{\ast}}`

Within the tree, `close` is used to identify closed branches.

`close with format`
Forest keylist register

= ⟨*key-value list*⟩

Additional TikZ keys to apply to the closure symbol. Empty by default.

Example: `close with format={red, font=}`

To replace a previously set value, rather than adding to it, use `close with format'` rather than `close with format`.

`close format` = ⟨*key-value list*⟩
Forest keylist register

Default: `font=\scriptsize`

Additional Ti*k*Z keys to apply to any annotation following closure of a branch.

Example: `close format={font=\footnotesize\sffamily, text=gray!75}`

To replace the default value of `close format`, rather than adding to it, use `close format'` rather than `close format`.

Example: `close format'={text=red}`

will produce red annotations in the default font size, whereas

Example: `close format={text=red}`

will produce red annotations in `\scriptsize`.

`subs with` = ⟨*symbol*⟩
Forest toks register

Default: `\ensuremath{\backslash}` (\)

Symbol to indicate variable substitution.

Example: `\text{:}`

Within the tree, `subs` is used to indicate variable substitution.

`subs right` = `true|false`
`not subs right`
Forest boolean register

Default: `true`

Determines whether variable substitution should be indicated to the right of the *wff*. The alternative is, again, to place it to the left of the *wff*. The following are equivalent to the default setting:

```
subs right
subs right=true
```

`subs left`
Forest style

Set `subs right=false`. The following are equivalent ways to place the annotations to the left:

```
subs right=false
not subs right
subs left
```

`just refs left` = `true|false`
`not just refs left`
Forest boolean register

Default: `true`

Determines whether line number references should be placed to the left of justifications. The alternative is to place them to the right of justifications. The following are equivalent to the default setting:

```
just refs left
just refs left=true
```

`just refs right`
Forest style

Set `just refs left=false`. The following are equivalent ways to place the references to the right:

```
just refs left=false
not just refs left
just refs right
```

Note that this setting *only affects the placement of line numbers specified using the cross-referencing system* explained in section 7.2. Hard-coded line numbers in justifications will be typeset as is.

`just format` *Forest keylist register* = ⟨*key-value list*⟩

Additional Ti*k*Z keys to apply to line justifications. Empty by default.

Example: `just format={red, font=}`

To replace a previously set value, rather than adding to it, use `just format'` rather than `just format`.

`line no format` *Forest keylist register* = ⟨*key-value list*⟩

Additional Ti*k*Z keys to apply to line numbers. Empty by default.

Example: `line no format={align=right, text=gray}`

To replace a previously set value, rather than adding to it, use `line no format'` rather than `line no format`. To change the way the number itself is formatted — to eliminate the dot, for example, or to put the number in brackets — redefine `\linenumberstyle` (see section 8).

`wff format` *Forest keylist register* = ⟨*key-value list*⟩

Additional Ti*k*Z keys to apply to *wff*s. Empty by default.

Example: `wff format={draw=orange}`

To replace a previously set value, rather than adding to it, use `wff format'` rather than `wff format`.

`proof statement format` *Forest keylist register* = ⟨*key-value list*⟩

Additional Ti*k*Z keys to apply to the proof statement. Empty by default.

Example: `proof statement format={text=gray, draw=gray}`

To replace a previously set value, rather than adding to it, use `proof statement format'` rather than `proof statement format`.

`highlight format` *Forest autowrapped toks register* = ⟨*key-value list*⟩

Default: `draw=gray, rounded corners`

Additional Ti*k*Z keys to apply to highlighted *wff*s.

Example: `highlight format={text=red}`

To apply highlighting, use the `highlight wff`, `highlight just`, `highlight line no` and/or `highlight line` keys (see section 7.2).

`merge delimiter` *Forest toks register* = ⟨*punctuation*⟩

Default: `\text{; } (; )`

Punctuation to separate distinct justifications for a single proof line. Note that prooftrees will issue a warning if it detects different justifications for a single proof line and will suggest using `move by` to avoid the need for merging justifications. In general, justifications ought not be merged because it is then less clear to which *wff*(s) each justification applies. Moreover, later references to the proof line will be similarly ambiguous. That is, `merge delimiter` ought almost never be necessary because it is almost always better to restructure the proof to avoid ambiguity.

7.2 Local Options

The following options affect the local structure or appearance of the tree and should typically be passed as options to the relevant node(s) within the tree.

`grouped`
`not grouped`
Forest boolean option

Indicate that a line is not an inference. When `single branches` is false, as it is with the default settings, this key is applied automatically and need not be given in the specification of the tree. When `single branches` is true, however, this key must be specified for any line which ought not be treated as an inference.

Example: `grouped`

7.2.1 Annotations

`checked`
Forest style

Mark a complex *wff* as resolved, discharging the line.

Example: `checked`

`checked`
Forest style

`=` ⟨*name*⟩

Existential elimination, discharge by substituting ⟨*name*⟩.

Example: `checked=a`

`close`
Forest style

Close branch.

Example: `close`

`close`
Forest style

`=` ⟨*annotation*⟩

`=` ⟨*annotation prefix*⟩`:`⟨*references*⟩

Close branch with annotation. In the simplest case, ⟨*annotation*⟩ contains no colon and is typeset simply as it is. Any required references to other lines of the proof are assumed to be given explicitly.

Example: `close={12,14}`

If ⟨*annotation*⟩ includes a colon, prooftrees assumes that it is of the form ⟨*annotation prefix*⟩`:`⟨*references*⟩. In this case, the material prior to the colon should include material to be typeset before the line numbers and the material following the colon should consist of one or more references to other lines in the proof. In typical cases, no prefix will be required so that the colon will be the first character. In case there is a prefix, prooftrees will insert a space prior to the line numbers. ⟨*references*⟩ may consist of either forest names (e.g. given by `name=` ⟨*name label*⟩ and then used as ⟨*name label*⟩) or forest relative node names (e.g. ⟨*nodewalk*⟩) or a mixture.

Example: `close={:negated conclusion}`

where `name=negated conclusion` was used to label an earlier proof line `negated conclusion`. If multiple references are given, they should be separated by commas and either ⟨*references*⟩ or the entire ⟨*annotation*⟩ must be enclosed in curly brackets, as is usual for Ti*k*Z and forest values containing commas.

Example: `close={:!c,!uuu}`

`subs`
Forest style

`=` ⟨*name*⟩/⟨*names*⟩

Universal instantiation, instantiate with ⟨*name*⟩ or ⟨*names*⟩.

Example: `subs={a,b}`

`just`
Forest autowrapped toks option

`=` ⟨*justification*⟩

`=` ⟨*justification prefix/suffix*⟩`:`⟨*references*⟩

Justification for inference. This is typeset in text mode. Hence, mathematical expressions must be enclosed suitably in dollar signs or equivalent. In the simplest case, ⟨*justification*⟩ contains no colon and is typeset simply as it is. Any required references to other lines of the proof are assumed to be given explicitly.

Example: `just=3 $\lor$D`

If ⟨*justification*⟩ includes a colon, prooftrees assumes that it is of the form ⟨*justification prefix/suffix*⟩`:`⟨*references*⟩. In this case, the material prior to the colon should include material to be typeset before or after the line numbers and the material following the colon should consist of one or more references to other lines in the proof. Whether the material prior to the colon is interpreted as a ⟨*justification prefix*⟩ or a ⟨*justification suffix*⟩ depends on the value of `just refs left`. ⟨*references*⟩ may consist of either forest names (e.g. given by `name=` ⟨*name label*⟩ and then used as ⟨*name label*⟩) or forest relative node names (e.g. ⟨*nodewalk*⟩) or a mixture. If multiple references are given, they should be separated by commas and ⟨*references*⟩ must be enclosed in curly brackets. If `just refs left` is true, as it is by default, then the appropriate line number(s) will be typeset before the ⟨*justification suffix*⟩.

Example: `just=$\lnot\exists$\elim:{!uu,!u}`

If `just refs left` is false, then the appropriate line number(s) will be typeset after the ⟨*justification prefix*⟩.

Example: `just=From:bertha`

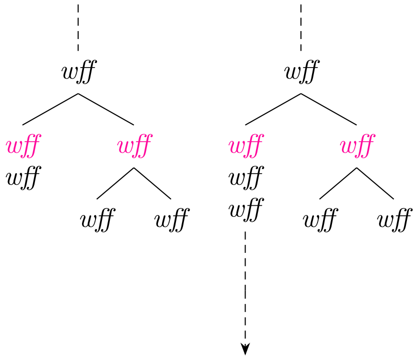
7.2.2 Moving

`move by` = ⟨*positive integer*⟩
Forest style

Move the content of the current line ⟨*positive integer*⟩ lines later in the proof. If the current line has a justification and the content is moved, the justification will be moved with the line. Later lines in the same branch will be moved appropriately, along with their justifications.

Example: `move by=3`

Note that, in many cases, prooftrees will automatically move lines later in the proof. It does this when it detects a condition in which it expects conflicting justifications may be required for a line while initially parsing the tree. Essentially, prooftrees tries to detect cases in which a branch is followed closely by asymmetry in the structure of the branches. This happens, for example, when the first branch's first *wff* is followed by a single *wff*, while the second branch's first *wff* is followed by another branch. Diagrammatically:



In this case, prooftrees tries to adjust the tree by moving lines appropriately if required.

However, this detection is merely structural — prooftrees does not examine the content of the *wff*s or justifications for this purpose. Nor does it look for slightly more distant structural asymmetries, conflicting justifications in the absence of structural asymmetry or potential conflicts with justifications for lines in other, more distant parallel branches. Although it is not that difficult to detect the *need* to move lines in a greater proportion of cases, the problem lies in providing general rules for deciding *how* to resolve such conflicts. (Indeed, some such conflicts might be better left unresolved e.g. to fit a proof on a single Beamer slide.) In these cases, a human must tell prooftrees if something should be moved, what should be moved and how far it should be moved.

Because simple cases are automatically detected, it is best to typeset the proof before deciding whether or where to use this option since prooftrees will assume that this option specifies movements which are required *in addition to* those it automatically detects. Attempting to move a line 'too far' is not advisable. prooftrees tries to simply ignore such instructions, but the results are likely to be unpredictable.

Not moving a line far enough — or failing to move a line at all — may result in the content of one justification being combined with that of another. This happens if `just` is specified more than once for the same proof line with differing content. prooftrees *does* examine the content of justifications for *this* purpose. When conflicting justifications are detected for the same proof line, the justifications are merged and a warning issued suggesting the use of `move by`.

7.2.3 Format: *wff*, justification & line number

`highlight wff`
`not hightlight wff`
Forest boolean option

Highlight *wff*.

Example: `highlight wff`

`highlight just`
`not hightlight just`
Forest boolean option

Highlight justification.

Example: `highlight just`

`highlight line no`
`not highlight line no`
Forest boolean option

Highlight line number.

Example: `highlight line no`

`highlight line`
`not highlight line`
Forest boolean option

Highlight proof line.

Example: `highlight line`

`line no options`
Forest autowrapped toks option

= ⟨*key-value list*⟩

Additional Ti*k*Z keys to apply to the line number for this line.

Example: `line no options={blue}`

`just options`
Forest autowrapped toks option

= ⟨*key-value list*⟩

Additional Ti*k*Z keys to apply to the justification for this line.

Example: `just options={draw, font=\bfseries}`

`wff options`
Forest autowrapped toks option

= ⟨*key-value list*⟩

Additional Ti*k*Z keys to apply to the *wff* for this line.

Example: `wff options={magenta, draw}`

Note that this key is provided primarily for symmetry as it is faster to simply give the options directly to forest to pass on to Ti*k*Z. Unless `wff format` is set to a non-default value, the following are equivalent:

```
wff options={magenta, draw}
magenta, draw
```

`line options`
Forest autowrapped toks option

= ⟨*key-value list*⟩

Additional Ti*k*Z keys to apply to this proof line.

Example: `line options={draw, rounded corners}`

`line no override`
Forest style

= ⟨*text*⟩

Substitute ⟨*text*⟩ for the programmatically-assigned line number. ⟨*text*⟩ will be wrapped by `\linenumberstyle`, so should not be anything which would not make sense in that context.

Example: `line no override={n}`

`no line no`
Forest style

Do not typeset a line number for this line. Intended for use in trees where `line numbering` is activated, but some particular line should not have its number typeset. Note that the number for the line is still assigned and the node which would otherwise contain that number is still typeset. If the next line is automatically numbered, the line numbering will, therefore, 'jump', skipping the omitted number.

Example: `no line no`

8 Macros

`\linenumberstyle` *macro* `{`⟨*number*⟩`}`

This macro is responsible for formatting the line numbers. The default definition is

```
\newcommand*\linenumberstyle[1]{#1.}
```

It may be redefined with `\renewcommand*` in the usual way. For example, if for some reason you would like bold line numbers, try

```
\renewcommand*\linenumberstyle[1]{\textbf{#1.}}
```

9 Extras

9.1 Steps

`every wff` *Forest long step*

A nodewalk long step which visits the proof statement and every *wff* exactly once in proof line number order. This is the default order used for tagging the tableau, but may be used for other purposes. As with the next step, this one should be used in `before annotating` or similar.

`wff from proof line no to` *Forest long step* `{`⟨*start*⟩`}{`⟨*end*⟩`}`

A long step which visits all *wffs* between proof lines numbered ⟨*start*⟩ and `{`⟨*end*⟩`}` inclusive. ⟨*start*⟩ and ⟨*end*⟩ must be proof line numbers in the tableau.

This step cannot be used until quite late in the tableau's processing, as it is valid only once line numbers have been assigned. Hence use of this step must always be delayed. For example, to colour the *wffs* in lines 3, 4 and 5 blue, you could add the following to the preamble:

```
before annotating={for nodewalk={wffs from proof line no to={3}{5}}{blue,typeset node}},
```

Note the use of `typeset node` to re-typeset the content. Without this option, the colour would have no effect.

9.2 Fit

`nodewalk to node` *Forest style* `=` ⟨*name*⟩`{`⟨*nodewalk*⟩`}`

A simple wrapper around forest's `fit to`, which is a Ti*k*Z key used to create a node fitted around a nodewalk using the Ti*k*Z fit library. This does not depend on the code used for tableaux and may be used in an ordinary `forest` environment. (But do not load prooftrees just for this!)

For example, adding the following to a tableau's preamble would create a node named `a` around all the *wffs* in lines 4 to 7 inclusive. Note that this does not include the line number or justification, if used, but only the *wffs* in the 'main' part of the proof.

```
nodewalk to node={a}{wffs from proof line no to={4}{7}},
```

`nodewalk node`
`nodewalk node+`
`+nodewalk node`
`nodewalk node'`
Forest wrapped style

`=` ⟨*key-value list*⟩

Default: `inner sep=0pt`

Style applied to any Ti*k*Z nodes created using `nodewalk to node`. The versions with `+` prepend/append to the existing style, while the `'` version replaces it. `nodewalk node` is aliased to `nodewalk node+`.

Example: `nodewalk node={draw=magenta,rounded corners},`

This would cause the options `inner sep=0pt,draw=magenta,rounded corners` to be applied to any nodes created by `nodewalk to node`.

Note that, despite any similarity in syntax, these are not forest options or registers, but just code wrappers around a simple Ti*k*Z style.

10 Advanced Configuration

forest's default Forest keylist option options may be used to customise tableaux if the provided options prove insufficient. In versions 0.9 and earlier, great care must be taken to avoid conflicts with prooftrees's use of these lists. In later versions, internal versions are reserved for prooftrees's use, enabling forest's to be used more freely by the user. Note that you should still avoid changing the basic structure of the proof. For example, deleting extant justifications or line numbers (as opposed to modifying their content or options), would end badly.

See section 13 for details of the typesetting process.

`before making annotations` = ⟨*key-value list*⟩
Forest keylist option

This Forest keylist option allows customisation after node positions are first computed by forest but before annotations are created. This is sometimes useful.

`before annotating` = ⟨*key-value list*⟩
Forest keylist option

This Forest keylist option allows customisation after annotations are created, but before they are attached to their corresponding *wffs*. I do not know if this option is useful or not.

The remaining options in this section are applicable only if tagging is active.

`before copying content` = ⟨*key-value list*⟩
Forest keylist option

Only really useful if tagging is active. This Forest keylist option allows the content of a node to be altered before it is copied for tagging. Changes made after `proof tree copy content` will affect only the visual representation.

Example: `P \supset Q, before copying content={content+={*}}, before typesetting nodes={blue},`

This would include the `*` into the content of the node used for tagging, but not the colouration.

`before tagging nodes` = ⟨*key-value list*⟩
Forest keylist option

Provided by the ext.tagging library. Only really useful if tagging is active (see section 12). Allow changes before tagged content for a node is finalised. This Forest keylist option is processed before annotations are added to a node's tagged content.

Example: `P \supset Q, before tagging nodes={alt text'={P horseshoe Q},},`

This would replace `P \supset Q` with `P horseshoe Q` in the content used for tagging[3].

`before collating tags` = ⟨*key-value list*⟩
Forest keylist option

Provided by the ext.tagging library. Only really useful if tagging is active (see section 12). This Forest keylist option is processed after annotations are added to a node's tagged content, but before that content is used for tagging.

---

[3] This is not the best way to handle the horseshoe, however. It would be better to define a dedicated macro to produce the symbol such as `\horseshoe` and assign an appropriate 'output intent', regardless of whether you choose to override the content in tagging.

Example: `P \supset Q, just=Ass, before collating tags={alt text'={P horseshoe Q},}`

This would prevent `Ass` from being used in the tagged content. Note that it would also lose any line number, so this should be added explicitly if required.

11 Memoization

Tableaux created by prooftrees cannot, in general, be externalised with Ti*k*Z's external library. Since pgf/Ti*k*Z, in general, and prooftrees, in particular, can be rather slow to compile, this is a serious issue. If you only have a two or three small tableaux, the compilation time will be negligible. But if you have large, complex proofs or many smaller ones, compilation time will quickly become excessive.

Version 0.9 does not cure the disease, but it does offer an extremely effective remedy for the condition. While it does not make prooftrees any faster, it supports the memoize package developed by forest's author, Sašo Živanović (2023). Memoization is faster, more secure, more robust and easier to use than Ti*k*Z's externalisation.

It is faster. It does not require separate compilations for each memoized object, so it is comparatively fast even when memoizing.

It is more secure. It requires only restricted shell-escape, which almost all TEX installations enable by default, so it is considerably more secure and can be utilised even where shell-escape is disabled.

It is more robust. It can successfully memoize code which defeats all ordinary mortals' attempts to externalize with the older Ti*k*Z library.

It is easier to use. It requires less configuration and less intervention. For example, it detects problematic code and aborts memoization automatically in many cases in which Ti*k*Z's external would either cause a compilation error or silently produce nonsense output, forcing the user to manually disable the process for relevant code.

It is compatible with tagging. The library used for tagging ensures that tagging data is not lost when forest trees are externalised with memoize.

There is always a 'but', but this is a pretty small 'but' as 'but's go.

But installation requires slightly more work. To reap the full benefits, you want to use either the `perl` or the `python` 'extraction' method[4]. There is a third method, which does not require any special installation, but this lacks several of the advantages explained above and is not recommended.

If you use TEX Live, you have `perl` already, but you may need to install a couple of libraries. `python` is not a prerequisite for TEX Live but, if you happen to have it installed, you will probably only need an additional library to use this method.

See *Memoize* (Živanović 2023) for further details.

Once you have the prerequisites setup, all you need do is load memoize *before* prooftrees.

```
\usepackage[extraction method=perl]{memoize}% or python
\usepackage{memoize-ext}
\usepackage{prooftrees}
```

After a single compilation, your document will have expanded to include extra pages. At this point, it will look pretty weird. After the next compilation, your document will return to its normal self, the only difference being the speed with which it does so as all your memoized tableaux will simply be included, as opposed to recompiled. Only when you alter the code for a

---

[4] A better `lua`-based solution is currently under development. Once this is available, no additional software will be required, at least for users of TEX Live.

tableau, delete the generated files, disable memoization or explicitly request it will the proof be recompiled.

Memoization is compatible with both prooftrees's cross-referencing system and LaTeX 2ε's cross-references, but the latter require an additional compilation. In general, if a document element takes n compilations to stabilise, it will take $n + 1$ compilations to complete the memoization process. See *Memoize* (Živanović 2023) for details.

12 Tagging

The infrastructure for tagging is provided by the ext.tagging and ext.utils forest libraries, which are part of forest-ext[5]. **These libraries are required regardless of whether tagging is used.**

If memoize is loaded (section 11), ext.tagging uses the framework provided by memoize-ext[6]. **This package is required if memoize is used, regardless of whether tagging is enabled.**

Tagging is *highly experimental* and the implementation will certainly change, as well, possibly, as the interface. Changes to the public interface will be avoided where reasonable. If documented interfaces do change, compatibility options will be provided if possible.

By default, tagging should largely 'just work' for straightforward tableaux. If tagging is active, an 'alternative text' (`alt text`) is automatically generated based on the tableau content[7]. The default aim is to tag tableaux *syntactically*, as opposed to semantically, in accordance with typical usage in logic[8]. If your document is not written in English, you will need to configure a few global options to provide translations. See section 12.1.

See also section 10.

Most of the few options are global and fairly straightforward.

12.1 Global Tagging Options

`tag`
Forest boolean register

`= true|false`

Automatically set according to current status of tagging. Alter at your peril! Whether tagging is active or not. **This register should not be set by the user[9]!** However, it may be safely read to conditionalise code.

`setup plug`
Forest toks register

`tableaux/alt`

`tag plug`
Forest toks register

`alt`

Default: `setup plug=tableaux/alt,tag plug=alt`

Note these keys are provided by ext.tagging.

The only choice with package-specific support is currently the `tableaux/alt` `setup plug`, which uses the library's default `alt` option for `tag plug`. It provides a customised configuration for `tag nodes` which constructs an `alt text` for all *wffs* and the `to prove` statement, if present. It also modifies the order in which tags are collated. Use of latex-lab's plugs for tikz will yield chaotic results at best, but more likely invalid structures or compilation errors. If you need something

---

[5] Rees 2026.

[6] **cfr-memoize-ext**.

[7] Whether this is a useful way to tag them I do not know. Some input from users of tableaux with screen-reading software is required. Contributions, suggestions or feedback seem exceedingly unlikely, but would be appreciated.

[8] This might seem at odds with the LaTeX Project's efforts to tag mathematical content which, as I understand it, is a *semantic* project. But the tension here is, of course, merely apparent, since the intended semantic content of tableaux is syntactic. In the LaTeX Project's sense, this package tries to provide *semantic* tagging. It just so happens that the relevant semantic content is concerned with *syntactic*, as opposed to *semantic*, methods.

[9] Note that setting this false will not result in an untagged tableau. Nor will it allow the user to tag the tableau manually. If you want to do either of those, see tagpdf (for the former) or ext.tagging (for the latter).

other than the current `tableaux/alt` and the options provided by the ext.tagging library do not suffice, file a feature request.

`tag check with` = ⟨*text*⟩
Forest toks register

Default: `discharged`

Text replacement for `check with` for tagging.

`tag close with` = ⟨*text*⟩
Forest toks register

Default: `closed`

Text replacement for `close with` for tagging.

`tag subs with` = ⟨*text*⟩
Forest toks register

Default: `substituted`

Text replacement for `subs with` for tagging.

`tag to prove` = ⟨*text*⟩
Forest toks register

Default: `To prove:`

Text to prepend to the proof statement when tagging.

For example, here's a possible setup for Welsh[10].

```
\forestset{%
  tag check with={cyflawnedig},
  tag close with={caead},
  tag sub with={enghreifftiwyd},
  tag to prove={Profir: },
}
```

12.2 Local Tagging Options

`alt text` = ⟨*text*⟩
Forest toks option

Provided by ext.tagging. `alt text` stores the content used to tag the proof statement and each *wff* in the tableau. prooftrees creates this content automatically from either the proof statement given to to prove or the content of the *wff*. Additional content is appended or prepended when `checked`, `close`, `subs` and/or `just` are used. If applicable, a line number is also added.

The content used for tagging the node may be supplemented or entirely overridden by the user at any stage, but direct use of the option must be delayed in order for the changes to be effective.

Example: `P \equiv Q, just=Ass, before collating tags={alt text'={P iff Q (Premise)},},` `checked,`

This would use precisely the specified content when tagging i.e. the checked marker, justification and any line number would be omitted.

Example: `P \equiv Q, just=Ass, before tagging nodes={alt text'={P iff Q (Premise)},},` `checked,`

The would use the specified content, together with the line number and justification, but would omit the checked marker.

See sections 10 and 13.

---

[10] I do not know if there is an extant terminology for logic. If you know of one, I'd be grateful if you could file a feature request letting me know.

13 Typesetting Process

This section provides a high-level description of the process `prooftree`/`tableau` uses to construct and typeset a proof. Further details can be found in the code documentation.

Most uses of prooftrees do not require knowledge — or, even, awareness of — the details described in this section. Indeed, earlier versions of the documentation did not include this section at all. The details may be of use to users who wish to modify tableaux in ways unsupported by the features documented in previous sections.

1. Initialise tagging, if applicable. This is largely a matter of setting latex-lab's plug for tikz to `noop`, setting some options for ext.tagging and resetting the *tagging keylist* `tag nodes`. This is necessary because a forest tree involves *many* uses of `tikzpicture` and the default tagging can result in erroneous structures and/or compilation errors and produces at best chaotic `marked content`.
2. Starts `forest` with a custom definition of `stages`. `tag tree stage` executes the code actually responsible for tagging the proof.

   Any keylist option described as 'Does nothing by default.' is explicitly intended for users to customise the process.

   Any key marked 'forest' is provided by forest and used unaltered.

   Any key marked 'ext.tagging' is provided by forest-ext and used unaltered.

   Any key marked '*Internal*' is used by this package in constructing and/or tagging the tableau. Like those used by ext.tagging and forest itself, you are both welcome to redefine these and welcome to keep the itsy-bitsy teeny-weeny little pieces if stuff breaks.

   Note that only those intended explicitly for user use *by this package* are marked as 'Does nothing by default.', but several other such items are similarly provided by forest and ext.tagging[11]

   See section 10, Živanović (2017) and forest-ext for details.

   Here is a (long!) step-by-step description of prooftrees's redefinition of `stages`.

   Stage 1 Execute the standard forest parsing for the `default preamble` and `preamble` with forest.

   ```
   for root'={%
     process keylist register=default preamble,
     process keylist register=preamble,
   },
   ```

   Stage 2 Process the forest keylist option `given options`. forest.

   Stage 3 Process the keylist option `before copying content`. Does nothing by default.

   Stage 4 Process the keylist option `proof tree copy content`. *Internal.*

   Stage 5 Process the keylist option `proof tree after copying content`. Does nothing by default.

   Stage 6 Process the keylist option `proof tree before typesetting nodes`. *Internal.*

   Stage 7 Process the forest keylist option `before typesetting nodes`. forest.

   Stage 8 Process the keylist option `proof tree ffurf`. *Internal.*

   Stage 9 Process the keylist option `proof tree symud awto`. *Internal.*

   Stage 10 Execute forest's `typeset nodes stage`. forest.

---

[11] Anything *beginning* `before` is probably OK, but you should check the other package's documentation to be sure.

Stage 11 Process the keylist option `proof tree before packing`. *Internal.*

Stage 12 Process the forest keylist option `before packing`. forest.

Stage 13 Execute forest's `pack stage`. forest.

Stage 14 Process the keylist option `proof tree before computing xy`. *Internal.*

Stage 15 Process the forest keylist option `before computing xy`. forest.

Stage 16 Execute forest's `compute xy stage`. forest.

Stage 17 Process the keylist option `before making annotations`. Does nothing by default.

Stage 18 Process the keylist option `proof tree creu nodiadau`. *Internal.*

Stage 19 Process the keylist option `before annotating`. Does nothing by default.

Stage 20 Process the keylist option `proof tree nodiadau`. *Internal.*

Stage 21 Process the keylist option `proof tree after annotations`. *Internal.*

Stage 22 Process the ext.tagging keylist option `before tagging nodes`. ext.tagging.

Stage 23 Process the ext.tagging keylist option `tag nodes`. ext.tagging.

Stage 24 Process the ext.tagging keylist option `before collating tags`. ext.tagging.

Stage 25 Process the ext.tagging keylist option `collate tags`. ext.tagging.

Stage 26 Process the ext.tagging keylist option `before tagging tree`. ext.tagging.

Stage 27 Execute ext.tagging's `tag tree stage`. ext.tagging.

Stage 28 Process the forest keylist option `before drawing tree`. forest.

Stage 29 Execute forest's `draw tree stage`. forest.

3. Applies style `proof tree`. **This style should NOT be used directly.**
4. Executes the content of `prooftree`/`tableau`'s mandatory argument.
5. Creates a root node with `name=` ⟨*proof statement*⟩.
6. Integrates the contents of the `prooftree`/`tableau`.

Note that prooftrees sets forest's `action character` to `@` before defining the `prooftree`/`tableau` environment.

14 Compatibility

Versions of prooftrees prior to 0.5 are incompatible with bussproofs, which also defines a `prooftree` environment. Version 0.6 is compatible with `bussproofs` provided

either bussproofs is loaded *before* prooftrees

or prooftrees is loaded with option `tableaux` (see section 4).

In either case, prooftrees will *not* define a `prooftree` environment, but will instead define `tableau`. This allows you to use `tableau` for prooftrees trees and `prooftree` for bussproofs trees.

References

Hodges, Wilfred (1991). *Logic: An Introduction to Elementary Logic.* Penguin.

Rees, Clea F. (2026). *forest-ext.* 0.1. 17th Jan. 2026. CTAN: `forest-ext`.

Tantau, Till (2015). *The TikZ and PGF Packages. Manual for Version 3.0.1a.* 3.0.1a. 29th Aug. 2015. URL: `http://sourceforge.net/projects/pgf`.

Živanović, Sašo (2016). *Forest: A PGF/TikZ-Based Package for Drawing Linguistic Trees.* 2.0.2. 4th Mar. 2016. URL: `http://spj.ff.uni-lj.si/zivanovic/`.

— (2017). *Forest: A PGF/TikZ-Based Package for Drawing Linguistic Trees.* 2.1.5. 14th July 2017. CTAN: `forest`.

— (2023). *Memoize.* 1.0.0. 10th Oct. 2023. CTAN: `memoize`.

Change History

v0.3
- General: First CTAN release. 32

v0.4
- General: Bug fix release: `forest` count register `line no shift` was broken; in some cases, an edge was drawn where no edge belonged. 32

v0.41
- General: Update for compatibility with forest 2.1. 32

v0.5
- General: Significant re-implementation leveraging the new argument processing facilities in forest 2.1. This significantly improves performance as the code is executed much faster than the previous pgfmath implementation. 32

v0.6
- General: Add compatibility option for use with bussproofs. Thanks to Peter Smith for suggesting this. 15

v0.7
- General: Fix bug reported at tex.stackexchange.com/q/479263/39222. 32
  - Implement forest boolean register `auto move`. The main point of this option is to allow automatic moves to be switched off if one teaches students to first apply all available non-branching rules for the tableau as a whole, as opposed to all non-branching rules for the sub-tree. The automatic algorithm is consistent with the latter, but not former, approach. The algorithm favours compact trees, which are more likely to fit on beamer slides. Switching the algorithm off permits users to specify exactly how things should or should not be moved. Thanks to Peter Smith for prompting this. 16

v0.8
- General: Add previously unnoticed dependency on amstext. 32
  - Attempt to fix straying closure symbols evident in documentation and a TEX SE question (`https://tex.stackexchange.com/q/619314/`). . . . 32
  - Documentation now loads enumitem, since it depended on it already anyway and specifies doc2 in options for ltxdoc as the code is incompatible with the current version. 31

v0.9
- General: Use `\NewDocumentEnvironment`, removing direct dependency on environ. 32

v0.9.1
- General: Switch to docstrip. 32

v0.9.2
- General: forest-ext is now required, since two of its libraries provide a framework for tagging forest trees. This applies even if tagging is not used. 31
  - Experimental support for tagging based on forest-ext. 32

v0.9.3
- General: memoize-ext is now required if memoize is loaded, since forest-ext now uses its framework for tagging forest trees. This applies even if tagging is not used. 31
  - More robust support for tagging memoized tableaux and changes for compatibility with changes in forest-ext. 32

Index

Features are sorted by kind. Page references are given for both definitions and comments on use. Underlined numbers refer to code line numbers; the remainder to pages.