Dednat6: Some Extra Features

Eduardo Ochs

January 16, 2019

At this moment the documentation of Dednat6 consists of:

- An article about Dednat6 for TUGBoat, called "Dednat6: An extensible (semi-)preprocessor for LuaLaTeX that understands diagrams in ASCII art", that explains all the main concepts and how they are implemented. Link:

  `http://angg.twu.net/LATEX/2018tugboat-rev1.pdf`

- The slides for a presentation with the same title (on the TUG2018 conference). Link:

  `http://angg.twu.net/LATEX/2018tug-dednat6.pdf`

- *This document*, that complements the article and the slides — with installation instructions, plus lots of details and technicalities. Link:

  `http://angg.twu.net/LATEX/2018dednat6-extras.pdf`

See also:
`http://angg.twu.net/dednat6.html`
`http://angg.twu.net/math-b.html#tug-2018`

1 Hey!

The slides for the TUG talk end with this call for feedback:

> I've stopped trying to document Dednat6 because
> 1) I don't have a mental image of who I am writing for,
> 2) I get *far too little feedback*,
> 3) all of the feedback that I got came from people who felt that I was not writing for them — my approach, tone and choice of pre-requisites were all wrong.
> If you would like to try Dednat6, get in touch, **let's chat** — *please*!
> *Maybe I can typeset in 20 minutes a diagram that took you*
> *a day, maybe I can implement an extension that you need...*

This still holds! I'm `eduardoochs@gmail.com`, get in touch!

2 Downloading

You can download the current version of dednat6 from http://angg.twu.net/dednat6.html, or from here: http://angg.twu.net/dednat6.zip. The .zip file contains documentation in PDF and source form. To download it, delete the PDFs and recompile them, do this:

```
rm -Rfv /tmp/dn6-test/
mkdir   /tmp/dn6-test/
cd      /tmp/dn6-test/
wget http://angg.twu.net/dednat6.zip
unzip dednat6.zip
rm -v *.pdf
lualatex 2018tugboat-rev1.tex
lualatex 2018tugboat-rev1.tex
lualatex 2018tug-dednat6.tex
lualatex 2018tug-dednat6.tex
lualatex 2018dednat6-extras.tex
lualatex 2018dednat6-extras.tex
lualatex 2018dednat6-minimal.tex
```

Dednat6 itself consists of just the contents of the `dednat6/` directory plus the file `dednat6load.lua`. You can run the code below to check that that's what is needed to compile `2018dednat6-minimal.tex`:

```
rm -Rfv /tmp/dn6-test-min/
mkdir   /tmp/dn6-test-min/
cd      /tmp/dn6-test-min/
wget http://angg.twu.net/dednat6.zip
unzip dednat6.zip "dednat6/**" dednat6load.lua 2018dednat6-minimal.tex
lualatex 2018dednat6-minimal.tex
```

3 The preamble

It *should* be possible to load Dednat6 from a .tex file with a single command, like this:

```
\documentclass{article}
  \directlua{dofile "dednat6load.lua"}
\begin{document}
(...)
\end{document}
```

but due to some quirks this is not possible at this moment, and we have to do this (the indented lines):

```
\documentclass{article}
  \usepackage{proof}
  \input diagxy
  \xyoption{curve}
\begin{document}
  \catcode`\^^J=10
  \directlua{dofile "dednat6load.lua"}
(...)
\end{document}
```

The ‘`\catcode`’ is needed to make the newlines in the TEX code generated by dednat6 be interpreted as newlines and not as ‘Ω’s (the “spurious omega problem”); ‘`\usepackage{proof}`’ loads a package for typesetting proof trees, ‘`\input diagxy`’ loads XYPic and the `diagxy` extension (sec.10), and `\xyoptioncurve` loads an extension for `diagxy` that allows drawing curved arrows.

One of the things that `\directlua{dofile "dednat6load.lua"}` does is that it runs this from Lua:

$$\texttt{output}(\texttt{preamble1})$$

`preamble1` is a chunk of TEX code defined in the file `dednat6/preamble.lua`, that contains material like this:

```
\def\defdiag#1#2{\expandafter\def\csname diag-#1\endcsname{\bfig#2\efig}}
\def\ifdiagundefined#1{\expandafter\ifx\csname diag-#1\endcsname\relax}
\def\diag#1{\ifdiagundefined{#1}
    \errmessage{UNDEFINED DIAGRAM: #1}
  \else
    \csname diag-#1\endcsname
  \fi
}
%
\def\expr#1{\directlua{output(tostring(#1))}}
\def\eval#1{\directlua{#1}}
\def\pu{\directlua{pu()}}
```

The file `dednat6/preamble.lua` also defines a ‘`preamble0`’ with the ‘`\usepackage`’s and ‘`\input`’s that would have to be run before the ‘`\begin{document}`’, and it explains in comments why this currently doesn’t work. See the source.

4 Other inference bars

All the examples of deduction trees in the TUGBoat article use ‘`-`’s for the inference bars in the ASCII art representation. If we use ‘`=`’s instead of ‘`-`’s we get double bars, and if we use ‘:’s we get a line of vertical dots instead of a bar:

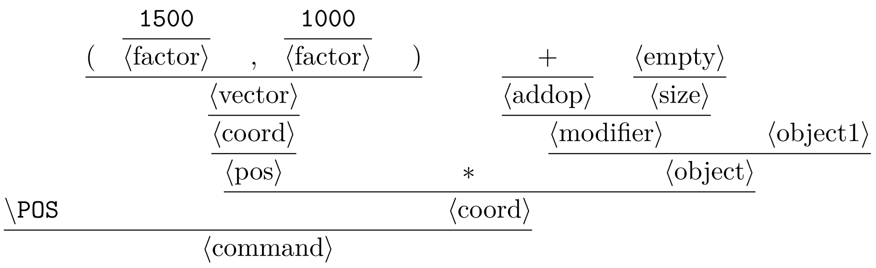
$$\dfrac{\Gamma, P \vdash R \quad \Gamma, Q \vdash R}{\Gamma, P \lor Q \vdash R} \qquad \Longrightarrow \qquad \dfrac{\begin{matrix}\Gamma \\ \vdots \\ P \lor Q\end{matrix} \quad \begin{matrix}\Gamma \quad [P]^1 \\ \vdots \\ R\end{matrix} \quad \begin{matrix}\Gamma \quad [Q]^1 \\ \vdots \\ R\end{matrix}}{R}\,1$$

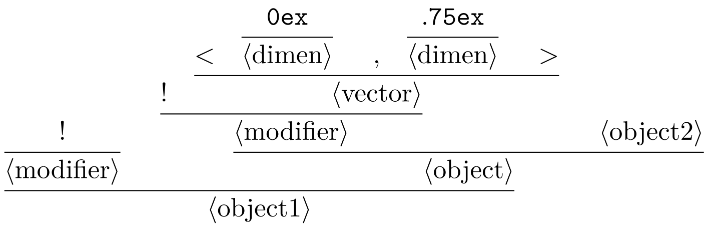
You can change the number of vertical dots by redefining the macro ‘\DeduceSym’. For example:

```
\makeatletter
% Original with 4 dots (from proof.sty):
% \def\DeduceSym{\vtop{\baselineskip4\p@ \lineskiplimit\z@
%     \vbox{\hbox{.}\hbox{.}\hbox{.}}\hbox{.}}}
% New, with 3 dots:
\def\DeduceSym{\vtop{\baselineskip4\p@ \lineskiplimit\z@
    \vbox{\hbox{.}\hbox{.}}\hbox{.}}}
\makeatother
```

5 Changing how tree nodes are LaTeX’ed

The trees below were produced by changing *temporarily* the function that converts each tree node from ascii to TeX code. TO DO: explain this; for the moment look at the source code of this section.





6 Abbrevs

The first Dednats did not support UTF-8, and the way to write a tree node that would display as ‘$a \to b$’ was to write it as ‘a->b’ after running addabbrevs("-

`>", "\to ")`. The module `abbrevs.lua` implements this, and `unabbrev(str)` parses `str` from left to right, at each point looking for the longest string starting at that point that is an abbrev and replacing it by its expansion, or leaving that character untouched if it doesn't have an expansion. Here is an example:

```
%L addabbrevs("->", "\\to ")
%:
%:  [a]^1  a->b
%:  -----------
%:       b       b->c
%:       ------------
%:            c
%:          ----1
%:          a->c
%:
%:          ^a->c
%:
$$\pu \ded{a->c}$$
```

→

$$\cfrac{\cfrac{\cfrac{[a]^1 \quad a \to b}{b} \qquad b \to c}{c}}{a \to c}\;1$$

Abbrevs are also used in 2D diagrams, in a more complex way. Section 2.2 of the TUGBoat article explains how the grid words create a table `nodes`, but it doesn't explain how the fields `.tex` and `.TeX` in a node affect how it is displayed. The code below creates nodes whose *tags* are `"A"`, `"B"`, `"C"`, `"D"`, and then changes the fields `.tex` and `.TeX` in some of these nodes. The TeX code for each node is calculated by the function `node_to_TeX`, that expects a node (a table) and returns a string. If `node_to_TeX` receives a node that has a `.TeX` field then it returns that field unchanged, surrounded by '`{}`'s; if it doesn't have a `.TeX` field but it has a `.tex` field then it returns the result of running `unabbrev` on that field and surrounding it with '`{}`'s; otherwise it returns the result of running `unabbrev` on the tag surrounding it with '`{}`'s. For example:

```
%D diagram nodes-and-abbrevs
%D 2Dx     100 +40
%D 2D  100 A -> B
%D 2D      |    |
%D 2D      v    v
%D 2D  +30 C -> D
%D 2D
%D (( B .tex= (a->b)
%D    C             .TeX= (a->b)
%D    D .tex= (a->b) .TeX= (a->b)
%L print("nodes:"); print(nodes)
%L print("A:", node_to_TeX(nodes["A"]))
%L print("B:", node_to_TeX(nodes["B"]))
%L print("C:", node_to_TeX(nodes["C"]))
%L print("D:", node_to_TeX(nodes["D"]))
%D    A B -> A C -> B D -> C D ->
%D ))
%D enddiagram
```

→

$$\begin{tikzcd} A \arrow[r] \arrow[d] & (a \to b) \arrow[d] \\ (a- > b) \arrow[r] & (a- > b) \end{tikzcd}$$

The output of the `print()`s is:

```
nodes:
{   1={"noden"=1, "tag"="A", "x"=100, "y"=100},
    2={"noden"=2, "tag"="B", "x"=140, "y"=100, "tex"="(a->b)"},
    3={"noden"=3, "tag"="C", "x"=100, "y"=130,                 "TeX"="(a->b)"},
    4={"noden"=4, "tag"="D", "x"=140, "y"=130, "tex"="(a->b)", "TeX"="(a->b)"},
  "A"={"noden"=1, "tag"="A", "x"=100, "y"=100},
  "B"={"noden"=2, "tag"="B", "x"=140, "y"=100, "tex"="(a->b)"},
  "C"={"noden"=3, "tag"="C", "x"=100, "y"=130,                 "TeX"="(a->b)"},
  "D"={"noden"=4, "tag"="D", "x"=140, "y"=130, "tex"="(a->b)", "TeX"="(a->b)"}
}
A:      {A}
B:      {(a\to b)}
C:      {(a->b)}
D:      {(a->b)}
```

7 Renaming

The word `ren` in the language for 2D diagrams eats the rest of the line, splits it at the ‘`==>`’, and splits the material before the ‘`==>`’ into a list of *tags*, $A_1, \ldots, A_n$, and the material after ‘`==>`’ into a list of *texs*, $B_1, \ldots, B_n$; these two lists must have the same length, and then `ren` runs `nodes[`A_i`].tex = `B_i for each i in $1, \ldots, n$. For example:

```
%D diagram ren
%D 2Dx     100    +30
%D 2D  100 A1 <-| A2
%D 2D      |       |
%D 2D      v       v
%D 2D  +30 A3 |-> A4
%D 2D
%D ren A1 A2 ==> A{\times}B A
%D ren A3 A4 ==> C B{\to}C
%D
%D (( A1 A2 <-| .plabel= a ({\times}B)
%D    A1 A3 -> A2 A4 ->
%D    A3 A4 |-> .plabel= b (B{\to})
%D ))
%D enddiagram
```

$\to$

$$\begin{tikzcd}
A{\times}B \arrow[d] & A \arrow[l, mapsto, "(\times B)"'] \arrow[d] \\
C \arrow[r, mapsto, "(B\to)"'] & B{\to}C
\end{tikzcd}$$

8 Arrow modifiers

The language for 2D diagrams in dednat6 has some words for curving and sliding arrows:

```
%D diagram curve-slide
%D 2Dx     100 +40
%D 2D  100 A   B
%D 2D
%D 2D  +40 D   C
%D 2D
%D (( A B ->               .plabel= b \text{one}
%D    A B -> .slide=  5pt  .plabel= a \text{two}
%D    A B -> .slide= 20pt  .plabel= a \text{three}
%D    B C -> .curve= _10pt .plabel= l \text{four}
%D    B C -> .curve=  ^5pt .plabel= r \text{five}
%D    C D ->
%D    C D -> .curve=  _5pt
%D    C D -> .curve=  _5pt .slide= -5pt
%D           .plabel= a \text{six}
%D ))
%D enddiagram
```

→

$$\begin{tikzcd}
A \arrow[r, "\text{one}"'] \arrow[r, shift left=2, "\text{two}"] \arrow[r, shift left=8, "\text{three}"] & B \arrow[d, bend right, "\text{four}"'] \arrow[d, bend left, "\text{five}"] \\
D & C \arrow[l] \arrow[l, bend right] \arrow[l, bend right, shift right=2, "\text{six}"']
\end{tikzcd}$$

The words ‘`sl^^`’, ‘`sl^`’, ‘`sl_`’, and ‘`sl__`’ are abbreviations for “`.slide= 5pt`”, “`.slide= 2.5pt`”, “`.slide= -2.5pt`”, “`.slide= -5pt`” respectively.

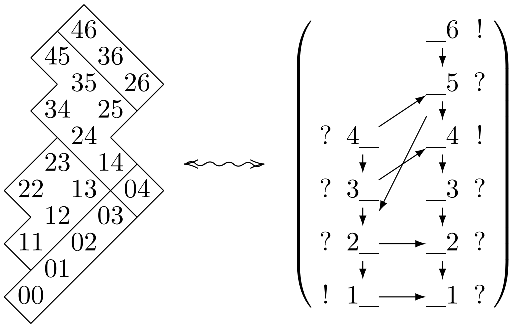
9 Strange modules

The code of Dednat6 — inside the directory `dednat6/` — is made of several `.lua` files that are *all* loaded by `dednat6.lua`; there is no provision yet for loading only the modules that are used in a given `.tex` file. This means that some modules that are only useful to the author of Dednat6 (Eduardo Ochs, a.k.a. “me”) are always loaded.

Most of these extra modules were written to handle the objects described in my series of papers about “Planar Heyting Algebras”, at:

`http://angg.twu.net/math-b.html#zhas-for-children-2`

Here’s an example of what they produce:



Even though these modules are not useful to other people some *ideas* in them may be. (TO DO: give examples!)

10 Versions of diagxy

Diagxy comes in two versions: the original one, `diagxy.tex`, that is loaded with `\input diagxy`, and another one, `xybarr.tex`, that is a module of xypic and is loaded with `\usepackage[barr,pdf]{xy}`. If you have a recent texlive, installed with, say,

`https://www.tug.org/texlive/quickinstall.html`

then you should have these .tex files and their documentation files at places like these:

```
/usr/local/texlive/2018/texmf-dist/tex/generic/barr/diagxy.tex
/usr/local/texlive/2018/texmf-dist/doc/generic/barr/diaxydoc.pdf
/usr/local/texlive/2018/texmf-dist/tex/generic/xypic/xybarr.tex
/usr/local/texlive/2018/texmf-dist/doc/generic/xypic/barrdoc.pdf
```

You can access their docs online at these URLs:
`http://www.math.mcgill.ca/barr/papers/diaxydoc.pdf`
`http://tug.ctan.org/tex-archive/macros/latex/contrib/xypic/doc/barrdoc.pdf`

It seems that loading diagxy with `\usepackage[barr,pdf]{xy}` is incompatible with lualatex, and thus with Dednat6. Until we get that fixed please use `\input diagxy` instead.

11 “A few samples”

The file `barrdoc.pdf` has a section called “A few samples” (5.9), that shows how to produce a certain big diagram in two ways; one, that we will not discuss, using ‘`\morphism`’, and another one, that we will call “Barr’s diagram”, that uses ‘`\node`’ and ‘`\arrow`’ instead. In the next two pages we will compare Barr’s diagram to an approximate translation of it to Dednat6.

Note: I don’t quite understand Barr’s code for the two outermost curved arrows — it *seems* that he resorts to low-level XYPic code in the “modifier” part of the shape parameter to create splined arrows, but I couldn’t figure out the exact meaning of “`` @`{c,(3000,0),(2700,2800),p} ``” in the XYPic reference manual...

```
% Source code for Barr's diagram:
%
$$\bfig
  \def\f{\bar f}
  \def\g{\bar g}
  \def\h{\bar h}
  \let\t\tau
  \node A11(0,2800)[(\h(\g\f))\t_A]
  \node A13(1200,2800)[((\h\g)\f)\t_A]
  \node A21(0,2400)[\h((\g\f)\t_A)]
  \node A22(600,2400)[\h(\g\f\t_A)]
  \node A23(1200,2400)[(\h\g(\f\t_A))]
  \node A32(600,2000)[\h(\g(\t_Bf))]
  \node A33(1200,2000)[(\h\g)(\t_Bf)]
  \node A34(1800,2000)[((\h\g)\t_B)f]
  \node A42(600,1600)[\h((\g\t_B)f)]
  \node A44(1800,1600)[(\h(\g\t_B))f]
  \node A52(600,1200)[\h((\t_C)g)f]
  \node A54(1800,1200)[(\h(\t_Cg))f]
  \node A62(600,800)[\h(\t_C(gf))]
  \node A63(1200,800)[(\h\t_C)(gf)]
  \node A64(1800,800)[\h(\t_C(gf))]
  \node A73(1200,400)[(\t_Dh)(gf)]
  \node A74(1800,400)[((\t_D)h)g]
  \node A75(2400,400)[(\t_D(hg))f]
  \node A83(1200,0)[\t_D(h(gf))]
  \node A85(2400,0)[\t_D((hg)f)]
  \arrow[A11`A13;]
  \arrow[A21`A11;]
  \arrow[A21`A22;]
  \arrow[A22`A23;]
  \arrow[A23`A13;]
  \arrow[A32`A22;\h(\g\t_f)]
  \arrow[A32`A33;]
  \arrow[A33`A23;(\h\g)\t_f]
  \arrow[A33`A34;]
  \arrow[A42`A44;]
  \arrow[A42`A32;]
  \arrow[A44`A34;]
  \arrow[A52`A42;\h(\t_gf)]
  \arrow[A52`A54;]
  \arrow[A54`A44;(\h\t_g)f]
  \arrow[A62`A52;]
  \arrow[A62`A63;]
  \arrow[A63`A64;]
  \arrow[A73`A63;\t_h(gf)]
  \arrow[A73`A74;]
  \arrow[A74`A64;\t_{(hg)f}]
  \arrow[A74`A75;]
  \arrow[A83`A73;]
  \arrow[A83`A85;]
  \arrow[A85`A75;]
  \arrow|r|/{@{>}@/_15pt/}/[A75`A34;\t_{hg}f]
  \arrow|l|/{@{>}@/^15pt/}/[A62`A21;\h(\t_C(gf))]
  \arrow|l|/{@{>}@`{c,(3000,0),(2700,2800),p}}/[A85`A13;\t_{hg}f]
  \arrow|r|/{@{>}@`{c,(-300,0),(-600,2400),p}}/[A83`A11;\t_{h(fg)}]
  \efig
$$
```

```
% Source code for its translation to Dednat6:
%
%D diagram barr-dednat6
%D 2Dx     100    +40    +40    +40    +40
%D 2D  100 A11 --------> A13
%D 2D       ^             ^
%D 2D       |             |
%D 2D  +27 A21 -> A22 -> A23
%D 2D       ^      ^      ^
%D 2D       |      |      |
%D 2D  +27  |     A32 -> A33 -> A34
%D 2D       |      ^             ^  ^
%D 2D       |      |             |   \
%D 2D  +27  |     A42 --------> A44   \
%D 2D       |      ^             ^     \
%D 2D        \     |             |      |
%D 2D  +27    \   A52 --------> A54     |
%D 2D          \   ^                    |
%D 2D           \  |                    |
%D 2D  +27        A62 -> A63 -> A64     |
%D 2D                     ^      ^      |
%D 2D                     |      |      |
%D 2D  +27               A73 -> A74 -> A75
%D 2D                     ^             ^
%D 2D                     |             |
%D 2D  +27               A83 --------> A85
%D 2D
%D ren A11     A13   ==>   (\h(\g\f))\t_A              ((\h\g)\f)\t_A
%D ren A21 A22 A23   ==>   \h((\g\f)\t_A) \h(\g\f\t_A) (\h\g(\f\t_A))
%D ren     A32 A33 A34     ==>            \h(\g(\t_Bf)) (\h\g)(\t_Bf) ((\h\g)\t_B)f
%D ren     A42     A44     ==>            \h((\g\t_B)f)               (\h(\g\t_B))f
%D ren     A52     A54     ==>            \h((\t_C)g)f                (\h(\t_Cg))f
%D ren     A62 A63 A64     ==>            \h(\t_C(gf)) (\h\t_C)(gf) \h(\t_C(gf))
%D ren         A73 A74 A75 ==>                          (\t_Dh)(gf) ((\t_D)h)g (\t_D(hg))f
%D ren         A83     A85 ==>                          \t_D(h(gf))            \t_D((hg)f)
%D
%D (( # Horizontal arrows:
%D    A11 A13 ->
%D    A21 A22 -> A22 A23 ->
%D    A32 A33 -> A33 A34 ->
%D    A42 A44 ->
%D    A52 A54 ->
%D    A62 A63 -> A63 A64 ->
%D    A73 A74 -> A74 A75 ->
%D    A83 A85 ->
%D
%D    # Simple vertical arrows:
%D    A11 A21 <-                       A13 A23 <-
%D    A22 A32 <- .plabel= r \h(\g\t_f) A23 A33 <- .plabel= r (\h\g)\t_f
%D    A32 A42 <-                       A34 A44 <-
%D    A42 A52 <- .plabel= r \h(\t_gf)  A44 A54 <- .plabel= r (\h\t_g)f
%D    A52 A62 <-
%D    A63 A73 <- .plabel= r \t_h(gf)   A64 A74 <- .plabel= r \t_{(hg)f}
%D    A73 A83 <- A75 A85 <-
%D
%D    # Curved vertical arrows:
%D    A75 A34 -> .curve= _15pt .plabel= r \t_{hg}f
%D    A62 A21 -> .curve= ^15pt .plabel= l \h(\t_C(gf))
%D    A83 A11 -> .mod= @`{c,(-300,-2835),(-800,-100),p} .plabel= r \t_{h(fg)}
%D    A85 A13 -> .mod= @`{c,(3000,-2000),(2700,-500),p} .plabel= l \t_{hg}f
%D
%D ))
%D enddiagram
%D
$$\pu
  \def\f{\bar f}
  \def\g{\bar g}
  \def\h{\bar h}
  \let\t\tau
  \diag{barr-dednat6}
$$
```

Barr's diagram:

$$\begin{tikzcd}
(\bar h(\bar g\bar f))\tau_A \arrow[rr] & & ((\bar h\bar g)\bar f)\tau_A & & \\
\bar h((\bar g\bar f)\tau_A) \arrow[u] \arrow[r] & \bar h(\bar g\bar f\tau_A) \arrow[r] & (\bar h\bar g(\bar f\tau_A)) \arrow[u] & & \\
& \bar h(\bar g(\tau_B f)) \arrow[u, "\bar h(\bar g\tau_f)"'] \arrow[r] & (\bar h\bar g)(\tau_B f) \arrow[u, "(\bar h\bar g)\tau_f"'] \arrow[r] & ((\bar h\bar g)\tau_B)f & \\
& \bar h((\bar g\tau_B)f) \arrow[u] \arrow[rr] & & (\bar h(\bar g\tau_B))f \arrow[u] & \\
& \bar h((\tau_C)g)f \arrow[u, "\bar h(\tau_g f)"'] \arrow[rr] & & (\bar h(\tau_C g))f \arrow[u, "(\bar h\tau_g)f"'] & \\
& \bar h(\tau_C(gf)) \arrow[u] \arrow[r] \arrow[uuuul, bend left, "\bar h(\tau_C(gf))"] & (\bar h\tau_C)(gf) \arrow[r] & \bar h(\tau_C(gf)) & \\
& & (\tau_D h)(gf) \arrow[u, "\tau_{h(gf)}"'] \arrow[r] & ((\tau_D)h)g \arrow[u, "\tau_{(hg)f}"'] \arrow[r] & (\tau_D(hg))f \arrow[uuuul, bend right, "\tau_{hg}f"] \\
& & \tau_D(h(gf)) \arrow[u] \arrow[rr] \arrow[uuuuuuull, bend left=60, "\tau_{h(fg)}"] & & \tau_D((hg)f) \arrow[u] \arrow[uuuuuuull, bend right=60, "\tau_{hg}f"']
\end{tikzcd}$$

My approximate translation of it to dednat6:

$$\begin{tikzcd}
(\bar h(\bar g\bar f))\tau_A \arrow[rr] & & ((\bar h\bar g)\bar f)\tau_A & & \\
\bar h((\bar g\bar f)\tau_A) \arrow[u] \arrow[r] & \bar h(\bar g\bar f\tau_A) \arrow[r] & (\bar h\bar g(\bar f\tau_A)) \arrow[u] & & \\
& \bar h(\bar g(\tau_B f)) \arrow[u, "\bar h(\bar g\tau_f)"'] \arrow[r] & (\bar h\bar g)(\tau_B f) \arrow[u, "(\bar h\bar g)\tau_f"'] \arrow[r] & ((\bar h\bar g)\tau_B)f & \\
& \bar h((\bar g\tau_B)f) \arrow[u] \arrow[rr] & & (\bar h(\bar g\tau_B))f \arrow[u] & \\
& \bar h((\tau_C)g)f \arrow[u, "\bar h(\tau_g f)"'] \arrow[rr] & & (\bar h(\tau_C g))f \arrow[u, "(\bar h\tau_g)f"'] & \\
& \bar h(\tau_C(gf)) \arrow[u] \arrow[r] \arrow[uuuul, bend left, "\bar h(\tau_C(gf))"] & (\bar h\tau_C)(gf) \arrow[r] & \bar h(\tau_C(gf)) & \\
& & (\tau_D h)(gf) \arrow[u, "\tau_{h(gf)}"'] \arrow[r] & ((\tau_D)h)g \arrow[u, "\tau_{(hg)f}"'] \arrow[r] & (\tau_D(hg))f \arrow[uuuul, bend right, "\tau_{hg}f"] \\
& & \tau_D(h(gf)) \arrow[u] \arrow[rr] \arrow[uuuuuuull, bend left=60, "\tau_{h(fg)}"] & & \tau_D((hg)f) \arrow[u] \arrow[uuuuuuull, bend right=60, "\tau_{hg}f"']
\end{tikzcd}$$

12 The REPL

Section 5 of the TUGBoat article — called “A read-eval-print-loop (REPL)” — describes a way to start a Lua REPL in the middle of the compilation of a `.tex` file. The .zip file for dednat6 includes a file `2018dednat6-repl.tex` that lets you play with the REPL by running just this:

$$\mathtt{lualatex2018dednat6} - \mathtt{repl.tex}$$

13 Other back-ends

Dednat6 can be seen as a front-end for `proof.sty` and `diagxy.tex`. It shouldn’t be hard to make it generate, say, code for Tikz instead of for diagxy, and code for Sam Buss’s `bussproofs.sty` instead of for `proof.sty`. If you are interested in this, and you know Tikz/bussproofs/whatever enough to give me examples of how the output should look, get in touch!