A Report on
HTML to LaTeX

MERAJ AHMED

2019MCS2565

COP701 Assignment 01: **HTML to LaTeX Converter**

Abstract

LATEX(LaTeX) is largely used in the academic field for the preparation of scientific documents. It used tagging convention to format text.
HTML is standard markup language for web pages. Many documents are in HTML file for which we want to extract its Latex version.
This project enables us to convert ant HTML file to its corresponding Latex file.

Introduction

The objective of this project is to develop a tool which takes input HTML file and produces an equivalent Latex file as output.
For this, we use a compiler tool to convert HTML file to its parse tree format. The parse tree is then converted to Latex output while traversing.
For this project, we use ANTLR compiler tool.

The following features of HTML are considered in this project:

1. head
2. body
3. title
4. a, href
5. font: size
6. center
7. br
8. p
9. h1, h2, h3, h4
10. ul, li, ol, ul, dl, dt, dd
11. div
12. u, b, i, em, tt, strong, small,
13. sub, sup
14. img: src, width, height,
15. table, th, tr, td

ANTLR

ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files.

We use ANTLR version 4.7.2 for our project.

Given a grammar file which uses another lexer file for the token, ANTLR can generate the parser for that language. That parser can automatically generate a parse tree to show us how the grammar matches the input.
ANTLR also automatically generates tree walkers that we can use to visit the nodes of those trees to execute application-specific code.
For each grammar rule derived from variable V (V → something), it gives us two functions enterV and exitV, we can specify application-specific code in these function.
In our project, we use this function to convert our HTML file to Latex equivalent file.
ANTLR v4 automatically rewrites left recursive grammar into non-left recursive equivalents. ANTLR automatically generates parse trees and tree walkers, there's no need for us to build tree grammars in v4.

PROJECT

We use ANLR v4 tool for our project.
We write all lexer rules in a separate file “HtmlLexer.g4”.
We write all parser rules in a separate file “Html.g4”.
Our grammar starting symbol is the “root”.

In “HtmlLexer.g4” we specify lexical rules for each opening and closing tag.
We also specify rule for Html text, Greek text and Latex character in this file.
We specify the rule for white space and tell antlr to skip white spaces.
We also specify rule for Html comments and Doctype and tell antlr to skip these tokens as we will not use these tokens in grammar.

In “Html.g4” we specify grammar rules. Our grammar starting symbol is “root”.
This file recursively defined the grammar for the tag.
For every grammar rule in this file, antlr creates two functions which can be used to write application-specific code. Such as for rule V → something, antlr generates two functions enterV and exitV. These functions have access to the context string which is string of all terminals. We can use that string and check some properties such as href link for a tag. So we can extract the link from that string and generates corresponding latex code.

“HtmlBaseListener.java” is generated java file by antlr. This file contains all the functions discussed above such as enterV and exitV. It also have some extra function like enterEveryRule, exitEveryRule, visitTerminal, visitErrorNode.

We don't directly use “HtmlBaseListener.java” instead we create our own java file “LatexTranslateListener” which extends “HtmlBaseListener.java” and override all the necessary functions.

“LatexTranslateListener.java” is a file created by us which extends “HtmlBaseListener.java” file and override all the necessary function.
In this file, we defined a variable StringBuilder sb which is a very important

variable.
For converting Html to Latex, we append partial latex code to sb variable.
Finally, sb is written into the file after full parsing.

“Main.java” is the important file in this project, it first takes the input file and passes it into the lexer

```
HtmlLexer lexer = new HtmlLexer(input);
```

It then creates a buffer of tokens pulled from the lexer

```
CommonTokenStream tokens = new CommonTokenStream(lexer);
```

It then creates a parser that feeds of the token buffer

```
HtmlParser parser = new HtmlParser(tokens);
```

It then begin parsing

```
ParseTree tree = parser.root();
```

We can print parse tree

```
System.out.println(tree.toStringTree(parser));
```

We create standard walker, it walks every node, and build up partial latex output

```
ParseTreeWalker walker = new ParseTreeWalker();
LatexTranslateListener extractor = new LatexTranslateListener();
walker.walk(extractor,tree);
```

After walk extractor has sb variable which has full latex output.
Finally, we write extractor.sb into a file.

So “Main.java” is the heart of the project as it shows the full process of grammar parsing. It first creates a lexer and then passes the input file to the lexer. Then we pull buffer of token generated from the input file.
Then we create a parser from the buffered tokens. From parser, we create a parse tree. This parse tree is a tree representation of the input Html file.
We then create a walker, while walking we convert the Html to Latex. When walking is complete then we write latex output to file.

“CustomMethods.java” is a helper class for “LatexTranslateListener.java”. We define a function in this file which will be used by “LatexTranslateListener.java”.

How to run

```
antlr4 HtmlLexer.g4

antlr4 Html.g4

javac -cp antlr-4.7.2-complete.jar *.java
//or
javac *.java

java Main sample.html sample.tex
```

ANTLR v4 must be installed in the system.
In the 3rd line, we have to compile all java file using antlr jar file. If this is not already included in classpath then we have explicitly specified antlr jar file. Our project folder has “antlr-4.7.2-complete.jar” file for explicitly specifying jar file.
In last line sample.html is input Html file name and sample.tex is output Latex file name.

We have also created a run.sh bash file which takes two arguments input file name and output file name.