FUNCTIONAL PROGRAMMING CS-IS-2010-1 MIDTERM EVALUATION PROJECT REPORT

Haskell Web Scraper

Saptarishi Dhanuka

Ashoka University

Contents

1 Problem Statement and Requirements

Assigned Project Statement

Develop a scraper using Haskell to extract text and code snippets separately.

1. **Input**: Scrape the text and code snippets from the given text source
2. **Output**: A Word document containing the text and `.txt` file containing the code.
3. **Method**: Write the algorithm to scrape (you can use the `tagsoup` library) and all the input-output facilities using Haskell. Do not use any other language.

1.1 Problem Description

The given web page is made of text and code snippets, which we need to scrape and extract separately into a `.docx` file containing the text portions and a `.txt` file which has the code snippets.
For this, we need to fetch the given web page, parse and analyze its HTML structure to identify the HTML tags of the code snippets and the tags of the rest of the text, so that we can effectively separate them into different documents.

1.2 Requirements

1. The scraper shall be written entirely in Haskell
2. The scraper shall get all text of the given page and write it into a `.docx` file.
3. The scraper shall get all the code snippets of the given page and write it into a `.txt` file.

2 Specifications

1. The scraper will use Haskell HTTP libraries for fetching the HTML content of the given web page.
2. The scraper will separate the code snippets from the rest of the textual content using an algorithm that utilizes the `tagsoup` library to parse the HTML content, along with other standard libraries for string and text handling.
3. The scraper will write the text into a Word document and the code snippets into a `.txt` file mainly using the `tagsoup` and `pandoc` libraries, along with some standard Haskell libraries.
4. The `.txt` file will be formatted such that the code snippets are visibly delimited for readability purposes.
5. The `.docx` file will be formatted in a mannger similar to the original web page in terms of demarcating headings, footnotes and the order of the text.
6. The scraper will handle errors gracefully.

3 Design and Architecture

3.1 High-Level Architecture

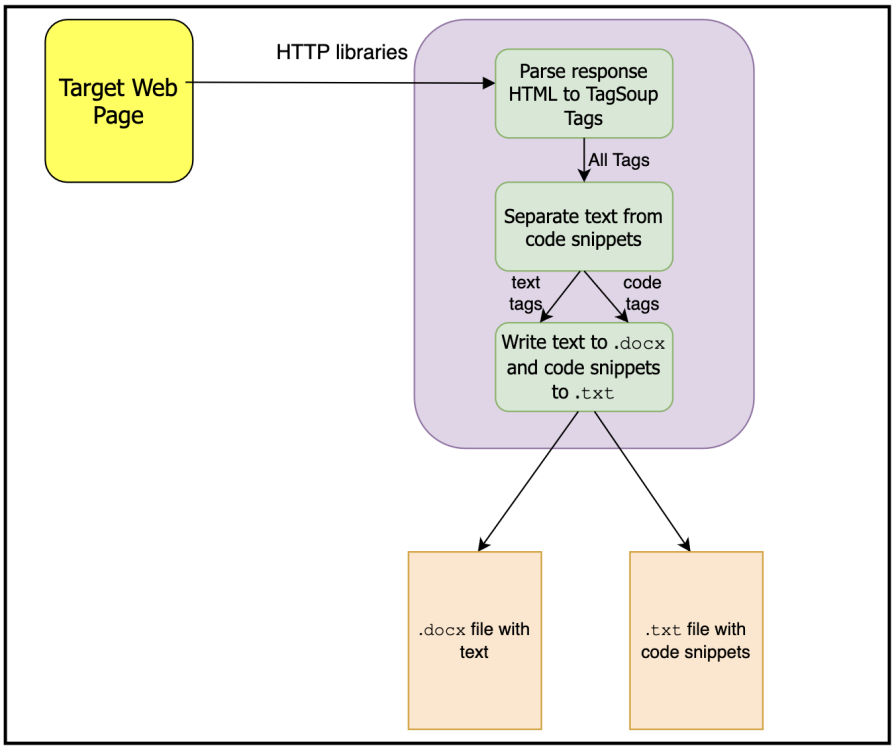

Figure 3.1: High-Level Architecture

The **high-level architecture** consists of the following:

1. Functionality for getting the target web page using HTTP libraries.
2. Parsing the response obtained from the HTTP libraries into Tags from the `tagsoup` library.
3. Separating the text from the code snippets using the descriptions of each Tag from the above Tags
4. Extracting the visible content from the text tags and writing them into a `.docx` file

5. Extracting the visible content from the code tags and writing them into a `.txt` file

The **high-level design** which implements the above architecture consists of the following:

1. A TLS manager for handling HTTPS requests, since the given URL is prefixed with `https`
2. Parsing the url into a request
3. Executing the request with the TLS manager
4. Get the body i.e. the HTML content from the response received after executing the request
5. Parse the HTML into a list of Tags according to the `tagsoup` library
6. Separate the Tags corresponding to the code from the Tags corresponding to the textual content. By inspecting the HTML, we can see that the code snippets are within `<pre>` tags, so we need to separate everything enclosed within these tags from the rest of the HTML content
7. Insert newlines between each `<pre>` tag for formatting purposes.
8. Convert the list of Tag Strings corresponding to the code and to the text each back into an HTML-formatted string, which we then convert into a `pandoc` document as intermediate representation
9. Convert the pandocs into another intermediate format which can then be written into the respective `.docx` and `.txt` files

4 Choice of Tools, Platforms, and Languages

5 Test Plan

6 Prototype Implementation Details

7 Plan for Completion