Typesetting Math, Part 2

We continue looking at issues in typesetting math. We won't cover everything in Lamport. We'll cover just the trickier topics.

Ellipsis and matrices

In math, we often use three dots to indicate the omitted portion of a sequence, e.g., "for $i = 1, 2, \ldots, n$." The three dots are known as an ***ellipsis***. You do *not* typeset them by just typing three periods: "for $i = 1, 2, ..., n$" looks bad, because the dots are too close together. Instead, you use the `\ldots` command: `for $i = 1, 2, \ldots, n$`.

A common error is to use `\ldots` for dots that should be higher above the baseline. (The ***baseline*** is the imaginary line on which the text is placed.) This error is most frequently made when the sequence includes operators in addition to items. For example, `$x_1 + x_2 + \ldots + x_n$` produces $x_1 + x_2 + \ldots + x_n$. Here, the dots are too low; they should be aligned with the middle of the plus signs. Replacing `\ldots` by `\cdots` does the trick: $x_1 + x_2 + \cdots + x_n$.

What about using `\cdots` within a sequence of multiplications, when you use `\cdot` ($\cdot$) to indicate multiplication? By analogy to the summation example, we would typeset the expression by replacing the plus signs by `\cdot`: `$x_1 \cdot x_2 \cdot \cdots \cdot x_n$`, producing $x_1 \cdot x_2 \cdot \cdots \cdot x_n$. That looks weird. Omit the `\cdot` operators immediately adjacent to the `\cdots`. That change gives `$x_1 \cdot x_2 \cdots x_n$`, producing $x_1 \cdot x_2 \cdots x_n$.

There are also vertical dots $\vdots$ produced by `\vdots` and diagonal dots $\ddots$ produced by `\ddots`. These symbols are typically used in matrices:

$$\left(\begin{array}{cccc} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{array} \right) .$$

Note the use of `\cdots` in the third column.

In my work, matrices come up often enough that I've made a simple macro to help me typeset them:

```
\newcommand{\matx}[2]{\left[\begin{array}{*{#1}{c}}#2\end{array}\right]}
```

The first argument gives the number of columns, and the second gives the guts of the matrix. Because my work with matrices usually requires me to subdivide them into blocks, I like to use square brackets rather than parentheses around the matrices; that's why the `\matx` macro uses `\left[` and `\right]`. Here's the above matrix, typeset with this command:

$$\left[\begin{array}{cccc} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{array}\right] .$$

Symbols

Most of what you need to know about symbols and delimiters appears in Sections 3.3.2 and 3.3.4 of Lamport.

The `\not` command is kind of cute. It puts a slash through the operator that follows it. I think the only time I ever use it is to show that something is not a member of a set: $x \not\in S$. Most other "negation" operators are either already given to you (e.g., $\neq$) or are silly (e.g., why use $\not\leq$ when you can use $>$ instead?).

Generally speaking, you should avoid using the summation-like symbols outside of math displays. They look pretty bad in the main text. In other words,

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

looks good, but $\sum_{i=0}^{n} i = n(n+1)/2$ does not look quite as good. You can put the limits of the in-line summation above and below the summation sign by using the `\displaystyle` command: $\displaystyle\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$. I do not recommend doing so normally. Instead, a good guideline is to use a math display for this sort of notation.

Log-like functions

There are two things to say about log-like functions.

First, use them. Don't typeset $logn$. Put in that backslash: $\log n$.

Second, you can make your own. For example, to make `\rank` a log-like function, define the macro

```
\newcommand{\rank}{\mathop{\operator@font rank}\nolimits}
```

Now you can typeset expressions like $\operatorname{rank} A = 0$.

Note that some of the log-like functions take arguments. An example is `\pmod`: $x \equiv y \pmod{n}$. Others can take limits, for example `\lim`:

$$\lim_{n\to\infty} x = 0 .$$

You get a similar behavior to summations when you use `\lim` in line: $\lim_{n\to\infty} x = 0$.

Digressions

There are a couple of things about the modulus functions that are worth mentioning. First, there are two of them, and they are for different purposes. `\bmod` is a binary operator: b is a divisor of a if $a \bmod b = 0$. Mathematically, this operator takes two operands and produces a result. In typesetting, the correct amount of space is left on either side of the operator. `\pmod` is *not* an operator; it states a relation. When we write

$x \equiv y \pmod{n}$, we are stating that if we have a congruence-modulo-n relation—let's call it C—then $(x, y) \in C$.

Now, I find that the space left before the left paren when you use `\pmod` is a bit excessive for in-line use. It's OK for displays, but a bit too much in line. For CLRS, I defined a related macro, `\imod`, that leaves a little less space. Here are the definitions of `\pmod` and `\imod` from the book files:

```
\def\pmod#1{\allowbreak\mkern12mu({\operator@font mod}\,\,#1)}
\def\imod#1{\allowbreak\mkern10mu({\operator@font mod}\,\,#1)}
```

(These definitions use the TEX-style way of defining macros.) In fact, even our definition of `\pmod` differs from the stock LATEX2e definition; the stock definition uses 18mu, whatever that is. Here's how they all differ:

Stock LATEX2e:	$x \equiv y \pmod{n}$
CLRS `\pmod`:	$x \equiv y \;\; (\mathrm{mod}\; n)$
CLRS `\imod`:	$x \equiv y \; (\mathrm{mod}\; n)$

Now for an unrelated digression.[1] Did you notice how when I defined macros above, the definitions were bracketed by `\makeatletter` and `\makeatother`? The problem is that @ means something to TEX some of the time. If I were better at TEX, I could give you the particulars. Anyway, if you're defining a macro and it contains @, then you cannot go wrong by putting `\makeatletter` before the macro definition and `\makeatother` afterward.

Arrays (and Tabular)

If you look at Section 3.3.3 of Lamport, you might be under the impression that arrays are a bit limited. Not so. The binding of my Lamport is cracked at Section C.10. There you can see all about arrays. The array and tabular environments are so similar that we can talk about them pretty much in one gulp.

Your best bet is to check out the big example on page 204. Note in particular the `\multicolumn`, `@`, `p`, `*`, `\hline`, and `\cline` options. A common phenomenon is, armed with the awesome power of vertical and horizontal lines, to get line-happy:

Year	Election winner
1980	Reagan
1984	Reagan
1988	G. W. H. Bush
1992	Clinton
1996	Clinton
2000	G. W. Bush

That is just too many lines. You can make the same point with fewer:

[1] Does that make it a digression from the digression? Do two digressions cancel each other out, like South Northfield, Vermont being just Field?

Year	Election winner
1980	Reagan
1984	Reagan
1988	G. W. H. Bush
1992	Clinton
1996	Clinton
2000	G. W. Bush

And even the top and bottom lines here are only to offset the tabular from the rest of the text; you could just as easily lose them, too.

Sometimes, you want to change the interline or intercolumn spacing in an array or table. For interline, add an optional extra space in the `\\` that ends each line. For example, here's a 3×3 array typeset normally:

$$\begin{array}{ccc} a & b & c \\ d & e & f \\ g & h & i \end{array}.$$

To add more space between lines, I suggest expressing it in terms of ex's. An ***ex*** is the height of a lowercase letter 'x'. Here's the same array with 2 ex of extra space between lines 1 and 2:

$$\begin{array}{ccc} a & b & c \\[2ex] d & e & f \\ g & h & i \end{array}.$$

You can use negative space to shrink the distance between lines 1 and 2:

$$\begin{array}{ccc} a & b & c \\[-2ex] d & e & f \\ g & h & i \end{array}.$$

To change the interline space uniformly, change `\arraystretch`. Note that `\arraystretch` is actually a command, so you have to use `\renewcommand` to change it. Here's the same array with the interline space 1.5 times the default:

$$\renewcommand{\arraystretch}{1.5}\begin{array}{ccc} a & b & c \\ d & e & f \\ g & h & i \end{array}.$$

You can use the `@` specification to change the spacing between columns. Here's the array with extra space between the first two columns:

$$\begin{array}{c@{\hspace{4em}}cc} a & b & c \\ d & e & f \\ g & h & i \end{array}.$$

To change the intercolumn space uniformly, set `\arraycolsep` or `\tabcolsep`. These are lengths, so you use the `\setlength` command. I set up macros to save and set these:

```
\newlength{\savearraycolsep}
\newcommand{\narrowarray}[1]{\setlength{\savearraycolsep}{\arraycolsep}%
    \setlength{\arraycolsep}{#1\arraycolsep}}
\newcommand{\normalarray}{\setlength{\arraycolsep}{\savearraycolsep}}
```

Here's the array with all intercolumn spacing 0.5 times the default:

$$\begin{array}{ccc} a & b & c \\ d & e & f \\ g & h & i \end{array} .$$

If you use a factor greater than 1, you stretch the intercolumn space. I usually am trying to reduce the intercolumn space, and so I think of it as narrowing the array.

Another great use for arrays is for the "cases" notation used in math. There, you use a left delimiter that's a curly brace and a null right delimiter:

$$n! = \left\{ \begin{array}{ll} 1 & \mbox{if } n = 0 \, , \\ n \cdot (n-1)! & \mbox{if } n > 0 \, . \end{array} \right.$$

There are a couple of things to note here. First, there's a comma after each case except the last (i.e., after "if $n = 0$"). Second, this comma has a space before it. Third, since this particular displayed equation ends a sentence, there's a period after the last case (i.e., after "if $n > 0$"). Fourth, there's a space before that period. Do *not* put the period after the null right delimiter:

$$n! = \left\{ \begin{array}{ll} 1 & \mbox{if } n = 0 \, , \\ n \cdot (n-1)! & \mbox{if } n > 0 \end{array} \right. \quad .$$

Although you might think that logically it makes sense—after all, the displayed equation ends the sentence—it just looks bizarre.

I have macros for doing such cases:

```
\renewcommand{\cases}[1]{\left\{ \begin{array}{ll}#1\end{array}\right.}
\newcommand{\cif}[1]{\mbox{if $#1$}}
```

I'd use it as

$$n! = \left\{ \begin{array}{ll} 1 & \mbox{if } n = 0 \, , \\ n \cdot (n-1)! & \mbox{if } n > 0 \, . \end{array} \right.$$

One-line, displayed formulas

Use either `\[ ... \]` or the equation environment. The former does not number equations, and the latter does. Some folks use `$$ ... $$` or the displaymath environment instead of `\[ ... \]`; they're all equivalent, but I like `\[ ... \]`.

The equation environment gives you numbered equations:

$$n! = \left\{ \begin{array}{ll} 1 & \mbox{if } n = 0 \, , \\ n \cdot (n-1)! & \mbox{if } n > 0 \, . \end{array} \right. \tag{1}$$

Now I can refer to equation (1).

A few things to note here. First, if you're not going to refer to an equation, then don't number it, unless you're forced to by peer pressure. By that, I mean that if all the equations around it are numbered, then it might look downright weird not to number this one. But that usually applies to eqnarray environments, not onesy displays.

Second, *use symbolic labels*. Don't refer to equation (1). As you can see, you can put all sorts of symbols in the label. It need not begin with "eq:". That's just a nice convention that we adopted in CLR, and I have continued using it to this day.

Third, when you refer to the equation, put parentheses around it. It is not equation 1. In fact, I have a macro

```
\newcommand{\eqnref}[1]{(\ref{#1})}
```

so now I can refer to equation (1).

Fourth, I like to call it equation (1) rather than Equation (1). When referring to chapters, sections, and parts, the name "Chapter", "Section", or "Part" should be capitalized. But when referring to non-sectional units, like equations, pages, or lines, do not capitalize. You wouldn't write, "See Page 34" or "see Line 7".

Fifth, remember to run the latex command twice when there are symbolic labels anywhere (not just in numbered equations). The first run puts the definition of the labels in the .aux file, and the second run resolves these definitions.

eqnarrays and the like

When you have multiline formulas, or you have multiple formulas that you want to line up nicely, use the eqnarray environment. If you use eqnarray, every line will be numbered as an equation, unless you put `\nonumber` on the line. Put labels on the numbered lines so that you can refer to them. You can suppress numbering of all lines by using the eqnarray* environment instead.

The eqnarray environment gives you three columns (plus equation numbers if you use them). The left column is flush right, the center column is centered, and the right column is flush left. Here's an example:

$$\begin{array}{rcl} T(n) & \leq & 2T(n/2) + \Theta(n) \qquad (2) \\ & = & O(n \lg n) \; . \end{array}$$

Here, we see a few things. First, you don't have to use anything in particular in the center column. Second, the first line is equation (2), but the second line is unnumbered. Third, I've ended the eqnarray with a period. Fourth, no \\ goes after the last line.

To see that the left column is flush right, here is an eqnarray* that holds multiple equations:

$$\begin{array}{rcl} \sum_{k=1}^{n} (ca_k + b_k) & = & c \sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k \; , \\ \sum_{k=0}^{n} k^2 & = & \dfrac{n(n+1)(2n+1)}{6} \; . \end{array}$$

The left column looks weird that way, so I have created a package called eqnarrays (notice that the top of this file has the line the `\usepackage{eqnarrays}`) that provides three variants on eqnarray. To make the left column be flush left, use eqnarrayl:

$$\begin{aligned}
\sum_{k=1}^{n}(ca_k + b_k) &= c\sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k \ , \\
\sum_{k=0}^{n} k^2 &= \frac{n(n+1)(2n+1)}{6} \ .
\end{aligned}$$

The eqnarrayr environment makes the right column be flush right. We found it useful for writing systems of difference constraints:

$$x_1 - x_2 \leq 0 \ , \tag{3}$$

$$x_1 - x_5 \leq -1 \ . \tag{4}$$

If you leave the `\mbox` commands out, the spacing isn't quite right. I don't recall why, nor do I recall how we discovered that using the `\mbox` commands fixed the problem. Notice also that by using eqnarrayr rather than eqnarrayr*, we get numbers for inequalities (3) and (4). All the environments in the eqnarrays package come in starred and unstarred versions.

The most useful environment in the eqnarrays package is eqnarrayreason. This environment adds an extra column on the right. This column is flush left and *not* in math mode. It's great for adding reasons why a line is valid. For example, here's how I would typeset a proof using indicator random variables for the hat-check problem. (Here, X is a random variable for how many customers get back their own hat, and X_i is an indicator random variable for the event that customer i gets back his own hat.)

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] && \text{(definition of } X\text{)} \\
&= \sum_{i=1}^{n} \mathrm{E}[X_i] && \text{(linearity of expectation)} \\
&= \sum_{i=1}^{n} \Pr\{\text{customer } i \text{ gets his own hat}\} && \text{(CLRS Lemma 5.1)} \\
&= \sum_{i=1}^{n} 1/n \\
&= 1 \ .
\end{aligned}$$

Note that even when there was no reason, I put in the `&`. I don't think it's absolutely necessary, but it seems like good form. Note also that the period ending the equation goes after the last text on its line, which is not necessarily the rightmost column.

Finally, a couple of things that Lamport mentions in Section 3.3.5, but are good to point out. First, if you continue a line and the continuation line starts with a $+$ or $-$, it could look wrong:

$$\begin{aligned}
X &= X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 \\
&\quad +X_8 + X_9 + X_{10} \ .
\end{aligned}$$

The $+$ before X_8 is assumed to be a unary operator, and so it is closer to X_8 than it would be if were a binary operator. You can fool LaTeX2e into thinking that it's a binary operator by giving it an empty `\mbox` as a left operand:

$$\begin{aligned} X &= X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 \\ &\quad + X_8 + X_9 + X_{10} \ . \end{aligned}$$

Sometimes you need to let the columns overlap. This situation often occurs when the first and some succeeding line are both long. For example,

$$\begin{aligned} O(D(n) + t(H)) + ((D(n) + 1) + 2\,m(H)) - (t(H) + 2\,m(H)) + \Theta(n \lg n) &= O(D(n)) + O(t(H)) - t(H) + \Theta(n \lg n) \\ &= O(D(n)) + \Theta(n \lg n) \ , \end{aligned}$$

We solve this problem by putting the left column of the first line on its own line, surrounding it by the command `\lefteqn`:

$$\begin{aligned} &O(D(n) + t(H)) + ((D(n) + 1) + 2\,m(H)) - (t(H) + 2\,m(H)) + \Theta(n \lg n) \\ &\quad = O(D(n)) + O(t(H)) - t(H) + \Theta(n \lg n) \\ &\quad = O(D(n)) + \Theta(n \lg n) \ , \end{aligned}$$

This still looks bad. We correct the problem by adding some horizontal space after the argument of `\lefteqn` but before `\\`:

$$\begin{aligned} &O(D(n) + t(H)) + ((D(n) + 1) + 2\,m(H)) - (t(H) + 2\,m(H)) + \Theta(n \lg n) \\ &\qquad\qquad\qquad\qquad = O(D(n)) + O(t(H)) - t(H) + \Theta(n \lg n) \\ &\qquad\qquad\qquad\qquad = O(D(n)) + \Theta(n \lg n) \ , \end{aligned}$$

Unfortunately, you have to use trial and error to determine the right amount of space to add.