The `natded` package*

Tikitu de Jager
tikitu@gmail.com

September 26, 2006

Abstract

The `natded` package provides macros for typesetting natural deduction proofs with assumptions in column format. Related packages are `proof`, `prooftree` and `bussproofs`, all of which however provide tree-format derivations.

Language note

This package was written for Dutch users, so the user-level macros have Dutch names. The documentation was produced in haste, and therefore is in English. I might get around to translating it at some point.

1 Introduction

Spelling out exactly how this all works takes a surprising number of words. You might want to `\usepackage[prop,pred]{natded}` and jump straight to the examples to get your fingers dirty. If you do, keep one thing in mind: the macros are *very* temperamental about spacing and similar. You can sometimes add spaces where there aren't any in the examples, but you'll pretty much never get away with taking them away.

1.1 Configuration

The file `natded.cfg` can be used to add arbitrary configuration code to the package. This file will be processed after the package and all option code, if present. The copy distributed with the package provides examples of the introduction and elimination rules used in the propositional and predicate calculus package options, and some other examples.

1.2 Usage

Add `\usepackage[`⟨*options*⟩`]{`⟨*natded*⟩`}` to the preamble of your document. Valid options are `prop`, `pred` and `intuit`. The first two define the standard rules for propositional and predicate calculus derivations, while `intuit` redefines the rule `negneg` to produce an error. (Note that this is *all* the proof-checking this package performs!)

*This document corresponds to `natded` v0.2, dated 06/09/25.

2 The macros

`afleiding` Each afleiding must be surrounded by an `afleiding` environment. Within such an environment you should only ever start lines with one of the following macros: `\premis` (introducing a premis), `\regel` (for any named rule, see "Defining rules" below), `\anonregel` for a rule-like line but not applying any named rule (e.g., "as for ψ"), or the `assumptie` environment. Any other typeset material on a line will definitely break the lines marking open assumptions and may also interfere with other lines, eat your face, or threaten the very integrity of the universe itself. You have been warned.

All the 'rule-like' macros (including `\premis`, but *not* including `\anonregel`) share more or less the following form:

```
\macroname[label] $content$ Rulename (ref1,ref2)
```

The label is always optional, and lets you label a line for later reference. The refs are also always optional, and refer back to previous lines (you can have any number of them). The dollar signs and spaces are *not* optional! If bits of your proof are disappearing, missing spaces are the first thing to check for.

`\premis` Since there's no rule involved, and never any need to reference previous lines, `\premis` is a bit simpler: `\premis[label] $content$`.

`\regel` This one has everything: `\regel[label] $content$ Rulename (refs)`.

`\anonregel` The anonymous regel is intended to be used for 'meta-comment' lines like "As for ψ" or "etc.". Because it's already out-of-format, as it were, it uses the basic LaTeX machinery to do the job: `\anonregel[`⟨*label*⟩`]{`⟨*content*⟩`}{`⟨*side-note*⟩`}`. Note however the the content *will* still be typeset in math mode, use `\text{}` to avoid this if necessary.

`assumptie` The `assumptie` environment uses the same argument conventions: `\begin{assumptie}[label] $content$`. Like `\premis`, there's never a rule name and never a need to refer back to previous lines. Apart from that it's a perfectly normal environment.

2.1 The standard rules (and package options)

`[prop]` Under package option `[prop]` you get the following rules:

Rule	typesets	Rule	typesets
`Iconj`	$I_\wedge$	`Econj`	$E_\wedge$
`Idisj`	$I_\vee$	`Edisj`	$E_\vee$
`Ineg`	$I_\neg$	`Eneg`	$E_\neg$
`Ipijl`	$I_\rightarrow$	`Epijl`	$E_\rightarrow$
`EFSQ`	EFSQ	`negneg`	$\neg\neg$

`[pred]` The package option `[pred]` adds four rules:

Rule	typesets	Rule	typesets
`Iuniv`	$I_\forall$	`Euniv`	$E_\forall$
`Iexist`	$I_\exists$	`Eexist`	$E_\exists$

(To use both options together: `\usepackage[prop,pred]{natded}`)

`[intuit]` The package option `[intuit]` effectively disables the `negneg` rule, by making it produce an error.

3 Examples

Here's what makes it all worthwhile. We'll begin with a very simple proof, of a very simple fact:

```
\begin{afleiding}
  \premis[penq] $p \land q$
  \regel[p]     $p$         Econj (penq)
  \regel[q]     $q$         Econj (penq)
  \regel        $q \land p$ Iconj (p,q)
\end{afleiding}
```

1. $p \land q$ prem.
2. p $\mathrm{E}_{\land}$: 1
3. q $\mathrm{E}_{\land}$: 1
4. $q \land p$ $\mathrm{I}_{\land}$: 2, 3

Notice how the layout of the code mimics the layout of the proof. I've chosen line labels that match the formula on that line; this is a good habit to get into, in case you want to rearrange things later. (Note, however, that within each afleiding the labels should be unique – LaTeX will complain if this is not the case, but not very loudly, so you need to keep and eye open.)

Next, something a little more extensive, with some assumptions.

```
\begin{afleiding}
  \premis[nnphi] $\lnot\lnot\phi$
  \premis[nnpsi] $\lnot\lnot\psi$
  \begin{assumptie}[npenp] $\lnot(\phi\land\psi)$
    \begin{assumptie}[phi] $\phi$
      \begin{assumptie}[psi] $\psi$
        \regel[penp] $\phi\land\psi$ Iconj (psi,phi)
        \regel       $\bot$          Eneg  (npenp,penp)
      \end{assumptie}%
      \regel[npsi] $\lnot\psi$  Ineg
      \regel       $\bot$       Eneg (nnpsi,npsi)
    \end{assumptie}%
    \regel[npsi2] $\lnot\psi$  Ineg
    \anonregel{}{(seeing the pattern?)}
    \regel        $\bot$       Eneg (nnpsi,npsi2)
  \end{assumptie}
  \regel $\lnot\lnot(\psi\land\phi)$  Ineg
\end{afleiding}
```

1.	$\neg\neg\phi$	prem.
2.	$\neg\neg\psi$	prem.
3.	$\neg(\phi \wedge \psi)$	ass.
4.	ϕ	ass.
5.	ψ	ass.
6.	$\phi \wedge \psi$	$I_\wedge$: 5, 4
7.	$\bot$	$E_\neg$: 3, 6
8.	$\neg\psi$	$I_\neg$
9.	$\bot$	$E_\neg$: 2, 8
10.	$\neg\psi$	$I_\neg$
11.		(seeing the pattern?)
12.	$\bot$	$E_\neg$: 2, 10
13.	$\neg\neg(\psi \wedge \phi)$	$I_\neg$

4 Advanced usage

Unless you're comfortable with LaTeX you're better off ignoring this for the moment. There is some extra machinery provided in the package for defining your own rules, through a configuration file.

4.1 Defining rules

One of the cute things this package does is turns "Eneg" in your proof into "$E_\neg$", and so on for all the standard rules. This is actually done by defining a macro `\Eneg`, and the shortcut mechanism to make this less work is also available for the user. The names of the rules can be (re)defined in the config file, so that if you prefer to call "Epijl" "MP" you can change it without too much trouble. All that's needed is to make a macro that typesets the name of the rule, then use that macro without the `\` in the proof. The double negation rule is defined like so: `\newcommand{\negneg}{\lnot\lnot}` (the macro will be evaluated in math mode), and after that definition you can use "negneg" as the name of a rule in your proofs.

`\mk@intro@regel` `\mk@elim@regel`

To make things easier, there are two macro-creation macros for elimination and introduction rules. `\mk@intro@regel{conj}{\land}` will create a macro `\Iconj`, while `\mk@elim@regel{disj}{\lor}` will create `\Edisj`. You'll have to either put these in the cfg file or surround them with `\makeatletter` and `\makeatother`, since they use the symbol `@` which is usually reserved for operation within packages.

4.2 Default contents of the config file

And finally (on the next page), the contents of the config file supplied with the distribution. You should feel free to edit this to your own taste (in that case, it might make sense to not use the package options enabling the default rules — don't forget you can copy bits of the package code too!).

```
% Introduction and elimination rules
% (Example only, so commented out -- these are provided by
% the package option [prop].)

    % rule name, no I
    %            vvvv
%\mk@intro@regel{conj}{\land}
    %                  ^^^^^
    %                  symbol for rule (will be subscripted)

    % rule name, no E
    %           vvvv
%\mk@elim@regel{disj}{\lor}
    %                 ^^^^
    %                 symbol for rule (will be subscripted)

% And then a general rule:

% Like me, you might think of 'pijl elimination' as 'modus ponens'
% (note the \textrm{}, to give roman instead of math type)

% rule name as macro (i.e., with '\')
%           vvv
\newcommand{\MP}{\textrm{MP}}
%                ^^^^^^^^^^^
%                complete replacement text (for math mode)

% Or you might want to write MP in your proofs but have it typeset
% as an elimination rule (this is part of what \mk@elim@regel does
% for you, but that also makes a rule E<rulename>):
%\newcommand{\MP}{\textrm{E}_{\rightarrow}}
```

5 Implementation

First off, let's be clear about one thing: *this is not clean TEX*. It's a horrible mess, and a competent TEXnician could probably reduce the size of the style file by half and make it more readable at the same time. Don't say I didn't warn you.

assumpties
afleidingregel
afleidingen

First we define a number of counters, to track the current proof, the line within the proof, and how many assumptions are active. The tokens register `ass@toks` will store the vertical bars showing active assumptions.

```
\newtoks\ass@toks
\newcounter{assumpties}
\newcounter{afleidingregel}
\newcounter{afleidingen}
\setcounter{afleidingen}{0}
```

We'll need to typeset the lines, corners and so on, so here are their definitions conveniently in one place: (the numerical details were trial-and-error, and should really be parameterised by font I suppose)

```
\newcommand{\ass@space}{\hspace{5pt}}
\newcommand{\ass@ul@hook}{\vrule height 3pt\hspace{-0.4pt}%
  \vrule width 0.5em height 3pt depth -2.6pt\hspace{-0.4pt}}
\newcommand{\ass@dr@hook}{\vrule depth -8pt\hspace{-0.4pt}%
  \vrule width 0.5em height 7.9pt depth -7.5pt\hspace{-0.4pt}}
\newcommand{\ass@u@fill}{\leaders\hbox{\vrule height 3pt
                                       depth -2.6pt}\hfill\hbox{}}
\newcommand{\ass@d@fill}{\leaders\hbox{\vrule height 7.9pt
                                       depth -7.5pt}\hfill\hbox{}}
```

Here's some helper code for adding and removing vertical lines as assumptions are added and retracted:

```
\newcommand{\ass@push}{%
  \edef\tmp{\the\ass@toks}%
  \global\ass@toks=\expandafter{\expandafter\vline\expandafter\ass@space\tmp}}
\def\ass@pop@sub#1\ass@space#2\e@ass@pop{\global\ass@toks={#2}}
\newcommand{\ass@pop}{\expandafter\ass@pop@sub\the\ass@toks\e@ass@pop}
\newcommand{\ass@print}{\unskip\the\ass@toks}
```

The user might (probably will) want to reuse line labels between proofs, so we need to make them globally unique. We turn the proof number into a roman numeral ('cos it's prettier that way) and do like so: user label `psi` in the second proof goes to `ii:line:psi`. So then we need forward-and-back conversion for labels and refs. (Also a fixed style for printing the line numbers inside the proofs.)

```
\newcommand{\proof@line@label}[1]{%
  \def\tempa{}\def\tempb{#1}%
  \ifx\tempa\tempb%
  \else\expandafter\label\expandafter{\roman{afleidingen}:line:#1}\fi}
\newcommand{\proof@line@ref}[1]{\expandafter\ref{\roman{afleidingen}:line:#1}}
\newcommand{\print@proof@line}{\arabic{afleidingregel}.}
```

It gets more complicated: the format for rules in a proof allows comma-separated lists of refs. This monster turns them into (nicely formatted) lists of proof-line-refs. (NB: it doesn't like spaces. And it's really really ugly.)

```
\newcommand{\proof@line@refs}[2]{%
  \def\tempa{}\def\tempb{#2}\ifx\tempa\tempb\else{#1}\fi%
```

```
  \line@refs #2,\e@line@refs}
\def\gobble@line@refs#1\e@line@refs{}
\def\line@refs#1,#2\e@line@refs{%
  \let\next@line@refs=\gobble@line@refs%
  \def\tempa{#1}\def\tempb{}\ifx\tempa\tempb\else%
    \proof@line@ref{#1}%
    \def\tempa{#2}\ifx\tempa\tempb\let\next@line@refs=\gobble@line@refs\else%
      \let\next@line@refs=\line@refs,
    \fi
  \fi
  \next@line@refs#2\e@line@refs%
}
```

`\proof@rule` Ah, finally the good stuff. This macro gets used by every line in a proof except the assumptions. Its arguments are a label (possibly empty; the labelling code will cope with that), the content of the line, and the side-note (usually the name of the rule applied and references to previous lines).

```
\newcommand{\proof@rule}[3]{\unskip\ass@print
  &\refstepcounter{afleidingregel}\print@proof@line\proof@line@label{#1}
  &\ensuremath{#2} &#3\\\ignorespaces}
```

`\anonregel` `\premis` `\regel` Now several user-level rule macros. The anonieme regel (anonymous line/rule) is provided for any proof lines that aren't according to any specified rule. It's just a handoff for `\proof@rule`.

```
\newcommand{\anonregel}[3][1]{\unskip\proof@rule{#1}{#2}{#3}}
```

A premis is a little more complicated: it uses the special formatting: `\premis[label] $content$`. The space is non-optional!

```
\def\premis{%
  \unskip\@ifnextchar[{\premis@i}{\premis@i[] }%]
}
\def\premis@i[#1] $#2${%
  \unskip\proof@rule{#1}{#2}{prem.}%
}
```

And the ugliest of 'em all, `\regel` has two optional arguments (the label and the references). Two example usages: `\regel[label] $content$ Rule (ref,ref2)` and `\regel $content$ Rule`. The code is based on what came out of Scott Pakin's python program `newcommand` (which wasn't quite up to the job but showed me the way). Note, again, spaces are non-optional.

```
\def\regel{%
  \unskip\@ifnextchar[{\regel@i}{\regel@i[] }%]
}
\def\regel@i[#1] $#2$ #3 {%
  \unskip\@ifnextchar({\regel@ii[#1] #3 $#2$ }{\regel@ii[#1] #3 $#2$ ()}%)
}
\def\regel@ii[#1] #2 $#3$ (#4){%
  \unskip\proof@rule{#1}{#3}{$\csname #2\endcsname$\proof@line@refs{: }{#4}}%
}
```

`assumptie` And finally, why this was all worthwhile: the assumptie (assumption) environment, which wraps pretty boxes around assumption-dominated lines in the proof.

```
\newenvironment{assumptie}{%
```

```
  \unskip\@ifnextchar[{\assumptie@sub}{\assumptie@sub[] }%]
}{\unskip%
  \addtocounter{assumpties}{-1}%
  \ifnum\value{assumpties}<0%
```

Despite the chatty tone of this next error message, LaTeX will not always warn about badly nested assumptions. Normally brace nesting (groups) matches environment nesting, so this is what LaTeX uses to check for those mistakes. But you can't add cells to a tabular through an intervening environment (read: group), so I close the group before typesetting the assumptie material, then fake opening it again for `\end{assumptie}`. That fake means we need the assumptie counter to make sure of proper nesting. Told you it was ugly.

```
  \errhelp{You seem to have retracted an assumption you haven't made. LaTeX
    would normally warn you that your environments don't match, but this is
    more helpful, don't you think?}%
  \errmessage{Retraction of an unmade assumption}%
  \fi
  \ass@pop
  \unskip\ass@print\ass@dr@hook\ass@d@fill
  &\hspace{-.5em}\ass@d@fill
  &\hspace{-.5em}\ass@d@fill\\[-.8em]%
  \begingroup\def\@currenvir{assumptie}\ignorespaces}

\def\assumptie@sub[#1] $#2${%
  \unskip%
  \stepcounter{assumpties}%
  \endgroup%
  \ass@print\ass@ul@hook\ass@u@fill
  &\refstepcounter{afleidingregel}\print@proof@line\proof@line@label{#1}
  &\ensuremath{#2} &ass.\ass@push\\\ignorespaces}
```

`afleiding` And the wrapper that makes it possible, by adding the tabular around each afleiding (proof) to keep everything lined up:

```
\newenvironment{afleiding}{%
  \setcounter{assumpties}{0}%
  \setcounter{afleidingregel}{0}%
  \stepcounter{afleidingen}%
  \ass@toks={}%
  \begin{tabular}{l@{\hspace{0.4em}}l@{\hspace{0.4em}}ll}%
}{
  \ifnum\value{assumpties}>0
  \errhelp{Some assumption(s) in the afleiding have not been retracted
    before the end of the proof. This means your afleiding is broken!}
  \errmessage{Assumptions still active at end of afleiding}
  \fi
  \end{tabular}
}
```

`\intro@regel` `\elim@regel` `\mk@intro@regel` `\mk@elim@regel` Now some helper functions for defining rule types. These will most likely be used in the cfg file.

```
\newcommand{\intro@regel}[1]{\textrm{I}_{#1}}
\newcommand{\elim@regel}[1]{\textrm{E}_{#1}}
\newcommand{\mk@intro@regel}[2]{%
  \expandafter\newcommand\csname I#1\endcsname{\intro@regel{#2}}}
```

```
\newcommand{\mk@elim@regel}[2]{%
  \expandafter\newcommand\csname E#1\endcsname{\elim@regel{#2}}}
```

Using these, define some standard rules as package options.

```
\DeclareOption{prop}{
  \mk@intro@regel{conj}{\land}
  \mk@elim@regel{conj}{\land}
  \mk@intro@regel{disj}{\lor}
  \mk@elim@regel{disj}{\lor}
  \mk@intro@regel{neg}{\lnot}
  \mk@elim@regel{neg}{\lnot}
  \mk@elim@regel{pijl}{\rightarrow}
  \mk@intro@regel{pijl}{\rightarrow}
  \newcommand{\EFSQ}{\textrm{EFSQ}}
  \newcommand{\negneg}{\ensuremath{\lnot\lnot}}
  \newcommand{\herhaling}{\textrm{her.}}
}
\DeclareOption{pred}{
  \mk@intro@regel{univ}{\forall}
  \mk@elim@regel{univ}{\forall}
  \mk@intro@regel{exist}{\exists}
  \mk@elim@regel{exist}{\exists}
}
```

The `[intuit]` package option is an odd one; it redefines the `negneg` rule to produce an error, since this error isn't allowed in intuitionistic derivations. Basically this is blatant showing off.

```
\DeclareOption{intuit}{
  \renewcommand{\negneg}{
    \errhelp{The double-negation rule is not allowed in intuitionistic
      derivations. If you meant to write a classical derivation, don't use
      the package option [intuit].}
    \errmessage{Double-negation rule in intuitionistic afleiding}
  }
}
```

Finally, we process the options (if any) and read the user config file.

```
\ProcessOptions\relax
\InputIfFileExists{natded.cfg}{}{}
```