Sample LaTeX Stuff

Dennis Ryan Storoshenko

Nov 10, 2014

1 Some Basic Typesetting

The first line of a section is never indented, which is standard for any good typesetter. Because of that section heading, it's obvious that this is a new paragraph, so an indent would just be redundant. You don't need to indent your paragraphs. Anything that comes after a blank line of space in your source file will be interpreted as starting a new paragraph. Note that just starting a new line in the source file does nothing.

You need that blank line to get an indent.
The "newline" backslashes like I used there will get you down a line, but no indent. To get an indent, you need a blank line in the source fine. Two newlines will get you a blank line in the output, but still no indent.

Like this. Also, any number of blank lines in the source file is interpreted as one blank line in the output.

One silly thing about LaTeX is that it does not support the standard quotation mark ”keys”. To get proper smart quotation marks, you need to use the prime key and the apostrophe "like this" The other thing to remember is that some common symbols have special functions in LaTeX, and so you need to type something special in order to see those symbols. Four that spring to mind are $, %, &, and #.

If you type a

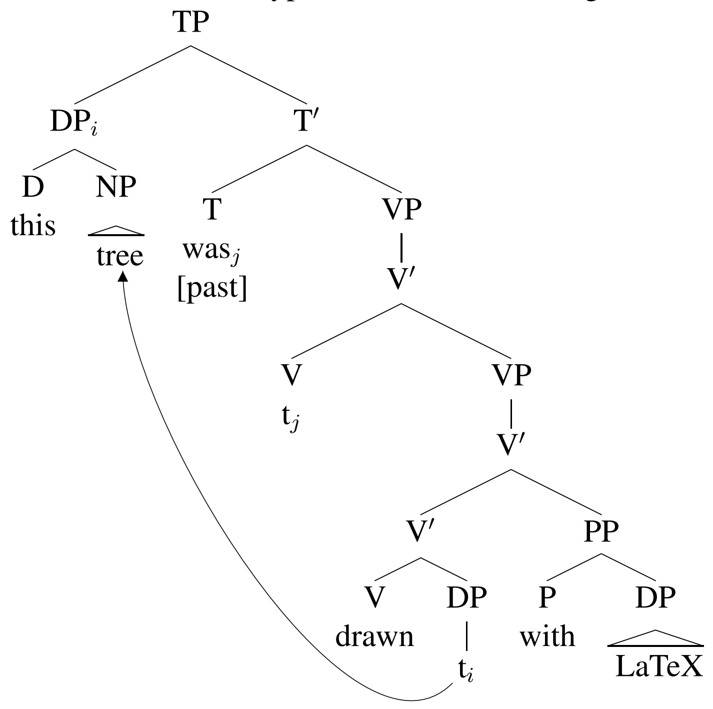
For the others, if you just leave a single dollar sign symbol sitting alone, you are likely to get an error. Type one and see what happens. If you look on the top right of the ShareLaTeX screen you get a red number in the Logs box. That red X over on the left margin tells you that there's a problem here. In TexShop this document would not even compile.

So what does that dollar sign do? The $ activates "Math Mode" which is what most people love about LaTeX. Anything that appears between two dollar signs is interpreted in this alternate mode. For regular text, this turns everything into pseudo-italics like so: v, because the assumption is that the only "real" letters you use in math mode are meant to be variables, and variables are usually in italics, so there you go. Except it's not quite italics either. g and *g* are different. The real reason most linguists and philosophers will want to get into here is that math mode gets you the Greek alphabet, and all of the logical symbols. So this is how you will get θ, λ, $\forall$, $\exists$, and so on. You can write longer expressions inside a single set of these: $[\![\text{eat}]\!] = \lambda x \lambda y.\text{eat}(y, x)$. Also note that math mode is needed for the greater than or equal than symbols, which are different again from $\langle$ and $\rangle$. Math mode also gives you access to simple arrows like $\rightarrow$. This file also comes with code that defines two functions that give you denotation brackets. `denplain` just puts the brackets around its argument $[\![\text{like so}]\!]$, while `denote` will $[\![\text{will include superscript variables}]\!]^g$.

Other things you may need are *italics* which can also be accessed using an alternate command: *like this*. **This is how to get bold.** You may also want to underline something. The scope of the brackets works differently with some functions, as shown in the last sentence. You just need to know this can happen and play with it when something comes out not quite right. The underscore symbol activates subscript. But, by default, it only subscripts the first character after the underscore, which is usually a$_p$roblem. Again, angle brackets define the scope$_{solution}$. Also, anything in a subscript is automatically in math mode (not just italics!). The same goes for superscripts which come, intuitively, from^{caret}. The last thing you might want is ~~strikeout~~. This is a good example of packages not playing well together though, as the package that defines the strikeout command changes what *does*. To test this, go back up to the top and get rid of the last package.

2 Trees and Arrows

To get a tree, just use the tree command and type in labelled bracketing as below.



The spaces here are crucial; whitespace is how the parser knows you mean that a node is done. This is also an exploitable weakness though, as it lets you put in a newline character and get multi-line nodes. If you are ever not sure, you can always use curly brackets to define the scope of a node. The curly brackets in front of the [past] feature are there just to make sure that those square brackets are read as text, rather than defining a branch of the tree.

You have an example of an arrow, the code that you need is given below the tree. The way to parse that command is roughly

```
[SourceCorner]{Source}[DestinationCorner]{Destination}{Length}.
```

You define nodes as points in your tree code by adding those `\node{}` commands. Note that you can just recycle the same definitions from one tree to the next. This can be a huge time-saver, as it lets you copy the basic code for the arrows from one tree to the next without having to re-type the whole command every time! The arrows literally start and end from where that definition appears (see how I put the label in between the “r” and the first “e” of “tree” in the first example there). You have eight “corner” values to choose from:

- `b` bottom
- `br` bottom right
- `bl` bottom left
- `t` top
- `tr` top right
- `tl` top left
- `r` right
- `l` left

By playing with these corner settings and changing the length of your arrow, you should be able to get something that works. This is where people get frustrated though: you can’t just drag and drop an arrow: you need to encode the endpoints and all these parameters, then re-compile every time you make an adjustment to see what it looks like.

You can also use similar code in bracketed examples:

(1) [[This phrase]$_i$ has been moved t$_i$.]

The `\node{}` command does not like being at the edge of a word. Sometimes you will get errors when trying to put these in. It is usually a good strategy to first make sure the tree compiles, then add the node commands, make sure it still complies again, and finally put in the arrows. You can use {} to create an invisible character at the edge that keeps the `\node{}` command happy. Another common problem is that these arrow drawing commands, especially the one in (1), will cause catastrophic errors if the arrow appears at the bottom of a page. If you get to a point where you get a mysterious error after putting some arrows into your document, see if an arrow is at the bottom of a page. Use a `\newpage{}` command to see if that solves your problem. Also, the arrows and the text cannot “see” each other, so you need to move your text down using `\vspace{}` to get things back to a nice distance.

3 Examples

Simple examples use the following structure:

(2) This is a simple example.

Sometimes you want a’s and b’s:(Chomsky, 1995, Chap. 2)

(3) a. This will be your first example.
    b. And this will be the second one.

You might have noticed what that last command after the last example does. If you want to have something be ungrammatical or questionable, put those symbols in the box:

(4) a. * Ungrammatical this sentence be should.
    b. ? Some people might could argue with this one too.

Of course, every now and then, we need to have examples in a foreign language. The markup is a bit complicated at first, but just remember that it matches up based on whitespace in the first two lines:

(5) Shona Reflexive Data (Storoshenko 2010)

a. John a-ka-zvi-pis-a.
   John SUBJ.1-REM.PST-REFL-burn-FV
   'John burned himself.'

b. *pro* Nda-ka-zvi-pis-a.
   pro.1^{st}.SG 1^{st}.SG.SUBJ-REM.PST-REFL-burn-FV
   'I burned myself.'

c. Mbudzi dza-ka-zvi-pis-a.
   goats.10 SUBJ.10-REM.PST-REFL-burn-FV
   'The goats burned themselves.'

Oh, and you see that label command up at the top of the code for this example? You callback to that label using (5), which is one of the functions we defined at the top. (5c) will also follow along to the correct subexample. The really cool bit is that you can copy and paste that huge messy chunk of code for the Shona data so that it is before all of the English examples, and all of the example numbers (including the reference callback!) will automatically change after one or two compiling runs. Test this out for yourself.

4 Sections and Structure

You see how to make sections from the code. Putting an asterisk before the curly bracket will suppress the section number.

An Alternative If you are working on a document and space is at a premium (say, you have a one-page limit), then this tool is a good way of giving your document some structure without wasting lines.

4.1 Subsection

4.1.1 It just keeps going...

4.1.2 List Environments

- This is bullet one.
- And bullet two.

♣ You can use the box to define your own bullet if you don't like the default.

  – And it just tabs everything in for you if you nest these.

Empty box suppressed the bullet.

- Just remember to end them all, or bad things happen.

1. This is the environment for a numbered list.
2. Second item
3. Third item

5 IPA

[gugəl] “tipa manual” [foɹ haw tə juwz ðɪs]

6 References and Your Library

The other huge power of LaTeX is in the citations. You need a separate .bib file of your own for this, as well as a .bst file. ShareLaTeX has some .bst files pre-installed, but I included linquiry2 here just in case. The two major ways of citing are where you are going to talk about Minimalism, and just that say according to Chomsky (1995), things can move covertly at LF. But, you might want to point out that alternative analyses for things like quantifier scope are available (Freedman & Frank, 2010).

One picky thing about this is that to get uppercase in titles to survive through from your .bib file into the .pdf, you need to put the letters inside curly brackets. You can see at the bottom how to trigger the bibliography to appear. The included .bib file has an article, a journal paper, a proceedings paper, and a thesis.

If you run this on ShareLaTeX, it seems that you just need to click Compile once, and it all goes. If you are running this locally on your own computer, you need a four step process:

1. Typeset as LaTeX
2. Typeset as BibTeX
3. Typeset as LaTeX
4. Typeset as LaTeX

You can see the first one as parsing through a looking for all of your citations, the second as grabbing the relevant info from your `.bib` file, the third as building your references list, and the fourth as updating all the in-line citations.

7 Tables and Figures

The commands for a table are given below:
The code above generates a table that does not completely look right, just to show what happens when you forget an hline command. Also that is what you leave if you want to have an empty cell.

Table 1: A Very Boring Table

top left	top right
bottom left	bottom right
	beside an empty cell

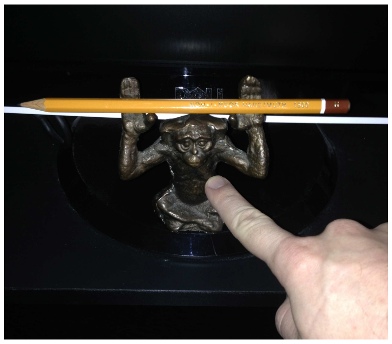


Figure 1: A Structural Ambiguity

The most frustrating thing about tables and figures is that you have only very loose control over where they go. That box in the first line is supposed to let you control it, but it's not very accurate if you are dealing with a very large table. You can refer to this as Table 1.

For graphics, the wrapping is similar, but different code. You need your image to be part of the same project (or in the same folder if running locally)[1] From what I can see here, to get the tree to work, and have an image, the image needs to be in .eps format. Like the table, you can call this using Figure 1. As you can see, this is an example of the floating figure not landing where we want it.

References

Chomsky, Noam. 1995. *The minimalist program*. Dordrecht: Foris.

Enc, Murvet. 1986. Towards a referential analysis of temporal expressions. *Linguistics and Philosophy* 9(4). 405–426.

Freedman, Michael & Robert Frank. 2010. Restricting inverse scope in stag. In *Proceedings of the 10th international workshop on tree adjoining grammars and related formalisms*, .

Kim, Wha-Chun Mary. 1976. *The theory of anaphora in Korean syntax*: Massachusetts Institute of Technology dissertation.

[1] Actually, if you set this up on your own computer, you can use more complicated filepath designations, but we don't need to get into that. This is how you do a footnote though.