# Creating pretty cross-format inference rules in Org mode

Mark Armstrong

Tuesday, March 17th, 2020

## Contents

Disclaimer: this project of mine was left incomplete; I'm uploading what I have for the moment, and hope to complete this idea at some later time, when I am using inference rules more.

Also, the solutions posted here have been tested only in plain HTML; I have not yet utilised the CSS that would be necessary to ensure the rules look consistent under different styles.

## 1 Introduction

In teaching and writing, I occasionally need to create inference/deduction rules.

In the past, when working only in LaTeX, I found the `bussproofs` package to be the best solution.

These days though, I work in Org, and want a simple method of writing these that satisfies these requirements.

1. The Org source
   (a) is at least as easy to type as plaintext (ideally easier), and
   (b) looks like an inference rule; i.e., there is relatively little setup noise.

2. It looks correct in *at least* HTML and PDF export; in particular,
   (a) the horizontal rule should be the correct length, and
   (b) there should be relatively little vertical space around the horizontal rule.

This gives us this rubric for judging methods:

| Criteria category | Criteria | Pass/Fail | Comments |
|---|---|---|---|
| 1. Org source | a. Ease of typing | | |
| | b. Appearance | | |
| 2. Export | a. Rule length | | |
| | b. Rule vspace | | |

This post documents my different attempts to fulfill these requirements.

## 2 Approach 1 Plaintext

Initially, I just wrote the rules in plaintext `src` blocks, i.e., `#+begin_src text`, `#+end_src text`.

Using ascii dashes isn't ideal; the rule should be solid.

```
 p   q
---- dashes
   r
```

Instead, we can use unicode `em` or `en` dashes.

```
 p   q
 em
   r
```

```
 p   q
 en
   r
```

In Org mode and HTML export, the `em` dashes work perfectly; they create a solid rule with the exact length of the the characters entered. But in the PDF export, they are wider than most characters, making the rule far too long. This is a particular problem when working with proof trees. Although this simple tree looks perfect in Org and HTML, it is gibberish in the PDF export.

```
A    B

C    D    E

     F
```

Possibly the issue with PDF export could be solved by tweaking the font used in LaTeX, but unfortunately restricting my font choice just to get pretty inference rules is too big of an ask.

In addition to the problems with horizontal length in the PDF, the vertical space around the rule is not ideal in both HTML and PDF. And I can see no way around this problem; it seems unlikely I will modify the height of the `em` dash character.

### 2.1 Conclusion

| Criteria category | Criteria | Pass/Fail | Comments |
|---|---|---|---|
| 1. Org source | a. Ease of typing | | By definition, as easy as plaintext. |
| | b. Appearance | | As good as it can get with fixed width and height. |
| 2. Export | a. Rule length | | Either dashed in Org/HTML, or too long in PDF. |
| | b. Rule vspace | | Regular character height is too much. |

## 3 Approach 2 Tables

There is an environment in Org which has horizontal rules builtin; the table!

There are several potential ways to use tables to create inference rule diagrams, and we go from the simple to the complex.

### 3.1 Approach 2.1 Starting out Basic tables

Let's start out with a simple table, with the contents centered.

| păăăq |
|:---:|
| r |

Right off the bat, this has the advantage that the horizontal rule is *exactly* the right width in every format, and its vertical spacing is *far* better than the plaintext solution.

This table was generated from

```
#+attr_HTML: :frame void
|  <c>  |
| păăăq |
|----|
|   r   |
```

where the `:frame void` setting for HTML omits the top and bottom rule that would usually be present for tables. The LaTeX export does not have these rules by default.

We must tweak the HTML slightly further, though; the top row is recognised as a header row, and so it's wrapped in `<th><\th>` and in the default style is thus bolded. We can define a new style for table headers as so:

```
#+HTML_HEAD: <style type="text/css">th {font-weight:
↪  normal;}</style>
```

and then repeat our table, with the setting `:class noboldheader` added. Additional settings may be necessary depending upon your CSS.

| păăăq |
|:---:|
| r |

There are two drawbacks that remain, that will require us to refine this appraoch; one relatively insignificant, and the other crippling.

1. In both the HTML and PDF export, we must in fact use *non-breaking* spaces to separate the premises; otherwise, the export will not retain the spacing between them.

2. **We are completely unable to write a name for the inference rule next to the horizontal rule**!

    - Org tables are rather simple, and so a row can only contain a horizontal rule or text, not both!

### 3.1.1 Conclusion

Given problem 2 above, it's not even worth evaluating this approach; it's not a valid solution. But it may be that we can extend it to become one.

## 3.2 Approach 2.2 Enabling names Manual horizontal rule

First and foremost, we need to solve the conflict between using the horizontal rule and the name of the inference rule.

Given the limitations of Org tables, it seems the only solution is to not use a proper horizontal rule. Instead, we'll have a cell containing a macro which creates rules in HTML and LaTeX.

```
#+Macro: bar1 @@latex:\hrulefill@@@@html:<hr>@@
```

Applying this

```
#+attr_HTML: :frame void :class noboldheader
|    <c>     |      |
| ăpăăăăqă   |      |
| {{{bar1}}} | name |
|     r      |      |
```

gives us

| | |
|---|---|
| ăpăăăăqă | |
| ——————— | name |
| r | |

But with this solution we lose the nice vertical spacing around the rule.

This is easy enough to solve in the HTML. We simply set the table's `cellpadding` to 0. At this point, we do have to introduce a non-breaking space before the `name`, or it will be directly next to the rule.

| | |
|---|---|
| ăpăăăăqă | |
| ——————— | ăname |
| r | |

On the LaTeX side, we do fortunately have the ability to manipulate vertical space on the page by calling `\vspace{_}` with negative amounts. We augment our macro

```
#+Macro: bar2 @@latex:\hrulefill\vspace{-0.3em}@@@@html:<hr>@@
```

and replace `{{{bar1}}}` with `{{{bar2}}}` to see that the gap above the rule is much improved,

| | |
|---|---|
| ăpăăăăqă | |
| ——————— | ăname |
| r | |

but the space above the rule is more concerning. For dealing with that, we can introduce a macro to wrap the premises for good measure, we also place the non-breaking spaces in there.

```
#+Macro: premise1 @@latex:\vspace{-0.7em}@@ă$1ăăă$2ă
```

and apply it to admire a perfect recreation of the spacing we had with a simple table.

$$\frac{\text{ăpăăăqă}}{\text{r}} \quad \text{ăname}$$

That said, we are still left with some smaller issues.

1. The `premise1` macro expects exactly two arguments; we would like to have a variable number of premises.
2. It would be better to allow a variable amount of space between premises.
3. The non-breaking space we entered before `name` to account for the lack of padding in the HTML creates too much space in the PDF.
4. The premises and rules should ideally use *math font*, not plain font.

We can solve all of these with further macro development.

## 3.3 Approach 2.3 One step forward, two steps back A macro for premises

Spoiler: the development here turns out to not be useable in a table cell; still, it is a useful step forward. It just causes a few steps away from a pretty solution in Org, which have to be corrected for afterward.

### 3.3.1 A bit of elisp

We start with a little bit of elisp code which our macro will use.

```
(let*
    ;; Find the last non-empty string,
  ((last-nonempty (position "" premises :test-not 'equal
  ↪  :from-end t))
    ;;  and take only up to that element of the list.
    (premises (seq-take premises (+ last-nonempty 1))))
  ;; Join the resulting list using the spacer, with the
  ↪  wrapper around
  ;; each element.
  (mapconcat (lambda (p) (format wrapper p)) premises spacer))
```

This code block has three `var` arguments:

- `premises`, which defaults to `()`, and should be a list of strings. Note that trailing emptystrings will be deleted the reason for this will be seen when we get to defining macros.
- `wrapper`, which defaults to `%s`, and which should be a format string this will be applied to each premise, to allow for formatting.
- `spacer`, which defaults to a string of three non-breaking spaces, this will be inserted between each premise.

### 3.3.2 Using the elisp: inline calls

We can make use of this code block, which is named `format-premises`, by an *inline source code call*, with the syntax such as

```
call_format-premises(wrapper="*%s*",
                     spacer=" , ",
                     premises='("hello" "world"))
```

See the Org manual regarding inline source code calls.

This call wraps each of the words `hello` and `world` with asterisks, which is the Org syntax for bold, and separates them with three dashes. Admire the results in the export: `*hello* , *world*`

Not quite right, actually. By default, the results are wrapped in

- verbatim emphasis delimiters, `=`, and outside of those,
- an invocation of the `results` macro, which as far as I can tell is simply used to demarcate the results, so that successive `C-c C-c` invocations will overwrite them, rather than prepend them.

This can be seen by hitting `C-c C-c` on the `call`, which results in the following being placed right after it.

```
{{{results(=*hello* , *world*=)}}}
```

We definitely don't want the verbatim wrapper, and can do without the `results` macro wrapper in the end use-case of this `call`, users would not be manually invoking it.

Thankfully, the wrappers can be replaced via the optional `<end header arguments>` argument, placed in (square) brackets after the arguments. These affect the result of the code block an additional optional argument, `<inside header arguments>`, would come before the arguments and would apply to the source block itself; we have no need of this. We could use `:results raw` to eliminate both wrappers. That is,

```
call_format-premises(wrapper="*%s*",
                     spacer=" , ",
                     premises='("hello" "world"))[:results
                      ↪  raw]
```

results in the expected output

```
*hello* , *world*
```

which exports as expected, as seen here: **hello** , **world**

### 3.3.3 Hiding the inline calls: introducing the `premise2` macro

Now, this invocation is too intrusive for our purposes. But we can prettify it into a macro! Note the linebreaks here are only for presentation; they cannot be present in the actual macro.

```
#+Macro: premise2
  call_format-premises(
     wrapper="$1",
     spacer="$2",
     premises='("$3" "$4" "$5" "$6"
                "$7" "$8" "$9" "$10"))
   [:results raw]
```

You'll notice that we have lost some flexibility in the translation: the macro allows for 8 premise arguments I say allows for because omitting arguments does not cause any errors whereas `format-premises` allows an arbitrary number. Unfortunately, it seems Org macros don't have support for an arbitrary number of arguments. In this case, it seems an acceptable loss; if you have more than 8 premises or honestly, more than 4 then this inference rule presentation is probably the wrong tool for you.

In any case, our macro looks promising; if we invoke it

```
{{{premise2(ăăă,/%s/,p,q)}}}
```

we get nicely formatted output: *p/ăăă/q*

### 3.3.4 Where it falls apart: no inline calls in tables

Unfortunately, as promising as this development was, we've reached a point where it falls apart. If we place our macro into the table,

```
#+attr_HTML: :frame void :cellpadding 0 :class noboldheader
|            <c>            |                 |
| {{{premise2(/%s/,ăăă,p,q)}}} |                 |
|         {{{bar2}}}        | @@html:ă@@name |
|            /r/            |                 |
```

we unfortunately notice on export that it just expands to the text of the `call`.

| call_format-premises(wrapper="*%s*",spacer="ăăă",premises='("p" "q" "" "" "" "" "" ""))[:results raw] |
|:---:|
| *r* |

This is a limitation of Org tables! They do not support inline code in their cells.

So, we find ourselves at a bit of a dead end. But there is a solution, which can make use of many of the pieces we constructed here; table formulas!

## 3.4 Approach 2.4 Automating premises 2.0 Stepping forward again

While we cannot place inline code invocations in table cells, there is a way to compute cell contents by invoking elisp code: table formulas!

At this point, the reader may wonder why I didn't immediately consider table formulas; the reason is the *appearance in Org* criteria. By placing the premise calculation in a table formula, the table is naturally less shaped like an inference rule. This means the first few solutions I work through here are not ideal by my standards. Thankfully, this can be mitigated with some cleverness; we'll get there eventually.

### 3.4.1 Writing the premises in a formula

As a first venture, consider:

```
#+attr_HTML: :frame void :cellpadding 0 :class noboldheader
|    <c>     |                              |
| /p/ăăă/q/  | @@latex:\vspace{-0.7em}@@ |
| {{{bar2}}} | modus-ponens                 |
|    /r/      |                              |
#+tblfm: @2$1='(org-sbe format-premises (wrapper "\"/%s/\"")
↪  (spacer "\"ăăă\"") (premises "p" "q"))
```

The function `org-sbe` (Org-source babel execute) is necessary to invoke code in a source block within this file. It takes as argument the name of the code block as a string or as, as we have chosen, a symbol, as well as lists for the arguments of the code block, in the form `(argument-name value value value)` so, allowing for lists of arguments where each value should be a string. The premise cell can then be auto-filled by invoking the table formula with `C-c C-c`. Also, notice we have moved the LaTeX code to move up the horizontal rule to the cell next to the premise, since we are no longer use a macro. With all that in place, here is the resulting table:

$$\frac{p/\breve{a}\breve{a}\breve{a}/q}{r} \quad \text{modus-ponens}$$

### 3.4.2 Writing the premises in a commented row

We can improve the appearance of solution in Org at least somewhat by finding a way to place the premises in the table itself, rather than in the formula.

Thankfully, Org tables account for use cases such as this; they include *comment rows*, similar to the alignment tag row. To use them, we must add a first column which will not be exported which contains a `/` for comment rows.

This row can also contain other markers; we actually make use of two of them immediately. A `_` indicates that the contents of the row are *names* for the cells below. A `^` is similar, but the names are for the cells *above*.

We can achieve better appearance in Org by using comment rows for

- the premises,
- the rule name,
- the conclusion, and
- the wrapper and spacer.

Then the end user needs only insert a table template, fill in the comment rows, and evaluate the formulas to get their inference rule.

$$\frac{p \quad q/\breve{a}\breve{a}\breve{a}/p}{\text{"}q\text{"}} \quad \text{modus ponens}$$

We also replaced the `bar2` macro with an identical one whose name looks a bit more like a horizontal rule. For reference, there are 10 dashes in the name, and they can easily be typed using `C-u 10 -`.

```
#+Macro: inf--------
↪  @@latex:\hrulefill\vspace{-0.3em}@@@@html:<hr>@@
```

### 3.5 Approach 2 conclusion (for now)

For the moment, the above is where I am leaving this project.

Ideally, what should be done next is to separate the tabular construction from the table itself entirely; the user would write the rule in simple table format, and either place a macro or invoke some elisp to generate a table which exports as the desired tabular.

## 4 Approach 3 `ditaa`

A final approach would be to generate images of the inference rules using `ditaa`, a program to generate images from ASCII diagram.

The benefit is that the generated images have a fixed appearance, regardless of the export filetype; the downside is that `ditaa` seems to use a poor choice of font for unicode, and so the resulting diagrams look rather unprofessional. A colleague noted they looked like they were made in Word; a true insult .

If I return to this project, I will generate examples.