prooftrees

Version v0.9.1 (SVN Rev: 11204)

Clea F. Rees[*]

2025/09/04

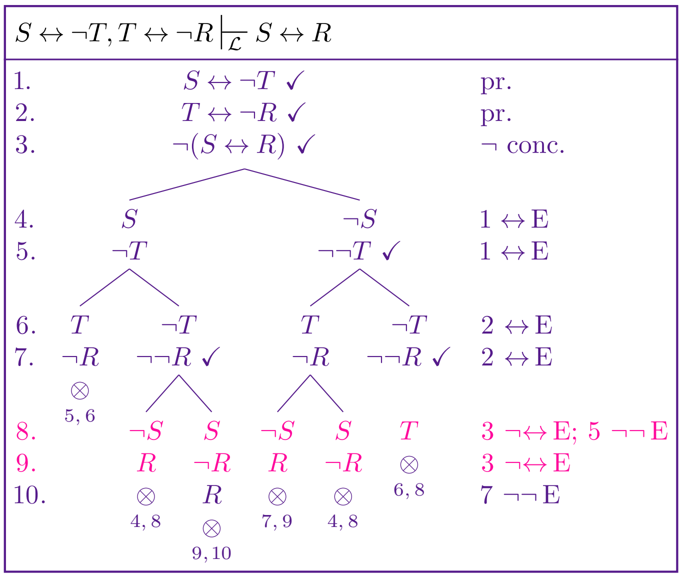
Abstract

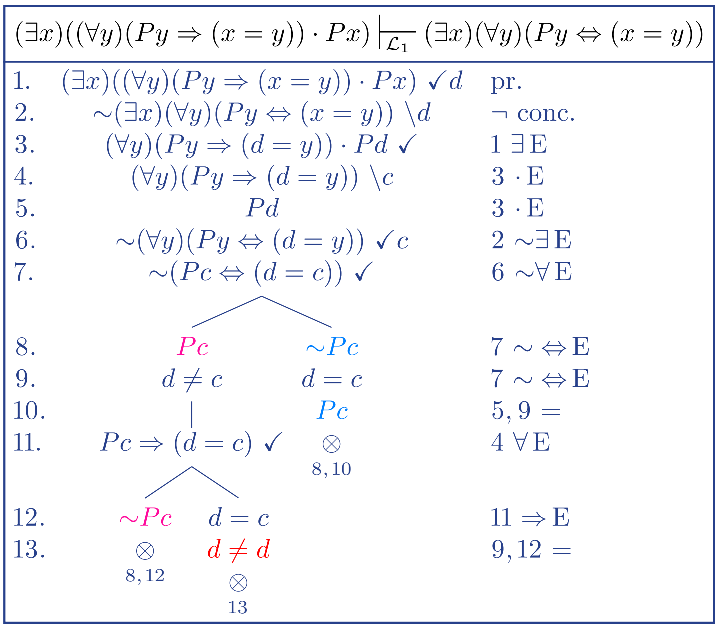
prooftrees is a LaTeX 2_ε package, based on forest, designed to support the typesetting of logical tableaux — 'proof trees' or 'truth trees' — in styles sometimes used in teaching introductory logic courses, especially those aimed at students without a strong background in mathematics. One textbook which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

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.

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`


$S \leftrightarrow \neg T, T \leftrightarrow \neg R \mid_{\overline{\mathcal{L}}} S \leftrightarrow R$
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 ⊗ ⊗ 7 ¬¬E
4, 8 ⊗ 7, 9 4, 8 6, 8
9, 10


$(\exists x)((\forall y)(Py \Rightarrow (x = y)) \cdot Px) \mid_{\overline{\mathcal{L}_1}} (\exists x)(\forall y)(Py \Leftrightarrow (x = y))$
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.

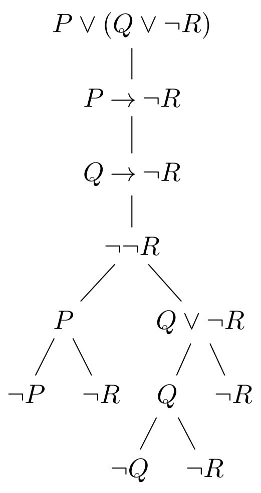
forest is a powerful and flexible package capable of all this and, indeed, a good deal more. It is not enormously difficult to customise particular trees to meet most of our desiderata. However, it is difficult to get things perfectly aligned even in simple cases, requires the insertion of 'phantom' 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_ε 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 Ti*k*Z (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.

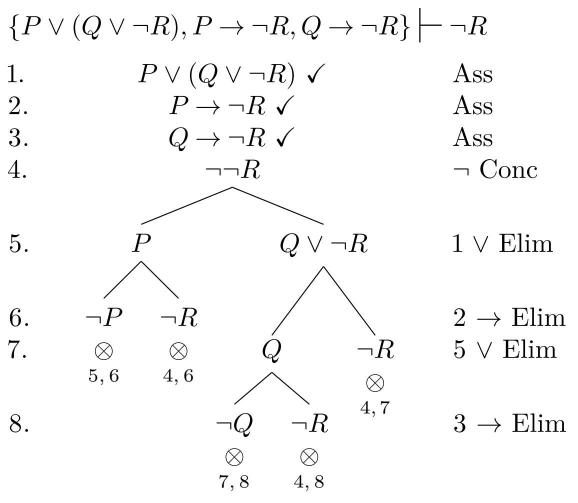
1 forest: default settings

```
\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}
```



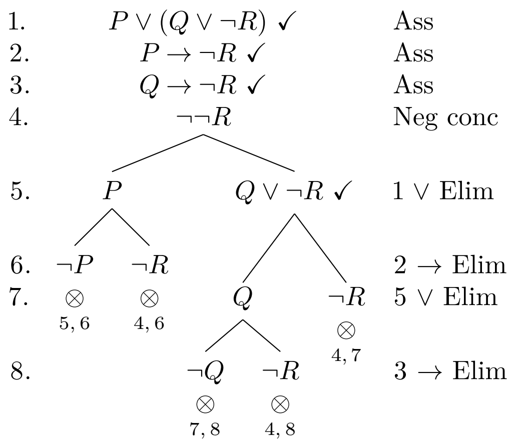
2 prooftrees: default settings

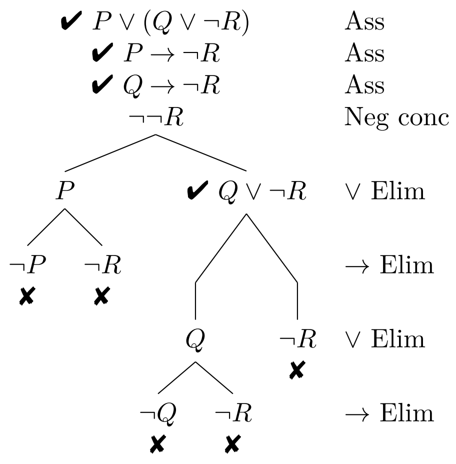
```
\begin{tableau}
  {
    to prove={\{P \vee (Q \vee \lnot R), P \lif
 \lnot R, Q \lif \lnot R\} \sststile{}{} \lnot
 R}
  }
  [P \vee (Q \vee \lnot R), just=Ass, checked
    [P \lif \lnot R, just=Ass, checked
      [Q \lif \lnot R, just=Ass, checked,
name=last premise
        [\lnot\lnot R, just={$\lnot$ 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}
```


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

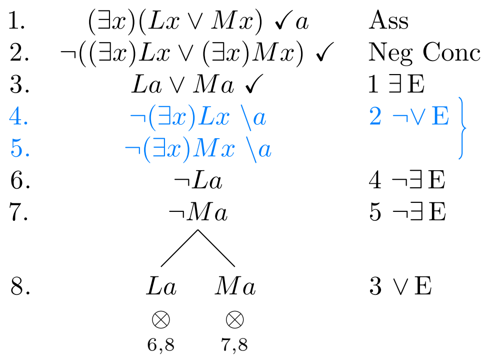
3 prooftrees: sample output

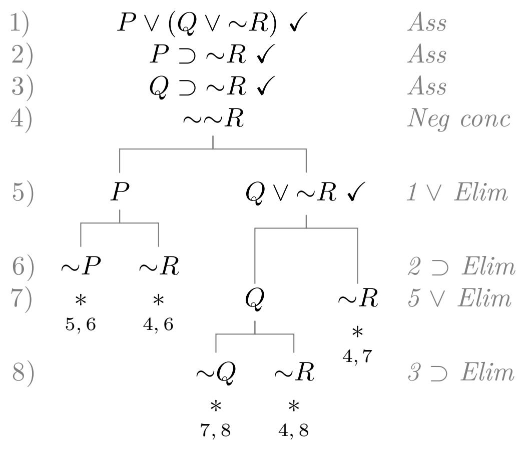
$\{P \lor (Q \lor \neg R), P \to \neg R, Q \to \neg R\} \vdash \neg R$


1. $P \lor (Q \lor \neg R)$ ✓ Ass
2. $P \to \neg R$ ✓ Ass
3. $Q \to \neg R$ ✓ Ass
4. $\neg\neg R$ Neg conc
5. P $Q \lor \neg R$ ✓ 1 ∨ Elim
6. $\neg P$ $\neg R$ 2 → Elim
7. ⊗ 5, 6 ⊗ 4, 6 Q $\neg R$ ⊗ 4, 7 5 ∨ Elim
8. $\neg Q$ $\neg R$ ⊗ 7, 8 ⊗ 4, 8 3 → Elim

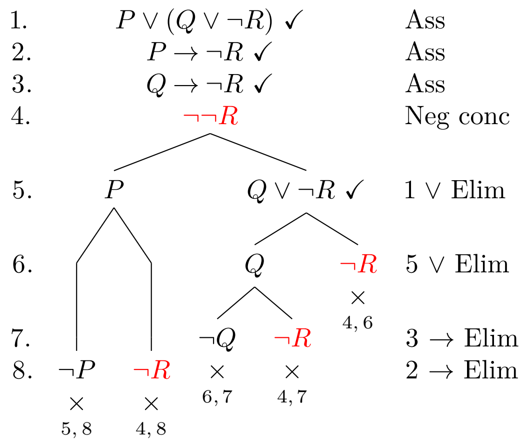

✔ $P \lor (Q \lor \neg R)$ Ass
✔ $P \to \neg R$ Ass
✔ $Q \to \neg R$ Ass
$\neg\neg R$ Neg conc
P ✔ $Q \lor \neg R$ ∨ Elim
$\neg P$ ✘ $\neg R$ ✘ → Elim
Q $\neg R$ ✘ ∨ Elim
$\neg Q$ ✘ $\neg R$ ✘ → Elim

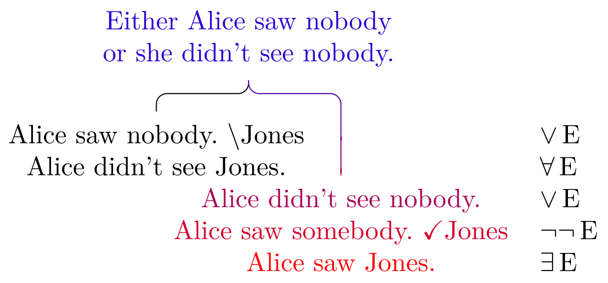
$(\exists x)(Lx \lor Mx) \vdash (\exists x)Lx \lor (\exists x)Mx$


1. $(\exists x)(Lx \lor Mx)$ ✓a Ass
2. $\neg((\exists x)Lx \lor (\exists x)Mx)$ ✓ Neg Conc
3. $La \lor Ma$ ✓ 1 ∃E
4. $\neg(\exists x)Lx \setminus a$ 2 ¬∨E
5. $\neg(\exists x)Mx \setminus a$
6. $\neg La$ 4 ¬∃E
7. $\neg Ma$ 5 ¬∃E
8. La Ma 3 ∨E
⊗ 6,8 ⊗ 7,8


1) $P \lor (Q \lor \sim R)$ ✓ Ass
2) $P \supset \sim R$ ✓ Ass
3) $Q \supset \sim R$ ✓ Ass
4) $\sim\sim R$ Neg conc
5) P $Q \lor \sim R$ ✓ 1 ∨ Elim
6) $\sim P$ $\sim R$ 2 ⊃ Elim
7) * 5, 6 * 4, 6 Q $\sim R$ * 4, 7 5 ∨ Elim
8) $\sim Q$ $\sim R$ * 7, 8 * 4, 8 3 ⊃ Elim

$\{P \lor (Q \lor \neg R), P \to \neg R, Q \to \neg R\} \therefore \neg R$

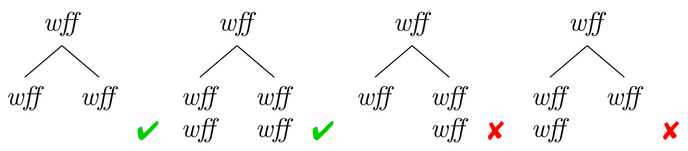

1. $P \lor (Q \lor \neg R)$ ✓ Ass
2. $P \to \neg R$ ✓ Ass
3. $Q \to \neg R$ ✓ Ass
4. $\neg\neg R$ Neg conc
5. P $Q \lor \neg R$ ✓ 1 ∨ Elim
6. Q $\neg R$ × 4, 6 5 ∨ Elim
7. $\neg Q$ $\neg R$ × 6, 7 × 4, 7 3 → Elim
8. $\neg P$ $\neg R$ × 5, 8 × 4, 8 2 → Elim


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

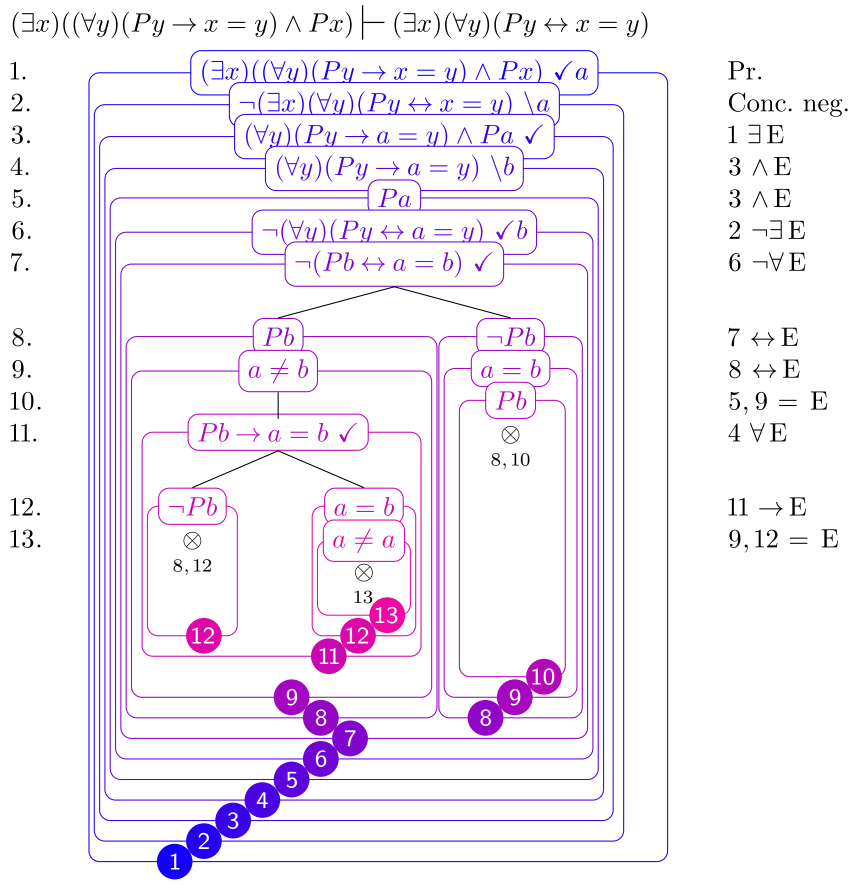
$$(\exists x)((\forall y)(Py \rightarrow x = y) \land Px) \vdash (\exists x)(\forall y)(Py \leftrightarrow x = y)$$

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


$(\exists x)((\forall y)(Py \rightarrow x = y) \land Px) \vdash (\exists x)(\forall y)(Py \leftrightarrow x = y)$
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 ∃E
4. $(\forall y)(Py \rightarrow a = y)$ \$b$ 3 ∧E
5. Pa 3 ∧E
6. $\neg(\forall y)(Py \leftrightarrow a = y)$ ✓b 2 ¬∃E
7. $\neg(Pb \leftrightarrow a = b)$ ✓ 6 ¬∀E
8. Pb | $\neg Pb$ 7 ↔E
9. $a \neq b$ | $a = b$ 8 ↔E
10. Pb 5, 9 = E
⊗ 8, 10
11. $Pb \rightarrow a = b$ ✓ 4 ∀E
12. $\neg Pb$ | $a = b$ 11 →E
⊗ 8, 12
13. $a \neq a$ 9, 12 = E
⊗ 13
1 2 3 4 5 6 7 8 9 8 9 10 11 12 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}
```

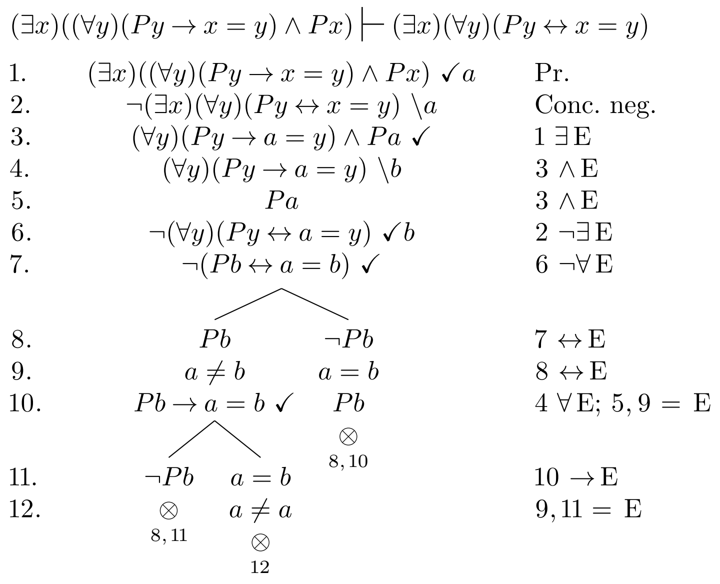
At this point, the main left-hand branch itself branches, so we have two trees 12 . Treating this in the same way as the earlier branch at 8 , we use two sets of square brackets nested within those for tree 12 , but with neither nested within the other. Since we also want to mark the leftmost branch 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:


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

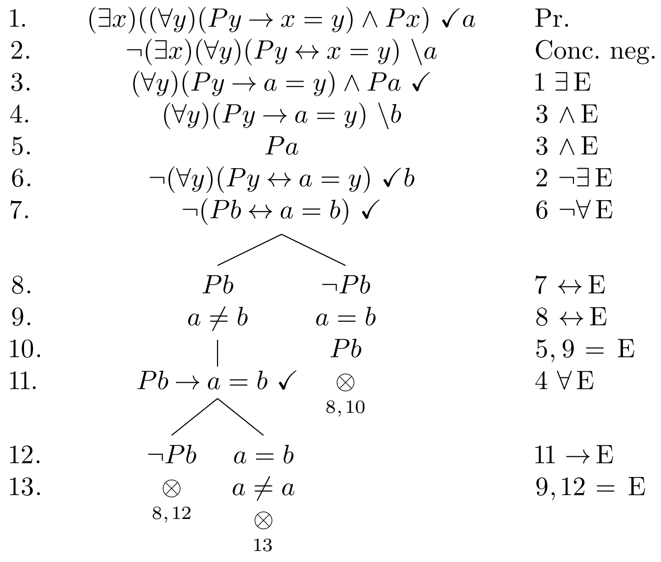
```
                    [{Pb \lif a = b}, checked, just=$\forall\elim$:mark, move by=1
```

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:

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


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

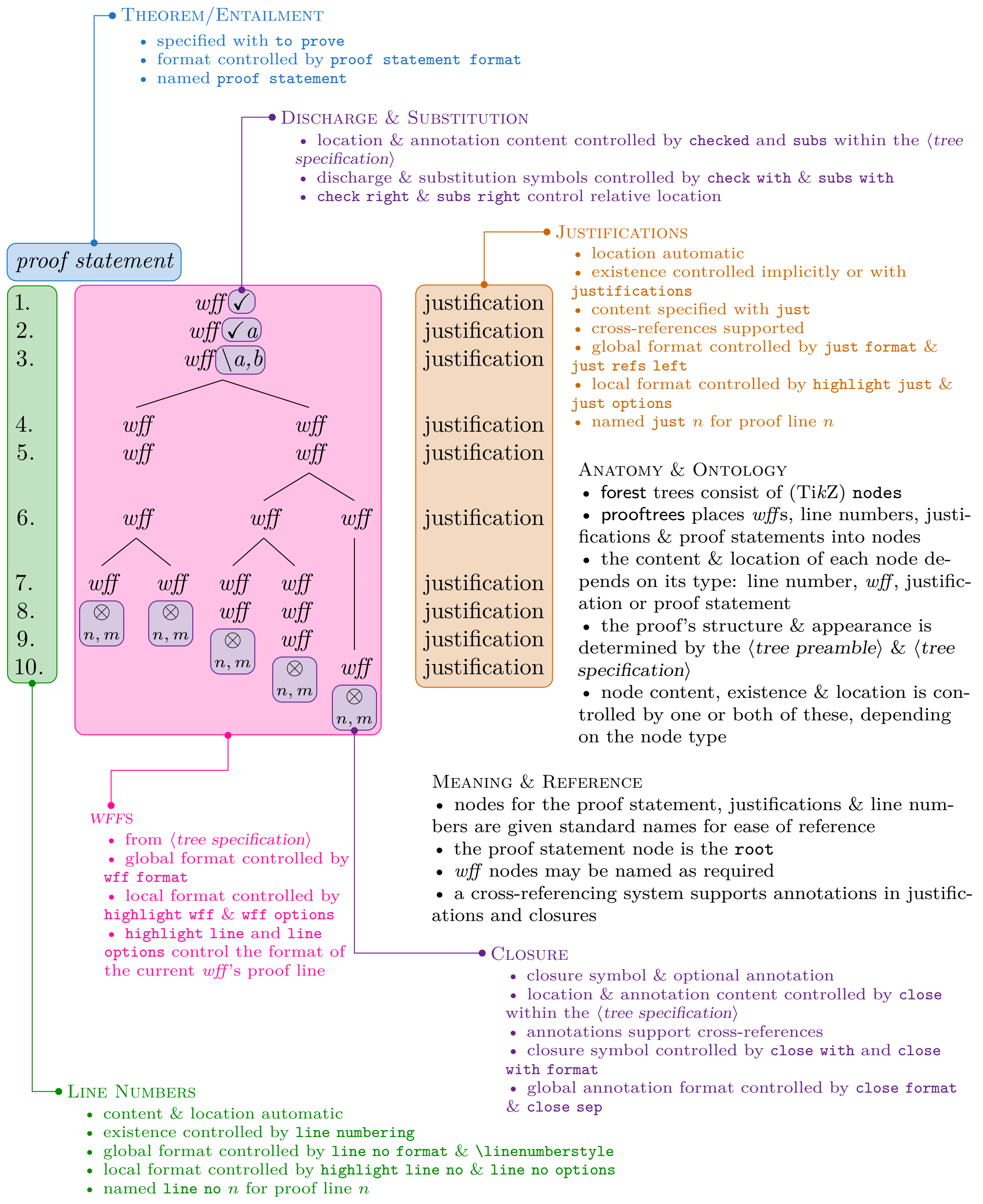
`\begin{tableau}{`⟨*tree preamble*⟩`}`⟨*tree specification*⟩`\end{tableau}`

A substitute for `prooftree`, 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 13 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` `= true|false`
`not auto move`
Forest boolean register

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` `= true|false`
`not line numbering`
Forest boolean register

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` `= true|false`
`not justifications`
Forest boolean register

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` `= true|false`
`not single branches`
Forest boolean register

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}` ($\otimes$)

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 Ti*k*Z 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 TikZ 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` = ⟨*key-value list*⟩
Forest keylist register

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

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

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

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

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` = ⟨*punctuation*⟩
Forest toks register

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 TikZ 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`

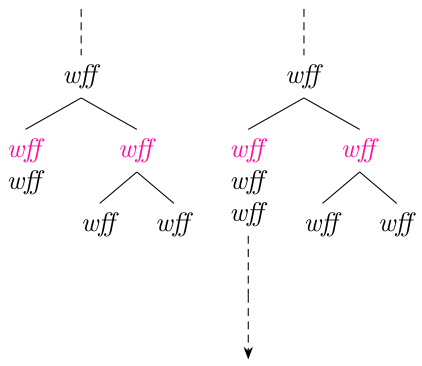
7.2.2 Moving

`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 12 for details of the typesetting process.

`before making annotations` *Forest keylist option*

= ⟨*key-value list*⟩

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` *Forest keylist option*

= ⟨*key-value list*⟩

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` *Forest keylist option*

= ⟨*key-value list*⟩

Only relevant 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 making tags` *Forest keylist option*

= ⟨*key-value list*⟩

Only relevant if tagging is active (see **??**). 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 making tags={ttoks'={P horseshoe Q},},`

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

`before getting tags` *Forest keylist option*

= ⟨*key-value list*⟩

Only relevant if tagging is active (see **??**). This Forest keylist option is processed after annotations are added to a node's tagged content, but before that content is used for tagging.

Example: `P \supset Q, just=Ass, before getting tags={ttoks'={P horseshoe Q},}`

[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.

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.

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. 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{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 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 2$_\varepsilon$'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 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. Increment a count and determine whether to tag the tableau.
2. Initialise tagging, if applicable. This is largely a matter of setting latex-lab's plug for tikz to `artifact`. 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`.
3. Starts `forest` with a custom definition of `stages`. Any keylist option prefixed with `proof tree` is used internally by prooftrees to process the tableau. `tag tableau 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. See section 10 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

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

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

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

   Stage 4 Process the keylist option `proof tree copy content`. Does nothing unless tagging.

   Stage 5 Process the keylist option `proof tree after copying content`.

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

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

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

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

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

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

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

   Stage 13 Execute forest's `pack stage`.

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

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

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

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

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

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

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

Stage 21 Process the keylist option `proof tree before drawing tree`.

Stage 22 Process the forest keylist option `before drawing tree`.

Stage 23 Process the keylist option `before making tags`. Does nothing unless by default.

Stage 24 Process the keylist option `proof tree make tags`. Does nothing unless tagging.

Stage 25 Process the keylist option `before getting tags`. Does nothing unless by default.

Stage 26 Process the keylist option `proof tree get tags`. Does nothing unless tagging.

Stage 27 Execute `tag tableau stage`. Does nothing unless tagging.

Stage 28 Execute forest's `draw tree stage`.

4. Applies style `proof tree`. **This style should NOT be used directly.**
5. Applies style `ttableau`. This does nothing unless tagging is enabled. **This style should NOT be used directly.**
6. Executes the content of `prooftree`/`tableau`'s mandatory argument.
7. Creates a root node with `name=` ⟨*proof statement*⟩.
8. Integrates the contents of the `prooftree`/`tableau`.

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

13 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.

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/.

— (2023). *Memoize*. 1.0.0. 10th Oct. 2023. URL: https://www.ctan.org/pkg/memoize.

14 Implementation

```
\NeedsTeXFormat{LaTeX2e}
\RequirePackage{svn-prov}
⟨!debug&!tag⟩\ProvidesPackageSVN[\filebase.sty]{$Id: prooftrees.dtx 11204 2025-09-04 03:23:15Z
cfrees $}[v0.9.1 \revinfo]
⟨debug⟩\ProvidesPackageSVN[\filebase-debug.sty]{$Id: prooftrees.dtx 11204 2025-09-04 03:23:15
cfrees $}[v0.9.1 \revinfo\ (debugging)]
\DefineFileInfoSVN
```

\prooftrees@enw

Define `\prooftrees@enw` to hold the name of the environment.

Default is to name the environment prooftree, this ensures backwards compatibility.

```
\newcommand*\prooftrees@enw{prooftree}
```

Allow users to change the name to tableau using tableaux.

```
\DeclareOption{tableaux}{\renewcommand*\prooftrees@enw{tableau}}
```

Just in case.

```
\DeclareOption{tableau}{\renewcommand*\prooftrees@enw{tableau}}
```

```
\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{forest}}
```

If `\prooftree` is not yet defined, set the name to `prooftree`; otherwise, use `tableau` to avoid conflict with bussproofs (which uses `prooftree` rather than `bussproof` as one might expect).

Is there some reason I didn't use a hook here? obviously hooks weren't a thing, but `\AtBeginDocument`? Oh, I guess I can't . . . .

```
\ifcsname prooftree\endcsname
  \renewcommand*\prooftrees@enw{tableau}%
\else
  \renewcommand*\prooftrees@enw{prooftree}%
\fi
```

Let users override the default `prooftree` in case they need to load bussproofs later.

```
\ProcessOptions
```

Load forest, but load maths packages later only if needed.

```
\RequirePackage{forest}[2016/12/04]
```

\linenumberstyle

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

How am I meant to describe these things when they aren't macros or environments?!

```
\newtoks\prooftrees@tableau@toks
```

Currently, keys starting `proof tree` or `tableau` and macros starting `prooftree` or `prooftree@` are intended for internal use only.

This does not apply to the environment `prooftree`.

Other keys and macros are intended for use in documents.

In particular, the style `proof tree` is **NOT intended to be used directly by the user and its direct use is **ABSOLUTELY NOT SUPPORTED IN ANY WAY, SHAPE OR FORM**; it is intended only for implicit use when the `prooftree` environment calls it.**

Don't use `@` in register/option names - the documentation is lying when it says non-alphanumerics will be converted to underscores when forming pgfmath functions ;)

```
\forestset{%
```

Line numbers

```
  declare boolean register={line numbering},
```

Default is for line numbers

```
line numbering,
```

Line justifications

```
declare boolean register={justifications},
```

Default is for no line justifications (b/c there's no point in enabling this if the user doesn't specify any content)

```
not justifications,
```

Single branches: explicitly drawn branches and a normal level distance between lone children and their parents

```
declare boolean register={single branches},
```

Default is for lone children to be grouped with their parents

```
not single branches,
```

```
declare boolean register={auto move},% ble mae'n bosibl, symud pethau'n awtomatig
```

Default: symud yn awtomatig

```
auto move,
```

Default will be set to the width of 99 wrapped in the line numbering style

```
declare dimen register={line no width},
```

Fallback default is 0pt

```
line no width'=0pt,
```

Amount by which to shift justifications away from the main tree

```
declare dimen register={just sep},
```

Default is 1.5em

```
just sep'=1.5em,
```

Distance of justifications from centre of inner tree; overrides just sep

```
declare dimen register={just dist},
just dist'=0pt,
```

Amount by which to shift line numbers away from the main tree

```
declare dimen register={line no sep},
line no sep'=1.5em,
```

Distance of line nos. from centre of inner tree; overrides line no sep

```
declare dimen register={line no dist},
line no dist'=0pt,
```

Distance between closure symbols and any following annotation

```
declare dimen register={close sep},
close sep'=.75\baselineskip,
declare dimen register={proof tree line no x},
proof tree line no x'=0pt,
declare dimen register={proof tree justification x},
proof tree justification x'=0pt,
declare dimen register={proof tree inner proof width},
proof tree inner proof width'=0pt,
declare dimen register={proof tree inner proof midpoint},
proof tree inner proof midpoint'=0pt,
```

Count the levels in the proof tree

```
declare count register={proof tree rhif lefelau},
proof tree rhif lefelau'=0,
```

Count the line numbers (on the left)

```
declare count register={proof tree lcount},
proof tree lcount'=0,
```

Count the justifications (on the right)

```
declare count register={proof tree jcount},
proof tree jcount'=0,
```

Adjustment for line numbering

```
declare count register={line no shift},
line no shift'=0,
declare count register={proof tree aros},
proof tree aros'=0,
declare toks register={check with},
check with={\ensuremath{\checkmark}},
declare boolean register={check right},
check right,
check left/.style={not check right},
declare toks register={subs with},
subs with={\ensuremath{\backslash}},
declare boolean register={subs right},
subs right,
subs left/.style={not subs right},
declare toks register={close with},
close with={\ensuremath{\otimes}},
declare keylist register={close format},
close format={font=\scriptsize},
declare keylist register={close with format},
close with format={},
declare toks register={merge delimiter},
merge delimiter={\text{; }},
declare boolean register={just refs left},
just refs left,
just refs right/.style={not just refs left},
declare keylist register={just format},
just format={},
declare keylist register={line no format},
line no format={},
declare autowrapped toks register={highlight format},
highlight format={draw=gray, rounded corners},
declare keylist register={proof statement format},
proof statement format={},
declare keylist register={wff format},
wff format={},
declare boolean={proof tree justification}{0},
declare boolean={proof tree line number}{0},
declare boolean={grouped}{0},
declare boolean={proof tree phantom}{0},
declare boolean={highlight wff}{0},
declare boolean={highlight just}{0},
declare boolean={highlight line no}{0},
declare boolean={highlight line}{0},
Autoforward={highlight line}{highlight just, highlight wff, highlight line no},
declare boolean={proof tree toing}{0},
declare boolean={proof tree toing with}{0},
declare boolean={proof tree rhiant cymysg}{0},
declare boolean={proof tree rhifo}{1},
declare boolean={proof tree arweinydd}{0},
declare autowrapped toks={just}{},
declare toks={proof tree rhestr rhifau llinellau}{},
declare toks={proof tree close}{},
declare toks={proof tree rhestr rhifau llinellau cau}{},
declare autowrapped toks={just options}{},
declare autowrapped toks={line no options}{},
declare autowrapped toks={wff options}{},
declare autowrapped toks={line options}{},
Autoforward={line options}{just options={#1}, line no options={#1}, wff options={#1}},
declare count={proof tree toing by}{0},
declare count={proof tree cadw toing by}{0},
declare count={proof tree toooing}{0},
declare count={proof tree proof line no}{0},
```

Keylists for internal storage

```
declare keylist={proof tree jrefs}{},
declare keylist={proof tree crefs}{},
```

Internal keylists for use in stages

```
declare keylist={proof tree ffurf}{},
declare keylist={proof tree symud awto}{},
declare keylist={proof tree creu nodiadau}{},
declare keylist={proof tree nodiadau}{},
```

Additional internal keylists so we don't pollute forest's and customisation is easier.

```
declare keylist={before copying content}{},
declare keylist={proof tree copy content}{},
declare keylist={proof tree after copying content}{},
declare keylist={proof tree before typesetting nodes}{},
declare keylist={proof tree before packing}{},
declare keylist={proof tree before computing xy}{},
declare keylist={proof tree before drawing tree}{},
```

Empty by default. Allow changes in between processing of standard keylists.

```
declare keylist={before making annotations}{},
declare keylist={before annotating}{},
```

Additions for tagging. These are not actually used yet, but make experimenting (with prooftrees-debug easier.

```
declare boolean register={tag},
tag=0,
declare toks register={plug},
declare toks register={tag check with},
tag check with={discharged},
declare toks register={tag close with},
tag close with={closed},
declare toks register={tag subs with},
tag subs with={substituted},
declare toks register={tag to prove},
tag to prove={To prove: },
declare keylist={before making tags}{},
declare keylist={proof tree make tags}{},
declare keylist={before getting tags}{},
declare keylist={proof tree get tags}{},
declare toks={ttoks}{},
```

> indicates use of process when it is the first token, preceding a list of instructions as opposed to pgfmath stuff

```
define long step={proof tree symud}{}{%
  root,sort by={>{O}{level},>{_O<}{1}{n children}},sort’=descendants
},
```

```
define long step={proof tree cywiro symud}{}{%
  root,if line numbering={n=2}{n=1},sort by={>{O}{level},>{_O<}{1}{n children}},sort’=desc
},
```

Updated version of defn. from saso’s code (forest2-saso-ptsz.tex) & https://chat.stackexchange.com/transcript/message/28321501#28321501

```
define long step={proof tree camau}{}{%
```

Angen +d - gweler https://chat.stackexchange.com/transcript/message/28607212#28607212

```
  root,sort by={>{O}{y},>{Ow1+d}{x}{-##1}},sort’={filter={descendants}{>{OO!&}{proof
tree rhifo}{proof tree phantom}}}%
},
```

coeden brif yn unig ar ôl i greu nodiadau

```
define long step={proof tree wffs}{}{%
  fake=root,if line numbering={n=2}{n=1},tree
},
```

Unlike the previous step, this includes any proof statement and ensures nodes are only visited once, which we want for tagging.

```
define long step={every wff}{}{%
  unique={name=proof statement,proof tree wffs}%
},
proof tree get tags processing order=every wff,
```

See https://tex.stackexchange.com/a/749854/39222 for example usage.

Cf. Sašo Živanović: https://tex.stackexchange.com/a/296771/

Cf. Sašo Živanović: https://chat.stackexchange.com/transcript/message/28484520#28484520

Is there any advantage to sorting here?

```
define long step={wffs from proof line no to}{n args=2}{
  sort by={>O{proof tree proof line no}},
  sort={filter={proof tree wffs}{> nO< nO> O! &&{#1-1}{proof tree proof line no}{#2+1}{proo
tree proof line no}{phantom}}}%
},
```

Mark discharge with optional name substituted into existential

For building `alt` text, we want to do this after content is copied but still before `before typesetting nodes` or `proof tree before typesetting nodes`.

```
checked/.style={%
  proof tree after copying content={%
    if check right={%
      content+'={\ \forestregister{check with}#1},
⟨tag⟩     if tag={%
⟨tag⟩       ttoks+/.process={Rw{tag check with}{ ##1#1}},
⟨tag⟩     }{},
    }{%
      +content'={\forestregister{check with}#1\ },
⟨tag⟩     if tag={%
⟨tag⟩       +ttoks/.process={Rw{tag check with}{##1#1 }},
⟨tag⟩     }{},
    },
  },
},
```

Mark substitution of name into universal

```
subs/.style={%
  proof tree after copying content={%
    if subs right={%
      content+'={\ \forestregister{subs with}#1},
⟨tag⟩     if tag={%
⟨tag⟩       ttoks+/.process={Rw{tag subs with}{ ##1#1}},
⟨tag⟩     }{},
    }{%
      +content'={\forestregister{subs with}#1\ },
⟨tag⟩     if tag={%
⟨tag⟩       +ttoks/.process={Rw{tag subs with}{##1#1 }},
⟨tag⟩     }{},
    },
  },
},
```

This now uses nodes rather than a label to accommodate annotations; closing must be done before packing the tree to ensure that sufficient space is allowed for the symbol and any following annotation; the annotations must be processed before anything is moved to ensure that the correct line numbers are used later, even if the references are given as relative node names

```
close/.style={%
  if={%
    >{__=}{#1}{}%
  }{}{%
    temptoksb={},
    temptoksa={#1},
    split register={temptoksa}{:}{proof tree close,temptoksb},
    if temptoksb={}{}{%
      split register={temptoksb}{,}{proof tree cref},
    },
  },

  proof tree after copying content={%
```

This node holds the closure symbol

```
    append={%
      [\forestregister{close with},
        not proof tree rhifo,
        proof tree phantom,
        grouped,
        no edge,
        process keylist register=close with format,
```

Adjust the distance between the closure symbol and any annotation

```
          proof tree before computing xy={%
            delay={%
```

Cywiro? Fel arall, bydda'r peth byth yn cael ei wneud achos proof tree phantom? Dim yn siwr o gwbl.

```
              l'=\baselineskip,%
              for children={%
                l/.register=close sep,
              },
            },
          },
          proof tree before drawing tree={%
            if={>{RR|}{line numbering}{justifications}}{%
              proof tree proof line no/.option=!parent.proof tree proof line no,
            }{},
          },
          if={%
            >{__=}{#1}{}%
          }{}{%
```

Don't create a second node if there's no annotation.

```
            delay={%
              append={%
```

This node holds the annotation, possibly including cross-references which will be relative to the node's grandparent.

```
                [,
                  not proof tree rhifo,
                  proof tree phantom,
                  grouped,
                  no edge,
                  process keylist register=close format,
                  if={%
                    >{O_=}{!parent,parent.proof tree close}{}%
                  }{}{content/.option=!{parent,parent}.proof tree close},
                  proof tree crefs/.option=!{parent,parent}.proof tree crefs,
                  delay={%
                    !{parent,parent}.proof tree crefs'={},
                  },
                  proof tree before drawing tree={%
                    if={>{RR|}{line numbering}{justifications}}{%
                      proof tree proof line no/.option=!{parent,parent}.proof tree proof
  line no,
                    }{},
                  },
                ]%
              },
            },
          },
        ]%
      },
    },
  },
```

Creates the line numbers on the left; note that it *does* matter that these are part of the tree, even though they do not need to be packed or to have xy computed; moreover, it matters that each is the child of the previous line number... so it won't do for them to *remain* siblings, even though that's fine when they are created.

```
proof tree line no/.style={%
  anchor=base west,
  no edge,
  proof tree line number,
  text width/.register=line no width,
  x'/.register=proof tree line no x,
  process keylist register=line no format,
  delay={%
    proof tree lcount'+=1,
    tempcounta/.process={RRw2+n}{proof tree lcount}{line no shift}{##1+##2},
    content/.process={Rw1}{tempcounta}{\linenumberstyle{##1}},% content i.e. the line
number
```

Name them so they can be moved later

```
    name/.expanded={line no \foresteregister{tempcounta}},%
    typeset node,
```

The initial location of most line numbers is incorrect and they must be moved

```
    if proof tree lcount>=3{%
```

Move the line number below the previous line number

```
      for previous={%
        append/.expanded={line no \foresteregister{tempcounta}}
      },
    }{},
  },
},
```

Creates the justifications on the right but does not yet specify any content

```
proof tree line justification/.style={%
  anchor=base west,
  no edge,
  proof tree justification,
  x'/.register=proof tree justification x,
  process keylist register=just format,
  delay={%
    proof tree jcount'+=1,
    tempcounta/.process={RRw2+n}{proof tree jcount}{line no shift}{##1+##2},
```

Name them so they can be moved

```
    name/.expanded={just \foresteregister{tempcounta}},
```

Angen i osgoi broblemau 'da highlight just/line etc.

```
    typeset node,
```

Correct the location as for the line numbers (cf. line no style)

```
    if proof tree jcount>=3{%
      for previous={%
        append/.expanded={just \foresteregister{tempcounta}},
      },
    }{},
  },
},
zero start/.style={%
  line no shift'+=-1,
},
```

Sets a proof statement

```
to prove/.style={%
  for root={%
    proof tree before typesetting nodes={%
      content={#1},
      phantom=false,
      baseline,
      if line numbering={anchor=base west}{anchor=base},
      process keylist register=proof statement format,

      if={>R{tag}}{%
⟨debug⟩          debug tagging=Copying to prove to ttoks,
        ttoks/.process={ORw2{content}{tag to prove}{##2\ \ensuremath{##1}}},
⟨debug⟩          debug tagging/.option=ttoks,
        proof tree get tags={%
⟨debug⟩            debug tagging=Pick up ttoks from to prove,
          pick up tag/.option=ttoks,
        },
      }{},

    },
    proof tree before computing xy={%
      delay={%
        for children={%
          l=1.5*\baselineskip,
        },
      },
    },
  },
},
```

This style should **NOT** be used directly in a forest environment - see notes at top of this file.

```
proof tree/.style={%
  for tree={%
```

manual 64

```
    parent anchor=children,
```

manual 64

```
    child anchor=parent,
    math content,
    delay={%
```

If we've got justifications, make sure nodes are created for them later and split out cross-references so we identify the correct nodes before anything gets moved, allowing the use of relative node names.

```
      if just={}{}{%
        justifications,
        temptoksa={},
        split option={just}{:}{just,temptoksa},
        if temptoksa={}{}{%
          split register={temptoksa}{,}{proof tree jref},
        },
      },

      if content={}{% if there’s no proof statement
        if level=0{}{%
          shape=coordinate,
        },
      }{},
    },
  },
  where level=0{%
```

No edges from phantom root or proof statement to children.

```
      for children={%
        proof tree before typesetting nodes={%
          no edge,
        },
      },
      delay={%
        if content={}{phantom}{},
```

Create the line numbers if appropriate.

```
        if line numbering={%
          parent anchor=south west,
          if line no width={0pt}{%
            line no width/.pgfmath={width("\noexpand\linenumberstyle{99}")},
          }{},
        }{},
      },
```

This is processed after computing xy.

```
      proof tree creu nodiadau={%
```

Count proof lines if necessary.

```
        if={>{RR|}{line numbering}{justifications}}{%
          proof tree rhif lefelau’/.register=line no shift,
          for proof tree camau={%
            if level>=1{%
              if={%
                >{OO<}{y}{!back.y}%
              }{%
                proof tree rhif lefelau’+=1,
                proof tree proof line no’/.register=proof tree rhif lefelau,
              }{%
                proof tree proof line no’/.register=proof tree rhif lefelau
              },
            }{},
          },
          proof tree inner proof midpoint/.min={%
            >{OOw2+d}{x}{min x}{##1+##2}%
          }{fake=root,descendants},
          proof tree inner proof width/.max={%
            >{OOw2+d}{x}{max x}{##1+##2}%
          }{fake=root,descendants},
          proof tree inner proof width-/.register=proof tree inner proof midpoint,
          proof tree inner proof midpoint+/.process={%
            Rw+d{proof tree inner proof width}{##1/2}%
          },
        }{},
```

Get the x position of line numbers and adjust the location and alignment of the proof statement.

```
      if line numbering={%
        proof tree line no x/.min={>{OOw2+d}{x}{min x}{##1+##2}}{fake=root,descendants},
        if={%
          > Rd= {line no dist}{0pt}%
        }{%
          proof tree line no x-/.register=line no sep,
        }{%
          tempdima/.register=proof tree inner proof width,
          tempdima:=2,
          if={%
            > RR< {line no dist}{tempdima}%
          }{}{%
            proof tree line no x/.register=proof tree inner proof midpoint,
            proof tree line no x-/.register=line no dist,
          },
        },
        proof tree line no x-/.register=line no width,
        for root={%
          tempdimc/.option=x,
          x'+/.register=proof tree line no x,
          x'-/.option=min x,
        },
```

create line numbers on left

```
        prepend={%
          [,
            proof tree line no,
```

() to group are required here - otherwise, the -1 (or -2 or whatever) is silently ignored. Most are created in the wrong place but proof tree line no moves them later.

```
            repeat={((proof_tree_rhif_lefelau)-1)-(line_no_shift)}{%
              delay n={proof_tree_lcount}{
                append={[, proof tree line no]},
              },
            },
          ]%
        },
      }{},
```

Get the x position of justifications and create the nodes which will hold the justification content, if required.

```
      if justifications={%
        proof tree justification x/.max={%
          >{OOw2+d}{x}{max x}{##1+##2}%
        }{fake=root,descendants},
        if={%
          > Rd= {just dist}{0pt}%
        }{%
          proof tree justification x+/.register=just sep,
        }{%
          tempdima/.register=proof tree inner proof width,
          tempdima:=2,
          if={%
            > RR< {just dist}{tempdima}%
          }{}{%
            proof tree justification x/.register=proof tree inner proof midpoint,
            proof tree justification x+/.register=just dist,
          },
        },
        append={%
          [,
            proof tree line justification,
```

Most are created in the wrong place but proof tree line justification moves them later.

```
          repeat={((proof_tree_rhif_lefelau)-1)-(line_no_shift)}{%
            delay n={proof_tree_jcount}{%
              append={[, proof tree line justification]},
            },
          }%
        ]%
      },
    }{},
  },
}{%
  delay={%
```

Automatically group lines if not using single branches.

```
    if single branches={}{%
      if n children=1{%
        for children={%
          grouped,
        },
      }{},
    },
  },
```

Apply wff-specific highlighting and additional TikZ keys.

```
  proof tree before typesetting nodes={%
    process keylist register=wff format,
    if highlight wff={node options/.register=highlight format}{},
    node options/.option=wff options,
  },
},
```

Processed before proof tree symud auto: adjusts the alignment of lines when some levels of the tree are grouped together either whenever the number of children is only 1 or by applying the grouped style to particular nodes when specifying the tree.

```
proof tree ffurf={%
  if auto move={%
    if single branches={%
      where={%
        >{O! _O< O &&}{grouped}{2}{level}{proof tree rhifo}%
      }{%
        if={%
          >{_O= _O< &}{1}{!parent.n children}{1}{!parent,parent.n children}%
        }{%
          not tempboola,
          for root/.process={Ow1}{level}{%
            for level={##1}{%
              if={%
                >{_O< _O= &}{1}{!parent.n children}{1}{n}%
```

```
                  }{%
                    tempboola,
                  }{},
                },
              },
              if tempboola={%
                proof tree toing,
              }{},
            }{},
          }{},
        }{},
        where={%
          >{O _O< O &&}{grouped}{1}{level}{proof tree rhifo}%
```

This searches for certain kinds of structural asymmetry in the tree and attempts to move lines appropriately in such cases - the algorithm is intended to be relatively conservative (not in the sense of 'cautious' or 'safe' but in the sense of 'reflection of the overlapping consensus of reasonable users' / 'what would be rationally agreed behind the prooftrees veil of ignorance'; however, I should have realised I actually had 'the overlapping concensus of reasonable Beamer users' in mind rather than 'the overlapping consensus of reasonable users', so there is now an option to turn it off; apologies if this comment previously misclassified you as 'unreasonable'; apologies for the inconvenience if you are an unreasonable user).

```
        }{%
          not tempboola,
          for root/.process={Ow1}{level}{%
            for level={##1}{%
              if={%
                >{_O< _O= &}{1}{!parent.n children}{1}{n}%
              }{%
                tempboola,
              }{},
            },
```

Sašo: https://chat.stackexchange.com/transcript/message/27874731#27874731, see also https://chat.stackexchange.com/transcript/message/27874722#27874722.

```
          },%
          if tempboola={%
            if n children=0{%
```

We're already moving the parent and the child will move with the parent, so we can just mark this and do nothing else.

```
              if={>{OO|}{!parent.proof tree toing}{!parent.proof tree toing with}}{%
                proof tree toing with,
              }{%
```

Don't move a terminal node even in case of asymmetry: instead, create a separate proof line for terminal nodes on this level which are only children, by moving children with siblings on this level down a proof line, without altering their physical location.

```
                for root/.process={Ow1}{level}{%
```

This makes the tree more compact and stops it looking silly.

```
                  for level={##1}{%
                    if={%
                      >{_O< _O= &}{1}{!parent.n children}{1}{n}%
                    }{%
```

This just serves to keep the levels nice for the sub-tree and ensure things align. We need this because we want to skip a level here to allow room for the terminal node in the other branch.

```
         for parent={%
```

We mark the parent to avoid increasing the line number of its descendants more than once.

```
          if proof tree rhiant cymysg={}{%
           proof tree rhiant cymysg,
           for descendants={%
            proof tree toing by'+=1,
           },
          },
         },
        }{},
       },
```

Sašo: https://chat.stackexchange.com/transcript/message/27874731#27874731, see also https://chat.stackexchange.com/transcript/message/27874722#27874722.

```
      },%
     },
     no edge,
    }{%
     if={%
      >{_O= _O< &}{1}{!parent.n children}{1}{!parent,parent.n children}%
```

Don't try to move if the node has more than 1 child or the grandparent has no more than that; otherwise, mark the node as one to move - we figure out where to move it later.

```
     }{%
      proof tree toing,
     }{no edge},
    },
   }{no edge},
  }{},
 }{},
},
```

Processed before typesetting nodes: if *this* could be done during packing, that would be very nice, even if the previous stuff can't be.

```
proof tree symud awto={%
 if auto move={%
  proof tree aros'=0,
  for proof tree symud={%
```

This relies on an experimental feature of forest, which is anffodus.

```
   if proof tree toing={%
    for nodewalk={fake=parent,fake=sibling,descendants}{do dynamics},
    delay n={\foresteregister{proof tree aros}}{%
     tempcounta/.max={%
      >{OOOOw4+n}{level}{proof tree toing by}{proof tree toooing}%
      {proof tree rhifo}{(##1+##2+##3)*##4}%
     }{parent,sibling,descendants},
     if tempcounta>=1{%
      if={%
       >{Rw1+n OOw2+n >}{tempcounta}{##1+1}{level}{proof tree toing by}{##1+##2}%
      }{%
       tempcounta-/.option=level,
       tempcounta'+=1,
                  move by/.register=tempcounta,
                }{no edge},
              }{no edge},
            },
            proof tree aros'+=4,
          }{},
        },
      }{},
    },
```

Processed after proof tree creu nodiadau and before before drawing tree: creates annotation content which may include cross-references, applies highlighting and additional TikZ keys to line numbers, justifications and to wffs where specified for entire proof lines.

```
    proof tree nodiadau={%
```

Resolve cross-refs in closures.

```
      where proof tree crefs={}{}{%
        split option={proof tree crefs}{,}{proof tree rhif llinell cau},
        if content={}{%
          content/.option=proof tree rhestr rhifau llinellau cau,
        }{%
          content+/.process={_O}{\ }{proof tree rhestr rhifau llinellau cau},
        },
        typeset node,
      },
```

Apply highlighting and additional TikZ keys to line numbers; initial alignment of numbers with proof lines.

```
      if line numbering={%
        for proof tree wffs={%
          if highlight line no={%
```

From Sašo's anti-pgfmath version - rhaid ddweud proof tree proof line no yn ddwywaith ?! dim yn bosibl i ailddefnyddio'r gyntaf ?!

```
            for name/.process={Ow1OOOw3}{proof tree proof line no}{line no ##1}{proof tree proof line no}{line no options}{y}{%
              node options/.register=highlight format,
              ##2,
              y'=##3,
              proof tree proof line no'=##1,
              typeset node,
            }%
          }{%
            if line no options={}{%
              if proof tree phantom={}{%
                for name/.process={Ow1OOw2}{proof tree proof line no}{line no ##1}{proof tree proof line no}{y}{%
                  y'=##2,
                  proof tree proof line no'=##1,
                }%
              },
            }{%
              for name/.process={Ow1OOOw3}{proof tree proof line no}{line no ##1}{proof tree proof line no}{line no options}{y}{%
                ##2,
                y'=##3,
                proof tree proof line no'=##1,
                typeset node,
              }%
            },
          },
        },
      }{},
```

Initial alignment of justifications with proof lines, addition of content, resolution of cross-references and application of highlighting and additional TikZ keys.

```
      if justifications={%
        for proof tree wffs={%
          if just={}{%
            if proof tree phantom={}{%
```

From Sašo's anti-pgfmath version - rhaid ddweud proof tree proof line no yn ddwywaith ?! dim yn bosibl i ailddefnyddio'r gyntaf ?!

```
              for name/.process={Ow1OOw2}{proof tree proof line no}{just ##1}{proof tree proof line no}{y}{%
                y'=##2,
                proof tree proof line no'=##1,
              }%
            },
          }{%
```

Puts the content of the justifications into the empty justification nodes on the right; because this is done late, the nodes need to be typeset again.

```
            if proof tree jrefs={}{}{%
```

Resolve cross-refs in justifications.

```
              split option={proof tree jrefs}{,}{proof tree rhif llinell},
              if just refs left={%
                +just/.process={O_}{proof tree rhestr rhifau llinellau}{\ },
              }{%
                just+/.process={_O}{\ }{proof tree rhestr rhifau llinellau},
              },
            },
```

Apply highlighting and additional TikZ keys to justifications, set content and merge any conflicting specifications, warning user if appropriate.

```
            if highlight just={%
```

From Sašo's anti-pgfmath version - rhaid ddweud proof tree proof line no yn ddwywaith ?! dim yn bosibl i ailddefnyddio'r gyntaf ?!

```
              for name/.process={Ow1OOOOw4}{proof tree proof line no}{just ##1}{proof tree proof line no}{just}{just options}{y}{%
                if={%
                 >{O_= O_= |}{content}{}{content}{##2}%
```

Gweler isod - o gôd Sašo.

```
                }{%
                  content={##2},
```

Avoid merging tags for merged justifications. We need this in four places: for merged and unmerged justifications with and without highlighting. This would have been easier with Peter Smith's preferred design . . . .

```
                }{%
```

```
          content+'={\foresteregister{merge delimiter}##2},
          TeX={\PackageWarning{prooftrees}{Merging conflicting justifications for line ##1! 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.}},
```

Avoid merging tags for merged justifications.

```
         },
         node options/.register=highlight format,
         ##3,
         y'=##4,
         proof tree proof line no'=##1,
         typeset node,
        }%^^A do NOT put a comma here!
       }{%
```

From Sašo's anti-pgfmath version - rhaid ddweud proof tree proof line no yn ddwywaith ?! dim yn bosibl i ailddefnyddio'r gyntaf ?!

```
        for name/.process={Ow1OOOOw4}{proof tree proof line no}{just ##1}{proof tree proof line no}{just}{just options}{y}{%
         if={%
```

From Sašo's anti-pgfmath version - I appreciate this is faster, but why is it **required**?!

```
          >{O_= O_= |}{content}{}{content}{##2}%
         }{%
          content={##2},
```

Avoid merging tags for merged justifications.

```
         }{%
          content+'={\foresteregister{merge delimiter}##2},
          TeX={\PackageWarning{prooftrees}{Merging conflicting justifications for line ##1! 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.}},
```

Avoid merging tags for merged justifications.

```
         },
         ##3,
         y'=##4,
         proof tree proof line no'=##1,
         typeset node,
        }%^^A do NOT put a comma here!
       }
      },
     },
    }{},
```

Apply highlighting and TikZ keys which are specified for whole proof lines to all applicable wffs.

```
    for proof tree wffs={%
     if proof tree phantom={}{%
      if highlight line={%
       for proof tree wffs/.process={OOw2}{proof tree proof line no}{line options}{%
        if proof tree proof line no={##1}{%
         node options/.register=highlight format,
         ##2,
        }{}%
       },
      }{%
       for proof tree wffs/.process={OOw2}{proof tree proof line no}{line options}{%
```

```
         if proof tree proof line no={##1}{##2}{},
        },
      },
      delay={typeset node},
    },
   },
  },
```

Initial alignment so we don't get proof line numbers incrementing due to varying height/depth of nodes, for example - when single branches is true and few nodes are grouped, this is also a reasonable first approximation.

```
  proof tree before packing={%
   for tree={%
    tier/.process={OOw2+nw1}{level}{proof tree toing by}{##1+##2}{tier ##1},
   },
```

If there's no proof statement, adjust the alignment of the proof relative to the surrounding text.

```
   for root={%
    if content={}{%
     !{n=1}.baseline,
    }{},
   },
  },
```

Adjust distance between levels for grouped nodes after tree is packed.

```
  proof tree before computing xy={%
   for tree={%
    if={%
     >{O _O< &}{grouped}{1}{level}%
```

Osgoi overlapping nodes, if posibl: cwestiwn https://tex.stackexchange.com/q/456254/.

```
    }{%
     not tempboola,
     tempcounta/.option=level,
     tempcountb/.option=proof tree toing,
     tempcountb+/.option=proof tree toooing,
     for nodewalk={fake=root, descendants}{if={> RO= On>  O! O! OOw2+nR= &&&&
       {tempcounta}{level} {!u.n children}{1} {proof tree arweinydd} {proof tree
phantom} {proof tree toing by} {proof tree toooing}{##1+##2} {tempcountb}
      }{tempboola}{}},
     if tempboola={}{l'=\baselineskip},
    }{},
   },
  },
```

Set final alignment for proof lines which have been moved by effectively grouping lead nodes and moving their subtrees accordingly - this requires that each line number and justification be the child of the previous one and that if justifications are used at all, then justifications exist for all proof lines, even if empty.

```
  proof tree before drawing tree={%
```

Correct the alignment of move by lines when single branches is false - o fersiwn anti-pgfmath Sašo.

```
   if={>{RR|R!&}{line numbering}{justifications}{single branches}}{%
```

Track cumulative adjustments to line numbers and justifications

```
      tempdimc'=0pt,
      for proof tree cywiro symud={%
```

Only examine the lead nodes - their descendants need the same (cumulative) adjustments

```
        if proof tree arweinydd={%
          tempdima'/.option=y,
```

If there are line numbers, we use the previous line number's vertical position

```
          if line numbering={%
            for name/.process={Ow1+nw1}{proof tree proof line no}{##1-1}{line no ##1}{%
   arafach ?
              tempdimb'/.option=y,
            }%
```

If not, we use the previous justification's vertical position

```
          }{%
            for name/.process={Ow1+nw1}{proof tree proof line no}{##1-1}{just ##1}{%
   arafach ?
              tempdimb'/.option=y,
            }%
          },
```

The parent (which will be a phantom) gets aligned with the previous line

```
          for parent={%
            y'/.register=tempdimb,
          },
```

Adjust so we align this line below the previous one (assuming we're going down)

```
          if tempdimb<={0pt}{%
            tempdimb'-=\baselineskip,
          }{%
            tempdimb'+=\baselineskip,
          },
```

How far are we moving?

```
          tempdimb'-/.register=tempdima,
```

Adjust this node and all descendants

```
          for tree={%
            y'+/.register=tempdimb,
          },
```

Deduct any tracked cumulative adjustments to line numbers and justifications

```
          tempdimb'-/.register=tempdimc,
```

Adjust the line numbers, if any

```
          if line numbering={%
            for name/.process={Ow1}{proof tree proof line no}{line no ##1}{%
              for tree={%
                y'+/.register=tempdimb,
              },
            }%
          }{},
```

Adjust the justifications, if any

```
          if justifications={%
```

t. 60 manual 2.1 rc1

```
            for name/.process={Ow1}{proof tree proof line no}{just ##1}{%
              for tree={%
                y'+/.register=tempdimb,
              },
            }%
          }{},
```

Add the adjustment just implemented to the tracked cumulative adjustments for line numbers and/or justifications

```
          tempdimc'/.register=tempdimb,
        }{},
      },
    }{},
    if={%
      > RR| {auto move}{single branches}%
    }{}{%
      where proof tree arweinydd={%
        for nodewalk={%
          save append={proof tree walk}{%
            current,
            do until={%
              > O+t_+t=! {content}{}%
            }{parent}%
          }%
        }{},
      }{},
      where level>=1{%
        if grouped={%
          if in saved nodewalk={current}{proof tree walk}{}{%
            no edge,
          },
        }{},
      }{},
    },
  },
},
```

This implements both the automated moves prooftrees finds necessary and any additional moves requested by the user - more accurately, it implements initial moves, which may get corrected later (e.g. to avoid skipping numbers or creating empty proof lines, which we assume aren't wanted).

```
move by/.style={%
  if={
   >{_n<}{0}{#1}%
```

Only try to move the node if the target line number exceeds the one i.e. the line number is to be positively incremented.

```
  }{%
    proof tree cadw toing by/.option=proof tree toing by,
    proof tree arweinydd,
    for tree={%
      if={%
       >{_n<}{1}{#1}%
```

Track skipped lines for which we won't be creating phantom nodes

```
        }{%
          proof tree toing by+=#1-2,
          proof tree toooing'+=1,
        }{},
      },
```

Insert our first phantom

```
      delay={%
        replace by={%
          [,
            if={%
             >{_n<}{1}{#1}%
            }{%
              child anchor=parent,
              parent anchor=parent,
            }{%
              child anchor=children,
              parent anchor=children,
            },
            proof tree phantom,
```

Sašo Živanović: https://chat.stackexchange.com/transcript/message/27990955#27990955.

```
            edge path/.option=!last dynamic node.edge path,
            edge/.option=!last dynamic node.edge,
            append,
            proof tree before drawing tree={%
              if={>{RR|}{line numbering}{justifications}}{%
                proof tree proof line no/.process={Ow1+n}{!parent.proof tree proof line no}{##1+1},
              }{},
            },
            if={%
             >{_n<}{1}{#1}%
```

If we are moving by more than 1, we insert a second phantom so that a node with siblings which is moved a long way will not get a unidirectional edge but an edge which looks similar to others in the tree (by default, sloping down a line or so and then plummeting straight down rather than a sharply-angled steep descent).

```
            }{%
              delay={%
                append={%
                  [,
                    child anchor=parent,
                    parent anchor=parent,
                    proof tree toing by=#1-2+proof_tree_cadw_toing_by,
                    proof tree phantom,
                    edge path/.option=!u.edge path,
                    edge/.option=!u.edge,
                    proof tree before drawing tree={%
                      if={>{RR|}{line numbering}{justifications}}{%
                        proof tree proof line no/.process={Ow1+n}{!n=1.proof tree proof line no}{##1-1},
                      }{},
                    },
                    append=!sibling,
                  ]%
                },
              },
            }{%
              if single branches={}{%
                delay={%
                  for children={%
                    no edge,
                  },
                },
              },
            },
          ]%
        },
      },
    }{%
      TeX/.process={Ow1}{name}{\PackageWarning{prooftrees}{Line not moved! I can only
  move things later in the proof. Please see the documentation for details. ##1}},
    },
  },
```

Get the names of nodes cross-referenced in closure annotations for use later

```
  proof tree cref/.style={%
    proof tree crefs+/.option=#1.name,
  },
```

Get the proof line numbers of the cross-referenced nodes in closure annotations, using the list of names created earlier.

```
  proof tree rhif llinell cau/.style={%
    if proof tree rhestr rhifau llinellau cau={}{}{%
      proof tree rhestr rhifau llinellau cau+={,\,},
    },
    proof tree rhestr rhifau llinellau cau+/.option=#1.proof tree proof line no,
  },
```

Get the names of nodes cross-referenced in justifications for use later.

```
  proof tree jref/.style={%
    proof tree jrefs+/.option=#1.name,
  },
```

Get the proof line numbers of the cross-referenced nodes in justifications, using the list of names created earlier.

```
  proof tree rhif llinell/.style={%
    if proof tree rhestr rhifau llinellau={}{}{%
      proof tree rhestr rhifau llinellau+={,\,},
    },
```

works according to Sašo's anti-pgfmath version

```
    proof tree rhestr rhifau llinellau+/.option=#1.proof tree proof line no,
  },
```

2018-02-19 ateb https://tex.stackexchange.com/a/416037/

```
  line no override/.style={%
    proof tree before drawing tree={
      for name/.process={Ow}{proof tree proof line no}{line no ##1}{
        content=\linenumberstyle{#1},
        typeset node,
      },
    },
},
```

2018-02-19 gweler uchod

```
no line no/.style={%
  proof tree before drawing tree={
    for name/.process={Ow}{proof tree proof line no}{line no ##1}{
      content=,
      typeset node,
    },
  },
},
```

Styles to make facilitate drawing around nodewalks.

```
prooftrees@nodewalk@node/.style={inner sep=0pt},
nodewalk node+/.code={%
  \pgfqkeys{/forest}{prooftrees@nodewalk@node/.append style={#1}}%
},
+nodewalk node/.code={%
  \pgfqkeys{/forest}{prooftrees@nodewalk@node/.prepend style={#1}}%
},
nodewalk node'/.code={%
  \pgfqkeys{/forest}{prooftrees@nodewalk@node/.style={#1}}%
},
nodewalk node/.forward to=/forest/nodewalk node+,
nodewalk to node/.style 2 args={%
  proof tree before drawing tree={%
    tikz+={%
      \node [fit to={#2},/forest/prooftrees@nodewalk@node] (#1) {};
    },
  },
},
```

Two styles for debugging. Despite the names, these are available in the non-debug package for largely historical reasons, but also because they probably do not cost much.

Style for use in debugging moves which displays information about nodes in the tree.

```
proof tree dadfygio/.style={%
  proof tree before packing={%
    for tree={%
      label/.process={OOOw3}{level}{proof tree toing by}{id}{%
        [red,font=\tiny,inner sep=0pt,outer sep=0pt, anchor=south]below:##1/##2/##3%
      },
    },
  },
  proof tree before drawing tree={%
    for tree={%
      delay={%
        tikz+/.process={Ow1}{proof tree proof line no}{%
          \node [anchor=west, font=\tiny, text=blue, inner sep=0pt] at (.east) {##1};
        },
      },
    },
  },
},
```

Debugging / dangos dimension stuff.

```
proof tree alino/.style={%
  proof tree before drawing tree={%
    tikz+/.process={%
```

```
   RRRRw4{proof tree inner proof midpoint}{line no width}{line no dist}{just dist}
   {
    \begin{scope}[densely dashed]
     \draw [darkgray] (##1,0) coordinate (a) -- (a |- current bounding box.south);
     \draw [green] (current bounding box.west) -- ++(##2,0) coordinate (b);
     \draw [blue] (b) -- ++(##3,0) coordinate (c);
     \draw [magenta] (c) -- ++(##4,0);
    \end{scope}
   }%
  },
 },
},
```

debug tagging is more expensive, so split this out.

ANGEN: dw i ddim yn meddwl bod crefs yn cynnwys explicit closures?

```
ttableau/.style={%
```

```
 if={>R{tag}}{%
  proof tree copy content={%
```

```
⟨debug⟩      debug tagging=Copying node contents,
```

```
   where content={}{}{%
```

```
⟨debug⟩       debug tagging=Copying node content to ttoks,
```

```
    ttoks+/.process={Ow{content}{\ensuremath{##1}}},
```

```
⟨debug⟩       debug tagging/.process={Ow{ttoks}{ttoks is ##1}},
```

```
   },
  },
  proof tree make tags={%
```

```
⟨debug⟩   debug tagging=Making tags,
```

```
   for unique={proof tree wffs}{%
    if={>OO!&{proof tree rhifo}{proof tree phantom}}
    {%
     if line numbering={%
      +ttoks={\ },
      +ttoks/.option=proof tree proof line no,
     }{},
     if justifications={%
⟨debug⟩     debug tagging={Looking for a justification ...},
```

Avoid merged justifications when tagging; duplicate shared justifications where possible.

```
      if just={}{%
       if={> O_= {!u.n children}{2}}{%
        if={>O_={!s.just}{}}{}{just/.option=!s.just,},
⟨debug⟩       debug tagging/.process={Ow{just}{from sibling just is ##1}},
       }{%
        temptoksa=,
        for nodewalk={%
         while nodewalk valid={u}{%
          u,
          if proof tree phantom={}{%
           if n children=2{%
            back=1,
            s,
            temptoksa/.option=just%
```

```
                    }{},
                    break,
                   }%
                  }%
                 }{},
                 just/.register=temptoksa,
⟨debug⟩         debug tagging/.process={Ow{just}{from ancestor sibling just is ##1}},
                },
               }{},
               if just={}{}{%
                ttoks+/.process={%
                 Ow{just}{\ ##1\ }%
                },
% ^^A              Ow+pw {proof tree proof line no}{%
% ^^A               O{!{name=just ##1}.content}%
% ^^A              }{\ ##1\ }%
               },
⟨debug⟩       debug tagging/.process={Ow{ttoks}{ttoks is now ##1}},
              }{},
⟨debug⟩      debug tagging/.process={Ow{ttoks}{ttoks is now ##1}},
              proof tree get tags={%
⟨debug⟩       debug tagging=Get tag from wff,
⟨debug⟩       debug tagging/.process={Ow{ttoks}{ttoks is now ##1}},
               pick up tag/.option=ttoks,
              },
             }{%
              if n children=0{%
               delay={%
⟨debug⟩        debug tagging=Leaf node,
⟨debug⟩        debug tagging=Get closure status,
                if={> O_=! O_=! | {proof tree crefs}{} {!uu.proof tree close}{}}
                {%
⟨debug⟩         debug tagging=Branch is closed,
⟨debug⟩         debug tagging/.process={Ow{proof tree crefs}{crefs: ##1}},
⟨debug⟩         debug tagging/.process={Ow{!uu.proof tree close}{!uu.proof tree close:
##1}},
⟨debug⟩         debug tagging/.process={Ow{content}{content: ##1}},
                 !uu.ttoks+/.process={ORw2{content}{tag close with}{\ ##2\ ##1\ }},
⟨debug⟩         debug tagging/.process={Ow{!uu.ttoks}{!uu.ttoks is now ##1}},
                }{%
⟨debug⟩         debug tagging=Branch is open,
                },
               },
              }{},
             },
            },
           },
```

Note that this method would not work for many forest trees and may fail for some tableaux, but should work for most proofs, I think.

```
           tag tableau/.style={
⟨debug⟩        debug tagging=Tag tableau,
            tempdima/.max={>OOw2+d{x}{max x}{####1+####2}}{tree},
            tempdima-/.min={>OOw2+d{x}{min x}{####1+####2}}{tree},
            tempdimb/.max={>OOw2+d{y}{max y}{####1+####2}}{tree},
            tempdimb-/.min={>OOw2+d{y}{min y}{####1+####2}}{tree},
            TeX/.process={%
              RRRw3{plug}{tempdima}{tempdimb}{\prooftrees@ttableau{####1}{####2}{####3}}%
            },
           },
         }{},
```

```
  },
```

```
⟨!debug⟩  tag tableau stage/.style={for root'=tag tableau,},
⟨!debug⟩  tag tableau/.style={},
```

Note this is not just default. It is the **only** option even vaguely compatible with tagging.

```
⟨tag⟩  alt/.style={%
⟨tag⟩    plug=alt,
⟨debug & tag⟩    debug tagging=Using plug alt,
⟨tag⟩    pick up tag/.code={%
⟨tag⟩      \toksapp\prooftrees@tableau@toks{##1 }%
⟨debug & tag⟩      \if@ttableau@dadfygio
⟨debug & tag⟩        \typeout{[Tag tableau debug]:: Appending toks ##1.}%
⟨debug & tag⟩      \fi
⟨tag⟩    },
⟨tag⟩  },
⟨tag⟩  alt,
⟨debug⟩  debug tagging/.code={},
% ^^A dadfygio >>>
}
⟨!debug⟩\bracketset{action character=@}
```

`prooftree` `tableau` `\forest`/`\endforest` from egreg's answer at https://tex.stackexchange.com/a/229608/

```
⟨!debug⟩\NewDocumentEnvironment{\prooftrees@enw}{ m +b }
⟨debug⟩\RenewDocumentEnvironment{\prooftrees@enw}{ m +b }
{%
  \global\advance\prooftrees@tableau@id by 1
  \prooftrees@ttableau@init
  \forest
    (%
```

Customised definition of stages - we don't use any custom stages, but we do use several custom keylists, where the processing order of these is critical.

```
      stages={%
```

Nothing is removed from the standard forest definition - we only change it by adding to it.

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

`proof tree before typesetting nodes`, `proof tree after copying content`, `proof tree before packing`, `proof tree before computing xy` and `proof tree before drawing tree` just avoid polluting forest's keylists so they can be used to customise the tableau. `proof tree copy content` is used only for tagging. These are internal lists. They should not generally be redefined or customised by users, as doing so may render the tree structure invalid or cause unexpected results.

In addition to the keylists provided by forest, `before copying content`, `before making annotations`, `before annotating`, `before making tags` and `before getting tags` are intended for users to customise the tableau at these points, if required.

```
        process keylist=before copying content,
⟨tag⟩       process keylist=proof tree copy content,
        process keylist=proof tree after copying content,
        process keylist=proof tree before typesetting nodes,
        process keylist=before typesetting nodes,
```

First two structural additions: process two custom keylists after before typesetting nodes and before typesetting nodes to shape the tree.

```
        process keylist=proof tree ffurf,
        process keylist=proof tree symud awto,
        typeset nodes stage,
        process keylist=proof tree before packing,
        process keylist=before packing,
        pack stage,
        process keylist=proof tree before computing xy,
        process keylist=before computing xy,
        compute xy stage,
```

Second two structural/content additions: process two custom keylists after computing xy and before before drawing tree to create and attach the annotations.

```
        process keylist=before making annotations,
        process keylist=proof tree creu nodiadau,
        process keylist=before annotating,
        process keylist=proof tree nodiadau,
```

Standardish

```
        process keylist=proof tree before drawing tree,
        process keylist=before drawing tree,
```

Hopefully for doing something useful for tagging. `proof tree make tags` and `proof tree get tags` currently do nothing, but will hopefully eventually be used to collect information for tagging the tableau. The 'public' keylists are described above.

```
        process keylist=before making tags,
⟨tag⟩       process keylist=proof tree make tags,
        process keylist=before getting tags,
⟨tag⟩       process keylist=proof tree get tags,
```

```
⟨debug & tag⟩        TeX={%
⟨debug & tag⟩          \if@ttableau@dadfygio
⟨debug & tag⟩            \typeout{[Tag tableau debug]:: Accumulated toks:}%
⟨debug & tag⟩            \ExpandArgs {o} \typeout{\the\prooftrees@tableau@toks}%
⟨debug & tag⟩          \fi
⟨debug & tag⟩        },
```

Try to produce some kind of useful stuff for tagging, if active. Does nothing right now.

```
⟨tag⟩       tag tableau stage,
```

Standard.

```
        draw tree stage,
      },
    )%
```

Apply the proof tree style, which sets keylists from both forest's defaults and our custom additions.

```
    proof tree,
```

Isolate (partly) for now.

```
⟨tag⟩    ttableau,
```

Insert user's preamble, empty or otherwise - this allows the user both to override our defaults (e.g. by setting a non-empty proof statement or a custom format for line numbers) and to customise the tree using forest's facilities in the usual way - BUT customisations of the latter kind may or may not be effective, may or may not have undesirable - not to say chaotic - consequences, and may or may not cause compilation failures (structural changes, in particular, should be avoided completely).

```
    #1,
    [, name=proof statement @#2]%
  \endforest
}{}

\ExplSyntaxOn
```

__prooftrees_memoize:n
__prooftrees_memoize:V

Internal macro so we don't memoize bussproofs's prooftree by mistake.

```
\cs_new_protected_nopar:Npn \__prooftrees_memoize:n #1
{
  \mmzset{
    auto = { #1 } { memoize },
  }
}
\cs_generate_variant:Nn \__prooftrees_memoize:n { V }
```

Paid â memoize bussproofs prooftree ....

```
\hook_gput_code:nnn { begindocument / before } { . }
{%
  \@ifpackageloaded{memoize}{
⟨!debug⟩    \__prooftrees_memoize:V \prooftrees@enw

⟨tag⟩    \tag_if_active:T
⟨tag⟩    {
⟨tag⟩      \mmzset{direct~ccmemo~input=true,}
⟨tag⟩    }
  }{
⟨!debug⟩    \newif\ifmemoizing\memoizingfalse
  }
```

tableau_property_ref_orig:nn
tableau_property_ref_orig:ee

```
  \@ifpackageloaded{memoize-ext}{
    \cs_new_eq:NN \__tableau_property_ref_orig:nn \__mmzx_property_ref_orig:nn
    \cs_new_eq:NN \c__tableau_nexpl_at_cctab  \c__mmzx_nexpl_at_cctab
  }{
    \cs_new_eq:NN \__tableau_property_ref_orig:nn \property_ref:nn
```

Otherwise ...

\c__tableau_nexpl_at_cctab

Not a macro but a constant. Allow @ and expl3 syntax in memos, but don't change cat codes of spaces or newlines.

```
    \cctab_const:Nn \c__tableau_nexpl_at_cctab {
      \cctab_select:N \c_code_cctab
      \makeatletter
      \int_set:Nn \tex_endlinechar:D { 13 }
      \char_set_catcode_space:n      { 9 }
      \char_set_catcode_space:n      { 32 }
      \char_set_catcode_active:n     { 126 } % tilde
    }
  }
  \cs_generate_variant:Nn \__tableau_property_ref_orig:nn {ee}

  \cs_if_exist:NF \checkmark
  {
    \sys_if_engine_opentype:TF
    {
      \RequirePackage{unicode-math}
    }{
      \RequirePackage{amssymb}
    }
  }
  \cs_if_exist:NF \text
  {
    \sys_if_engine_opentype:TF
    {
      \RequirePackage{unicode-math}
    }{
      \RequirePackage{amstext}
    }
  }
```

`\toksapp` Copy of LaTeX's `\addto@hook`. Not used if LuaTeX is used, which defines it as a primitive, or if collargs is loaded (e.g. for memoize), which provides a more complicated version. David Carlisle: https://chat.stackexchange.com/transcript/message/68194858#68194858.

```
  \cs_if_free:NT \toksapp
  {
    \cs_new:Npn \toksapp#1#2{#1\expandafter{\the #1#2}}
  }
}
```

`\prooftrees@tableau@id` Not a macro, but no idea how to mark it in ltxdoc correctly.

```
\newcount\prooftrees@tableau@id
```

`\if@ttableau@dadfygio` for debugging tagging

```
\newif\if@ttableau@dadfygio
\@ttableau@dadfygiofalse
```

`\l__tableau_toks_tl` Variable because I can't work out how to pass the toks directly.

```
\tl_new:N \l__tableau_toks_tl
```

`\prooftrees@tableau@toks` Not a macro, but no idea how to mark it in ltxdoc correctly.

```
\newtoks \prooftrees@tableau@toks
```

`tableau_pgftikz_tag_bbox:nnn`
`tableau_pgftikz_tag_bbox:enn`

```
\cs_new_nopar:Npn \__tableau_pgftikz_tag_bbox:nnn #1#2#3
```

```
{
  \__tableau_pgftikz_tag_bbox_aux:eenn
  {
    \__tableau_property_ref_orig:ee {#1}{xpos}
  }
  {
    \__tableau_property_ref_orig:ee {#1}{ypos}
  }
  {#2}{#3}
}
\cs_generate_variant:Nn \__tableau_pgftikz_tag_bbox:nnn {enn}
```

eau_pgftikz_tag_bbox_aux:nnnn
eau_pgftikz_tag_bbox_aux:eenn

```
\cs_new_nopar:Npn \__tableau_pgftikz_tag_bbox_aux:nnnn #1#2#3#4
{
  \dim_to_decimal_in_bp:n {#1sp}
  \c_space_tl
  \dim_to_decimal_in_bp:n {#2sp}
  \c_space_tl
  \dim_to_decimal_in_bp:n {#1sp+#3}
  \c_space_tl
  \dim_to_decimal_in_bp:n {#2sp+#4}
}
\cs_generate_variant:Nn \__tableau_pgftikz_tag_bbox_aux:nnnn {eenn}
```

Sockets. No idea how to index.

I am not sure `init` does anything at all useful which couldn't be done better with a macro?

```
\socket_new:nn {tableaux/tagsupport/tableau/init}{0}
\socket_new:nn {tableaux/tagsupport/tableau}{2}
\socket_new:nn {tableaux/tagsupport/tableau/mmzx}{2}
\socket_new_plug:nnn {tableaux/tagsupport/tableau/init}{tag}
{
```

Can't use noop due to collection of mcs even if tagging suspended; `artifact` gets rid of these. I think some combination of: `\tag_suspend:n {\tableau} \tag_resume:n {\tableau} \luamml_ignore:` should work, but I have no idea how to get it right. I also suspect I'm not *meant* to get rid of mcs, but the result is so horribly chaotic otherwise.

Assigning these labex-lab plugs prevents the chaos of errors produced by the defaults. Basically, if the tableau will involve n `tikzpicture` environments, we have to completely ignore the first $n-1$, even though this means we also lose all marked content.

```
  \socket_assign_plug:nn {tagsupport/tikz/picture/begin}{artifact}
  \socket_assign_plug:nn {tagsupport/tikz/picture/end}{artifact}
}
```

Shamelessly copied from latex-lab.

```
\socket_new_plug:nnn {tableaux/tagsupport/tableau}{alt}
{
  \tag_mc_end_push:
  \tag_struct_begin:n
  {
    tag=Figure,
    alt=\l__tableau_toks_tl
  }
```

s@\the\prooftrees@tableau@id

Modified version of the latex-lab code as we can't use pgf `remember picture` etc.

```
  \cs_new:cpe {prooftrees@tableau@mark@pos@\the\prooftrees@tableau@id}
  {
    \__tableau_pgftikz_tag_bbox:enn {prooftrees-tableau-id\the\prooftrees@tableau@id}
    {#1}{#2}
  }

  \tag_struct_gput:ene
  {\tag_get:n {struct_num}}
  {attribute}
  {
    /O /Layout /BBox~
    [
      \use:c
      {prooftrees@tableau@mark@pos@\the\prooftrees@tableau@id}
    ]
  }
  \tag_struct_end:
  \tag_mc_begin_pop:n{}
}
```

A version of the above which appends to the context memo rather than executing the code. For `alt`, externalisation seems to work at least somewhat with tagging. (Also `actualtext`. That seems a bad choice for a tableau, but could be provided if required.) But it changes the structure, so I'm not at all sure here.

This requires modifying the cat code regime inside the context memo (or providing 2e-style names for all the cases where these aren't available, including ones internal to prooftrees.

Somewhere, the memoization code introduces extra space when tagging, but since it doesn't work properly anyway, I find myself lacking the will to pin it down. Not even the non-memoized version works, for that matter, so worrying about this version is for the birds' birds, indeed.

```
\socket_new_plug:nnn {tableaux/tagsupport/tableau/mmzx}{alt}
{
  \gtoksapp\mmzCCMemo{
    \csname cctab_begin:c\endcsname {c__tableau_nexpl_at_cctab}
    \global\advance\prooftrees@tableau@id~by~1\relax
    \tex_savepos:D
    \property_record:ee {prooftrees-tableau-id\the\prooftrees@tableau@id}
    {xpos,ypos}
    \tex_savepos:D
    \tag_mc_end_push:
    \tag_struct_begin:n
  }
  \xtoksapp\mmzCCMemo{
    \c_left_brace_str
    tag=Figure,
    alt=
    \c_left_brace_str
  }
  \exp_args:NNV \gtoksapp\mmzCCMemo \l__tableau_toks_tl
  \xtoksapp\mmzCCMemo{
    \c_right_brace_str
    \c_right_brace_str
  }
  \gtoksapp\mmzCCMemo{
    \cs_new:cpe {prooftrees@tableau@mark@pos@\the\prooftrees@tableau@id}
    {
      \__tableau_pgftikz_tag_bbox:enn {prooftrees-tableau-id\the\prooftrees@tableau@id}
      {#1}{#2}
    }
    \tag_struct_gput:ene
    {\tag_get:n {struct_num}}
    {attribute}
    {
      /O /Layout /BBox~
      [
        \use:c
        {prooftrees@tableau@mark@pos@\the\prooftrees@tableau@id}
      ]
    }
    \tag_struct_end:
    \tag_mc_begin_pop:n{}
    \cctab_end:
  }
}
```

`\prooftrees@ttableau@init`
`\__tableau_ttableau_init:`

I think I don't really get the 'plug' concept. It is surely pointless to assign and immediately use one in a package which defines the relevant socket? That is, wouldn't a macro or just the code do equally well but faster?

```
⟨!debug⟩\cs_new_nopar:Npn \__tableau_ttableau_init:
⟨debug⟩\cs_set_nopar:Npn \__tableau_ttableau_init:
{

  \tag_if_active:T{
    \PackageError{prooftrees}{Prooftrees~is~currently~incompatible~with~
      tagging.~
      The~safest~approach~is~to~compile~tableaux~standalone~and~include~
      as~images~with~detailed~alt~text.
    }{Since~you~insist,~you~can~enable~basic~tagging~support~by~loading~
      prooftrees-debug~after~prooftrees.~
      The~code~certainly~requires~LuaLaTeX~and~probably~produces~an~
      invalid~structure,~but~verapdf~validates~it~and~the~code~produces~
      alt~text~automatically.~
      I~do~not~know~how~to~make~the~structure~valid.~
      For~a~logic~package,~this~is~doubly~unfortunate.
    }
    \def\forest##1\endforest{Tableau~\the\prooftrees@tableau@id}
  }

  \tag_if_active:TF{
    \forestset{tag=1}
    \global\prooftrees@tableau@toks{}
⟨debug⟩    \if@ttableau@dadfygio
⟨debug⟩      \typeout{Tagging~is~active.}
⟨debug⟩      \forestset{
⟨debug⟩        debug~tagging/.code={
⟨debug⟩          \typeout{[Tag~tableau~debug]::~##1}
⟨debug⟩        },
⟨debug⟩      }
⟨debug⟩      \typeout{Assigning~and~using~tag~plug~to~socket~
⟨debug⟩        tableaux/tagsupport/tableau/init.
⟨debug⟩      }
⟨debug⟩    \fi
    \socket_assign_plug:nn {tableaux/tagsupport/tableau/init}{tag}
    \socket_use:n {tableaux/tagsupport/tableau/init}
    \def\pgfsys@begin@text{}
    \def\pgfsys@end@text{}
  }{
    \forestset{tag=0}
⟨debug⟩    \if@ttableau@dadfygio
⟨debug⟩      \typeout{Tagging~is~not~active.}
⟨debug⟩     \fi
  }

}
⟨!debug⟩\cs_new_eq:NN \prooftrees@ttableau@init \__tableau_ttableau_init:
⟨debug⟩\cs_set_eq:NN \prooftrees@ttableau@init \__tableau_ttableau_init:
```

\prooftrees@ttableau
__tableau_ttableau:nnn

```
⟨!debug⟩ \cs_new_nopar:Npn \__tableau_ttableau:nnn #1#2#3
⟨debug & tag⟩  \cs_set_nopar:Npn \__tableau_ttableau:nnn #1#2#3
{

% ^^A \tag_resume:n {\tableau}
  \tex_savepos:D
  \property_record:ee {prooftrees-tableau-id\the\prooftrees@tableau@id}
  {xpos,ypos}
  \tex_savepos:D
  \tl_set:No \l__tableau_toks_tl {\the\prooftrees@tableau@toks}
  \socket_assign_plug:nn {tableaux/tagsupport/tableau}{#1}
  \ifmemoizing
    \socket_assign_plug:nn {tableaux/tagsupport/tableau/mmzx}{#1}
  \fi
  \socket_use:nnn {tableaux/tagsupport/tableau} {#2}{#3}
  \socket_use:nnn {tableaux/tagsupport/tableau/mmzx} {#2}{#3}
% ^^A  \tag_suspend:n {\tableau}

}
⟨!debug⟩ \cs_new_eq:NN \prooftrees@ttableau \__tableau_ttableau:nnn
⟨debug & tag⟩  \cs_set_eq:NN \prooftrees@ttableau \__tableau_ttableau:nnn

\ExplSyntaxOff
```

Change History

v0.3

General: First CTAN release. 29

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. . . . 29

v0.41

General: Update for compatibility with forest 2.1. 29

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. 29

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. . . . 29

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. 29

Attempt to fix straying closure symbols evident in documentation and a TEX SE question (`https://tex.stackexchange.com/q/619314/`). 29

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. 29

v0.9

General: Add support for memoize and utilise for documentation. 57

Use `\NewDocumentEnvironment`, removing direct dependency on environ. 29

v0.9.1

`\__tableau_property_ref_orig:ee`: Use mmzx-ext stuff if loaded, but don't require for now, anyway. 57

`\__tableau_ttableau:nnn`: Added `\__tableau_ttableau:nnn`, `\prooftrees@ttableau`. 62

`\__tableau_ttableau_init:`: Added `\__tableau_ttableau_init:`. 61

`\c__tableau_nexpl_at_cctab`: Add `\c__tableau_nexpl_at_cctab`. 57

`\if@ttableau@dadfygio`: Add `\if@ttableau@dadfygio`. 58

`\l__tableau_toks_tl`: Add `\l__tableau_toks_tl`. 58

`\prooftrees@tableau@id`: Add `\prooftrees@tableau@id`. 58

`\prooftrees@tableau@toks`: Add `\prooftrees@tableau@toks`. 58

`\toksapp`: Add `\toksapp`using format definition of `\addto@hook`, in case this is not primitive/already defined. 58

General: `tag tableau stage` following forest pattern and noop default style. 55

Adapt memoize config if tagging or provide conditional. 57

Add `nodewalk node+`, `nodewalk node'`, `+nodewalk node`, `nodewalk node` and `nodewalk to node`. 52

Add checked markers to `ttoks` if tagging. . . . 34

Add substitution markers to `ttoks` if tagging. 35

Add to `ttoks` if tagging. 38

Add ttableau style for experiments with tagging. 53

Added for tagging experiments. 30

Additions for tagging: tag, plug, tag check with, tag close with, tag subs with, tag to prove, proof tree get tags, before getting tags, before making tags, proof tree make tags, ttoks . . . 33

Adjust stage processing for new keylists. 55

Avoid merging tags for merged justifications. . 45

Delay appending closures to avoid copying into `ttoks` when tagging. 35

Don't load amssymb or amstext unconditionally. 30

Experimental `alt` plug for tagging. 55

Experimental tagging style, `ttableau`. 57

New internal keylists. 33

New long step `proof tree every wff`. 34

New long step `wff from proof line no to`. . 34

New public keylists. 33

Renamed `every wff`. 34

Switch to docstrip. 29

Try to tag tableau. 54

Use unicode-math rather than amssymb and amstext on Unicode engines, but, in any case, only load what's needed. 58

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.