MiniLaTeX: A subset of LaTeX for the Web

James Carlson

Abstract

MiniLaTeX is a no-setup subset of LaTeX that can be rendered on the fly to HTML. One can use it to build web apps with true HTML display viewable on any device from smart phone to tablet to desktop. Typesetting occurs in real time, and error messages are displayed in-line in the rendered text. MiniLaTeX documents can be exported to standard LaTeX.

We describe (a) MiniLaTeX the language, (b) the main features of the document management application `minilatex.lamdera.app`, and (c) some of the technical work required to implement an on-the-fly LaTeX-to-HTML compiler.

A video version of this paper is at `https://youtu.be/TAIYpCc3VV0`.

1 Introduction

MiniLaTeX is a no-setup subset of LaTeX that comes with an on-the-fly compiler to HTML:

- **No setup.** Just begin typing. No preamble with `\usepackage`, `\begin{document}`, etc., is needed.
- **On-the-fly.** A typical editor for MiniLaTeX presents two windows: on the left is the source text, on the right is the rendered text. Changes to the source text are immediately reflected in the rendered text window.
- **Errors.** Errors are immediately flagged in color in place in the rendered text.

The compiler is written in Elm (`elm-lang.org`), a strictly typed language of pure functions. Elm is a good language for writing MiniLaTeX apps because (a) it is designed for building web applications and (b) it has an excellent, high-performance library of parser combinators (`package.elm-lang.org/packages/elm/parser/latest`), akin to Haskell's `parsec`.

Here are two links to apps that use MiniLaTeX.

- `demo.minilatex.app`: a simple no-signin app for experimenting with MiniLaTeX. Feel free to edit the text you find there, or clear it and write something new. Since there is no setup and since it is interactive, the demo app is also a good tool for learning LaTeX.
- `minilatex.lamdera.app/g/34`: This link takes you to some class notes, hosted on `minilatex.lamdera.app`.

`minilatex.lamdera.app`, while still in alpha test, is a full content-management system for creating, editing, and distributing MiniLaTeX documents. Some ways to access it:

- **Guest access.** Sign in to `minilatex.lamdera.app` as `guest` with password `minilatex` to explore documents that have been made public by their authors. Both the source and rendered text are available, so you can see how MiniLaTeX documents are written.
- **Registered user.** Create and edit documents on a desktop computer or tablet.
- **Smart phone.** Use in read-only mode on a smart phone. Students can read class notes and problem sets this way.

2 Features of MiniLaTeX

Below is a summary of MiniLaTeX features. See the manual (`minilatex.lamdera.app/g/21`) for more details.

- **Macros and Environments.** Environments include theorem, problem, definition, etc., as well as equation, align, verbatim, and more. The `\maketitle`, `\section`, `\cite`, `\eqref` macros and many others work as expected. Unimplemented macros are rendered verbatim, but colored red to indicate their status.
- **Macro definitions.** One can define macros for both math mode and text mode in MiniLaTeX.
- **Export.** MiniLaTeX documents can be exported to standard LaTeX, complete with the necessary `\usepackage`, `\begin{document}` ... `\end{document}`, and any needed macro definitions. Exported documents are ready to process with `pdflatex`. An example of an exported document compiled to PDF using TeXShop is at `noteimages.s3.amazonaws.com/anharmonic_oscillator.pdf`.
- **Images.** A macro
  `\image{URL}{Caption}{Format}`
  is provided to place images in a MiniLaTeX document. Provision is made to render images in exported documents. In addition, there is an `svg` environment for rendering SVG images from SVG source code.
- **Unicode.** MiniLaTeX accepts Unicode (UTF-8) input. More needs to be done to accommodate Unicode in exported documents.
- **Paragraph-centric.** MiniLaTeX is "paragraph-centric", meaning that the smallest unit of recompilation is the *logical paragraph*. A logical paragraph is either an ordinary paragraph or an outer begin-end block delimited above and below by a blank line.

- **Additions.** Besides the `\image` command, various other commands exist in MiniLaTeX but not standard LaTeX. These commands are provided with suitable macro definitions on export so that a MiniLaTeX document can always be compiled with standard LaTeX tools. Here are some examples (colors are grayscaled in the printed *TUGboat*; apologies). Text can be colored blue using the `\blue` macro: *I am feeling blue*. Text can be highlighted using the `\highlight` macro. The teacher said that all work on our class project is due by October 1. There is also a `\strike` macro: Please delete this ~~very bad word~~. For a complete list of additions, see `minilatex.lamdera.app/g/21`.

An important part of the MiniLaTeX project is to properly define the subset of LaTeX to be supported. The current rule of thumb for this is "good enough to write my lecture notes, class hand-outs, and problem sets". Feedback on this issue is much appreciated.

3 Some features of minilatex.lamdera.app

`minilatex.lamdera.app` hosts a content management system which supports creating, editing, and distributing documents from a searchable repository. Users can search by title, author, tags, etc. Documents can be collaboratively edited, shared by url, and versioned on Github through a simple user interface. Documents can also be exported to PDF.

In addition to the familiar `\href` macro, `\xlink` and `\ilink` are provided in MiniLaTeX. The `\xlink` macro is used to make a link from one document to another in `minilatex.lambdera.app`. Thus, one can say `\xlink{21}{Manual}` to make a link to document 21 with label *Manual*.

The `\ilink` macro is similar. It has the same syntax, but is used in constructing a page of links which functions as a table of contents or *index document*. In this way, one can assemble many documents into one to make a book. For an example, see the class notes at `minilatex.lamdera.app/g/34`. It is worth looking at the source to see how it is done.

4 The MiniLaTeX Compiler

The MiniLaTeX compiler consists of two parts, a *parser* and a *renderer*. The first is a function

$$\textit{parse}\text{: Source text} \to \text{AST},$$

where *AST* stands for *Abstract Syntax Tree*. This is a tree with nodes like

```
LXString "Pythagoras says"
```

and

```
InlineMath "a^2 + b^2 = c^2"
```

The tree thus expresses a grammatical analysis of the source text, identifying its "parts of speech" and putting them in relationship to one another. The second is a function

$$\textit{render}\text{: AST} \to \text{HTML}.$$

The compiler is the composite of these two functions:

$$\text{compile} = \text{render} \circ \text{parse}$$

The strategy that makes writing such a compiler feasible is *divide and conquer*. One writes the parser using parser combinators. One then constructs a function which renders the text-mode LaTeX to HTML, passing the math-mode text on to either MathJax (`https://mathjax.org`) or KaTeX (`https://katex.org`) for rendering.

Video: Making a LaTeX-to-HTML parser in Elm (`https://youtu.be/dmDA7iziSgs`).

4.1 MiniLaTeX's AST

Every value in a statically typed language like Haskell, ML, or Elm, has a *type*. Below is the type of the AST for the MiniLaTeX compiler. Writing down this type definition was the first step in writing the parser.

```
type LatexExpr
    = LXString String
    | Comment String
    | Item Int LatexExpression
    | InlineMath String
    | DisplayMath String
    | SMacro String (List LatexExpr)
                    (List LatexExpr) LatexExpr
    | Macro String (List LatexExpr)
                               (List LatexExpr)
    | Environment String (List LatexExpr)
                                      LatexExpr
    | LatexList (List LatexExpr)
    | NewCommand String Int LatexExpr
    | LXError (List (DeadEnd Context Problem))
```

Note that the definition of `LatexExpr` refers to itself, hence is recursive. This is typical of the definitions of types of structured trees. To give an idea of how the parser works, consider the following examples. In each, the first line is source text, the others constitute the resulting AST (line breaks are editorial).

```
> Pythagoras
LXString "Pythagoras"

> \strong{Pythagoras}
Macro "strong" [] [LatexList
                        [LXString "Pythagoras"]]

> \strong{Pythagoras} says that $a^2 + b^2 = c^2$
Macro "strong" [] [LatexList
                        [LXString "Pythagoras"]]
  , LXString "says  that "
  , InlineMath "a^2 + b^2 = c^2"
```

The parser is defined in roughly 500 lines of Elm code and consists of a set of functions which call upon one another. The top-level parser function is given below. Note the close correspondence between its parts and the parts of the type definition. It is built using `oneOf`, a combinator which takes a list of parsers as arguments and which returns a parser as value.

```
latexExpression : LXParser LatexExpr
latexExpression =
    oneOf
        [ texComment
        , displayMathDollar
        , displayMathBrackets
        , inlineMath
        , newcommand
        , macro
        , smacro
        , words
        , lazy (\_ -> environment)
        ]
```

We give one more example, the `macro` parser. It uses the combinators `(|=)` and `(|.)` to *sequence* parsers, thereby forming a new one. The resulting "parser pipeline" operates in the following way. The phrase `|= macro` parses the name of the macro. Then `|= itemList optionalArg` recognizes the list of optional arguments, and `|= itemList arg` does the same for the regular macro arguments. Finally, `|. whitespace` "eats" but ignores whatever white space it finds. The values found by the `(|=)` phrases are taken as arguments of the constructor `Macro` for the `LatexExpr` type, and the result of this function call is a value of type `LatexExpr`.

In this example, `whitespace` is a parser for white space, which can come in different flavors depending on context, e.g., spaces only or spaces and newlines.

```
macro : LXParser () -> LXParser LatexExpr
macro =
    succeed Macro
        |= macroName
        |= itemList optionalArg
        |= itemList arg
        |. whitespace
```

One can continue down the rabbit hole, explaining the parsers `macroName`, `optionalArg`, `arg` and the combinators `itemList`, etc., but we stop here.

What is important to understand is that fundamentally there are only three things in something like the MiniLaTeX parser: primitive parsers, combinators that choose among alternatives, and combinators that sequence other parsers. As a note, eating trailing whitespace is important because in the present setup, lexing and parsing are not separate operations.

4.2 Rendering

The top-level rendering function is much like the top-level parsing function. It analyzes the type of a LatexExpr and dispatches the appropriate renderer, which may in turn call other rendering functions, including the top level one. And so on, down the next rabbit hole of function calls we go. The renderer module constitutes roughly 1300 lines of code.

4.3 Code

Code for the MiniLaTeX compiler as well as the demo app can be found at `github.com/jxxcarlson/meenylatex`. (The strange name is to reserve the name `github.com/jxxcarlson/minilatex` for a future stable version with a polished API.)

The code for the `minilatex.lamdera.app` application is at `https://github.com/jxxcarlson/lamdera-minilatex-app`. All code is open source.

5 Feedback

I am very interested in feedback from the community regarding features, bugs, etc. Of special interest is the subset of LaTeX used: what should it be? Comments to jxxcarlson at gmail.

6 Acknowledgements

I would like to thank Evan Czaplicki, Ilias Van Peer, Mario Rogic, and Luke Westby, all of the Elm community, Davide Cervone, MathJax, and the team at KaTeX.org for their generous and invaluable help. I also wish to thank the Simons Foundation (`https://simonsfoundation.org`) for its support of this project.

This document was originally written in MiniLaTeX and is available at `minilatex.lamdera.app/g/22`.

◇ James Carlson
`jxxcarlson (at) gmail dot com`
`https://minilatex.lamdera.app`