A Package for Logical Symbols

Rett Bull

October 10, 2009

The package `logicsym` supplies logical symbols for classes like Pomona College's Math 123, CS 80, and CS 81.

Known conflicts: the `logicsym` package redefines existing symbols `\mp` (minus-plus) and `\Re`.

1 Options and Customization

1.1 Compatiblity

This version introduces a number of name changes—for consistency. Use the option `oldnames` to add the previous names.

1.2 Truth Constants

There are three options for truth constants: `topbottom`, `tfletters`, and `tfwords`. The commands `\False` and `\True` may be used in any mode. The default font for words and letters is italic, more specifically `\mathit`, but it may be changed by redefining `\Truthconstantfont`.

The default is `topbottom`. Only one of the three options may be used.

1.3 Conjunction

The options for conjunction are `wedge` and `ampersand`. The default is `wedge`.

1.4 Implication

The choices for arrows are `singlearrows` or `doublearrows`. The default is `doublearrows`.

The "if and only if" connective matches the arrow style but can be replaced with $\equiv$ by specifying the option `equivalence`.

1.5 Turnstiles

There is no option for turnstiles, but their size (relative to other symbols) may be adjusted by redefining `\TurnstileScale`. The default is 1.0.

1.6 Crossing Out

In natural deduction inferences, we often want to show that an assumption has been discharged by crossing it out. A slash often works well for single letters, but it is too narrow for longer formulas. We provide a macro `\Crossbox` to make a wide slash, but unfortunately it is difficult to draw a simple diagonal across a TEX box.

There are three options. The first is `crossboxgraphicx` which uses the `graphicx` package to rotate a line segment. It loads `graphicx` automatically; if you want to pass options to `graphicx`, load that package before `logicsym`. The default is `crossboxgraphicx`.

The second option is `crossboxdvips` which inserts raw Postscript code that can be processed by the `dvips` driver. It *requires* `dvips`. This option is present for historical reasons.

The third option is `crossboxlatex` which uses the line-drawing facilities of the LATEX `picture` environment. It works in all cases, but the possible slopes of lines are limited, and the results will be less satisfying.

2 Usage

2.1 Truth Constants

As mentioned earlier `\False` and `\True` may be used in text mode or math mode.

2.2 Symbols

The connectives `\Not`, `\Or`, `\And`, `\Implies`, `\Iff`, and `\Followsfrom` are available in math mode. The quantifiers are `\Forall` and `\Exists`. The first order equality symbol `\Equals` is rendered as $\approx$.

There are relational "turnstile" symbols available in math mode as

- `\Proves`, `\Notproves`, `\Revproves`, and `\Provesequiv`;
- `\Models`, `\Notmodels`, `\Revmodels`, and `\Modelsequiv`; and
- `\Forces` and `\Notforces`.

There are symbols for the standard sets of numbers: `\Nat`, `\Int`, `\Re`, and `\Cx`. There are also symbols for the logical structures: `\modelA`, `\modelB`, `\modelN`, `\modelR`, `\modelZ`. These require the `\mathbb` and `\mathfrak` commands, which comes from the `amssym` package. Alternately, one can use the `mtpro2` package with the `mtpbb` and `mtpfrak` options, or the `lucimatx` package.

There are abbreviations `\mp` or `\Mp` for *modus ponens* and *Modus ponens*, respectively.

The command `\notR` creates the letter R with a slash through it for the negation of relations.

`\False`	$\bot$	F	*False*
`\True`	$\top$	T	*True*
`\Or`	$\vee$		
`\Not`	$\neg$		
`\And`	$\wedge$	$\&$	
`\Implies`	$\rightarrow$	$\Rightarrow$	
`\Followsfrom`	$\leftarrow$	$\Leftarrow$	
`\Iff`	$\leftrightarrow$	$\Leftrightarrow$	$\equiv$
`\Equals`	$\approx$		

Table 1: The constants and connectives, and their renditions.

`\ModelA`	$\mathfrak{A}$	`\Nat`	$\mathbb{N}$
`\ModelB`	$\mathfrak{B}$	`\Int`	$\mathbb{Z}$
`\ModelN`	$\mathfrak{N}$	`\Re`	$\mathbb{R}$
`\ModelR`	$\mathfrak{R}$	`\Cx`	$\mathbb{C}$
`\ModelZ`	$\mathfrak{Z}$	`\NotR`	$\not R$

Table 2: The special symbols.

`\Models`	$\models$	`\Proves`	$\vdash$	`\Forces`	$\Vdash$
`\Notmodels`	$\not\models$	`\Notproves`	$\nvdash$	`\Notforces`	$\nVdash$
`\Revmodels`	$=\!\mid$	`\Revproves`	$\dashv$		
`\Modelsequiv`	$\models\!=\!\mid$	`\Provesequiv`	$\vdash\dashv$		

Table 3: The turnstile symbols.

2.3 Inferences

There are facilities for the creation of natural-deduction style inferences.

For discharging assumptions there is `\Crossbox`. The macro `\Crossout` may be used instead when the assumption is a single letter, but usually `\Crossbox` looks better.

The command `\Infer[<align>]{<top>}{<bottom>}{<label>}` produces

$$\frac{<\texttt{top}>}{<\texttt{bottom}>}<\texttt{label}>$$

The label may be empty. The optional argument `<align>` is the alignment relative to the current environment. It may be `b`, `t`, or `c`, just as with arrays and tabular matter. The default alignment is `b` and can be changed by redefining `\InferAlign`.

The components of an inference are created with the `array` environment. There can be up to four components above or below the bar. Use the `\multicolumn` macro to balance the spacing when the numbers of components above and below the bar differ.

The command `\VStack` is intended for use as the top part of an inference. The result of `\VStack{X\\Y\\Z}` is

$$\begin{array}{l} X \\ Y \\ Z \end{array}$$

2.4 Boxed Proofs

There are facilities for construction of "boxed proofs." Here are the primitives:

- Environment `BoxProof`. The first column contains line numbers, the second column formulas which are set in math mode, and the third column is the justification in text mode.

  There is one required parameter, an integer indicating the maximum level of nesting.

- `\line` starts a line of the proof. Just a formula and a justification, separated by an ampersand.

- `\linenn` is like `\line` but no line number is produced.

- `\StartBox` and `\StopBox` delimit the beginning and end of a box.

- `\label` and `\ref` may be used in the natural way to record step numbers. The `\label` macro must be invoked *after* the `\line` macro.

As of version 1.61, there is a variation `QBoxProof`, which is used in proof boxes for formulas with quantifiers, as in the text by Huth and Ryan. There is an additional column, between the line number and the formula, to declare "fresh variables." A `\line` or `\linenn` macro must be followed by two ampersands, even if the fields are empty.

3 The Code

3.1 Preamble and Options

We first require the `ifthen` and `array` packages.

```
\RequirePackage{ifthen,array}
```

`\And` `\Implies` `\Followsfrom` `\Iff`

Next, we declare the options,

```
\DeclareOption{tfletters}{\def\tf@constant{letters}}
\DeclareOption{tfwords}{\def\tf@constant{words}}
\DeclareOption{topbottom}{\def\tf@constant{topbottom}}
\DeclareOption{ampersand}{\newcommand{\And}{\mathop{\&}}}
\DeclareOption{wedge}{\newcommand{\And}{\wedge}}
\DeclareOption{singlearrows}{%
  \def\Implies{\mathop{\rightarrow}}
  \def\Followsfrom{\mathop{\leftarrow}}
  \def\Iff{\mathop{\leftrightarrow}}}
\DeclareOption{doublearrows}{%
  \def\Implies{\mathop{\Rightarrow}}
  \def\Followsfrom{\mathop{\Leftarrow}}
  \def\Iff{\mathop{\Leftrightarrow}}}
\DeclareOption{equivalence}{\AtBeginDocument{\let\Iff\equiv}}
\DeclareOption{oldnames}{%
  \AtBeginDocument{%
    \let\notR\NotR
    \let\Re\Real
    \let\models\Models
    \let\notmodels\Notmodels
    \let\revmodels\Revmodels
    \let\modelequiv\Modelsequiv
    \let\proves\Proves
    \let\notproves\Notproves
    \let\revproves\Revproves
    \let\provesequiv\Provesequiv
    \let\crossout\Crossout
    \let\crossbox\Crossbox}}
\DeclareOption{crossboxgraphicx}{\def\cb@style{cbgraphicx}}
\DeclareOption{crossboxlatex}{\def\cb@style{cblatex}}
\DeclareOption{crossboxdvips}{\def\cb@style{cbdvips}}
```

and process them.

```
\ExecuteOptions{topbottom,doublearrows,wedge,crossboxgraphicx}
\ProcessOptions
```

3.2 Basic Symbols

`\False` `\True` `\Truthconstantfont`

Next come the truth constants.

```
\newcommand{\Truthconstantfont}{\mathit}
\newcommand{\truth@constant}[3]{%
\ensuremath{%
\ifthenelse{\equal{\tf@constant}{letters}}{\Truthconstantfont{#1}}{%
\ifthenelse{\equal{\tf@constant}{words}}{\Truthconstantfont{#2}}{#3}}}}
\newcommand{\True}{\truth@constant{T}{true}{\top}}
\newcommand{\False}{\truth@constant{F}{false}{\bot}}
```

`\Not` `\Or` `\Forall` `\Exists` `\Equals`

Then we define the connectives that were not determined by options.

```
\newcommand{\Not}{\neg}
\newcommand{\Or}{\vee}
\newcommand{\@quantifier}[2]{#1#2\,}
\newcommand{\Forall}{\@quantifier\forall}
\newcommand{\Exists}{\@quantifier\exists}
\newcommand{\Equals}{\approx}
```

`\ModelA` `\ModelB` `\ModelN` `\ModelR` `\ModelZ`

The special letters use `\mathfrak`

```
\newcommand{\ModelA}{\mathfrak{A}}
\newcommand{\ModelB}{\mathfrak{B}}
\newcommand{\ModelN}{\mathfrak{N}}
\newcommand{\ModelR}{\mathfrak{R}}
\newcommand{\ModelZ}{\mathfrak{Z}}
```

`\Nat` `\Int` `\Re` `\Cx` `\NotR`

and `\mathbb`.

```
\newcommand{\Nat}{\mathbb{N}}
\newcommand{\Int}{\mathbb{Z}}
\renewcommand{\Re}{\mathbb{R}}
\newcommand{\Cx}{\mathbb{C}}
\newcommand{\NotR}{\mathbin{\Crossout{R}}}
```

`\mp` `\Mp`

The Latin uses emphasis.

```
\renewcommand{\mp}{\emph{modus ponens}}
\newcommand{\Mp}{\emph{Modus ponens}}
```

3.3 Turnstiles

`\TurnstileScale`

The turnstile symbols are drawn with “rules.” They are all the same width and same height. They are raised just enough so that the vertical midpoint lines up with the crossbar on +. They can be scaled by redefining `\TurnstileScale`. Here are the lengths and the basic rules.

```
\newcommand{\TurnstileScale}{1.0}
\newlength{\turnstile@height}
\newlength{\turnstile@width}
\newlength{\turnstile@thickness}
\newlength{\turnstile@sep}
\newlength{\turnstile@templength}
\newlength{\turnstile@lift}
\newsavebox{\turnstile@box}
\newcommand{\turnstile@singlevertical}{%
  \rule[-0.5\turnstile@height]{\turnstile@thickness}{\turnstile@height}}
\newcommand{\turnstile@doublevertical}{%
  \turnstile@singlevertical%
  \hspace{\turnstile@sep}%
  \turnstile@singlevertical}
```

```
\newcommand{\turnstile@singlehorizontal}{%
  \rule[-0.5\turnstile@thickness]{\turnstile@width}{\turnstile@thickness}}
\newcommand{\turnstile@doublehorizontal}{%
  \setlength{\turnstile@templength}{\turnstile@sep}%
  \addtolength{\turnstile@templength}{-0.5\turnstile@thickness}%
  \rule[\turnstile@templength]{\turnstile@width}{\turnstile@thickness}%
  \hspace{-\turnstile@width}%
  \rule[-\turnstile@templength]{\turnstile@width}{\turnstile@thickness}}
```

Here are the outer macros, that create the turnstile symbols.

```
\newcommand{\turnstile@setup}{%
  \setlength{\turnstile@height}{1.8ex}
  \setlength{\turnstile@height}{\TurnstileScale\turnstile@height}
  \setlength{\turnstile@width}{0.72em}
  \setlength{\turnstile@width}{\TurnstileScale\turnstile@width}
  \setlength{\turnstile@thickness}{0.05\turnstile@height} %{0.1ex}
  \setlength{\turnstile@sep}{3\turnstile@thickness} %{0.3ex}
  \settoheight{\turnstile@lift}{${+}$}
  \setlength{\turnstile@lift}{0.5\turnstile@width}}
\newcommand{\turnstile@make}[1]{%
  \turnstile@setup%
  \sbox{\turnstile@box}{\raisebox{\turnstile@lift}{#1}}%
  \mathrel{\usebox{\turnstile@box}}}
\newcommand{\turnstile@negate}[1]{%
  \turnstile@setup%
  \sbox{\turnstile@box}{%
    \hbox to\turnstile@width{\hss${\big/}$\hss}%
    \hspace{-\turnstile@width}\raisebox{\turnstile@lift}{#1}}%
  \mathrel{\usebox{\turnstile@box}}}
\newcommand\turnstile@sequence[2]{%
  \turnstile@setup%
  \sbox{\turnstile@box}{%
    \raisebox{\turnstile@lift}{#1\hspace{0.3\turnstile@width}#2}}%
  \mathrel{\usebox{\turnstile@box}}}
```

These are the details for the different symbols.

```
\newcommand{\turnstile@single}{%
  \turnstile@singlevertical%
  \hspace{-\turnstile@thickness}%
  \turnstile@singlehorizontal}
\newcommand{\turnstile@single@reverse}{%
  \turnstile@singlehorizontal%
  \hspace{-\turnstile@thickness}%
  \turnstile@singlevertical}
\newcommand{\turnstile@double}{%
  \turnstile@singlevertical%
  \hspace{-\turnstile@thickness}%
  \turnstile@doublehorizontal}
\newcommand{\turnstile@double@reverse}{%
  \turnstile@doublehorizontal%
  \hspace{-\turnstile@thickness}%
  \turnstile@singlevertical}
\newcommand{\turnstile@forces}{%
  \turnstile@doublevertical%
  \hspace{-\turnstile@thickness}%
  \addtolength{\turnstile@width}{-\turnstile@sep}%
  \addtolength{\turnstile@width}{-\turnstile@thickness}%
  \turnstile@singlehorizontal}
```

`\Models` `\Revmodels` `\Notmodels` `\Modelsequiv`

And finally, the user macros for models, proves, and forces

```
\newcommand{\Models}{\turnstile@make{\turnstile@double}}
\newcommand{\Revmodels}{\turnstile@make{\turnstile@double@reverse}}
\newcommand{\Modelsequiv}{%
  \turnstile@sequence{\turnstile@double}{\turnstile@double@reverse}}
\newcommand{\Notmodels}{\turnstile@negate{\turnstile@double}}
```

`\Proves` `\Revproves` `\Notproves` `\Provesequiv`

```
\newcommand{\Proves}{\turnstile@make{\turnstile@single}}
\newcommand{\Revproves}{\turnstile@make{\turnstile@single@reverse}}
\newcommand{\Provesequiv}{%
  \turnstile@sequence{\turnstile@single}{\turnstile@single@reverse}}
\newcommand{\Notproves}{\turnstile@negate{\turnstile@single}}
```

`\Forces` `\Notforces`

```
\newcommand{\Forces}{\turnstile@make{\turnstile@forces}}
\newcommand{\Notforces}{\turnstile@negate{\turnstile@forces}}
```

3.4 Inference

`\Crossout`

Here is the code to cross out a single letter.

```
\newlength{\crossoutwidth}
\newcommand{\Crossout}[1]{%
  \settowidth{\crossoutwidth}{$#1$}%
  \hbox to\crossoutwidth{\hss${\big/}$\hss}%
  \hspace{-\crossoutwidth}#1}
```

The code to cross out a longer formula is complicated by the difficulty in drawing slanted rules in TEX and LATEX. We provide three options. The idea is to draw the diagonal of a box with the same height and depth as the formula—and slightly wider. By default, the extra space on the left and right is 0.3 em. The factor can be changed by redefining `\CrossboxMargin`.

`\CrossboxMargin`

```
\newcommand{\CrossboxMargin}{0.3}
```

`\CrossboxLineThickness`

When possible, the line thickness is set in terms of the current ex-size.

```
\newcommand{\CrossboxLineThickness}{0.1}
```

In all cases, we assume that the box containing the formula is wider than it is tall. The slash runs through the center of the box and touches the left and right sides. It is as steep as possible without extending below the lower left corner or above the upper left corner.

`\Crossbox`

```
\newbox\thecrossbox@
\newlength\crossbox@MarginWidth
```

```
\newlength\crossbox@Height
\newlength\crossbox@Depth
\newlength\crossbox@Width
\newcommand{\Crossbox}[1]{%
\setbox\thecrossbox@=\hbox{\ensuremath{#1}}
\setlength{\crossbox@MarginWidth}{\CrossboxMargin em}
\setlength{\crossbox@Height}{\ht\thecrossbox@}
\setlength{\crossbox@Depth}{\dp\thecrossbox@}
\setlength{\crossbox@Width}{\wd\thecrossbox@}
\hspace*{1\crossbox@MarginWidth}
\box\thecrossbox@
\addtolength{\crossbox@Width}{1\crossbox@MarginWidth}
\hspace{-\crossbox@Width}
\addtolength{\crossbox@Width}{1\crossbox@MarginWidth}
\crossbox@Diagonal}
```

`\crossbox@Diagonal` One option for drawing the diagonal is to use the limited range of slanted lines in the LaTeX `picture` environment. It is a simple, brute-force search through the possibe slopes—not pretty!

```
\ifthenelse{\equal{\cb@style}{cblatex}}{
\newbox\thecrossbox@
\newlength{\cb@TempA}
\newlength{\cb@TempB}
\newcommand{\greater@test}[2]{
  \setlength{\cb@TempA}{#2\crossbox@Width}
  \setlength{\cb@TempB}{#1\crossbox@Height}
  \ifthenelse{\lengthtest{\cb@TempB<\cb@TempA}}}
\newcommand{\setslope}[2]{
  \def\line@cmd{\line(#1,#2)}
  \setlength{\cb@TempA}{#2\crossbox@Width}
  \divide\cb@TempA#1\relax
  \setlength{\cb@TempB}{0.5\crossbox@Height}
  \addtolength{\cb@TempB}{-0.5\cb@TempA}
  \addtolength{\cb@TempB}{-\crossbox@Depth}}
\newcommand{\crossbox@Diagonal}{%
\addtolength{\crossbox@Height}{\crossbox@Depth}
\setlength{\unitlength}{\crossbox@Width}
\greater@test{6}{1}{\setslope{1}{0}}{
\greater@test{5}{1}{\setslope{6}{1}}{
\greater@test{4}{1}{\setslope{5}{1}}{
\greater@test{3}{1}{\setslope{4}{1}}{
\greater@test{5}{2}{\setslope{3}{1}}{
\greater@test{2}{1}{\setslope{5}{2}}{
\greater@test{5}{3}{\setslope{2}{1}}{
\greater@test{3}{2}{\setslope{5}{3}}{
\greater@test{4}{3}{\setslope{3}{2}}{
\greater@test{5}{4}{\setslope{4}{3}}{
\greater@test{6}{5}{\setslope{5}{4}}{
\greater@test{1}{1}{\setslope{6}{5}}{
                    \setslope{1}{1}}}}}}}}}}}}}
\raisebox{\cb@TempB}
         {\begin{picture}(1,0)\put(0,0){\line@cmd{1}}\end{picture}}}
}{}
```

Another option is to use `dvips` specials. In this case and the next one, we can meet the corners of the box exactly and adjust the line thickness to the font size.

We must know the scale, in dots per inch, of the Postscript output. The default setting is 1200. It can be changed with the `\DPIfactor` macro. Alternately, one can pass the flag `-D1200` to `dvips`.

`\DPIfactor`

```
\ifthenelse{\equal{\cb@style}{cbdvips}}{
\newcount\crossbox@Ht
\newcount\crossbox@Wd
\newcount\crossbox@Dp
\newcount\crossbox@Tk
\newdimen\dpifactor
\newlength{\crossbox@Thickness}
\newcommand{\DPIfactor}[1]{\dpifactor=1in\relax\divide\dpifactor#1\relax}
\DPIfactor{1200}
\newcommand{\crossbox@Diagonal}{
\crossbox@Ht=\crossbox@Height \divide\crossbox@Ht\dpifactor
\crossbox@Wd=\crossbox@Width  \divide\crossbox@Wd\dpifactor
\crossbox@Dp=\crossbox@Depth  \divide\crossbox@Dp\dpifactor
\setlength{\crossbox@Thickness}{\CrossboxLineThickness ex}
\crossbox@Tk\crossbox@Thickness \divide\crossbox@Tk\dpifactor
\setlength{\unitlength}{\crossbox@Width}
\begin{picture}(1,0)
\put(0,0){\special{ps:
 matrix currentmatrix currentpoint translate newpath
 \the\crossbox@Tk\space setlinewidth
 0 \the\crossbox@Dp\space moveto
 \the\crossbox@Wd\space
 \the\crossbox@Ht\space \the\crossbox@Dp\space add neg
 lineto stroke
 setmatrix
}}
\end{picture}}
}{}
```

The remaining option is to draw a horizontal line and use the `graphicx` package to rotate it.

Arithmetic and trigonometry in TEX is not fun. We use the `fp` package to do ease the burden.

```
\ifthenelse{\equal{\cb@style}{cbgraphicx}}{
\RequirePackage{graphicx}
\RequirePackage{fp}
\newcount\crossbox@Count
\newcommand{\crossbox@Diagonal}{
\addtolength{\crossbox@Height}{\crossbox@Depth}
\crossbox@Count\crossbox@Height
\FPset\crossbox@Ht{\the\crossbox@Count}
\crossbox@Count=\crossbox@Width
\FPset\crossbox@Wd{\the\crossbox@Count}
\FPupn{\crossbox@Diag}
      {2 1 \crossbox@Ht{} \crossbox@Wd{} div copy mul add root}
\FPupn{\crossbox@Angle}
      {\crossbox@Ht{} \crossbox@Wd{} div arctan 180 mul pi div}
\FPround{\crossbox@Angle}{\crossbox@Angle}{1}
\raisebox{-\crossbox@Depth}
         {\rotatebox{\crossbox@Angle}
                    {\rule{\crossbox@Diag\crossbox@Width}
                          {\CrossboxLineThickness ex}}}}
}{}
```

`\Infer` `\VStack` `\InferAlign` Finally, we create the inference diagrams and stacks of formulas.

```
\newlength\@InferFullSep
\newcommand\InferStretch{1}
\newcommand\InferAlign{b}
\newcommand\Infer{\@ifnextchar[{\@Infer}{\@Infer[\InferAlign]}}
\def\@Infer[#1]#2#3#4{%
  \setlength{\@InferFullSep}{\InferStretch\normalbaselineskip}%
  \renewcommand\arraystretch{1}\ensuremath{
  \ifthenelse{\equal{#4}{}}{\@@Infer[#1]{#2}{#3}}{\@@@Infer[#1]{#2}{#3}{#4}}\relax}}
\def\@@Infer[#1]#2#3{\begin{array}[#1]{cccc}#2\\ \hline#3\end{array}}
\def\@@@Infer[#1]#2#3#4{\begin{array}[#1]{@{}c@{}c@{}c@{}c@{}}
  \@@Infer[b]{#2}{#3}&\raisebox{0.5\@InferFullSep}{$\,#4$}\end{array}}
\newcommand\VStack[1]{\ensuremath{
  \begin{array}[b]{@{}c@{}}\vphantom{\raisebox{0.5ex}{(}}#1\end{array}}\relax}
```

3.5 Boxed Proofs

And now for the boxed proofs. This should be viewed as experimental code. The boxes look good, but the horizontal alignment is off, and there is extra vertical space on top when the proof starts with a box. In a pinch, the alignment can be fixed with brute force using `\raise`.

There are seven counters.

```
\newcounter{bprf@State}
\newcounter{bprf@TempCount}
\newcounter{bprf@LineNumber}
\newcounter{bprf@NestCount}
\newcounter{bprf@NestLimit}
\newcounter{bprf@PendingStarts}
\newcounter{bprf@PendingStops}
\renewcommand{\thebprf@LineNumber}{\arabic{bprf@LineNumber}}
```

There are two lengths, to separate boxes vertically and horizontally.

```
\newlength{\bprf@HLineSep}
\newlength{\bprf@VLineSep}
```

The `BoxProof` environment is a tabular environment. There is an optional parameter for the vertical alignment, if one wishes to change it. The state is encoded as a counter: −1 for the top of the environment—no lines produced, 0 for the middle of the environment, and +1 for the final cleanup.

BoxProof

```
\newenvironment{BoxProof}[2][t]
{\setcounter{bprf@State}{-1}%
 \setcounter{bprf@LineNumber}{0}%
 \setcounter{bprf@NestCount}{0}%
 \setcounter{bprf@NestLimit}{#2}%
 \setcounter{bprf@PendingStarts}{0}%
 \setcounter{bprf@PendingStops}{0}%
 \setlength{\bprf@HLineSep}{1ex}%
 \setlength{\bprf@VLineSep}{1ex}%
 \let\item\bprf@Item%
 \let\line\bprf@Line%
 \let\linenn\bprf@Linenn%
 \ifthenelse{\boolean{bprf@QBox}}
   {\begin{tabular}[#1]
     {@{}r<{\bprf@HSpace{\arraycolsep}}
      @{}>{\bprf@LeftBlock{\vline}$}l<{$\bprf@HSpace{\arraycolsep}}
      @{}>{$\bprf@HSpace{\arraycolsep}}l<{$\bprf@HSpace{\arraycolsep}}%
      @{}>{\bprf@RightBlock{\vline}}l@{}}}
   {\begin{tabular}[#1]
     {@{}r<{\bprf@HSpace{\arraycolsep}}
      @{}>{\bprf@LeftBlock{\vline}$}l<{$\bprf@HSpace{\arraycolsep}}
      @{}>{\bprf@RightBlock{\vline}}l@{}}}
}{\setcounter{bprf@State}{1}\bprf@MakeHLines\end{tabular}}
```

The QBoxProof environment is like BoxProof, except that there is an additional column for “fresh” variables. It really ought to be corrected and generalized.

QBoxProof

```
\newboolean{bprf@QBox}
\setboolean{bprf@QBox}{false}
\newenvironment{QBoxProof}[2][t]
{\setboolean{bprf@QBox}{true}\begin{BoxProof}[#1]{#2}}
{\end{BoxProof}}
```

The macros `\item` and `\item*` are depricated, as of version 1.61. The difficulty is that an ampersand cannot appear immediately after an `\item`; it gets “eaten” while `\item` is searching for an asterisk. The `\item` commands still exist for compatibility (and a simple, but annoying, way around it is to insert `{}` or `\relax` between `\item` and an ampersand), but `\line` and `\linenn` are the preferred replacements for `\item` and `\item*`.

The macro that will eventually become `\line` is `\bprf@Item`. The height adjustment for the first line is a kludge that only works for one box. I don't know why. It is inspired by the macro `\firsthline`, and uses `\backup@length`, from the `array` package.

\bprf@Item

```
\newcommand{\bprf@ItemPreamble}{%
 \ifthenelse{-1=\value{bprf@State}}%
    {\ifthenelse{0=\value{bprf@PendingStarts}}%
      {}{\bprf@MakeFirstHLines}%
     \setcounter{bprf@State}{0}}%
    {\bprf@MakeHLines}}
\newcommand{\bprf@Item}{\@ifstar\bprf@LineNN\bprf@Line}
\newcommand{\bprf@LineNN}{\bprf@ItemPreamble&\ignorespaces}
\newcommand{\bprf@Line}{%
  \bprf@ItemPreamble%
  \refstepcounter{bprf@LineNumber}\xdef\@currentlabel{\thebprf@LineNumber}%
  \thebprf@LineNumber.&\ignorespaces}
\newcommand{\bprf@MakeFirstHLines}{%
  \backup@length\value{bprf@PendingStarts}\bprf@HLineSep%
  \backup@length0.5\backup@length%
  \advance\backup@length\value{bprf@PendingStarts}\arrayrulewidth%
  \advance\backup@length\ht\@arstrutbox%
  \advance\backup@length\dp\@arstrutbox%
  \vspace*{-\backup@length}
  \bprf@MakeHLines}
```

\StartBox \StopBox

The macros \StartBox and \StopBox simply increment counters. The work is done when the next line starts or is at the end of the environment.

```
\newcommand{\StartBox}{%
 \addtocounter{bprf@PendingStarts}{1}\ignorespaces}
\newcommand{\StopBox}{%
 \addtocounter{bprf@PendingStops}{1}\ignorespaces}
```

The macros \bprf@Narrow and \bprf@Widen adjust the number of nested boxes.

```
\newcommand{\bprf@Narrow}{%
 \ifthenelse{\value{bprf@NestCount}<\value{bprf@NestLimit}}
   {\addtocounter{bprf@NestCount}{1}}
   {\@latex@warning{Boxes are nested too deeply}}\ignorespaces}
\newcommand{\bprf@Widen}{%
 \ifthenelse{0<\value{bprf@NestCount}}
   {\addtocounter{bprf@NestCount}{-1}}
   {\@latex@warning{Attempt to close non-existant box}}\ignorespaces}
```

These cycle through the pending stops and starts. We avoid extra vertical space when a box opens before the first line of the proof and when one closes after the last line.

```
\newcommand{\bprf@MakeHLines}{%
 \whiledo{0<\value{bprf@PendingStops}}%
   {\addtocounter{bprf@PendingStops}{-1}%
    \bprf@Gap\bprf@Widen\bprf@HLine%
    \ifthenelse{1=\value{bprf@State}\and0=\value{bprf@PendingStops}}%
      {}{\bprf@Gap}}%
 \ifthenelse{-1=\value{bprf@State}\and0<\value{bprf@PendingStarts}}%
   {\addtocounter{bprf@PendingStarts}{-1}%
    \bprf@HLine\bprf@Narrow\bprf@Gap}{}%
 \whiledo{0<\value{bprf@PendingStarts}}%
   {\addtocounter{bprf@PendingStarts}{-1}%
    \bprf@Gap\bprf@HLine\bprf@Narrow\bprf@Gap}%
 \ifthenelse{1=\value{bprf@State}}{}{\crcr}\ignorespaces}
```

The macro `\bprf@HRule` creates a horizontal line that will span two columns.

```
\newcommand{\bprf@HRule}{\leaders\hrule\@height\arrayrulewidth\hfill}
```

The macro `\bprf@Gap` creates the “filler” on either side of a horizontal line.

```
\newcommand{\bprf@Gap}[1][0.5]{%
  \crcr\omit&\omit%
  \rule{0pt}{#1\bprf@HLineSep}% for vertical spacing when nesting is 0
  \bprf@LeftBlock{\rule{\arrayrulewidth}{#1\bprf@HLineSep}}%
  \ifthenelse{\boolean{bprf@QBox}}{&\omit}{}&\omit%
  \bprf@RightBlock{\rule{\arrayrulewidth}{#1\bprf@HLineSep}}}
```

The macro `\bprf@HLine` creates the horizontal line itself.

```
\newcommand{\bprf@HSpace}[1]{\rule{#1}{\z@}}
\newcommand{\bprf@HLine}{%
  \crcr\omit&\omit%
  \setcounter{bprf@TempCount}{1}%
  \whiledo{\value{bprf@TempCount}<\value{bprf@NestLimit}}%
     {\rule{\arrayrulewidth}{\arrayrulewidth}%
      \ifthenelse{\value{bprf@NestCount}<\value{bprf@TempCount}}%
         {\rule{\bprf@VLineSep}{\arrayrulewidth}}{\bprf@HSpace{\bprf@VLineSep}}%
      \stepcounter{bprf@TempCount}}%
  \rule{\arrayrulewidth}{\arrayrulewidth}%
  \rule{\arraycolsep}{\arrayrulewidth}%
  \ifthenelse{\boolean{bprf@QBox}}{\bprf@HRule&\omit}{}\bprf@HRule&\omit%
  \setcounter{bprf@TempCount}{\value{bprf@NestLimit}}%
  \addtocounter{bprf@TempCount}{-1}%
  \whiledo{0<\value{bprf@TempCount}}%
     {\rule{\arrayrulewidth}{\arrayrulewidth}%
      \ifthenelse{\value{bprf@NestCount}<\value{bprf@TempCount}}%
         {\rule{\bprf@VLineSep}{\arrayrulewidth}}{\bprf@HSpace{\bprf@VLineSep}}%
      \addtocounter{bprf@TempCount}{-1}}%
  \rule{\arrayrulewidth}{\arrayrulewidth}}
```

The macros `\bprf@LeftBlock` and `\bprf@RightBlock` create the vertical lines in a row of the proof. The argument is the command to create one vertical line.

```
\newcommand{\bprf@LeftBlock}[1]{%
  \setcounter{bprf@TempCount}{1}%
  \whiledo{\value{bprf@TempCount}<\value{bprf@NestLimit}}
     {\ifthenelse{\value{bprf@NestCount}<\value{bprf@TempCount}}
         {\bprf@HSpace{\arrayrulewidth}}{#1}%
      \bprf@HSpace{\bprf@VLineSep}%
      \stepcounter{bprf@TempCount}}%
  \ifthenelse{0<\value{bprf@NestLimit}\and
              \value{bprf@NestCount}=\value{bprf@NestLimit}}
     {#1}{\bprf@HSpace{\arrayrulewidth}}%
  \bprf@HSpace{\arraycolsep}}
\newcommand{\bprf@RightBlock}[1]{%
  \setcounter{bprf@TempCount}{\value{bprf@NestLimit}}%
  \addtocounter{bprf@TempCount}{-1}%
  \whiledo{0<\value{bprf@TempCount}}
     {\ifthenelse{\value{bprf@TempCount}<\value{bprf@NestCount}}
```

```
      {#1}{\bprf@HSpace{\arrayrulewidth}}%
    \bprf@HSpace{\bprf@VLineSep}%
    \addtocounter{bprf@TempCount}{-1}}%
\ifthenelse{0=\value{bprf@NestCount}}
   {\bprf@HSpace{\arrayrulewidth}}{#1}%
\bprf@HSpace{\arraycolsep}}
```