Typesetting semantics in LaTeX

Aidan Malanoski

February 15, 2023

The goal of this document is to provide readers with basic skills for typesetting semantics in LaTeX. I have tried not to assume too much LaTeX knowledge from readers, so it should be accessible even to people starting out in LaTeX. Feel free to copy-paste commands and such from the `.tex` file.

For a lot of semantics typesetting, you'll be in math mode. When you want your math-y stuff (including logic) to be on the same line as your text (*i.e.*, you want an "in-line" equation), you enter math mode by putting a single dollar sign $ on either side of the math-y stuff. For example, the code `$2 \times 2 = 4$` gives $2 \times 2 = 4$. If you want your math-y stuff to be on its own line, then you use two dollar signs instead of one. For example, the code `$$2 \times 2 = 4$$` gives the following:

$$2 \times 2 = 4$$

If your math-y stuff needs to be on multiple lines, then you can use the `\align` environment from the `mathtools` package (the `amsmath` package also provides an `\align` environment, but we'll use `mathtools` since it provides other useful functionality). To load the package, you put `\usepackage{mathtools}` in the "preamble," which is the part of the `.tex` file before the `\begin{document}` command. Generally speaking, you load other LaTeX packages the same way.

To create an `align` environment, you first type `\begin{align}`. As the name suggests, the `align` environment lets you align the lines of your math-y stuff. To do this, you oput an ampersand (&) before the character where you want the lines to be aligned. You separate lines using two backslashes (`\\`). This is LaTeX's general "new line" command—it's not specific to the `align` environment). When you're done writing your math-y stuff, you put `\end{align}`. Here's an example of some multi-line math-y stuff made in an `align` environment.

$$2x + 4 = 14 \tag{1}$$
$$2x = 10 \tag{2}$$
$$x = 5 \tag{3}$$

To make this, I used the code in (1). Note that by putting ampersands before the equals signs in the code, I aligned the equations on the equals signs.

(1)
```
\begin{align}
2x + 4 &= 14 \\
2x &= 10 \\
x &= 5
\end{align}
```

If you don't want the "tags" (*i.e.*, the line numbers) on the right side of the page, then you can use the `align*` environment instead. Just replace `\begin{align}` with `\begin{align*}` and replace `\end{align}` with `\end{align*}`. You can add or change the tags using the `tag` command. Type this command before the new line command `\\` on the line where you want it to go. For example, if I change the first line of (1) to `2x + 4 &= 14 \tag{bloop} \\`, I get the following:

$$2x + 4 = 14 \tag{bloop}$$
$$2x = 10 \tag{4}$$
$$x = 5 \tag{5}$$

As alluded to above, you need to be in math mode to type logic symbols. The command `\land` will give you the logical conjunction ('and') symbol $\land$, `\lor` will give you the logical disjunction ('or') symbol $\lor$, `\rightarrow` will give you the material implication ('if then') symbol $\rightarrow$, and `\leftrightarrow` will you give you the the biconditional ('if and only if') symbol $\leftrightarrow$.

You'll probably be using set theory, too. Once again, you'll need to be in math mode. The `\cup` command produces the union symbol $\cup$, `\cap` produces the intersection symbol $\cap$, `\subseteq` produces the subset symbol $\subseteq$, `\subset` produces the proper subset symbol $\subset$, and `\emptyset` produces the empty set symbol $\emptyset$. However, you can get a prettier empty set symbol $\varnothing$ by using the `\varnothing` command from the `amssymb` package.

Curly brackets are produced with the commands `\{` and `\}`. Typing the curly brackets directly, without the backslash, will not work: at best, they won't show

up, and at worst, you'll get an error. This is because curly brackets are used by the LaTeX code to group things together and specify the scope of commands.

Angle brackets are produced with the commands `\langle` and `\rangle` for the left angle bracket $\langle$ and right angle bracket $\rangle$, respectively. To produce double brackets, you need to load the package `stmaryrd`. The commands are `\llbracket` and `\rrbracket` for the left bracket $[\![$ and right bracket $]\!]$, respectively. The commands to produce angle brackets and double brackets only work in math mode.

Typing all these brackets gets old pretty fast. Fortunately, the `\mathtools` has a command `\DeclarePairedDelimiter` that can make our life easier. It'll be easier to explain how this command works after seeing an example. The preamble of this document contains the command in (2). This creates a new command `\denote` that places double brackets around its input. For example, `$\denote{dog}$` gives $[\![dog]\!]$. Note that the command goes in math mode. As you can see, plain text entered in math mode will appear in italics; if you don't want that, use the `\textrm` command. If I change the previous example to `$\denote{\textrm{dog}}$`, it will produce $[\![\text{dog}]\!]$.

(2) `\DeclarePairedDelimiter\denote\llbracket\rrbracket`

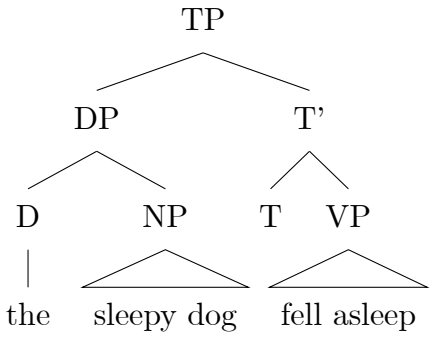
Now, let's take another look at (2). As we just saw, `\DeclarePairedDelimiter` is a command that makes new commands (specifically, commands to make "delimiters"). To use this command, you first type `\DeclarePairedDelimiter`. Then you type the name you want for the new command—in this case, `\denote`. Next, you put the command to make the thing you want as the left delimiter. Here, we want a left double bracket $[\![$, so we put `\llbracket`. Finally, you put the command to make the thing you want as the right delimiter. This is a right double bracket $]\!]$, so we put `\rrbracket`. In the preamble to this document, I've also defined a command `\set` for putting things in curly brackets and a command `\type` for putting things in angle brackets.

Finally, you may occasionally find yourself typesetting trees while doing semantics. While there a number of packages for creating trees, the `forest`[1] package seems to be increasingly popular. To load this package, put `\usepackage[linguistics]{forest}` in the preamble. In LaTeX, square brackets are used for optional inputs to a command. In this case, we are telling the `forest` package to load its settings for producing linguistic trees. To produce trees, you create a `forest` environment, and then write the tree structure using bracket notation. To produce triangles, you put a comma after the node name, then put `roof`. For example, (3a) produces (3b).

[1] http://tug.ctan.org/info/forest-quickstart/forest-quickstart.pdf

(3) a. `[TP [DP [D [the]] [NP [sleepy dog, roof]]] [T' [T ] [VP [fell asleep, roof]]]]`

b.



If you want to annotate a node with its semantic type, then you can put the node in curly brackets, put a new line `\\` after the node label, and put the semantic type. For example, by putting `{TP\\$t$}` instead of just `TP`, I can add a semantic type annotation below the node name. The type-annotated version of (3b) is shown in (4b), and the code that produces it is shown in (4a).

(4) a. `[{TP\\$t$} [{DP\\$e$} [{D\\$\type{\type{e,t}, e}$} [{the\\$\type{\type{e,t}, e}$}]] [{NP\\$\type{e,t}$} [sleepy dog, roof]]] [{T'\\$\type{e,t}$} [{T\\$\varnothing$} ] [{VP\\$\type{e,t}$} [fell asleep, roof]]]]`

b.