The pfsteps package

Jesse A. Tov
`tov@ccs.neu.edu`

This document corresponds to pfsteps v0.4, dated 2011/04/04.

Contents

1 Introduction

This package provides three distinct facilities for writing mathematical proofs: proof step labeling, proof sequences, and the `byCases` environment for case analysis.

Proof step labeling. The package provides a set of commands for numbering proof steps locally and referring back to those numbers.[1] For example, to get

[1]The idea is based on Didier Rémy's locallabel package (`http://cristal.inria.fr/~remy/latex/`), but the execution is different. In locallabel, a proof step number is printed and labeled at the same time, whereas in this package, printing a proof step number merely sets the current label so that a subsequent `\pflabel` command can then attach a label to it. This is appropriate for writing other commands that generate proof step numbers automatically but don't always

> (1) Socrates is a man, and (2) all men are mortal. Therefore, by (1–2), Socrates is mortal.

we might type:

```
\usepfcounter[socrates]~Socrates is a man, and
\usepfcounter[men]~all men are mortal.
Therefore, by \pfref{socrates,men}, Socrates is mortal.
```

The `\usepfcounter` command prints the next proof step number, and if given an optional argument, associates that label name with the proof step number. Alternatively, if we leave the optional argument out, we can capture the most recent proof step number using the `\pflabel` command. For example, for the first line above, we could have instead written:

```
\usepfcounter~Socrates is a man\pflabel{socrates}, and
```

We can then refer to proof step labels using the `\pfref` command, which takes a comma-separated list of proof step labels.

Finally, the proof step labels are local. The command `\resetpfcounter` ends the current numbering run, starts at 1 again, and allows reusing the same labels as the previous numbering run. The package does not currently support refering to proof steps outside the current numbering run.

Proof sequences. We provide environment `pfsteps` and `pfsteps*` for line-by-line proofs with justifications. The `pfsteps` environment puts proof steps in math mode, and the `pfsteps*` environment puts steps in text mode. The lines are numbered using the proof step labeling commands described above. Combined together this supports interspersing proof steps with explanatory text. The `pfsteps`[`*`] environments work like the `enumerate` list environment, except that their numbering uses proof labels, which can be named locally with `\pflabel`, and they defines a command `\BY`, which places proof step justifications to the right. For example, to get:

> (1) Socrates is a man.
>
> (2) All men are mortal.
>
> Therefore,
>
> (3) Socrates is mortal. by (1–2)

we can type:

```
\resetpfcounter
\begin{pfsteps*}
  \item Socrates is a man.  \pflabel{socrates}
```

require the user to name them. Also, unlike locallabel, we store proof step labels between runs, so that forward references work.

```
  \item All men are mortal. \pflabel{men}
\end{pfsteps*}
Therefore,
\begin{pfsteps*}
  \item Socrates is mortal.
          \BY{\pfref{men,socrates}}
\end{pfsteps*}
```

We also (optionally) define an abbreviated syntax, which looks like this:

```
\pfstepstextmode
«@socrates Socrates is a man.
•@men All men are mortal.»
Therefore,
«Socrates is mortal.  \BY{\pfref{men,socrates}}»
```

The `byCases` environment. We provide the `byCases` environment for proofs by cases. For example, to get this:

By cases on n:

Case 0**.**

Something.

Case $n' + 1$**.**

Something else.

Otherwise.

There is no otherwise!

type this:

```
By cases on $n$:
\begin{byCases}
  \case{0} Something.
  \case{n' + 1} Something else.
  \otherwise There is no otherwise!
\end{byCases}
```

1.1 Requirements & Other Packages

The pfsteps package depends on the listproc package, which is a non-standard LaTeX package available at http://www.ccs.neu.edu/~tov/code/latex/.

It cooperates with several other packages if they are loaded, and can load them by request:

hyperref If loaded, this package is used to create hyperlinks to proof steps from proof step references.

mathpartir If loaded, we define an `\icase` command within the `byCases` environment for considering inference rules by cases. This non-standard package is available at http://cristal.inria.fr/~remy/latex/.

ucs, inputenc If these packages are loaded, we can use them to define a nice notation for writing sequence of proof steps. The inputenc package must be loaded with the `utf8x` option.

2 Package Options

The package provides several options, which we document here.

`atsign, noatsign` *default:* true

This option controls whether `@` may be used inside the `pfsteps`[`*`] environment as a shorthand for `\pflabel`. This is on by default, but supplying the `noatsign` option will turn it off.

`hyperref, nohyperref` *default:* detect

This controls integration with the hyperref package. If neither option is specified, then we attempt to detect hyperref and use it if it's detected. Passing the `nohyperref` option will prevent this integration, even if hyperref is loaded. Passing the `hyperref` option will cause hyperref to be loaded if it wasn't already. Because hyperref likes to be loaded after other packages, it's probably best not to specify either of these options and load it yourself after other packages.

`loadunicode, noloadunicode` *default:* state of `unicode` option

This option has no effect unless the `unicode` option is turned on *explicitly*, in which case it defaults to true as well. If this option is on, we load two packages:

```
\RequirePackage{ucs}
\RequirePackage[utf8x]{inputenc}
```

Turning this option off by supplying `noloadunicode` will prevent those packages from being loaded. (Note that if the `unicode` option is on, you probably need to load these or something like them in order for it to work, or even for pfsteps to load properly.)

`mathpartir, nomathpartir` *default:* detect

As with the `hyperref` option, this controls integration with another package—in this case, Didier Rémy's mathpartir. We try to detect mathpartir by default, but specifying `nomathpartir` will prevent detection and integration, whereas explicitly passing the `mathpartir` option will cause it to be loaded.

`unicode, nounicode` *default:* false

If this option is turned on, we define « and » as shorthand for the `pfsteps` environment and • as `\item` within the environment. By default, enabling this option enables `loadunicode` as well.

Alternatively, it's possible to manually turn on the shorthand notation using different characters with `\pfstepsSetupUnicode`.

3 Command Reference

3.1 Proof Step Labeling

`\resetpfcounter [⟨proof-step⟩]`

Reset the proof step counter to ⟨*proof-step*⟩ (default 0), which starts a new proof section. Labels from before using this command become no longer accessible, but their names may be reused. By default, this is called by every `\case` command in the `byCases` environment, but it may be disabled by redefining `\byCasesEveryCase`.

`\usepfcounter [⟨label-name⟩]`

Increment and print the proof counter. If ⟨*label-name*⟩ is given, then we associate that name with the new proof step value. To change how the proof counter is formatted, redefine `\pfcounteranchor`.

`\pflabel {⟨label-name⟩}`

Associate ⟨*label-name*⟩ with the current value of the proof step counter.

`\pfref {⟨label-name⟩`$_1$`,...,⟨label-name⟩`$_k$`}`

Print references to the proof steps associated with one or label names, which must be separated by commas. The proof step numbers are sorted, and then adjacent numbers are compressed into ranges.

`\steppfcounter [⟨label-name⟩]`

Increment the proof counter, but don't print it. If ⟨*label-name*⟩ is given, then we associate that name with the new proof step value.

`\thepfcounter`
`\thepfsectioncounter`

These show the current proof counter and the proof section counter, respectively. (The proof section counter is incremented by each call to `\resetpfcounter`.

`\pfcounteranchor {⟨proof-step⟩}`
`\pfcounterref {⟨proof-step⟩}`

These are used to format proof step anchors and references. By default, they just place parentheses around their arguments. Redefine them to change how proof step numbers appear.

3.2 Proof Sequences

```
\begin{pfsteps}
  \item ⟨math⟩ [\BY{⟨justification⟩}]
  ⋮
\end{pfsteps}
```

Make a list of numbered proof steps. Each `\item` is numbered using `\usepfcounter`, so several instances of the `pfsteps` environment in sequence will continue numbering from each to the next instead of starting each at 1. (To reset the numbering, use `\resetpfcounter`). Optionally, `\BY{`⟨*justification*⟩`}` will print the provided justification for the given step in a column on the right.

Within the extent of this environment, a macro `\AND` is defined, which prints the word "and" with appropriate space around it in math mode.

```
\begin{pfsteps*}
  \item ⟨text⟩ [\BY{⟨justification⟩}]
  ⋮
\end{pfsteps*}
```

Like the `pfsteps` environment, but the proof steps are in text rather than math mode.

```
@⟨label-name⟩␣
```

Inside the `pfstep`[`*`] environments, this is equivalent to `\pflabel{`⟨*label-name*⟩`}`, unless option `noatsign` is supplied, in which case `@` has no special meaning inside the proof step environments.

`\proofleftskip=`⟨*dimen*⟩	*default:* `2pc`

This dimension parameter determines the space reserved to the left of each proof step for the step number.

`\proofrightwidth=`⟨*dimen*⟩	*default:* `0.3\linewidth`

This dimension parameter determines the width used for proof step justifications printed in the right column by `\BY`.

3.2.1 Unicode Shorthand

If option `unicode` is enabled, then this notation is defined:

```
«⟨math⟩ [\BY{⟨justification⟩}]
•⟨math⟩ [\BY{⟨justification⟩}]
⋮
»
```

This is equivalent to using the `pfsteps` environment, where the first `\item` is started implicitly, and the Unicode bullet • starts subsequent items.

```
\pfstepsmathmode
\pfstepstextmode
```

Toggle the meaning of «... » between math mode (like environment `pfsteps`) and text mode (like environment `pfsteps*`). The initial state is math mode.

```
\pfstepsSetupUnicode {⟨start-code⟩} {⟨stop-text⟩} {⟨item-code⟩}
```

If option `nounicode` is set, then this command may be used to customize the `pfsteps` environment shorthand notation to use other characters. The arguments are as follows:

⟨*start-code*⟩ The Unicode code point, in decimal, for starting the notation. When setup automatically using the `unicode` option, this is `171`, which is the code point for «.

⟨*stop-text*⟩ The actual text for terminating the notation. When setup by the `unicode` option, this is ».

⟨*item-code*⟩ The Unicode code point, in decimal, for the `\item` separator. When setup by the `unicode` option, this is `8226`, which is the code point for •.

Note that the first and third arguments are decimal numbers representing Unicode code points, whereas the second argument is the actual symbol or sequence of symbols to use.

3.3 The `byCases` Environment

```
\begin{byCases}
  ⟨by-cases-item⟩ ...
\end{byCases}
```

Introduce a case analysis section. Its contents must be a sequence of items, which may be constructed out of several commands detailed below. This is similar to the `description` environment, but it changes the behavior of `\item` and defines several other commands within its scope. These are the commands defined or affected by `byCases`:

```
\item [⟨math⟩] ⟨text⟩
```

Insert a case list item. If ⟨*math*⟩ is supplied, then the "bullet" is "**Case** `$`⟨*math*⟩`$`**.**" If ⟨*math*⟩ is not supplied, then we begin the item with "**Otherwise.**"

```
\case [⟨extra⟩] {⟨math⟩} ⟨text⟩
```

Insert a case list item with "**Case** `$`⟨*math*⟩`$`**.**' Unlike `\item`, this always puts a line break before ⟨*text*⟩. To change the appearance or language, redefine `\byCasesCaseTemplate`.

The optional argument ⟨*extra*⟩ is some text to insert after the case head but before the line break. The default value is `\byCasesEveryCase`.

```
\otherwise [⟨extra⟩] ⟨text⟩
```

Insert a case list item, with "**Otherwise.**" Unlike `\item`, this always puts a line break before ⟨*text*⟩. To change the appearance or language, redefine `\byCasesOtherwiseTemplate`.

The optional argument ⟨*extra*⟩ is some text to insert after the case head but before the line break. The default value is `\byCasesEveryOtherwise`.

```
\icase [⟨rule-name⟩] {⟨premises⟩}
       {⟨conclusion⟩} [⟨where-clause⟩]
\icase* [⟨inferrule*-opts⟩] {⟨premises⟩}
        {⟨conclusion⟩} [⟨where-clause⟩]
```

These commands are defined only if the `mathpartir` package is detected or loaded explicitly with the `mathpartir` option. In that case, they typeset a case using an inference rule. The non-starred variant uses `\inferrule` and the starred variant uses `\inferrule*`. Thus, the first optional argument to `icase` is a rule name and the first optional argument to `icase*` is a key-value list of options understood by `\inferrule*`.

The second and third arguments are the premises and conclusion to put in the inference rule.

Then, the final, optional argument, allows specifying a side condition, which will be printed after the inference rule like " **where** `$`⟨*where-clause*⟩`$`**.**" The text can be changed by redefining `byCasesWhereTemplate`.

```
\byCasesEveryCase
\byCasesEveryOtherwise
```

These are the default values for the optional argument `[`⟨*extra*⟩`]` of `\case` and `\otherwise`. The initial value of `\byCasesEveryCase` is `\resetpfcounter`, so that every case implicitly resets the proof counter. The initial value of `\byCasesEveryOtherwise` points to `\byCasesEveryCase`. These can be redefined to avoid automatic proof counter resets, or to change the behavior of `\case` and `\otherwise` in some other ways. Or, they can be ignored on a case-by-case basis by passing a blank for ⟨*extra*⟩.

command	default
`\byCasesCaseTemplate {⟨case-text⟩}`	`\textbf{Case {⟨case-text⟩}.}`
`\byCasesOtherwiseTemplate`	`\textbf{Otherwise.}`
`\byCasesWhereTemplate`	`\textbf{where}`

These are used by the `\case`, `\otherwise`, and `\icase` templates to create the text “Case,” “Otherwise”, and “where” along with the styling. Redefine them to change how these are presented.

4 Implementation

We begin by loading the listproc package:

```
\RequirePackage{listproc}
```

4.1 Package Options

`\pfsteps@set`
`\pfsteps@option`

These macros make it easy to add package options. Each use of `\pfsteps@option{⟨name⟩}` creates both the named option ⟨*name*⟩ and its opposite, `no`⟨*name*⟩, and sets things up so that we can detect whether an option has been explicitly set, explicitly unset, or left to default.

```
\newcommand*\pfsteps@set[3][]{
  \expandafter\let\csname #1pfsteps@#2\endcsname#3
}
\newcommand*\pfsteps@option[2][\iffalse]{
  \pfsteps@set[if]{#2}#1
  \pfsteps@set[if]{#2@set}\iffalse
  \DeclareOption{#2}{
    \pfsteps@set[if]{#2}\iftrue
    \pfsteps@set[if]{#2@set}\iftrue
  }
  \DeclareOption{no#2}{
    \pfsteps@set[if]{#2}\iffalse
    \pfsteps@set[if]{#2@set}\iftrue
  }
}
```

`\ifpfsteps@atsign`

This sets up all the options. The first four of them default to *true*. Then, we process the package options.

```
\pfsteps@option[\iftrue]{atsign}
\pfsteps@option[\iftrue]{hyperref}
\pfsteps@option[\iftrue]{loadunicode}
\pfsteps@option[\iftrue]{mathpartir}
\pfsteps@option{unicode}
\ProcessOptions
```

If both the `unicode` and `loadunicode` options are set, we load the relevant packages.

```
\ifpfsteps@unicode
  \ifpfsteps@loadunicode
    \RequirePackage{ucs}
    \RequirePackage[utf8x]{inputenc}
  \fi
\fi
```

If mathpartir has been explicitly set, load it.

```
\ifpfsteps@mathpartir
  \ifpfsteps@mathpartir@set
    \RequirePackage{mathpartir}
  \fi
\fi
```

If hyperref has been explicitly set, load it.

```
\ifpfsteps@hyperref
  \ifpfsteps@hyperref@set
    \RequirePackage{hyperref}
  \fi
\fi
```

4.2 Proof Step Numbering

This section is based on Didier Rémys locallabel package. It differs in terms of the protocol for when proof steps are defined versus displayed.

`\pfcounteranchor` `\pfcounterref`

User redefinable commands for formatting proof step numbers:

```
\newcommand{\pfcounteranchor}[1]{(#1)}
\newcommand{\pfcounterref}[1]{(#1)}
```

`\c@pfsteps@pfc@global` `\c@pfsteps@pfc@local`

Two counters for proof steps, one to keep track of how many times we've reset the count, and the other to keep the current step count.

```
\newcounter{pfsteps@pfc@global}
\newcounter{pfsteps@pfc@local}
```

`\resetpfcounter` `\thepfcounter` `\thepfsectioncounter` `\steppfcounter`

Simple counter management:

```
\newcommand{\resetpfcounter}[1][0]
  {\stepcounter{pfsteps@pfc@global}\setcounter{pfsteps@pfc@local}{#1}}
\newcommand{\thepfcounter}
  {\the\value{pfsteps@pfc@local}}
\newcommand{\thepfsectioncounter}
  {\the\value{pfsteps@pfc@global}}
\newcommand{\steppfcounter}[1][\relax]{%
  \addtocounter{pfsteps@pfc@local}{1}%
   \ifx\relax#1\relax\else
     \pflabel{#1}%
   \fi
}
```

`\usepfcounter` To advance and print the proof counter. We use `\pfsteps@hypertarget`, which delegates to `\hypertarget` if hyperref has been detected.

```
\newcommand{\usepfcounter}[1][\relax]{%
  \steppfcounter[#1]%
  \pfsteps@hypertarget{pfc:\thepfsectioncounter:\thepfcounter}{%
    \pfcounteranchor{\thepfcounter}%
  }%
}
```

`\pfsteps@pfc@cs` `\pfsteps@pfc@` `\pfsteps@strip` These are helper macros for turning a proof label name into the name of the command that we store its number in. `\pfsteps@pfc@cs` actually returns the control sequence, whereas `\pfsteps@pfc@` just returns the name of the control sequence. Both use `\pfsteps@strip` to remove spaces from the proof label.

```
\newcommand{\pfsteps@pfc@cs}[1]
  {\csname\pfsteps@pfc@{\pfsteps@strip#1 \@empty}\endcsname}
\newcommand{\pfsteps@pfc@}[1]
  {pfsteps@pfc@\pfsteps@strip#1 \@empty @\thepfsectioncounter}
\def\pfsteps@strip#1 #2{%
  #1%
  \ifx#2\@empty\else\expandafter\pfsteps@strip\fi
  #2}
```

`\pflabel` This associates a name with the current state of the proof counter. It both defines a macro in current session for producing backward proof step references write away, and writes code to the auxiliary file so that forward proof step referenced work in the next run. To make this work smoothly, it actually defines two macros whose names are based on the current proof state. The main one is used to hold the proof step number. The extra macro, which ends with `@thisrun`, is defined for the current session but not in the auxiliary file. We can then use `@thisrun` to detect attempts to reuse the same label name in the same proof section in the same run, in order to issue a warning.

```
\newcommand{\pflabel}[1]
  {\expandafter\ifx\csname\pfsteps@pfc@{#1}@thisrun\endcsname\relax
     \expandafter\xdef\csname\pfsteps@pfc@{#1}\endcsname
       {\thepfcounter}%
     \expandafter\gdef\csname\pfsteps@pfc@{#1}@thisrun\endcsname
       {}%
     \immediate\write\@auxout{
       \noexpand\pfsteps@def@label
         {#1}{\thepfsectioncounter}{\thepfcounter}
     }%
   \else
     \PackageWarning{pfsteps}
        {Proof step (#1) already defined in this section}%
   \fi}
```

`\pfsteps@def@label` This is the command written to the auxiliary file, so we have it defined here in order to run it when loading the auxiliary file.

```
\newcommand*{\pfsteps@def@label}[3]{
  \expandafter\gdef
    \csname pfsteps@pfc@#1@#2\endcsname
    {#3}
}
```

`\pfref` For creating proof step references. The bulk of this is a large listproc list expression, which does several steps: parse the argument as a comma-separated list of label names; map over those names to build pairs of the raw number as a sorting key and the formatted (possibly hyperlinked) references, or suitable error messages if the label isn't defined; sort by the numeric key, which is the proof step number, and compress adjacent keys into ranges.

```
\newcommand*{\pfref}[1]
{{\ListExprTo
    {\Compress[\@apply@group\@firstoftwo]
     {\Sort[\@apply@group\@firstoftwo]
      {\Map
       {%
        {\@ifundefined{\pfsteps@pfc@{##1}}
           {-1}
           {\csname\pfsteps@pfc@{##1}\endcsname}}%
        {\@ifundefined{\pfsteps@pfc@{##1}}
           {\PackageWarning{pfsteps}
              {Proof step (##1) not yet defined in this section}%
            \textbf{?}}
           {\pfsteps@hyperlink
             {pfc:\thepfsectioncounter:\pfsteps@pfc@cs{##1}}
             {\pfsteps@pfc@cs{##1}}}}}
       {\List{#1}}}}}
    \pfsteps@pfref@list
```

We finish up by setting singleton proof steps to print as-is and ranges to print with en dashes. We set `\listitem` to print the first list item with no punctuation and to redefine itself to print commas for subsequent items.

```
  \let\listitem\pfsteps@pfref@listitem@first
  \def\@single##1{\@secondoftwo##1}%
  \def\@range##1##2{\@secondoftwo##1--\@secondoftwo##2}%
  \pfcounterref{\pfsteps@pfref@list}%
}}
```

`\pfsteps@pfref@listitem@first` `\pfsteps@pfref@listitem@rest` `\pfref` uses these to print a comma separated list.

```
\newcommand\pfsteps@pfref@listitem@first[1]{%
  #1\let\listitem\pfsteps@pfref@listitem@rest
}
\newcommand\pfsteps@pfref@listitem@rest[1]{%
  , #1\let\listitem\pfsteps@pfref@listitem@rest
}
```

`\pfsteps@hypertarget` `\pfsteps@hyperlink` This is the hyperref compatibility layer. We initially define our compatibility commands to ignore the first argument (which is an anchor name) and just print the second. Then, if the hyperref option is enabled, either by default or by choice, we wait until the preamble ends and attempt to detect hyperref. If it's detected, we redefine the compatibility macros to use the real things.

```
\newcommand\pfsteps@hypertarget[2]{#2}
\newcommand\pfsteps@hyperlink[2]{#2}
\ifpfsteps@hyperref
  \AtBeginDocument{
    \ifcsname hypertarget\endcsname
      \let\pfsteps@hypertarget=\hypertarget
      \let\pfsteps@hyperlink=\hyperlink
    \fi
  }
\fi
```

4.3 Proof Sequences

\proofleftskip
\proofrightwidth

Length parameters for configuring spacing of the pfsteps environment.

```
\newlength{\proofleftskip}
\newlength{\proofrightwidth}
\setlength{\proofleftskip}{2pc}
\setlength{\proofrightwidth}{0.3\linewidth}
```

pfsteps
pfsteps*

The pfsteps and pfsteps* environments are both defined in terms of the underlying pfsteps@with environment, which takes an argument to determine whether proof steps are in math mode.

```
\newenvironment{pfsteps}
        {\begin{pfsteps@with}$}
        {\end{pfsteps@with}}
\newenvironment{pfsteps*}
        {\begin{pfsteps@with}{}}
        {\end{pfsteps@with}}
```

pfsteps@with

The implementation of the pfsteps environment. The whole thing is set in a \trivlist, using \item for line breaks.

```
\newenvironment{pfsteps@with}[1]
{
  \leavevmode\begingroup
  \setlength{\parskip}{0pt}%
  \trivlist
  \raggedright
  \setlength{\leftskip}{1.5\proofleftskip}
```

Save some commands that we want to redefine in here.

```
  \let\pfstepsSavedItem\item
  \let\pfstepsSavedLabel\label
  \let\pfstepsSavedQedhere\qedhere
```

\AND
\BY
\item
\label
\qedhere

We then setup several commands that are internal to the environment. The interesting ones are \BY and \item. For \BY, we break out of math mode if we're in it, and then set the justification in a minipage of the configured with. However, right before the minipage, we use `\penalty-1` to encourage a page break if there isn't enough room left on the line for the justification box, and then we `\hfill` to right align it. The result is that justifications appear cleanly to the right of proof steps, on new lines only when necessary.

```
  \newcommand\AND[1][and]{\mathrel{\mbox{##1}}}
  \newcommand\BY[2][by]
    {\pfsteps@unmath{\penalty-1 \mbox{~}\hfill%
     \begin{minipage}[t]{\proofrightwidth}%
       \raggedright##1 ##2%
     \end{minipage}}}
```

For `\item`, we break out of math mode if necessary, then start a new underlying `\item`, generate a proof step number, and optionall go back into math mode.

```
  \def\pfstepsItem{%
    \pfsteps@stopmath
    \pfstepsSavedItem\mbox{}\kern-1.25\proofleftskip
    \makebox[\proofleftskip]{\hfill\usepfcounter}\kern0.25\proofleftskip
    #1\relax}
```

For `\qedhere`, we need to temporarily break out of math mode, or the QED box ends up in a weird place.

```
  \def\pfstepsQedhere{\pfsteps@unmath{\pfstepsSavedQedhere}}
```

We redefine `\item`, `\label`, and `\qedhere` to use the proof steps versions of the same.

```
  \let\item\pfstepsItem
  \let\label\pflabel
  \let\qedhere\pfstepsQedhere
```

```
  \ifpfsteps@atsign
    \pfsteps@setup@atsign
  \fi
  \relax
}
{
  \pfsteps@stopmath
  \endtrivlist\endgroup
  \noindent\ignorespaces
}
```

`\pfsteps@stopmath`
`\pfsteps@unmath`

Two commands for getting us out of math mode, but only if we're in it. `\pfsteps@unmath` takes an argument to set outside of math mode, and then reinstates math mode only if we were in math mode to begin with. This is useful compared to creating an `\mbox` in math mode, because it *ends* the current math environment, which will then let us do something like start a new list item before entering math mode again.

```
\newcommand\pfsteps@stopmath{\ifmmode$\fi}
\newcommand\pfsteps@unmath[1]{\ifmmode$\relax#1\relax$\else\relax#1\relax\fi}
```

`\pfsetps@setup@atsign`

A macro to make `@` active and define it to alias `\pflabel`. A bit tricky, since we also want `@` in the name of the command.

```
{
  \def\atsign{@}
  \catcode`\@=\active\relax
  \expandafter\gdef\csname pfsteps\atsign setup\atsign atsign\endcsname{
    \catcode`\@=\active\relax
    \gdef@##1 {\pflabel{##1}}
  }
}
```

`\pfstepsmathmode`
`\pfstepstextmode`

Commands to determine whether the Unicode short hand is in math or text mode.

```
\newcommand\pfstepsmathmode{\def\pfsteps@unicode@arg{$}}
\newcommand\pfstepstextmode{\def\pfsteps@unicode@arg{\relax}}
```

`\pfstepsSetupUnicode`
`\pfsteps@unicode@startpfsteps`
`steps@unicode@startpfsteps@kont`
`\pfsteps@unicode@item`

For setting up the Unicode `pfsteps` shortcut. This associates the code points of the start and item sequences with commands that implement them, and then defines those commands. We default to math mode.

```
\newcommand\pfstepsSetupUnicode[3]{
  \DeclareUnicodeCharacter{#1}{\pfsteps@unicode@startpfsteps}
  \DeclareUnicodeCharacter{#3}{\pfsteps@unicode@item}
  \def\pfsteps@unicode@startpfsteps
    {\begingroup
     \ifpfsteps@atsign\catcode`\@=\active\relax\fi
     \pfsteps@unicode@startpfsteps@kont}
  \def\pfsteps@unicode@startpfsteps@kont##1#2
    {\begin{pfsteps@with}\pfsteps@unicode@arg\item##1\end{pfsteps@with}%
     \endgroup}
  \def\pfsteps@unicode@item{\item}
  \pfstepsmathmode
}
```

If the `unicode` option is set, then we setup unicode with left and right guillemets as the delimiters and bullet as the item separator. The second argument to `\pfstepsSetupUnicode` below, which appears empty in the documentation, is the right guillemet ». The two numbers, 171 and 8226, are the code points for left guillemet and bullet, respectively.

```
\ifpfsteps@unicode
  \pfstepsSetupUnicode{171}{}{8226} %
\fi
```

4.4 The byCases Environment

`\byCasesEveryCase`
`\byCasesEveryOtherwise`
`\byCasesOtherwiseTemplate`
`\byCasesCaseTemplate`
`\byCasesWhereTemplate`

User configuration macros. The first two cause the proof step to be automatically reset for every case item, and the last three specify how case items are to appear.

```
\newcommand\byCasesEveryCase{\resetpfcounter}
\newcommand\byCasesEveryOtherwise{\byCasesEveryCase}
\providecommand{\byCasesOtherwiseTemplate}{\textbf{Otherwise.}}
```

```
\providecommand{\byCasesCaseTemplate}[1]{\textbf{Case {#1}.}}
\providecommand{\byCasesWhereTemplate}{\textbf{where}}
```

`byCases` This environment is based on the `description` environment built-in to LaTeX. However, we also bring `\case` and `\otherwise` into scope by aliasing them to the actual definitions (below).

```
\newenvironment{byCases}
  {%
    \begingroup
    \let\case\byCases@case
    \let\otherwise\byCases@otherwise
```

Package mathpartir integration: if the option is set and the command is in scope, then we bring `\icase` into scope.

```
    \ifpfsteps@mathpartir
      \ifcsname inferrule\endcsname\let\icase\byCases@icase\fi
    \fi
    \list{}{\labelwidth\z@ \itemindent-\leftmargin
            \let\makelabel\byCases@label}%
  }
  {%
    \endlist
    \endgroup
  }
```

`\item` This is the implementation of `\item` labels for `byCases` lists.

```
\newcommand*\byCases@label[1]{%
  \hspace\labelsep
  \normalfont~\strut
  \expandafter\ifx#1\relax\relax
    \byCasesOtherwiseTemplate
  \else
    \byCasesCaseTemplate{\normalfont${#1}$}%
  \fi
}
```

`\case`
`\otherwise`
`\AND`
`\byCases@case`
`\byCases@otherwise`
`\pfsteps@reallynopagebreak`

These are the actual definitions of `\case` and `\otherwise` that `\byCases` brings into scope with accessible names. Mostly, they delegate to `\item` and then produce a line break while suppressing any page break. In `\case`, it defines `\AND` to produce a properly spaced text " and " in math mode, just for the scope of the item label.

```
\newcommand*\byCases@case[2][\byCasesEveryCase]
  {\item[{\let\AND\byCases@and #2}]\strut#1\pfsteps@reallynopagebreak}
\newcommand*\byCases@otherwise[1][\byCasesEveryOtherwise]
  {\item[]\strut#1\pfsteps@reallynopagebreak}
\newcommand\pfsteps@reallynopagebreak{\par\nopagebreak\@nobreaktrue}
\newcommand\byCases@and[1][and]{\mathrel{\mbox{\textbf{#1}}}}
```

`\icase`
`\byCases@icase`
`\byCases@icase@start`
`\byCases@icase@nostar`

The first thing `\icase` does is detect whether it is being called as `\icase` or as `\icase*` and dispatches accordingly. These then select either `\inferrule` or `\inferrule*`.

```
\newcommand*\byCases@icase{
  \@ifnextchar* \byCases@icase@star \byCases@icase@nostar
}
\def\byCases@icase@nostar{\byCases@icase@i{\inferrule}}
\def\byCases@icase@star*{\byCases@icase@i{\inferrule*}}
```

`\byCases@icase@i`
`\byCases@icase@opts`
`\byCases@icase@noopts`

The next thing to check for is the first optional argument; we dispatch accordingly and pass the version of `inferrule` to use along with the optional argument, if necessary, to `byCases@icase@ii`.

```
\newcommand*\byCases@icase@i[1]{
  \@ifnextchar [{\byCases@icase@opts{#1}}{\byCases@icase@noopts{#1}}
}
\def\byCases@icase@opts#1[#2]{\byCases@icase@ii{#1[#2]}}
\def\byCases@icase@noopts#1{\byCases@icase@ii{#1}}
```

`\byCases@icase@ii`
`\byCases@icase@where`
`\byCases@icase@nowhere`

This macro receives three arguments: (`#1`) the variant of `inferrule` along with any optional argument, (`#2`) the premises, and (`#3`) the conclusion. It then checks for the final, optional argument which specifies a where clause, and dispatches accordingly.

```
\newcommand*\byCases@icase@ii[3]{
  \@ifnextchar [
    {\byCases@icase@where{#1}{#2}{#3}}
    {\byCases@icase@nowhere{#1}{#2}{#3}}
}
\def\byCases@icase@where#1#2#3[#4]{
  \case{#1{#2}{#3}\AND[\byCasesWhereTemplate]#4}%
}
\def\byCases@icase@nowhere#1#2#3{\case{#1{#2}{#3}}}
```

Change History

v0.1
    General: Initial documented release 1
v0.2
    General: Included listproc.sty 1
v0.3
    General: Input encoding bug fix for right guillemet 15
v0.4
    `pfsteps`: Spacing bug fixes: Sets parskip to 0pt, so we don't get really wide spacing in proofs. Avoids weird behavior when proof steps are too long. 13

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.