CSE 150: Notation Examples

Leif Walsh

September 6, 2010

For most of the following, you will need to include `amsmath`, `amsfonts`, `amsthm`, and sometimes `amssymb` in your source, by putting the `\usepackage{amsmath}` and the like in the preamble.

1 Sets

1.1 Normal Sets

To denote sets, just go into math mode and use a capital letter. For literal sets, be sure to escape (\) the braces.

Example

`$A$` = "A"
`$\emptyset$`= "$\emptyset$"
`$\{1,2,apple\}$`= "$\{1, 2, apple\}$"

1.2 Sets of Numbers

For "bold" sets (naturals, reals, complex numbers), enter math mode and use `\mathbb`.

Note, `\mathbb` requires `amsfonts`.

Example

`$\mathbb{R}$` = `$\mathbb R$` = "$\mathbb{R}$"

1.3 Set Operations

All of these must be in math mode.

Union: `\cup`, `\bigcup`

Intersection: `\cap`, `\bigcap`

Difference: `\setminus`, `-`

Symmetric Difference: `\triangle` (I think it's usually a bit smaller than what this produces; if anyone comes across a better command, let me know)

Complement: `\overline`

Example

`$( (A \cup B) \cap (B \setminus \varnothing) ) \triangle \overline{C}$`
= "$((A \cup B) \cap (B \setminus \emptyset)) \triangle \overline{C}$"

1.4 Summation-ish notation

Make sure you are in `displaymath` mode, not `inline` math mode, and use `\bigcup` or `\bigcap`. Usage is as follows:

```
\bigcup_{<lower condition>}^{<upper condition>}
```

Example

```
\bigcup_{i=0}^{10} \mathbb{Z}_i
```

$$\bigcup_{i=0}^{10} \mathbb{Z}_i$$

```
\bigcap_{q \in \mathbb Q} \{qi | i \in \mathbb Z\}
```

$$\bigcap_{q \in \mathbb{Q}} \{qi | i \in \mathbb{Z}\}$$

2 Boolean Logic

2.1 Operators

See the examples.

Example

`$P \lor Q$` = "$P \lor Q$"
`$P \land Q$` = "$P \land Q$"
`$\lnot P$` = "$\neg P$"
`$P \Rightarrow Q$` = "$P \Rightarrow Q$"
`$P \Leftrightarrow Q$` = "$P \Leftrightarrow Q$"

2.2 Expressions

Convention dictates that boolean expressions may be grouped with square brackets or parens, depending on the author's mood or desire for readability. Note that in LaTeX, these do not need to be escaped like curly brackets do.

2.3 Truth Tables

Some practice with the `\tabular` environment is necessary, but see the following for a brief introduction.

Example

```
\begin{tabular}[h]{c|c|c}
$P$ & $Q$ & $P \Rightarrow Q$ \\
\hline
F & F & T \\
F & T & T \\
T & F & F \\
T & T & T
\end{tabular}
```

P	Q	$P \Rightarrow Q$
F	F	T
F	T	T
T	F	F
T	T	T

3 Errata (for want of a better term)

3.1 Overloading the "equals" Operator

In the most strict places, a "colon-equals" replaces the assignment sense of the equals sign, and a single equals represents only the comparison (it's an equivalence relation!). Most of the time, a single equals sign can do both and you can infer the meaning from context, and sometimes (especially when programmers are writing), a single equals will only be assignment and a double equals will be comparison. Note that in all cases, for assignment, the left operand is the variable taking the assignment and the right operand is the value which the left operand is being assigned.

Additionally, in place of the colon-equals assignment operator, many people use a simple `\leftarrow` ($\leftarrow$, read "takes the value of" or something equivalenl). This is probably more common in math circles than CS circles (particularly unpopular among C programmers, I hear), but I kind of like it, and it usually looks cleaner than equals or colon-equals (and making it look less like debuggable code tends to make it less scary to read). I strongly suggest the use of this.

Example

`$a := b$` = "$a := b$" [Assignment]

`$(a == b) \Rightarrow Q$` = "$(a == b) \Rightarrow Q$" [Comparison]

`$a = b \cdot 2$` = "$a = b \cdot 2$" [You really can't tell, can you?]

`$P \leftarrow a = b$` = "$P \leftarrow a = b$" [Assignment of a boolean value obtained by a comparison]

3.2 (Ordered) Lists / Arrays

A literal list is typically square bracket-delimited and comma-separated. A fixed-size list (like a tuple or a vector) is typically paren-delimited instead. If the list is named (L, say), the ith element would be referenced by $L[i]$ (more common with variable-length lists) or L_i (more common with fixed-length lists) (`$L[i]$` and `$L_i$`, respectively). A range of elements could be denoted by a Python-style 'slice' ($L[i:j]$) or an ellipsis ($L[i..j]$ or $L[i \ldots j]$), but be sure to remind your reader which, if either, ends are inclusive (also, ranges are typically not done with subscript notation). Typically the first index is inclusive and the second is exclusive, but it's always nice to make sure they are on the same page with you.

Example

Variable-length list: `$[1, 2, 3, 4, 5]$` = "$[1, 2, 3, 4, 5]$"

Fixed-length list (k-tuple): `$(a_1, a_2, \ldots, a_k)$` = "$(a_1, a_2, \ldots, a_k)$"

Setting the nth element of the list A: `$A[n] := 3$` = "$A[n] := 3$"

Often, we may never name a tuple, and we will just always refer to its contents: "Let $(a_1, a_2, \ldots, a_n)$ be the n-tuple where $a_0 := 0$, $a_1 := 1$, and $\forall i \in \mathbb{N}, a_{i+2} = a_i + a_{i+1}$" (see the source for that one)

3.3 Defining your own commands

3.3.1 Commands

One of the cool things about LaTeXis that you can program in it! You can define new functions with the `\newcommand` command, and new environments (like `displaymath` and `tabular`) with `\newenvironment`. Look online to see how those work, but one of the simple things you can do is create commands that act like `\log` and `\sin` (in that they aren't italic and add a space in math mode).

Example

First, you would put the following in the preamble (or a file you include with `\usepackage`):

```
\newcommand{\myfunc}{\mathop\mathrm{myfunc}}}
```

Now, in your `.tex` file, you can use `\myfunc` like any other no-argument function:

`$O(n \myfunc n)$` = "$O(n \operatorname{myfunc} n)$"

3.3.2 Arguments

If you would like your command to have arguments, specify the number of arguments after the name of the command in square brackets, and then use "#1", "#2", and the like in your definition.

Example

To include arguments, do something like this:
`\newcommand{\mysquare}[1]{#1^2}`
And then in the `.tex`:
`$\mysquare{x}$` = `$\mysquare x$` = "x^2"

3.3.3 Theorem environments

Just like `\newcommand` and `\newenvironment`, you can create "theorem" style environments with, you guessed it, `\newtheorem`. A "theorem" environment is numbered, and you can, when you create a new theorem environment, specify which other theorem to number it inside of. You can also specify which other theorem environments to interlace numbering with (so they use the same counter, effectively). After you define a theorem, you use it by adding `\begin{name}`, where "name" is the name you gave the environment.

(Note: You get much better results, and have a few more options I didn't discuss, by using the `amsthm` package.)

The `proof` environment (defined in `amsthm`) is a great way to prove your theorems, as you might expect. This will give you a nice little italicized "Proof:" before your proof, and a well-placed □ after.

There are some more details about theorem and proof environments in the well-written page http://www.math.uiuc.edu/~hildebr/tex/theorems.html .

Example

(In a preamble or included package:)
`\newtheorem{mythmenv}{Theorem}[section]`
`\newtheorem{mylemenv}[mythmenv]{Lemma}`
(Now, in the `.tex`:)

```
\begin{mythmenv}$1 + 1 + 1 = 3$\end{mythmenv}

\begin{proof}
First, we need a tiny lemma:

\begin{mylemenv}$1 + 1 = 2$\end{mylemenv}
\begin{proof}
\emph{Hand waving}
\end{proof}

And another:

\begin{mylemenv}$1 + 2 = 3$\end{mylemenv}
\begin{proof}
\emph{More hand waving}
\end{proof}
```

```
Now, $1 + 1 + 1 = 1 + 2 = 3$
\end{proof}
```

... which gives us:

Theorem 3.1. $1 + 1 + 1 = 3$

Proof. First, we need a tiny lemma:

Lemma 3.2. $1 + 1 = 2$

Proof. Hand waving □

And another:

Lemma 3.3. $1 + 2 = 3$

Proof. More hand waving □

Now, $1 + 1 + 1 = 1 + 2 = 3$ □

3.3.4 Dealing with math mode

Note that the above commands only work if you are already in math mode. If you would like to have commands that act as though they are in math mode whether or not they are embedded in math mode text, use the `\ensuremath` command (in LaTeX2e), or look up the usage of `\ifmmode`, which will allow you to set a command like

```
\newcommand{\mathify}[1]{\ifmmode{#1}\else\mbox{$#1$}\fi}
```

Example

```
\newcommand{\abse}[1]{\ensuremath{\left| #1 \right|}}
\newcommand{\abs}[1]{\mathify{\left| #1 \right|}}
Text \abse{x} more text = "Text |x| more text"
Text $\abse{x}$ more text = "Text |x| more text"
Text \abs{x} more text = "Text |x| more text"
Text $\abs{x}$ more text = "Text |x| more text"
```