Functional Programming
Project Report

A Functional HTML Paser

Université Saint-Joseph de Beyrouth
Faculté d'ingénierie et d'architecture
Institut national des télécommunications et de l'informatique

December 12, 2024
Submitted by

Ghady Youssef
`ghady.youssef@net.usj.edu.lb`

Antoine Karam
`antoine.karam3@net.usj.edu.lb`

Joseph Samara
`joseph.samara@net.usj.edu.lb`

1 Introduction

In this report, we present a functional HTML parser written in Haskell. This tool provides core features such as HTML parsing, DOM tree manipulation, and can output other documents, such as markdown.

2 Parser Combinators

Libraries such as *parsec* provide APIs for working with monadic parser combinators. This allows us to write grammars in a declarative way. Our HTML parser is written using parser combinators, and we provide our own implementation based on [2, 1].

Using parser combinators, we can construct complex parsers from simpler ones. Combinators such as `choice`, `many`, `char`, and `string` allow us to compose complex patterns based on our needs. We heavily rely on monads to chain multiple parsers, which enables us to write cleaner code and improve error handling.

Each parser sequentially consumes input from the input stream, and at each stage, it could potentially fail and return an error instead of the consumed input. This makes it easy to chain multiple parsers in a `do` block or use `<|>` to express an alternative path in the grammar.

3 Grammar

A simple LL(1) grammar suffices for our simple parser. This grammar encodes a subset of the HTML language.

$$\begin{aligned} \text{html} &\rightarrow \text{tagOpen children tagClose} \mid \epsilon \\ \text{children} &\rightarrow \textbf{text} \mid \text{html} \end{aligned}$$

4 DOM-like API

We also provide a set of functions that allow the user to manipulate the `DOMTree` such as `findById` or `findByClass` and `addElement`.

5 Markdown Compilation

Our library allows the user to parse HTML documents and translate them into corresponding markdown files. Any irrelevant HTML tag that does not have an equivalent in markdown is stripped out.

6 Diff Computations

We implemented the `diff` function which computes the differences between two HTML documents. The algorithm implemented in this function is inspired by the React Virtual DOM reconciliation [3]. First, we check if the two elements of the tree are different. If so, we replace the entire DOM subtree. If both elements are of the same type, we check the attributes and compute the differences in both the attributes and the children of the node. The signature for this function is as follows:

```
diff :: DOMTree -> DOMTree -> [Patch]
```

7 Future Work

7.1 Enhanced Error Handling

Since our parser is based on the implementation of monadic parser combinators, we can benefit from the `Monad` and `Alternative` type classes to gracefully handle errors and display meaningful messages.

7.2 Enhanced Parsing for More HTML Tags

We can extend support to include more HTML tags, such as `<strong>`, `<b>`, `<i>`, and self-closing tags.

Currently, our implementation does not handle cases where a `TextNode` and other `HTMLElement`s are siblings. For example:

```
<p>Hello <span>World</span></p>
```

This fails because our implementation supports either `TextNode` **or** `HTMLElement` as the contents of an existing `DOMTree`, but not both simultaneously.

7.3 Support for More Markdown Generation

We can add support for advanced HTML tags and implement proper handling of nested lists with indentation for both `<ol>` and `<ul>`, as well as support for tables.

We can adjust the formatting slightly and consider using parser combinators to build our Markdown generator. Currently, our parser combinator takes input as a `String`; we can extend it to accept input as a `DOMTree` as well.

References

[1] Heitor Toledo Lassarote de Paula. Parser Combinators in Haskell. Online, 2021.

[2] Daan Leijen and Erik Meijer. Parsec: Direct Style Monadic Parser Combinators For The Real World. 12 2001.

[3] The React Team. Reconciliation. Online.