drv
derivation trees with METAPOST*

almost a user's guide†

Laurent Méhats
laurent.mehats@gmail.com

documented version: 0.95

$$
\dfrac{\dfrac{}{\Pi \vdash (A \to D) \to E}\,\pi \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A, \Gamma \vdash B}\,\gamma \quad \dfrac{}{\Delta \vdash C}\,\delta}{A, \Gamma, \Delta \vdash B \wedge C}\,\wedge_R \qquad \dfrac{}{B \wedge C, \Theta \vdash D}\,\theta}{A, \Gamma, \Delta, \Theta \vdash D}\,\text{cut}}{\Gamma, \Delta, \Theta \vdash A \to D}\,\to_R \qquad \dfrac{}{E, \Upsilon \vdash F}\,\upsilon}{\Gamma, \Delta, \Theta, (A \to D) \to E, \Upsilon \vdash F}\,\to_L}{\Gamma, \Delta, \Theta, \Pi, \Upsilon \vdash F}\,\text{cut}
$$

Contents

*Available on CTAN. You don't need to know METAPOST to use this package.
†Feel free to improve! (E.g., by correcting the poor English.) Last update: January 11, 2010.

1 Usage

1.1 Structure of a METAPOST file using drv

Preamble

```
input drv;
verbatimtex %&latex
⟨LaTeX preamble⟩
\begin{document}
etex;
```

Figures

```
⟨optional drv tunings⟩
beginfig(⟨index⟩)
  ⟨judgment and inference declarations⟩
  draw drv_tree;
  ⟨optional extra METAPOST code⟩
endfig;
```

Postamble

```
end
```

For each "`beginfig(`⟨*index*⟩`)`, `endfig;`" pair in a file ⟨*jobname*⟩`.mp`, METAPOST generates an Encapsulated PostScript file ⟨*jobname*⟩`.`⟨*index*⟩.

1.2 Running METAPOST

You have to run *at least twice*

`mpost` ⟨*jobname*⟩`.mp`

(once more if you use sub-tree delimiters, see § 2.4). On the first run METAPOST collects the LaTeX code generated by `drv` declaration macros and writes it to the file ⟨*jobname*⟩`-delayed.mp`. On the second run METAPOST preprocesses the LaTeX code in ⟨*jobname*⟩`-delayed.mp` and then typesets the derivation trees.

If you get an error on the first run then it comes from the `drv`/METAPOST code. If you get an error on the second run then it comes from the LaTeX code. In both cases, correct the error (see Appendix A), delete ⟨*jobname*⟩`-delayed.mp` and run "`mpost` ⟨*jobname*⟩`.mp`" twice again (a `makefile` can do that for you).

1.3 LaTeX inclusion commands

Encapsulated PostScript files ⟨*jobname*⟩.⟨*index*⟩ generated by METAPOST can be included in LaTeX documents using the `\includegraphics{`⟨*jobname*⟩.⟨*index*⟩`}` command from the `graphicx.sty` (or `graphics.sty`) package[1].

However `drv` provides ways to set the baseline of derivation tree pictures (see § 3.9 and § 4.3). Then I suggest using the following `\drv{`⟨*jobname*⟩.⟨*index*⟩`}` command which is such that the baseline of the included picture coincides with the baseline of the inclusion point.

```
\usepackage{graphicx}
\makeatletter
\def\Gin@def@bp#1\relax#2#3{\gdef#2{#3}}
\newsavebox{\graphicsbox}
\newcommand*{\drv}[1]{%
\sbox{\graphicsbox}{\includegraphics{#1}}%
\raisebox{\Gin@lly bp}%
{\usebox{\graphicsbox}}}
\makeatother
```

The code for `\drv` was suggested by Josselin NOIREL on the `fr.comp.text.tex` Usenet group.

2 Judgment and inference declarations

2.1 jgm and nfr

`jgm` ⟨*nat*⟩ ⟨*str list*⟩

⟨*nat*⟩	judgment index
⟨*str list*⟩	sub-judgments *math-mode* LaTeX code

`nfr` ⟨*nat*⟩ `(`⟨*nat list*⟩`) (`⟨*str*⟩`,` ⟨*id*⟩`)`

⟨*nat*⟩	inference index
⟨*nat list*⟩	list of premise indices
⟨*str*⟩	inference label *math-mode* LaTeX code
⟨*id*⟩	inference line style identifier (`0`, `1`, `2`, `3`, `4`, `5`, `6` or `_`)

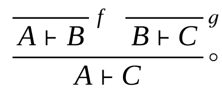
"`jgm` ⟨*nat*⟩" declares a judgment which index is ⟨*nat*⟩ while "`nfr` ⟨*nat*⟩" declares an inference which conclusion is the index ⟨*nat*⟩ judgment (you can declare a judgment before or after the corresponding inference, no matter).

A premise index ⟨*nat*⟩ refers to the sub-tree ending with the index ⟨*nat*⟩ judgment. A list of premise indices may be arbitrary long.

[1]You may get standalone picture files (e.g., transparent PNG for inclusion in a webpage) from each ⟨*jobname*⟩.⟨*index*⟩ file as described in Appendix E.

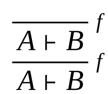
First example

```
beginfig(110)
jgm 0 "A\vdash B";
jgm 1 "B\vdash C";
jgm 2 "A\vdash C";
nfr 0 () ("f", 1);
nfr 1 () ("g", 1);
nfr 2 (0, 1) ("\circ", 1);
draw drv_tree;
endfig;
```



Sub-judgments

```
beginfig(111)
jgm 0 "A\vdash B";
jgm 1 "A", "\vdash", "B";
nfr 0 () ("f", 1);
nfr 1 (0) ("f", 1);
draw drv_tree;
endfig;
```



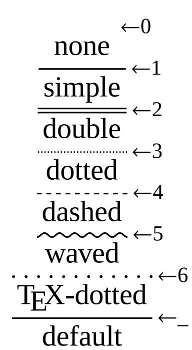
The outputs induced by

```
jgm 0 "A\vdash B";    and    jgm 1 "A", "\vdash", "B";
```

are the same. Using the latter declaration, you can manipulate sub-judgments independently from each-other (see § 6).

Inference line styles

```
beginfig(120)
jgm 0 "\text{none}";
jgm 1 "\text{simple}";
jgm 2 "\text{double}";
jgm 3 "\text{dotted}";
jgm 4 "\text{dashed}";
jgm 5 "\text{waved}";
jgm 6 "\text{\TeX-dotted}";
jgm 7 "\text{default}";
nfr 0 ( ) ("\leftarrow 0", 0);
nfr 1 (0) ("\leftarrow 1", 1);
nfr 2 (1) ("\leftarrow 2", 2);
nfr 3 (2) ("\leftarrow 3", 3);
nfr 4 (3) ("\leftarrow 4", 4);
nfr 5 (4) ("\leftarrow 5", 5);
nfr 6 (5) ("\leftarrow 6", 6);
nfr 7 (6) ("\leftarrow\_", _);
draw drv_tree;
endfig;
```



The default inference line style is set by the `drv_path_style` macro (see § 3.6).

Declarations order Declarations may occur in any order.

```
beginfig(130)
% preorder declarations
jgm 0 "0";
  jgm 1 "00";
    jgm 2 "000";
    jgm 3 "001";
    jgm 4 "002";
  jgm 5 "01";
    jgm 6 "010";
    jgm 7 "011";
    jgm 8 "012";
  jgm 9 "02";
    jgm 10 "020";
    jgm 11 "021";
    jgm 12 "022";
nfr 0 (1, 5, 9) ("a", _);
  nfr 1 (2, 3, 4) ("b", _);
    nfr 2 () ("c", _);
    nfr 3 () ("d", _);
    nfr 4 () ("e", _);
  nfr 5 (6, 7, 8) ("f", _);
    nfr 6 () ("g", _);
    nfr 7 () ("h", _);
    nfr 8 () ("i", _);
  nfr 9 (10, 11, 12) ("j", _);
    nfr 10 () ("k", _);
    nfr 11 () ("l", _);
    nfr 12 () ("m", _);
draw drv_tree;
endfig;
```

```
beginfig(131)
% postorder declarations
    jgm 0 "000";
    jgm 1 "001";
    jgm 2 "002";
  jgm 3 "00";
    jgm 4 "010";
    jgm 5 "011";
    jgm 6 "012";
  jgm 7 "01";
    jgm 8 "020";
    jgm 9 "021";
    jgm 10 "022";
  jgm 11 "02";
jgm 12 "0";
    nfr 0 () ("a", _);
    nfr 1 () ("b", _);
    nfr 2 () ("c", _);
  nfr 3 (0, 1, 2) ("d", _);
    nfr 4 () ("e", _);
    nfr 5 () ("f", _);
    nfr 6 () ("g", _);
  nfr 7 (4, 5, 6) ("h", _);
    nfr 8 () ("i", _);
    nfr 9 () ("j", _);
    nfr 10 () ("k", _);
  nfr 11 (8, 9, 10) ("l", _);
nfr 12 (3, 7, 11) ("m", _);
draw drv_tree;
endfig;
```

$$\dfrac{\dfrac{\dfrac{}{000}{\scriptstyle c}\quad\dfrac{}{001}{\scriptstyle d}\quad\dfrac{}{002}{\scriptstyle e}}{00}{\scriptstyle b}\quad\dfrac{\dfrac{}{010}{\scriptstyle g}\quad\dfrac{}{011}{\scriptstyle h}\quad\dfrac{}{012}{\scriptstyle i}}{01}{\scriptstyle f}\quad\dfrac{\dfrac{}{020}{\scriptstyle k}\quad\dfrac{}{021}{\scriptstyle l}\quad\dfrac{}{022}{\scriptstyle m}}{02}{\scriptstyle j}}{0}{\scriptstyle a} \tag{130}$$

$$\dfrac{\dfrac{\dfrac{}{000}{\scriptstyle a}\quad\dfrac{}{001}{\scriptstyle b}\quad\dfrac{}{002}{\scriptstyle c}}{00}{\scriptstyle d}\quad\dfrac{\dfrac{}{010}{\scriptstyle e}\quad\dfrac{}{011}{\scriptstyle f}\quad\dfrac{}{012}{\scriptstyle g}}{01}{\scriptstyle h}\quad\dfrac{\dfrac{}{020}{\scriptstyle i}\quad\dfrac{}{021}{\scriptstyle j}\quad\dfrac{}{022}{\scriptstyle k}}{02}{\scriptstyle l}}{0}{\scriptstyle m} \tag{131}$$

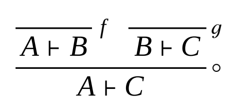
Code for the title page derivation tree

```
beginfig(100)
jgm 0 "\Gamma, \Delta, \Theta, \Pi, \Upsilon\vdash F";
  jgm 1 "\Pi\vdash (A\to D)\to E";
  jgm 2 "\Gamma, \Delta, \Theta, (A\to D)\to E, \Upsilon\vdash F";
    jgm 3 "\Gamma, \Delta, \Theta\vdash A\to D";
      jgm 4 "A, \Gamma, \Delta, \Theta\vdash D";
        jgm 5 "A, \Gamma, \Delta\vdash B\wedge C";
          jgm 6 "A, \Gamma\vdash B";
          jgm 7 "\Delta\vdash C";
        jgm 8 "B\wedge C, \Theta\vdash D";
    jgm 9 "E, \Upsilon\vdash F";
nfr 0 (1, 2) ("\text{cut}", 1);
  nfr 1 () ("\pi", 4);
  nfr 2 (3, 9) ("\to_L", 1);
    nfr 3 (4) ("\to_R", 1);
      nfr 4 (5, 8) ("\text{cut}", 1);
        nfr 5 (6, 7) ("\wedge_R", 1);
          nfr 6 () ("\gamma", 2);
          nfr 7 () ("\delta", 1);
        nfr 8 () ("\theta", 3);
    nfr 9 () ("\upsilon", 2);
draw drv_tree;
endfig;
```

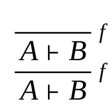
2.2 dcl

dcl enables the simultaneous declarations of a judgment and of the corresponding inference: "dcl ⟨*nat*⟩ (⟨*nat list*⟩) (⟨*str*⟩, ⟨*id*⟩) ⟨*str list*⟩" is a shorthand for "jgm ⟨*nat*⟩ ⟨*str list*⟩; nfr ⟨*nat*⟩ (⟨*nat list*⟩) (⟨*str*⟩, ⟨*id*⟩)".

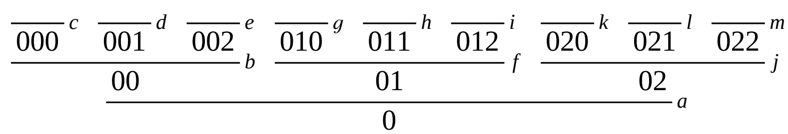
```
beginfig(140)
dcl 0 () ("f", 1) "A\vdash B";
dcl 1 () ("g", 1) "B\vdash C";
dcl 2 (0, 1) ("\circ", 1) "A\vdash C";
draw drv_tree;
endfig;
```



```
beginfig(141)
dcl 0 () ("f", 1) "A\vdash B";
dcl 1 (0) ("f", 1) "A", "\vdash", "B";
draw drv_tree;
endfig;
```



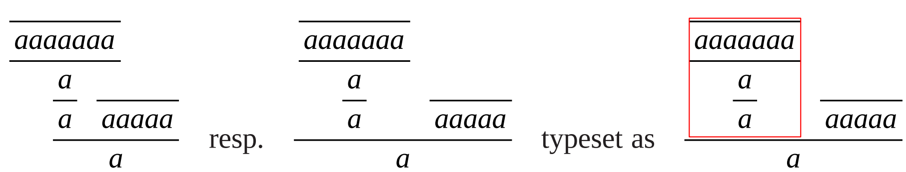
```
beginfig(150)
dcl 0 (1, 5, 9) ("a", _)      "0";
  dcl 1 (2, 3, 4) ("b", _)      "00";
    dcl 2 () ("c", _)             "000";
    dcl 3 () ("d", _)             "001";
    dcl 4 () ("e", _)             "002";
  dcl 5 (6, 7, 8) ("f", _)      "01";
    dcl 6 () ("g", _)             "010";
    dcl 7 () ("h", _)             "011";
    dcl 8 () ("i", _)             "012";
  dcl 9 (10, 11, 12) ("j", _)   "02";
    dcl 10 () ("k", _)             "020";
    dcl 11 () ("l", _)             "021";
    dcl 12 () ("m", _)             "022";
draw drv_tree;
endfig;
```



2.3 bxd and mvd

bxd A premise index ⟨*nat*⟩ can be replaced with "bxd ⟨*nat*⟩" so that the whole sub-tree ending with the index ⟨*nat*⟩ judgment behaves as if it was enclosed within a box.

```
      beginfig(160)
      dcl 0 (1, 4) ("", _)      "a";
resp. dcl 0 (bxd 1, 4) ("", _) "a";
        dcl 1 (2) ("", _)       "a";
          dcl 2 (3) ("", _)     "a";
            dcl 3 () ("", _)    "aaaaaaa";
        dcl 4 () ("", _)        "aaaaa";
      draw drv_tree;
      endfig;
```

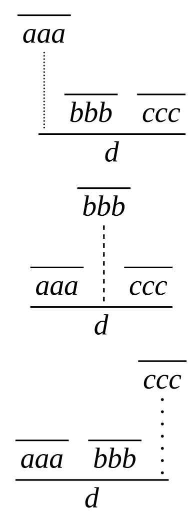


mvd A premise index ⟨*nat 1*⟩ in an inference declaration can be replaced with "`mvd` ⟨*nat 1*⟩ `(`⟨*nat 2*⟩`,` ⟨*id*⟩`)`" so as to declare ⟨*nat 2*⟩ "phantom" inference steps starting from the index ⟨*nat 1*⟩ judgment. The "phantom" inference steps are intended to be drawn as a path using the path-style ⟨*id*⟩.

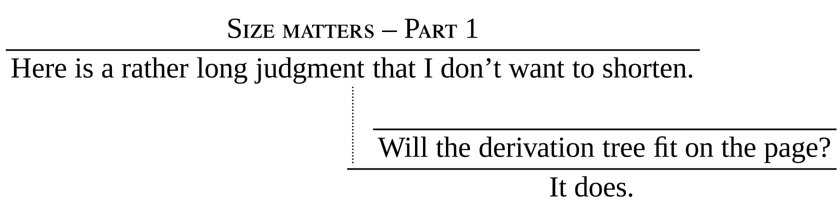
`mvd` ⟨*nat 1*⟩ `(`⟨*nat 2*⟩`,` ⟨*id*⟩`)`

⟨*nat 1*⟩	index of the origin judgment
⟨*nat 2*⟩	number of phantom steps
⟨*id*⟩	phantom steps path-style identifier (`0`, `1`, `2`, `3`, `4`, `5`, `6` or `_`)

```
     beginfig(170)
     jgm 1 "aaa";
     jgm 2 "bbb";
     jgm 3 "ccc";
     jgm 4 "d";
     nfr 1 () ("", _);
     nfr 2 () ("", _);
     nfr 3 () ("", _);
     nfr 4 (mvd 1 (2, 3), 2, 3) ("", _);
resp. nfr 4 (1, mvd 2 (2, 4), 3) ("", _);
resp. nfr 4 (1, 2, mvd 3 (2, 6)) ("", _);
     draw drv_tree;
     endfig;
```




```
beginfig(180)
jgm 0 "\textsc{Size matters -- Part 1}";
jgm 1 "\text{Here is a rather long judgment"& % string concatenation
      " that I don't want to shorten.}";
jgm 2 "\text{Will the derivation tree fit on the page?}";
jgm 3 "\text{It does.}";
nfr 0 () ("", 0);
nfr 1 (0) ("", 1);
nfr 2 () ("", 1);
nfr 3 (mvd 1 (2, 3), 2) ("", 1);
draw drv_tree;
endfig;
```



2.4 Sub-tree delimiters and labels

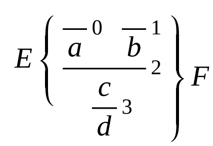
`Nfr` The `Nfr` declaration macro is an alternative for `nfr` that enables the typesetting of delimiters.

`Nfr` ⟨*nat*⟩ `(`⟨*nat list*⟩`)` `(`⟨*str 1*⟩`,` ⟨*str 2*⟩`,` ⟨*str 3*⟩`,` ⟨*id*⟩`)`

⟨*nat*⟩	inference index
⟨*nat list*⟩	list of premise indices
⟨*str 1*⟩	inference label *math-mode* LaTeX code
⟨*str 2*⟩	left delimiter label *math-mode* LaTeX code
⟨*str 3*⟩	right delimiter label *math-mode* LaTeX code
⟨*id*⟩	inference line style identifier (`0`, `1`, `2`, `3`, `4`, `5`, `6` or `_`)

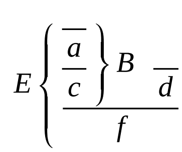
If both ⟨*str 2*⟩ and ⟨*str 3*⟩ are the empty string `""` then `Nfr` behaves exactly the same way as `nfr`. However, if ⟨*str 2*⟩ is a non-empty string then a delimiter is placed to the left of the sub-tree ending with the index ⟨*nat*⟩ judgment and ⟨*str 2*⟩ is attached to it as a label. The same way, if ⟨*str 3*⟩ is a non-empty string then a delimiter is placed to the right of the sub-tree ending with the index ⟨*nat*⟩ judgment and ⟨*str 3*⟩ is attached to it as a label. Both ⟨*str 2*⟩ and ⟨*str 3*⟩ may be non-empty strings. You may use `"{}"` as a string argument to get a delimiter without a label.

```
beginfig(190)
jgm 0 "a";
jgm 1 "b";
jgm 2 "c";
jgm 3 "d";
Nfr 0 () ("0", "", "", _);
Nfr 1 () ("1", "", "", _);
Nfr 2 (0, 1) ("2", "E", "", _);
Nfr 3 (2) ("3", "", "F", _);
draw drv_tree;
endfig;
```



`Dcl` The `Dcl` declaration macro is a shorthand for `jgm` and `Nfr` in the same way as `dcl` is a shorthand for `jgm` and `nfr`.

```
beginfig(200)
Dcl 0 () ("", "", "", _) "a";
Dcl 1 (0) ("", "", "B", _) "c";
Dcl 2 () ("", "", "", _) "d";
Dcl 3 (1, 2) ("", "E", "", _) "f";
draw drv_tree;
endfig;
```



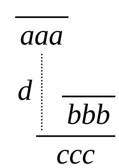
Mvd The `Mvd` macro is an alternative for `mvd` that enables the attachment of labels to phantom steps paths.

`Mvd` ⟨*nat 1*⟩ `(`⟨*nat 2*⟩`,` ⟨*str 1*⟩`,` ⟨*str 2*⟩`,` ⟨*id*⟩`)`

⟨*nat 1*⟩	index of the origin judgment
⟨*nat 2*⟩	number of phantom steps
⟨*str 1*⟩	left label *math-mode* LaTeX code
⟨*str 2*⟩	right label *math-mode* LaTeX code
⟨*id*⟩	phantom steps path-style identifier (`0`, `1`, `2`, `3`, `4`, `5`, `6` or `_`)

If ⟨*str 1*⟩ is a non-empty string then it is attached as a label to the left of the phantom steps path. The same way, if ⟨*str 2*⟩ is a non-empty string then it is attached as a label to the right of the phantom steps path. Both ⟨*str 1*⟩ and ⟨*str 2*⟩ may be non-empty strings.

```
beginfig(210)
jgm 1 "aaa";
jgm 2 "bbb";
jgm 3 "ccc";
nfr 1 () ("", _);
nfr 2 () ("", _);
nfr 3 (Mvd 1 (2, "d", "", 3), 2) ("", _);
draw drv_tree;
endfig;
```



3 drv tunings

`drv` tuning macros set the parameters according to which derivation trees are typeset. You have to call these macros *outside* figure environments (delimited by "`beginfig(`⟨*index*⟩`)`, `endfig;`" pairs).

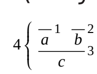
3.1 drv_font_size

`drv_font_size` ⟨*str*⟩

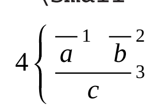
⟨*str*⟩ LaTeX font-size command

```
"\tiny"
"\scriptsize"
"\footnotesize"
"\small"
"\normalsize"    * default *
"\large"
"\Large"
etc.
```

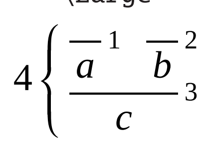
`"\tiny"`



`"\small"`



`"\Large"`



3.2 drv_math_style

`drv_math_style` (⟨*id*⟩, ⟨*str*⟩)

⟨*id*⟩	component identifier (`drv`, `jdg`, `ilb`, `dlb` or `plb`)		
	`drv`	derivation trees	∗ *default style:* `"\displaystyle"` ∗
	`jdg`	judgments	∗ *default style:* `"\textstyle"` ∗
	`ilb`	inference labels	∗ *default style:* `"\scriptstyle"` ∗
	`dlb`	delimiter labels	∗ *default style:* `"\textstyle"` ∗
	`plb`	phantom steps labels	∗ *default style:* `"\textstyle"` ∗
⟨*str*⟩	LaTeX math-style command		

"`drv_math_style (drv, —);`"

`"\displaystyle"`	`"\textstyle"`	`"\scriptstyle"`
$4\left\{\dfrac{\overline{a}^{\,1}\quad\overline{b}^{\,2}}{c}\,3\right.$	$\textstyle 4\left\{\frac{\overline{a}^{\,1}\quad\overline{b}^{\,2}}{c}\,3\right.$	$\scriptstyle 4\left\{\frac{\overline{a}^{\,1}\quad\overline{b}^{\,2}}{c}\,3\right.$

"`drv_math_style (jdg, —);`"

`"\displaystyle"`	`"\textstyle"`	`"\scriptstyle"`
$4\left\{\dfrac{\overline{\displaystyle\bigwedge_{i\in I} A_i}^{\,1}\quad\overline{b}^{\,2}}{c}\,3\right.$	$4\left\{\frac{\overline{\bigwedge_{i\in I} A_i}^{\,1}\quad\overline{b}^{\,2}}{c}\,3\right.$	$4\left\{\frac{\overline{\scriptstyle\bigwedge_{i\in I} A_i}^{\,1}\quad\overline{\scriptstyle b}^{\,2}}{\scriptstyle c}\,3\right.$

"`drv_math_style (ilb, —);`"

`"\textstyle"`	`"\scriptstyle"`	`"\scriptscriptstyle"`
$\dfrac{\overline{a}\,1\quad\overline{b}\,2}{c}\,3$	$\dfrac{\overline{a}^{\,1}\quad\overline{b}^{\,2}}{c}\,{}^{3}$	$\dfrac{\overline{a}^{\,\scriptscriptstyle 1}\quad\overline{b}^{\,\scriptscriptstyle 2}}{c}\,{}^{\scriptscriptstyle 3}$

Notice that the math-style of derivation trees determines the math-style of judgments (and of labels) in the same way as the math-style of fractions determines the math-style of numerators and denominators.

3.3 drv_scale

`drv_scale` (⟨*id*⟩, ⟨*float*⟩)

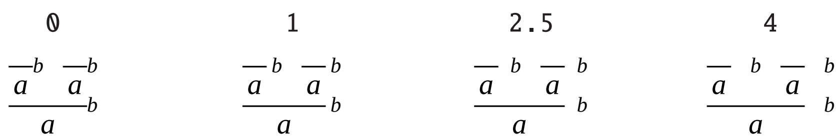
⟨*id*⟩	scale identifier (`clr`, `prm`, `jdg` or `ilb`)		
	`clr`	nice explanation soon (see examples)	∗ *default scale: 1* ∗
	`prm`	nice explanation soon (see examples)	∗ *default scale: 1* ∗
	`jdg`	nice explanation soon (see examples)	∗ *default scale: 1* ∗
	`ilb`	nice explanation soon (see examples)	∗ *default scale: 1* ∗
⟨*float*⟩	scale value		

“drv_scale (clr, —);”

“drv_scale (prm, —);”

“drv_scale (jgm, —);”

“drv_scale (ilb, —);”

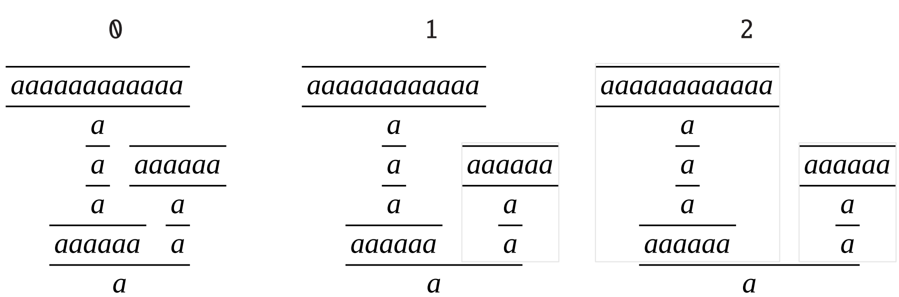


3.4 drv_junction_style

This macro sets the default way the premises of an inference are horizontally joined.

drv_junction_style ⟨*id*⟩
⟨*id*⟩ junction style identifier (0, 1 or 2)

- 0 “fully-interlacing”
- 1 “semi-interlacing” ∗ *default* ∗
- 2 “non-interlacing”



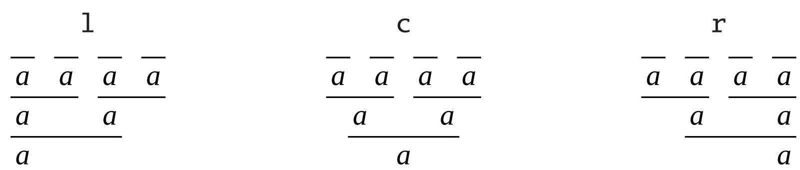
3.5 drv_alignment_style

This macro sets the default way a judgment is horizontally aligned relatively to its premises.

`drv_alignment_style` ⟨*id*⟩

⟨*id*⟩ alignment style identifier (`l`, `c` or `r`)

- `l` left
- `c` centered ∗ *default* ∗
- `r` right



3.6 drv_path_style

`drv_path_style (`⟨*id 1*⟩`,` ⟨*id 2*⟩`)`

⟨*id 1*⟩ path-type identifier (`iln` or `phm`)

- `iln` inference lines ∗ *default style:* `1` ∗
- `phm` phantom steps paths ∗ *default style:* `3` ∗

⟨*id 2*⟩ path-style identifier (`0`, `1`, `2`, `3`, `4`, `5` or `6`)

3.7 drv_labels_position

`drv_labels_position (`⟨*id 1*⟩`,` ⟨*id 2*⟩`)`

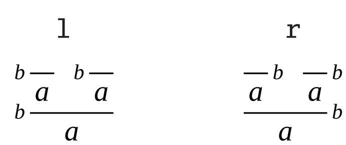
⟨*id 1*⟩ label-type identifier (`ilb`, `plb` or `dlb`)

- `ilb` inference labels ∗ *default position:* `r` ∗
- `dlb` delimiter labels ∗ *default position:* `l` ∗
- `plb` phantom steps labels ∗ *default position:* `l` ∗

⟨*id 2*⟩ position identifier (`l` or `r`)

- `l` left
- `r` right

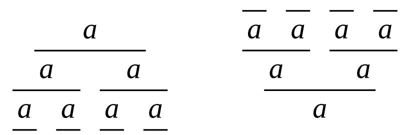
"`drv_labels_position (ilb, —);`"



Setting a default position for delimiter labels (thus for delimiters) and for phantom steps labels may be useful in conjunction with declaration macros taking optional label arguments (see § 5.2).

3.8 drv_roots_position

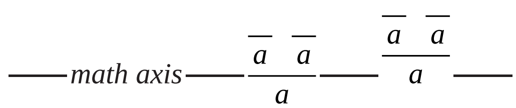
drv_roots_position ⟨*id*⟩
⟨*id*⟩ position identifier (t or b)
t top
b bottom ∗ *default* ∗



3.9 drv_axis_reference

The baseline of derivation tree pictures is set in such a way that their math axis coincides either with the axis of their root inference line or with the math axis of their root judgment according to the default behaviour set by drv_axis_reference.

drv_axis_reference ⟨*id*⟩
⟨*id*⟩ reference identifier (iln or jdg)
iln root inference line axis ∗ *default* ∗
jdg root judgment math axis



Notice that drv_axis_reference is irrelevant if you don't use the \drv inclusion command (see § 1.3).

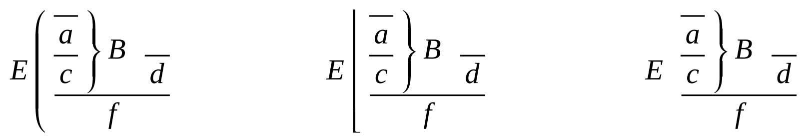
3.10 drv_left_delimiter and drv_right_delimiter

drv_left_delimiter ⟨*str*⟩
⟨*str*⟩ left delimiter math-mode LaTeX code
"(" (
"[" [
"\lbrace" { ∗ *default* ∗
"\langle" ⟨
etc.

drv_right_delimiter ⟨*str*⟩
⟨*str*⟩ right delimiter math-mode LaTeX code
")")
"]"]
"\rbrace" } ∗ *default* ∗
"\rangle" ⟩
etc.

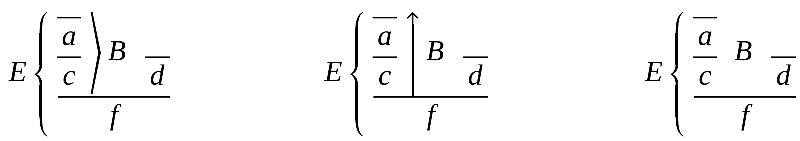
"drv_left_delimiter —;"

"(" "\lfloor" "."



"drv_right_delimiter —;"

"\rangle" "\uparrow" "."



3.11 drv_box_mode

When in "box mode", derivation trees are typeset in such a way that all sub-trees behave as if they were enclosed within boxes (that is as if all premise indices were prefixed with bxd, see § 2.3).

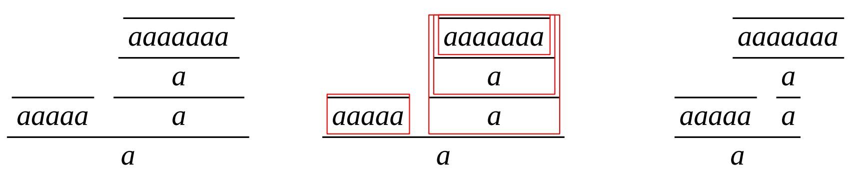
drv_box_mode ⟨*id*⟩

⟨*id*⟩ status identifier (on or off)

on

off ∗ *default* ∗

on typeset as off



3.12 drv_fraction_mode

drv typesets derivation trees in such a way that: the distance from the axis of an inference line to the math axis of a judgment above it is always the same (num_hg, see § 6.1); the distance from the axis of an inference line to the math axis of a judgment below it is always the same (den_dp, see § 6.1). When in "fraction mode", if roots are at bottom then the height of leaf judgments above which there is no inference line is ignored (the depth of root judgments is always ignored); if roots are at top then the depth of leaf judgments below which there is no inference line is ignored (the height of root judgments is always ignored). This mode may cause overlaps when used in conjunction with interlacing junction-styles (0 and 1).

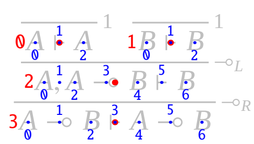
drv_fraction_mode ⟨*id*⟩
⟨*id*⟩ status identifier (on or off)
 on ∗ *default* ∗
 off

on	off	typeset as
$\dfrac{\dfrac{\overbrace{A,\Gamma\vdash B}\quad\overline{B,\Delta\vdash C}}{A,\Gamma,\Delta\vdash C}\,\text{cut}}{\Gamma,\Delta\vdash A\to C}\,{\to_R}$	$\dfrac{\dfrac{\overbrace{A,\Gamma\vdash B}\quad\overline{B,\Delta\vdash C}}{A,\Gamma,\Delta\vdash C}\,\text{cut}}{\Gamma,\Delta\vdash A\to C}\,{\to_R}$	$\dfrac{\dfrac{\overline{\overbrace{A,\Gamma\vdash B}}\quad\overline{B,\Delta\vdash C}}{A,\Gamma,\Delta\vdash C}\,\text{cut}}{\Gamma,\Delta\vdash A\to C}\,{\to_R}$

3.13 drv_proof_mode

drv_proof_mode ⟨*id*⟩
⟨*id*⟩ status identifier (on or off)
 on
 off ∗ *default* ∗

on



off

$$\dfrac{\dfrac{\dfrac{}{A\vdash A}\,1\quad\dfrac{}{B\vdash B}\,1}{A, A\multimap B\vdash B}\,{\multimap_L}}{A\multimap B\vdash A\multimap B}\,{\multimap_R}$$

Red numbers (resp. dots) refer to judgment indices (resp. central points, see § 6.1) while blue numbers (resp. dots) refer to sub-judgment indices (resp. central points, see § 6.1).

4 Pictures, bounding boxes and math axis

4.1 drv_freeze and drv_tree

drv composes derivation trees with respect to judgment and inference declarations only once the drv_freeze macro is called. This is usually done by drv_tree which is a macro that returns a picture. You may however call drv_freeze yourself if you have no need for the whole derivation tree picture that drv_tree would otherwise return (Section 6.1 illustrates such a situation).

drv composes derivation trees essentially according to the algorithm for composing fractions described in Appendix G of the TEXbook (see [2, 3]). In particular, `drv` uses "`\fontdimen`" parameters so that the derivation tree pictures it generates should integrate smoothly within any document, whatever the fonts you use. As an example, compare the following fractions (the first one is composed by `drv` while the second one is composed by the standard `\frac` command).

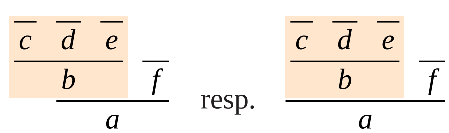
$$\frac{\gamma}{\delta} \quad \frac{\gamma}{\delta}$$

4.2 drv_bbox

`drv_bbox` ⟨*nat*⟩
⟨*nat*⟩ sub-tree root index

"`drv_bbox` ⟨*nat*⟩" returns a METAPOST closed path (see [1, Section 4]) standing for the bounding box of the sub-tree ending with the index ⟨*nat*⟩ judgment. `drv_bbox` calls `drv_freeze` if necessary.

```
      beginfig(410)
      dcl 0 (1, 5) ("", _)      "a";
resp. dcl 0 (bxd 1, 5) ("", _)  "a";
        dcl 1 (2, 3, 4) ("", _)   "b";
          dcl 2 () ("", _)          "c";
          dcl 3 () ("", _)          "d";
          dcl 4 () ("", _)          "e";
        dcl 5 () ("", _)          "f";
      fill drv_bbox 1 withcolor (255, 230, 205)/255; % rgb color
      draw drv_tree;
      endfig;
```

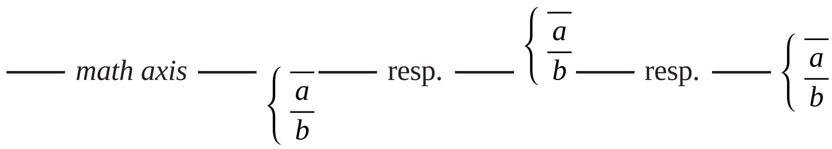


4.3 drv_axis

`drv_axis` locally overrides `drv_axis_reference` (see § 3.9), allowing you to explicitly set the math axis of a tree once it has been drawn.

`drv_axis` (⟨*id*⟩, ⟨*nat*⟩)
⟨*id*⟩ reference type identifier (`iln`, `jdg` or `dlm`)
  `iln` inference line axis
  `jdg` judgment math axis
  `dlm` delimiter axis
⟨*nat*⟩ reference index

```
      beginfig(420)
      Dcl 0 ( ) ("",   "", "", _) "a";
      Dcl 1 (0) ("", "{}", "", _) "b";
      draw drv_tree;
      drv_axis (iln, 0);
resp. drv_axis (jdg, 1);
resp. drv_axis (dlm, 1);
      endfig;
```



Notice that `drv_axis` is irrelevant if you don't use the `\drv` inclusion command.

5 Low level inference declaration macros

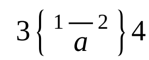
5.1 NFR, DCL and MVD

NFR The `NFR` declaration macro is the lowest level one. It allows you to specify all the labels and styles of an inference.

`NFR` ⟨*nat*⟩ (⟨*nat list*⟩) (⟨*str 1*⟩, ⟨*str 2*⟩, ⟨*str 3*⟩, ⟨*str 4*⟩, ⟨*id 1*⟩, ⟨*id 2*⟩, ⟨*id 3*⟩)

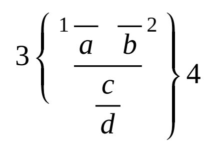
⟨*nat*⟩	inference index
⟨*nat list*⟩	list of premise indices
⟨*str 1*⟩	left inference label *math-mode* LaTeX code
⟨*str 2*⟩	right inference label *math-mode* LaTeX code
⟨*str 3*⟩	left delimiter label *math-mode* LaTeX code
⟨*str 4*⟩	right delimiter label *math-mode* LaTeX code
⟨*id 1*⟩	junction style identifier (`0`, `1`, `2`, `3` or `_`) `0` fully-interlacing `1` semi-interlacing `2` non-interlacing `3` user specified (tricky, see § 6.3) `_` default (set by `drv_junction_style`, see § 3.4)
⟨*id 2*⟩	alignment style identifier (`l`, `c`, `r`, `u` or `_`) `l` left `c` centered `r` right `u` user specified (tricky, see § 6.3) `_` default (set by `drv_alignment_style`, see § 3.5)
⟨*id 3*⟩	inference line style identifier (`0`, `1`, `2`, `3`, `4`, `5`, `6` or `_`)

```
beginfig(430)
jgm 0 "a";
NFR 0 () ("1", "2", "3", "4", _, _, _);
draw drv_tree;
endfig;
```



DCL The DCL declaration macro is a shorthand for `jgm` and `NFR` in the same way as `dcl` is a shorthand for `jgm` and `nfr`.

```
beginfig(440)
DCL 0 ()     ("1", "", "", "", _, _, _) "a";
DCL 1 ()     ("", "2", "", "", _, _, _) "b";
DCL 2 (0, 1) ("", "", "3", "", _, _, _) "c";
DCL 3 (2)    ("", "", "", "4", _, _, _) "d";
draw drv_tree;
endfig;
```

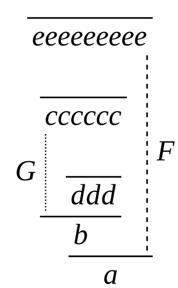


MVD The `MVD` macro is a generalization of `Mvd` that allows you to specify the alignment style of phantom inferences.

`MVD` ⟨*nat 1*⟩ `(`⟨*nat 2*⟩`,` ⟨*str 1*⟩`,` ⟨*str 2*⟩`,` ⟨*id 1*⟩`,` ⟨*id 2*⟩`)`

⟨*nat 1*⟩	index of the origin judgment
⟨*nat 2*⟩	number of phantom steps
⟨*str 1*⟩	left label *math-mode* LaTeX code
⟨*str 2*⟩	right label *math-mode* LaTeX code
⟨*id 1*⟩	alignment style identifier (`l`, `c` or `r`)
⟨*id 2*⟩	phantom steps path-style identifier (`0`, `1`, `2`, `3`, `4`, `5`, `6` or `_`)

```
beginfig(450)
jgm 0 "a";
  jgm 1 "b";
    jgm 2 "cccccc";
    jgm 3 "ddd";
  jgm 4 "eeeeeeeee";
nfr 0 (1, MVD 4 (5, "", "F", r, 4)) ("", _);
  nfr 1 (MVD 2 (2, "G", "", l, 3), 3) ("", _);
    nfr 2 () ("", _);
    nfr 3 () ("", _);
  nfr 4 () ("", _);
draw drv_tree;
endfig;
```



```
beginfig(460)
jgm 0 "\textsc{Size matters -- Part 2}";
jgm 1 "\text{Here is an even longer judgment"&
      " that I don't want to shorten either.}";
jgm 2 "\text{This time I'm pretty sure that the"&
      " derivation tree won't fit on the page.}";
jgm 3 "\text{It does! Amazing.}";
nfr 0 () ("", 0);
nfr 1 (0) ("", 1);
nfr 2 () ("", 1);
nfr 3 (MVD 1 (2, "", "", l, 3), 2) ("", 1);
draw drv_tree;
endfig;
```

SIZE MATTERS – PART 2

Here is an even longer judgment that I don't want to shorten either.

This time I'm pretty sure that the derivation tree won't fit on the page.

It does! Amazing.

5.2 Optional labels

NFR_opt The NFR_opt declaration macro is an alternative for NFR that lets you specify labels at your option.

NFR_opt ⟨*nat*⟩ (⟨*nat list*⟩) (⟨*str list 1*⟩) (⟨*str list 2*⟩) (⟨*id 1*⟩, ⟨*id 2*⟩, ⟨*id 3*⟩)

⟨*nat*⟩	inference index
⟨*nat list*⟩	list of premise indices
⟨*str list 1*⟩	list of inference labels *math-mode* LaTeX code
⟨*str list 2*⟩	list of delimiter labels *math-mode* LaTeX code
⟨*id 1*⟩	junction style identifier (0, 1, 2, 3 or _)
⟨*id 2*⟩	alignment style identifier (l, c, r, u or _)
⟨*id 3*⟩	inference line style identifier (0, 1, 2, 3, 4, 5, 6 or _)

The list ⟨*str list 1*⟩ may contain zero, one or two strings specifying inference labels. If no label is specified then no label is attached to the inference line. If two labels are specified then the first one is attached to the left and the second one to the right. Finally, if one label only is specified then it is attached either to the left or to the right of the inference line depending on the default inference labels position set by drv_labels_position (see § 3.7).

The same way, ⟨*str list 2*⟩ may contain zero, one or two strings specifying delimiter labels. If one label only is specified then it is attached to a delimiter placed either to the left or to the right of the sub-tree ending with the index ⟨*nat*⟩ judgment depending on the default delimiter labels position set by `drv_labels_position`.

As an example, "`nfr` ⟨*nat*⟩ `(`⟨*nat list*⟩`)` `(`⟨*str*⟩`,` ⟨*id*⟩`)`" behaves exactly the same way as "`NFR_opt` ⟨*nat*⟩ `(`⟨*nat list*⟩`)` `(`⟨*str*⟩`)` `()` `(_, _,` ⟨*id*⟩`)`".

`DCL_opt` The `DCL_opt` declaration macro is a shorthand for `jgm` and `NFR_opt` in the same way as `DCL` is a shorthand for `jgm` and `NFR`.

`MVD_opt` The `MVD_opt` macro is an alternative for `MVD` that lets you specify labels at your option.

`MVD_opt` ⟨*nat 1*⟩ `(`⟨*nat 2*⟩`)` `(`⟨*str list*⟩`)` `(`⟨*id 1*⟩ ⟨*id 2*⟩`)`

⟨*nat 1*⟩	index of the origin judgment
⟨*nat 2*⟩	number of phantom steps
⟨*str list*⟩	list of labels *math-mode* LaTeX code
⟨*id 1*⟩	alignment style identifier (`l`, `c` or `r`)
⟨*id 2*⟩	phantom steps path-style identifier (`0`, `1`, `2`, `3`, `4`, `5`, `6` or `_`)

The list ⟨*str list*⟩ may contain zero, one or two strings specifying labels. If one label only is specified then it is attached either to the left or to the right of the phantom steps path depending on the default phantom steps labels position set by `drv_labels_position` (see § 3.7).

5.3 User defined declaration macros (tricky)

Here are the METAPOST headers for `NFR`, `MVD`, `NFR_opt` and `MVD_opt`.

```
NFR[](text PRM)(expr lilb, rilb, ldlb, rdlb)(suffix jsty, asty, isty)
MVD[](expr num, lplb, rplb)(suffix asty, psty)
NFR_opt[](text PRM)(text ILB)(text DLB)(suffix jsty, asty, isty)
MVD_opt[](expr num)(text PLB)(suffix asty, psty)
```

"`[]`" in the header of a macro indicates that this macro expects a numeric argument referred to as "`@`" in its body. "`text`", "`expr`" and "`suffix`" specify argument types (see [1, Section 10]). You may use `NFR`, `MVD`, `NFR_opt` and `MVD_opt` to define your own declaration macros. As an example, here are possible definitions for `Nfr` and `Mvd`.

```
vardef Nfr[](text PRM)(expr ilb, ldlb, rdlb)(suffix isty)=
  NFR_opt[@](PRM)(ilb)(ldlb, rdlb)(_, _, isty);
enddef;

vardef Mvd[](expr num, lplb, rplb)(suffix psty)=
  MVD[@](num, lplb, rplb, _, psty) % Mvd returns an index, no ‘;’!
enddef;
```

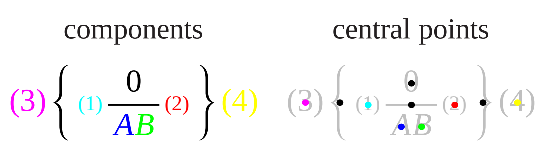
6 Inside derivation trees

6.1 Components, distinguished points and dimensions

Components Once `drv_freeze` has been called, all the components of a derivation tree are accessible independently from each-other.

```
beginfig(470) % components
DCL 6 () ("", "", "", "", _, _, 0) "0";
DCL 7 (6) ("(1)", "(2)", "(3)", "(4)", _, _, 1) "A", "B";
drv_freeze; % usually called by drv_tree
draw jdg[6];                           % judgment             0
draw sbj[7][0] withcolor (0, 0, 1); % sub-judgment         A
draw sbj[7][1] withcolor (0, 1, 0); % sub-judgment         B
draw l_ilb[7]  withcolor (0, 1, 1); % left inference label  (1)
draw r_ilb[7]  withcolor (1, 0, 0); % right inference label (2)
draw l_dlb[7]  withcolor (1, 0, 1); % left delimiter label  (3)
draw r_dlb[7]  withcolor (1, 1, 0); % right delimiter label (4)
draw l_dlm[7];                         % left delimiter
draw iln[7];                           % inference line
draw r_dlm[7];                         % right delimiter
endfig;
```

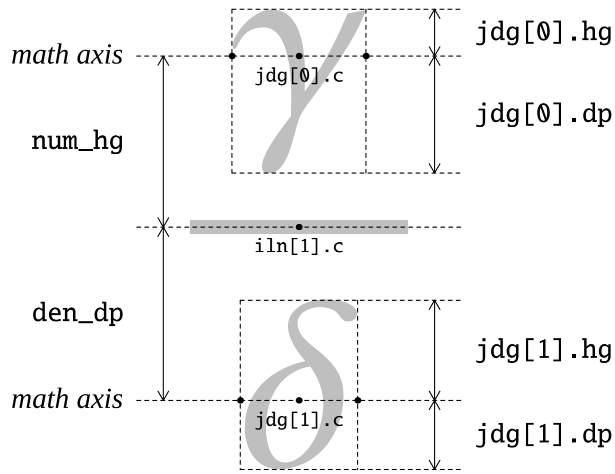
Distinguished points Three distinguished points are associated with each component ⟨*cpn*⟩, namely ⟨*cpn*⟩`.l`, ⟨*cpn*⟩`.c` and ⟨*cpn*⟩`.r` that lie respectively to the left, at the center and to the right of the component math axis.



Dimensions Two dimensions are associated with each component ⟨*cpn*⟩, a depth ⟨*cpn*⟩`.dp` and a height ⟨*cpn*⟩`.hg` that both are relative to the component math axis. Two overall dimensions are associated with each derivation tree, `den_dp` and `num_hg`. The former refers to the depth of a judgment math axis relatively to the axis of an inference line above it while the latter refers to the height of a judgment math axis relatively to the axis of an inference line below it. Depths are negative while heights are positive.

```
beginfig(470)
dcl 0 ()  ("", 0) "\gamma";
dcl 1 (0) ("", 1) "\delta";
draw drv_tree;
endfig;
```

(The picture below may look weird if you don't use scalable fonts.)



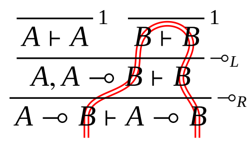
6.2 drv_styled

drv_styled allows you to draw METAPOST paths using drv path-styles.

⟨*path*⟩ drv_styled ⟨*id*⟩

⟨*path*⟩	METAPOST path expression
⟨*id*⟩	path style identifier (0, 1, 2, 3, 4, 5 or 6)

```
beginfig(490)
jgm 4 "A", "\vdash", "A";
jgm 5 "B", "\vdash", "B";
jgm 6 "A", ",", "A", "\multimap", "B", "\vdash", "B";
jgm 7 "A", "\multimap", "B", "\vdash", "A", "\multimap", "B";
nfr 4 () ("1", _);
nfr 5 () ("1", _);
nfr 6 (4, 5) ("\multimap_{L}", _);
nfr 7 (6) ("\multimap_{R}", _);
drv_freeze;
draw (sbj[7][2].c shifted (0, -num_hg) ..
      sbj[7][2].c {up} ..
      sbj[6][4].c ..
      sbj[5][0].c .. tension 1.05 ..
      sbj[5][2].c ..
      sbj[6][6].c ..
      sbj[7][6].c {down} ..
      sbj[7][6].c shifted (0, -num_hg))
drv_styled 2 withcolor (1, 0, 0);
draw drv_tree;
endfig;
```



6.3 User specified junction and alignment styles (tricky)

`drv` composes derivation trees by stating geometrical constraints to be solved by METAPOST. These constraints express how the components of a derivation tree must be arranged with respect to each-other. In the example about dimensions (see above), such a constraint could be that the vertical distance from `iln[1].c` to `jdg[0].c` has to be `num_hg`, which could be *stated* in the METAPOST syntax as "`ypart jdg[0].c=ypart iln[1].c+num_hg`" (this is *not* an affectation).

You can prevent `drv` from stating *horizontal* constraints about premises junction or judgments alignment by using the junction style `3` or the alignment style `u` of the `NFR` and `NFR_opt` macros (see § 5.1, 5.2). In such cases, you have to state your own constraints. All the constraints related to a derivation tree must be stated *before* `drv_freeze` is called. METAPOST will complain if the constraints you state are insufficient, redundant or inconsistent.

User specified junction style The *horizontal* constraints you state should express how the premises of the inference have to be joined.

```
beginfig(500)
jgm 0 "{\cdot}";
jgm 1 "{\cdot}";
jgm 2 "\text{You may check that the distance"&
      " between the two dots above is 5 cm.}";
NFR 0 () ("", "", "", "", _, _, _);
NFR 1 () ("", "", "", "", _, _, _);
NFR 2 (0, 1) ("", "", "", "", 3, _, _); % caution: 3
xpart jdg[1].c=xpart jdg[0].c+5cm;
draw drv_tree;
endfig;
```

$$\frac{\overline{\cdot} \qquad\qquad\qquad\qquad \overline{\cdot}}{\text{You may check that the distance between the two dots above is 5 cm.}}$$

User specified alignment style The *horizontal* constraints you state should express how the inferred judgment has to be aligned with respect to its premises.

```
     beginfig(510)                                         % "\vdash":
     jgm 0 "B, A, \Gamma", "\vdash", "C";                  % sbj[0][1]
     jgm 1 "A, \Gamma", "\vdash", "B\multimap C";          % sbj[1][1]
     jgm 2 "\Gamma", "\vdash", "A\multimap(B\multimap C)"; % sbj[2][1]
     NFR_opt 0 () () () (_, _, 0);
     NFR_opt 1 (0) ("\multimap_R") () (_, u, 1);           % caution: u
     NFR_opt 2 (1) ("\multimap_R") () (_, u, 1);           % caution: u
     xpart sbj[1][1].c=xpart sbj[0][1].c;
resp. xpart sbj[1][1].l=xpart sbj[0][1].r;
     xpart sbj[2][1].c=xpart sbj[1][1].c;
resp. xpart sbj[2][1].l=xpart sbj[1][1].r;
     draw drv_tree;
     endfig;
```

$$\dfrac{\dfrac{B, A, \Gamma \vdash C}{A, \Gamma \vdash B \multimap C}\multimap_R}{\Gamma \vdash A \multimap (B \multimap C)}\multimap_R \quad \text{resp.} \quad \dfrac{\dfrac{B, A, \Gamma \vdash C}{A, \Gamma \vdash B \multimap C}\multimap_R}{\Gamma \vdash A \multimap (B \multimap C)}\multimap_R$$

References

[1] John D. HOBBY. *A User's Manual for METAPOST*, 2009

[2] Bogusław JACKOWSKI. *Appendix G illuminated*. *TUGboat*, 27(1):83–90, 2006.

[3] Donald E. KNUTH. *The TEXbook*. Addison-Wesley, 1984.

[4] Greg RESTALL. *Proof Theory and Philosophy*. Book in progress, 2006.

[5] Denis ROEGEL. *The MetaObj tutorial and reference manual*, 2002.

[6] Lutz STRASSBURGER. *Proof Nets and the Identity of Proofs*. INRIA, 2006.

A Debugging and proofing

Debugging

Recall that you have to run "`mpost` ⟨*jobname*⟩`.mp`" *at least twice* (once more if you use sub-tree delimiters). If you get an error on the first run then it comes from the `drv`/METAPOST code. If you get an error on the second run then it comes from the LATEX code.

Error on the first run METAPOST behaves essentially as TEX/LATEX when it finds and error (see [1, Debugging]). It stops, “explains” the error in some way (look for the line starting with an exclamation mark `!`), shows some lines of context, and asks you what to do next (answer `h` to get some help or `x` to terminate the run). If you’re lucky, the error comes from an inconsistency that `drv` can detect. In such a case the explanation should be quite understandable.

```
46 beginfig(520)
47 jgm 0 "A\vdash B";
48 jgm 1 "B\vdash C";
49 jgm 2 "A\vdash C";
50 jgm 3 "C\vdash D";
51 jgm 4 "A\vdash D";
52 nfr 0 () ("f", _);
53 nfr 1 () ("g", _);
54 nfr 2 (0, 1) ("\circ", _);
55 nfr 3 () ("h", _);
56 nfr 4 (0, 3) ("\circ", _);
57 draw drv_tree;
58 endfig;
```

METAPOST error message.

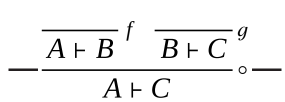
```
! drv (fig. 520): 0 has been used
already as a premise index for
inference declaration 2.
⟨error context⟩
l.56 nfr 4 (0, 3) ("\circ", 1)
                                ;
?
```

Error on the second run METAPOST fails to preprocess the LATEX code in ⟨*jobname*⟩`-delayed.mp` and suggests that you “`see mpxerr.log`” which is a regular LATEX log-file. This file shows you which part of the LATEX code is faulty but unfortunately not where to find it in ⟨*jobname*⟩`.mp`.

Proofing

Recall that for each “`beginfig(`⟨*index*⟩`)`, `endfig;`” pair in a file ⟨*jobname*⟩`.mp`, METAPOST generates an Encapsulated PostScript file ⟨*jobname*⟩`.`⟨*index*⟩. In addition, `drv` generates a LATEX file ⟨*jobname*⟩`-proof.tex` that contains a copy of the LATEX preamble in ⟨*jobname*⟩`.mp` and includes each ⟨*jobname*⟩`.`⟨*index*⟩[2] picture file using the `\drv` inclusion command, together with some information about its text size, math style and math axis. As an example, regarding the first figure on page 5, processing `drv-guide-proof.tex` produces:

`drv-guide.110` (`\normalsize`, `\displaystyle`)



[2]To METAPOST users: proof file generation does not take `outputtemplate` into account yet.

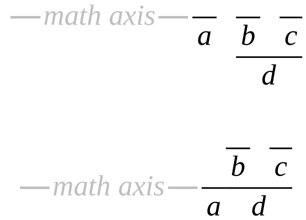
B Derivation forests

You may declare several derivation trees within a single figure environment. Then trees are drawn from left to right in the order their roots are declared, and in such a way that: root judgments are horizontally aligned; the distance between two trees is the same as the distance between two non-interleaving premises (and thus is affected by `drv_scale`, see § 3.3).

```
beginfig(530)
% first tree
dcl 10 () ("", _) "a";
% second tree
dcl 20 (21, 22) ("", _) "d";
  dcl 21 () ("", _) "b";
  dcl 22 () ("", _) "c";
draw drv_tree;
endfig;
```

You can however state constraints (horizontal or vertical, at your option) specifying the relative positioning of trees before `drv_freeze` is called.

```
beginfig(531)
% first tree
dcl 10 () ("", _) "a";
% second tree
dcl 20 (21, 22) ("", _) "d";
  dcl 21 () ("", _) "b";
  dcl 22 () ("", _) "c";
% relative positionning
ypart jdg[10].c=ypart jdg[22].c;
resp. xpart iln[10].r=xpart iln[20].l;
draw drv_tree;
endfig;
```



Notice that the math axis of a forest is set according to `drv_axis_reference` (see § 3.9) relatively to the first declared root (you can override this behaviour by using `drv_axis`, see § 4.3).

`drv_root`

`drv_root` locally overrides `drv_roots_position` (see § 3.8), allowing you to explicitly set the position of a root.

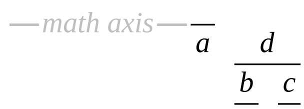
`drv_root` (⟨*nat*⟩, ⟨*id*⟩)

⟨*nat*⟩ root index

⟨*id*⟩ position identifier (`t` or `b`)

- `t` top
- `b` bottom

```
beginfig(533)
% first tree
dcl 10 () ("", _) "a";
% second tree
dcl 20 (21, 22) ("", _) "d";
  dcl 21 () ("", _) "b";
  dcl 22 () ("", _) "c";
drv_root (20, t); % root at top!
draw drv_tree;
endfig;
```



Then again, you can state constraints specifying the relative positioning (e.g., the overlapping) of trees before `drv_freeze` is called.

```
drv_left_delimiter "\downarrow";
drv_right_delimiter "\uparrow";

beginfig(540)
% first tree
jgm 10 "A\vdash D";
Nfr 10 (11, 14) ("\circ", "h\circ (g\circ f)", "", 1);
  dcl 11 (12, 13) ("\circ", 1) "A\vdash C";
    dcl 12 () ("f", 2) "A\vdash B";
    dcl 13 () ("g", 3) "B\vdash C";
  dcl 14 () ("h", 4) "C\vdash D";
% second tree
jgm 20 "\phantom{A\vdash D}"; % hidden judgment
Nfr 20 (21, 22) ("\circ", "", "(h\circ g)\circ f", 1);
  dcl 21 () ("f", 2) "A\vdash B";
  dcl 22 (23, 24) ("\circ", 1) "B\vdash D";
    dcl 23 () ("g", 3) "B\vdash C";
    dcl 24 () ("h", 4) "C\vdash D";
drv_root (20, t); % root at top!
% overlapping
jdg[10].c=jdg[20].c;
draw drv_tree;
endfig;
```

(The resulting figure is in Appendix D on page 33.)

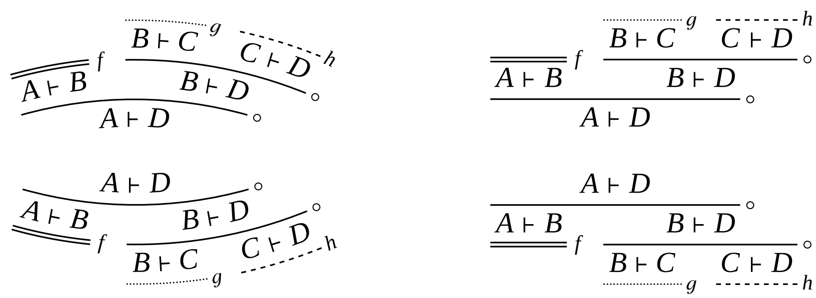
C Radial mode (beta version)

A few more tuning macros (see § 3) are available that allow you to manipulate *radial* trees rather than "linear" ones.

`drv_radial_mode`

`drv_radial_mode` ⟨*id*⟩
 ⟨*id*⟩ status identifier (`on` or `off`)
 `on`
 `off` ∗ *default* ∗

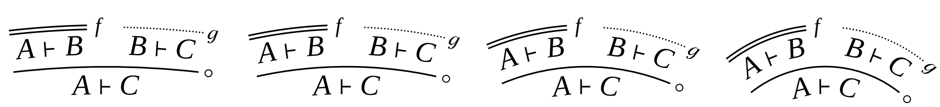
`on` `off`



`drv_scale (crv, —)`

`drv_scale (crv,` ⟨*float*⟩`)`
 `crv` scale identifier
 ⟨*float*⟩ scale value ∗ *default: 1* ∗

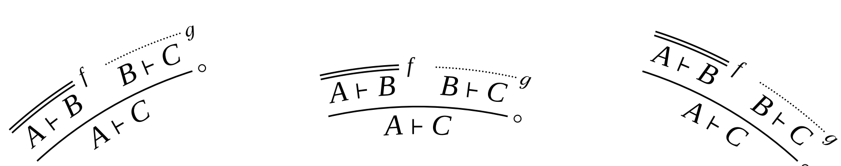
`0.5` `1` `2` `4`



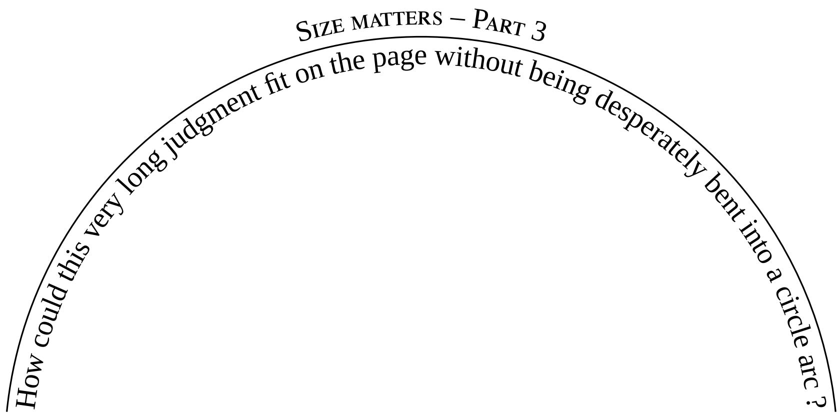
`drv_azimuth`

`drv_azimuth` ⟨*float*⟩
 ⟨*float*⟩ azimuth degree ∗ *default: 90* ∗

`120` `90` `60`



The azimuth of a derivation tree is that of the central point of its root judgment. Notice that both `drv_scale (crv, —)` and `drv_azimuth` are irrelevant when not in radial mode.

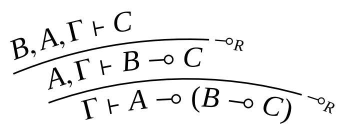


User specified junction and alignment styles (tricky)

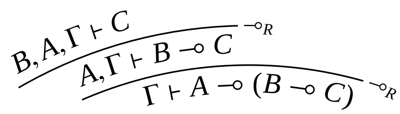
When in radial mode, `drv` composes derivation trees by stating *angular* constraints rather than horizontal ones. To this end, three distinguished angles are associated with each component ⟨*cpn*⟩, namely ⟨*cpn*⟩`.lng`, ⟨*cpn*⟩`.cng` and ⟨*cpn*⟩`.rng` that refer to the relative angles of ⟨*cpn*⟩`.l`, ⟨*cpn*⟩`.c` and ⟨*cpn*⟩`.r` respectively. (Radial constraints are essentially the same as vertical ones; however, each component comes also with a radius ⟨*cpn*⟩`.rad` and an origin point ⟨*cpn*⟩`.org`.)

You can prevent `drv` from stating angular constraints about premises junction or judgments alignment by using the junction style `3` or the alignment style `u` of the `NFR` and `NFR_opt` macros (see § 5.1, 5.2). Then again, you have to state your own constraints. As an example, compare the code below with the one for figure 510 on page 26.

```
     beginfig(590)                                         % "\vdash":
     jgm 0 "B, A, \Gamma", "\vdash", "C";                  % sbj[0][1]
     jgm 1 "A, \Gamma", "\vdash", "B\multimap C";          % sbj[1][1]
     jgm 2 "\Gamma", "\vdash", "A\multimap(B\multimap C)";  % sbj[2][1]
     NFR_opt 0 () () () (_, _, 0);
     NFR_opt 1 (0) ("\multimap_R") () (_, u, 1);           % caution: u
     NFR_opt 2 (1) ("\multimap_R") () (_, u, 1);           % caution: u
     sbj[1][1].cng=sbj[0][1].cng;
resp. sbj[1][1].lng=sbj[0][1].rng;
     sbj[2][1].cng=sbj[1][1].cng;
resp. sbj[2][1].lng=sbj[1][1].rng;
     draw drv_tree;
     endfig;
```



resp.



D Gallery

Here are two simple derivation trees (figures 600, 601).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a \vdash a}\,\text{id} \quad \dfrac{}{1 \vdash 1}\,\text{id}}{a \otimes a \multimap 1 \vdash 1}\,\overset{\multimap}{\eta}}{a \multimap 1 \vdash a \multimap 1}\,\overset{\multimap}{\varepsilon} \quad \dfrac{}{1 \vdash 1}\,\text{id}}{1 \multimapinv (a \multimap 1) \otimes a \multimap 1 \vdash 1}\,\overset{\multimapinv}{\eta}}{1 \multimapinv (a \multimap 1) \vdash 1 \multimapinv (a \multimap 1)}\,\overset{\multimapinv}{\varepsilon} \quad \dfrac{}{1 \vdash 1}\,\text{id}}{1 \multimapinv (a \multimap 1) \otimes (1 \multimapinv (a \multimap 1)) \multimap 1 \vdash 1}\,\overset{\multimap}{\eta}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a \vdash a}\,\text{id} \quad \dfrac{}{1 \vdash 1}\,\text{id}}{a \otimes a \multimap 1 \vdash 1}\,\overset{\multimap}{\eta}}{a \vdash 1 \multimapinv (a \multimap 1)}\,\overset{\multimapinv}{\varepsilon} \quad \dfrac{}{1 \vdash 1}\,\text{id}}{a \otimes (1 \multimapinv (a \multimap 1)) \multimap 1 \vdash 1}\,\overset{\multimap}{\eta}}{(1 \multimapinv (a \multimap 1)) \multimap 1 \vdash a \multimap 1}\,\overset{\multimap}{\varepsilon} \quad \dfrac{}{1 \vdash 1}\,\text{id}}{1 \multimapinv (a \multimap 1) \otimes (1 \multimapinv (a \multimap 1)) \multimap 1 \vdash 1}\,\overset{\multimapinv}{\eta}$$

Here are the `drv` version of a derivation tree found in [4, p. 57] and an alternative for it (figures 610, 611).

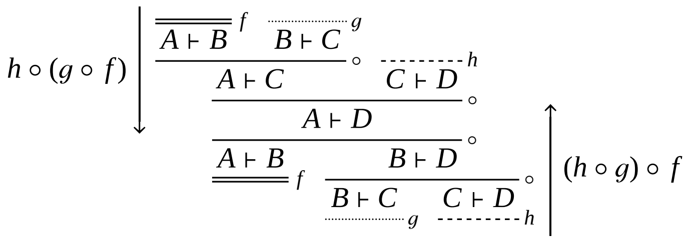
$$\dfrac{\dfrac{p \vdash p}{p \wedge (q \vee (r_1 \wedge r_2)) \vdash p}\,\wedge L_1 \quad \dfrac{\dfrac{\dfrac{q \vdash q}{q \vdash q \vee (r_1 \wedge r_2)}\,\vee R_1 \quad \dfrac{\dfrac{\dfrac{r_1 \vdash r_1}{r_1 \wedge r_2 \vdash r_1}\,\wedge L_1 \quad \dfrac{r_2 \vdash r_2}{r_1 \wedge r_2 \vdash r_2}\,\wedge L_2}{r_1 \wedge r_2 \vdash r_1 \wedge r_2}\,\wedge R}{r_1 \wedge r_2 \vdash q \vee (r_1 \wedge r_2)}\,\vee R_2}{q \vee (r_1 \wedge r_2) \vdash q \vee (r_1 \wedge r_2)}\,\vee L}{p \wedge (q \vee (r_1 \wedge r_2)) \vdash q \vee (r_1 \wedge r_2)}\,\wedge L_2}{p \wedge (q \vee (r_1 \wedge r_2)) \vdash p \wedge (q \vee (r_1 \wedge r_2))}\,\wedge R$$

$$\mathrm{Id}_{q \vee (r_1 \wedge r_2)}\left\{\begin{array}{c} \mathrm{Id}_{r_1 \wedge r_2}\left\{\dfrac{\dfrac{r_1 \vdash r_1}{r_1 \wedge r_2 \vdash r_1}\,\wedge L_1 \quad \dfrac{r_2 \vdash r_2}{r_1 \wedge r_2 \vdash r_2}\,\wedge L_2}{r_1 \wedge r_2 \vdash r_1 \wedge r_2}\,\wedge R\right. \\ \vdots \\ \dfrac{\dfrac{q \vdash q}{q \vdash q \vee (r_1 \wedge r_2)}\,\vee R_1 \quad \dfrac{}{r_1 \wedge r_2 \vdash q \vee (r_1 \wedge r_2)}\,\vee R_2}{q \vee (r_1 \wedge r_2) \vdash q \vee (r_1 \wedge r_2)}\,\vee L \end{array}\right.$$

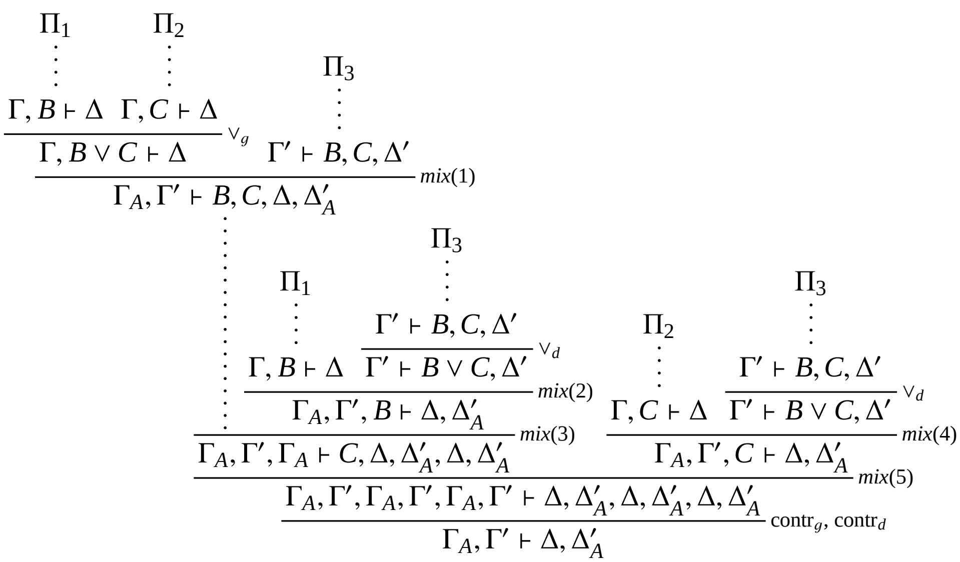
$$\vdots$$

$$\dfrac{\dfrac{p \vdash p}{p \wedge (q \vee (r_1 \wedge r_2)) \vdash p}\,\wedge L_1 \quad \dfrac{}{p \wedge (q \vee (r_1 \wedge r_2)) \vdash q \vee (r_1 \wedge r_2)}\,\wedge L_2}{p \wedge (q \vee (r_1 \wedge r_2)) \vdash p \wedge (q \vee (r_1 \wedge r_2))}\,\wedge R$$

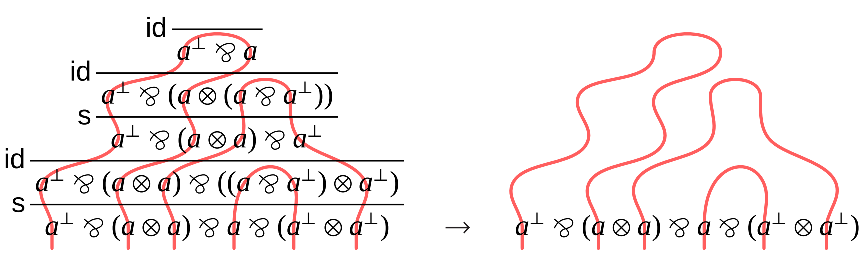
Here are overlapping trees with opposite directions (figure 540, code on page 29).



Here is the drv version of a derivation tree found in [5, p. 86] (figure 620).



Here are the drv versions of derivation trees found in [6, p. 50] (figures 630, 631).



Here is a continued fraction (figure 640).

$$1+\cfrac{a}{2+\cfrac{b}{3+\cfrac{c}{4+\cfrac{d}{\cdots}}}}$$

E Standalone picture files

Given a PostScript file ⟨*jobname*⟩.⟨*index*⟩ generated by METAPOST, you may get a standalone PDF file (with embedded fonts) ⟨*jobname*⟩-⟨*index*⟩`.pdf` by running

`mptopdf` ⟨*jobname*⟩.⟨*index*⟩

(`mptopdf` should be part of your TEX distribution). Next you may get a standalone PS file ⟨*jobname*⟩-⟨*index*⟩`.ps` by running

`pdftops` ⟨*jobname*⟩-⟨*index*⟩`.pdf`

(`pdftops` is part of the Xpdf software package). Finally you may get a standalone *transparent* PNG file ⟨*jobname*⟩-⟨*index*⟩`.png` by running

`convert` ⟨*jobname*⟩-⟨*index*⟩`.ps` ⟨*jobname*⟩-⟨*index*⟩`.png`

(`convert` is part of the ImageMagick software package). Notice that you can run `convert` on ⟨*jobname*⟩-⟨*index*⟩`.pdf` but then the PNG file you get is not transparent.

F Related packages

- `bussproofs.sty` (Samuel R. BUSS);
- `mathpartir.sty` (Didier RÉMY);
- `proof.sty` (Makoto TATSUTA);
- `prooftree.sty` (Paul TAYLOR);
- the `Ptree` constructor from `metaobj.mp` (Denis ROEGEL, see [5]);
- `semantic.sty` (Peter Møller NEERGAARD and Arne John GLENSTRUP);
- `trfrac.sty` (Kevin W. HAMLEN);
- `virginialake.sty` (Alessio GUGLIELMI).

Some of these are described on Peter SMITH's "LATEX for Logicians" webpage.