BlaTeX

Rushi Shah
TJHSST

Abstract

Introduction

Markdown and HTML are the standard tools used to write standard tech blogs with. But they have weak support for embedding mathematical formulas, and are not conducive to writing for an extended period of time. So instead of writing posts in Markdown, my project allows writers to create posts in LaTeX (a powerful type-setting and markup language) and generate a static-site that can be deployed to any hosting service.

Background

My project, called BlaTeX, is a static-site compiler written in Haskell that compiles LaTeX posts into a deploy-able site. It is inspired by a similar Ruby tool called Jekyll that compiles Markdown posts into a static, deploy-able site. However, it allows the user to write posts in LaTeX rather than Markdown (because LaTeX is a more powerful tool) and is overall a more lightweight blogging software.

Another popular content-management system for blogging is Wordpress. However, Wordpress posts are edited in the Wordpress Administration Panel as "What you see is what you get". Neither Markdown nor LaTeX are "What you see is what you get", and thus Wordpress caters to a different audience (a group of mostly non-technical authors).

Development & Techniques

Requirements. This command line utility requires the Haskell Platform and LaTeX. The Haskell Platform includes GHC (a Haskell compiler) and Cabal, which is Haskell's primary package manager. LaTeX is needed to build LaTeX posts into their resulting PDFs. After the Haskell Platform has been installed, installing BlaTeX is simple:

```
> cabal install blatex
```

Overview. The project is a command line utility that is written in Haskell. Haskell is a statically-typed functional programming language that makes heavy use of the REPL. Libraries were tested/experimented with independently in their own files. After the necessary functions were written, they were integrated into the final product. The finished project was assembled into an executable file and deployed to Haskell's package manager (Hackage).

Limitations. One potential limitation of this project is that it only compiles links to PDF files. Because LaTeX can't generate HTML, the blog will only be a series of interlinked PDF files rather than a series of interlinked HTML files. This is not necessarily a limitation, but it may interrupt the status quo of typical blogs made of HTML pages.

Dates are formatted in `DD MONTH YEAR` which contributes to the minimalist, classy theme of PDF blog posts; this date format is also used in the internal implementation. However, this becomes an issue if multiple posts are authored on same day because they will not be displayed in the correct order.

Research Theory and Design Criteria

Error Handling. Error handling was one of the most complex aspects of this project. Haskell is a strongly typed language and utilizes Type Classes for a large portion of functionality to produce elegant and expressive code. Series of functions that could potentially create errors were chained together with an instance of an Applicative Functor. Originally the Maybe Applicative was used, but in order to produce meaningful error messages for the user, I switched to the Either Applicative. A simplified example of code used to handle some potential errors is as follows:

```haskell
createPost s t = Post <$> pure s <*> title <*> author <*> date <*> pure t
  where
    date = (getCommandValue "date" t)
    author = (getCommandValue "author" t)
    title = (getCommandValue "title" t)
```

In this code, the `getCommand` function will either find the command within the abstract syntax tree, or the given command was never used. If the command was never used, no post can be created because the requisite information was missing. This series of function calls are tied together with `<*>`, the "splat" operator on Applicative instances. If one computation fails, the ultimate computation will also fail.

Input/Output. Because Haskell is a purely functional programming language, no side effects are permitted. This makes input/output rather difficult, and thus `Monad`'s are used to simulate the desired side effects. However, ubiquitous use of Monads defeats the purpose of the purely functional nature of Haskell. Thus, the input-output code was confined to a single module where almost all effect-ful computations were handled. The main function was included here that glued together

all pure functions. This function included code to automatically initate the proper
file structure and skeleton files for a new user through system commands. It also
included code for the building process. The output of this process is:

```
Getting directory contents
Turning directory contents into posts
All posts are well formed
Turning posts into an HTML element
Reading the layout file
Inserting HTML element into layout file
Writing resulting file into index.html
Success building!
```

If an error is raised at any point in the process, the process will halt and display
the error message for the user to troubleshoot.

Developmental Procedures

In general terms: a directory containing posts is examined by BlaTeX, each
post is parsed into an abstract syntax tree, information from this tree is parsed into
a Haskell Post data structure, each post is stored into a list, the list is sorted by
the date, each object is converted into it's corresponding HTML, and this HTML is
outputted into an output file at a user-specified location.

The `Post` Algebraic Data Strucure. In Haskell, the equivalent of an "ob-
ject" (used in object oriented programming languages), is an "algebraic data type
(ADT)". To represent each post, a `Post` algebraic data type was defined with meta-
information for each post. The final definition for the Post was:

```
data Post = Post {
  fileName :: String
  , postTitle :: String
  , postAuthor :: String
  , postDate :: DateTime
  , syntaxTree :: LaTeX
    }
    deriving (Eq, Show)
```

This defines the structure of a Post based on each attribute it would have and
what type that attribute would have.

BlazeHTML. The first part of the project that I tacked was generating HTML elements that would need to be inserted into the final file. Each post object would generate the following LI element:

```html
<li class='blog-post'>
    <a class='post-link' href='posts/example-post.pdf'>
        Example Post
    </a>
    <div class='post-date'>
        1 January 2015
    </div>
</li>
```

HaTeX. The second part of the project required that I read in LaTeX files for the posts and use them to extract certain meta-information about each post. This included the title of the post (inserted into the `.post-link` tag) and the date of the post (used to sort posts and displayed in the `.post-date` tag).

TagSoup. The user supplies a layout file which defines the format and style of their blog. BlaTeX finds the line `<ul id='blog-posts'></ul>` and inserts all the posts at that point. In order to do so, the project reads in an HTML file and derives it's abstract syntax tree with a package called TagSoup. TagSoup converts an HTML file into a list of HTML elements, and the HTML generated with BlazeHTML is inserted at the specified point.

Dates. Support for dates was one of the final undertakings addressed. The Data.Dates package was used to parse the LaTeX dates. This allowed me to sort the list of posts chronologically.

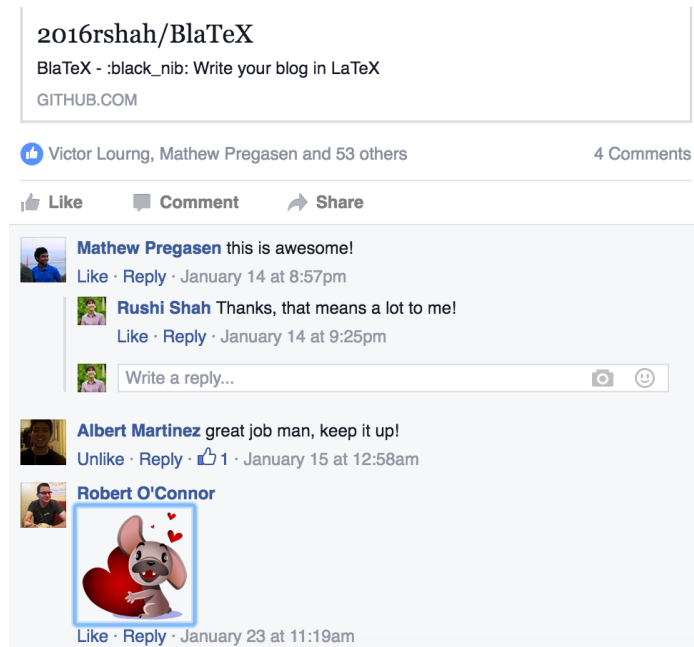## Internal Testing and Analysis

Because this tool was written in Haskell (a statically-typed, compiled programming language), most insidious bugs were filtered out at compile time. Haskell development is also strongly integrated with using a Haskell REPL (read-eval-print-loop). Thus, each function was experimented with in the REPL after being written and before being pushed to production.

User Interface Testing and Analysis

In order to test this project, I used it to create and maintain `http://rshah.org/blog`. Through my own use I discovered multiple improvements that could be made to the user interface and experience, which I promptly implemented.
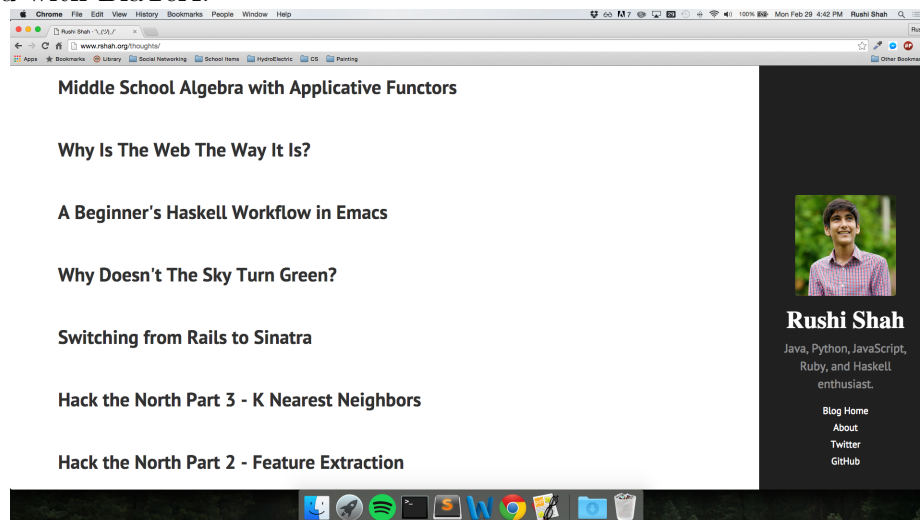
I also posted my project in a Facebook group for functional programming and recieved an overwhelmingly positive response:

It is also an open source project on Github, and met a similarly positive reaction from that community (with 19 Github "stars").

2016rshah/BlaTeX

BlaTeX - :black_nib: Write your blog in LaTeX

GITHUB.COM

Victor Lourng, Mathew Pregasen and 53 others   4 Comments

👍 Like   💬 Comment   ➤ Share

Mathew Pregasen this is awesome!
Like · Reply · January 14 at 8:57pm

Rushi Shah Thanks, that means a lot to me!
Like · Reply · January 14 at 9:25pm

Write a reply...

Albert Martinez great job man, keep it up!
Unlike · Reply · 👍 1 · January 15 at 12:58am

Robert O'Connor

Like · Reply · January 23 at 11:19am

Visual representation of Results

My blog at `http://rshah.org/blog` is a visual representation of a static site compiled with BlaTeX.

Middle School Algebra with Applicative Functors

Why Is The Web The Way It Is?

A Beginner's Haskell Workflow in Emacs

Why Doesn't The Sky Turn Green?

Switching from Rails to Sinatra

Hack the North Part 3 - K Nearest Neighbors

Hack the North Part 2 - Feature Extraction

Rushi Shah

Java, Python, JavaScript, Ruby, and Haskell enthusiast.

Blog Home
About
Twitter
GitHub

Results, Discussion, and Conclusion

Results

This project resulted in a simple, configurable, and useful tool called BlaTeX that lets users write posts in a powerful typesetting language and compile their posts into a beautiful static-site.

Discussion

BlaTeX benefits mathemeticians, computer-scientists, and students of both disciplines who would like to maintain a technical blog about their work. It allows them to write their posts with the same advanced tools they use for their professional work.

The project is also a milestone for the language used to write it. Often the popularity of computer science languages is based heavily on the amount of open-source projects implemented with that language. Haskell is often disparaged because it is rarely used for personal projects, however the Haskell community is trying to refute this claim. Thus, BlaTeX contributes to the growing population of popular projects implemented in Haskell and demonstrates that Haskell is a reasonable choice for future open-source projects.

LaTeX is widely constrained to only being used in large-scale academic or professional projects, which is why it is such a mature piece of software. However BlaTeX broadens the scope of LaTeX itself by allowing people to utilize the strenght of LaTeX on a day-to-day basis.

Future work can concentrate on the limitations outlined above, and specifically the fact that as of right now all posts are PDFs. Pandoc is a tool that converts between file formats (and is also written in Haskell). In order to address the potential limitation that BlaTeX doesn't produce HTML posts, further extensions could explore Pandoc conversions to HTML prior to static-site compilation.

Conclusion

BlaTeX was ultimately an experiment in niche fields. It caters to a (admittedly small) set of people who are either LaTeX enthusiasts, Haskell enthusiasts, or tech-savvy blogging enthusiasts. LaTeX is widely popular in certain fields (specifically math and computer-science), but virtually unknown past that. Haskell attracts fervent users who are passionate about and proud of the code they produce, but is not as popular as more main-stream languages like Java. It may seem like there are a lot of bloggers out there, but only a subset of them are tech-savvy enough to consider a software like this. These factors imply that BlaTeX would not be a very significant project: relevant to only a small population. But just because that population is small does not mean it does not mean it does not exist or can be ignored. Thus the significance of BlaTeX does not lie in its broad reach, but rather in its commitment to solve a very specific problem very well.