1 Functional and display math

I have been thinking about a way to have functional and readable mathematics in technical documents. It has always bothered me that I have to write a LaTeX version of an equation, and then a separate implementation of the equation in code somewhere. At least twice these separate representations have not agreed!

One solution might be if my functional code could be converted to LaTeX easily. I explore one simple approach to this here. It is somewhat inspired by this work here http://oremacs.com/2015/01/23/eltex/ on writing LaTeX in emacs-lisp, and from my work with org-mode in mixing narrative text, LaTeX and code.

The idea is to use emacs-lisp for the code, so it is functional, but provide an alternative output for the *same code* for a document conversion. In other words, we accept there is more than one version we need: a functional version for working, and a consumption version for presentation. We will generate the consumption version from the functional version.

I know emacs-lisp is not ideal for mathematics the way we are accustomed to seeing it, but it enables the idea I want to explore here so we will try it.

Here is the simplest example I could come up with for functional math. We can run it ourselves, and verify it is correct.

```
(+ 1 2 3)
```

```
6
```

Now, I can change the meaning of this code temporarily, so that it not only evaluates the form, but also represents the equation and result in LaTeX code. If this was incorporated into a preprocessor of the document, we could have a functional version representing our equations, in code form, and a presentation version generated from this version.

```
(cl-flet ((+ (lambda (&rest args)
               (concat
                "$"
                (mapconcat #'number-to-string args " + ")
                " = "
                (number-to-string (eval `(+ ,@args)))
                "$"))))
  (+ 1 2 3))
```

$1 + 2 + 3 = 6$

Getting this to a truly functional approach would require a lot of work, basically creating transformation functions for many, many kinds of mathematical functions, and a lot of other kinds of logic. For example, (+ 1 2 (+ 3 4)) would not render correctly with the code above.

Here is an example that generates a fraction from a division.

```
(cl-flet ((/ (lambda (&rest args)
               (format
                "$\\frac{%s}{%s} = %s$"
                (car args)
                (mapconcat 'number-to-string (cdr args) " \\cdot ")
                (number-to-string (eval `(/ ,@args)))))))
  (/ 1.0 2.0 3.0))
```

$\frac{1.0}{2.0 \cdot 3.0} = 0.16666666666666666$